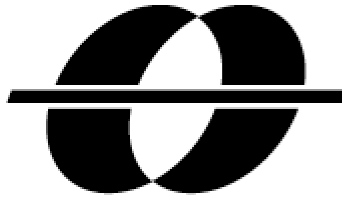

MÄLARDALEN UNIVERSITY
SWEDEN

Model Based Testing for Non-Functional Requirements

Master Thesis in Software Engineering
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

June, 2010

Vijaya Krishna Cherukuri
krisveejay.83@gmail.com

Piyush Gupta
piyushguptaiet@gmail.com

Supervisors:
Pär Pålhed, Ericsson AB and Antonio Cicchetti, Mälardalen University

Examiner:
Mikael Sjödin, Mälardalen University

Contents

ABBREVIATIONS **9**

TERMINOLOGY **11**

1. INTRODUCTION **12**

1.1 ORGANIZATION OF THE THESIS 12

2. BACKGROUND **13**

2.1 TESTING (BLACK-BOX AND WHITE BOX) 13
2.2 WHAT IS MODEL BASED TESTING? 13
2.2.1 Designing the test model 14
2.2.2 Selection of test generation criteria 14
2.2.3 Test Generation 14
2.2.4 Test Execution 14
2.3 MODES OF MODEL BASED TESTING 14
2.4 SCOPE OF MODEL BASED TESTING 15
2.5 BENEFITS OF MODEL BASED TESTING 15
2.6 MBT IN ERICSSON 16

3. THESIS **18**

3.1 PROBLEM DEFINITION 18
3.2 GOALS 18
3.3 LIMITATIONS 18

4. MBT FOR NON-FUNCTIONAL REQUIREMENTS TESTING **20**

4.1 PROBLEMS IN MODELING NON-FUNCTIONAL REQUIREMENTS 20
4.2 NEW PROPOSED METHOD FOR NON-FUNCTIONAL REQUIREMENTS VERIFICATION USING MBT 21
4.3 PROS AND CONS OF THE NEW PROCESS 21

5. TOOLS OVERVIEW **24**

5.1 QTRONIC 24
5.1.1 Workflow 24
5.1.1.1 Modeling SUT 24
5.1.1.2 Test Configuration and Generation 26
5.2 MOTES 27
5.2.1 Workflow 27
5.2.1.1 Modeling IUT 27
5.2.1.2 Prepare Test Data 28
5.2.1.3 Importing Models and Test data 28
5.2.1.4 Defining Test Coverage/Goal 28
5.2.1.5 Selecting Test Generation Engine 29
5.2.1.6 Generating the test cases 29
5.3 MATELO 29
5.4 MODELJUNIT 31
5.4.1 Workflow 31

5.4.1.1 Modeling SUT ... 31
5.4.1.2 Test Configuration ... 33
5.4.1.3 Test Generation ... 34
5.5 MBT Tigris ... 34
5.5.1 Work flow ... 34
5.5.1.1 Modeling SUT ... 35
5.5.1.2 Test Configuration ... 35
5.5.1.3 Test Generation ... 37

6. TOOLS COMPARISON ... 38

6.1 Functionality and Usability ... 38
6.1.1 Graphical User Interface ... 38
6.1.2 Libraries ... 45
6.1.3 Design Rules ... 45
6.1.3.1 Modeling Strategy ... 45
6.1.3.2 Test Data Specification ... 46
6.1.3.3 Test Configuration Definition ... 46
6.1.3.4 Modeling Language ... 46
6.1.4 Document Generation ... 47
6.1.5 Requirement Tracking ... 47
6.2 Flexibility ... 47
6.2.1 Operating System ... 47
6.2.2 Tools Integration ... 48
6.2.3 Script Languages ... 48
6.2.4 Adaptability ... 49
6.3 Pricing ... 49
6.3.1 Licensing and Support ... 49
6.3.2 Education ... 50
6.4 Support for Non Functional Testing ... 50
6.5 Applicability in current SUT Environment ... 50

7. CASE STUDIES ... 51

7.1 System Under Test : IP Security Protocol ... 51
7.2 Tools Selection ... 53
7.3 Case Study with ModelJUnit ... 53
7.3.1 Objectives of case study ... 53
7.3.1.1 Modeling the SUT ... 54
7.3.1.2 Test Generation ... 57
7.4 Case Study with Qtronic ... 57
7.4.1 Objectives of case study ... 58
7.4.1.1 Modeling the SUT ... 58

8. DISCUSSION ... 63

8.1 Qtronic ... 63
8.2 ModelJUnit ... 66

9. CONCLUSION ... 69

10. FUTURE WORK ... 70

11. REFERENCES 71

Index of Tables

Table 1 : Sample QML code 25
Table 2 : Sample ModelJUnit model code 32
Table 3: List of Generators for MBT Tigris 36
Table 4: List of Stop Criteria for MBT Tigris 36
Table 5 : Test Configuration code for ModelJUnit 56
Table 6: QML code for handling maximum number of objects creation 61
Table 7 : Data Values for the Initial Functional Models in Qtronic 63
Table 8 : Data values for Requirement 1 using Approaches 2 and 5 in Qtronic 64
Table 9 : Data values for Requirement 2 using approach 5 in Qtronic 65
Table 10 : Data values of Requirement 3 with different look-ahead depths 65
Table 11 : Data Values of the initial functional model in ModelJUnit 67
Table 12 : Data Values of Requirement 1 modeling in ModelJUnit 67
Table 13 : Data Values of Requirement 2 modeling in ModelJUnit 68

Index of Figures

Figure 1: MBT Process 15
Figure 2: MBT process in Ericsson 17
Figure 3: MBT process in Ericsson for Modeling Functional Requirements 17
Figure 4: New Process flow for Verification of Non-Functional Requirement 22
Figure 5: Workflow of Qtronic 24
Figure 6 : Modeler Provided with Qtronic 26
Figure 7: Workflow in Motes [12] 27
Figure 8: MaTeLo Modeler Window 30
Figure 9: MaTeLo Testor Window 30
Figure 10: ModelJUnit Generated Graphical EFSM Model 33
Figure 11: Workflow of MBT Tigris 35
Figure 12: Qtronic GUI Main Window 39
Figure 13 : Motes Project Structure 39
Figure 14: Motes Directive Definition Window 40
Figure 15 : Motes Main GUI Window 40
Figure 16 : Motes GUI showing Error Message 41
Figure 17: ModelJUnit GUI Main Window 42
Figure 18: ModelJUnit GUI Test Configuration Window 42
Figure 19: MBT Tigris Main Window 43
Figure 20: MaTeLo Modeler Windows 44
Figure 21: MaTeLo Testor Window 44
Figure 22: IPSEC Implementation in the System 52
Figure 23: Graphical model covering functional requirements of SUT 55
Figure 24: Graphical model covering both functional and Non-Functional requirements of SUT 56
Figure 25: Existing Qtronic Functional Model 59
Figure 26 : Qtronic approach for NFR by introducing new state 60
Figure 27: Addition of new transition approach in Qtronic 62

Abstract

Model Based Testing (MBT) is a new-age test automation technique traditionally used for Functional Black-Box Testing. Its capability of generating test cases by using model developed from the analysis of the abstract behavior of the System under Test is gaining popularity. Many commercial and open source MBT tools are available currently in market. But each one has its own specific way of modeling and test case generation mechanism that is suitable for varied types of systems. Ericsson, a telecommunication equipment provider company, is currently adapting Model Based Testing in some of its divisions for functional testing. Those divisions haven't yet attempted adapting Model Based Testing for non-functional testing in a full-pledged manner. A comparative study between various MBT tools will help one of the Ericsson's testing divisions to select the best tool for adapting to its existing test environment. This also helps in improving the quality of testing while reducing cost, time and effort. This thesis work helps Ericsson testing division to select such an effective MBT tool. Based on aspects such as functionality, flexibility, adaptability, performance etc., a comparative study is carried out on various available MBT tools and a few were selected among them: Qtronic, ModelJUnit and Elvior Motes.

This thesis also helps to understand the usability of the selected tools for modeling of non-functional requirements using a new method. A brief idea of modeling the non-functional requirements is suggested in this thesis. A System under Test was identified and its functional behavior was modeled along with the non functional requirements in Qtronic and ModelJUnit. An experimental analysis, backed by observations of using the new proposed method indicates that the method is efficient enough to carry out modeling non-functional requirements along with modeling of functional requirements by identifying the appropriate approach.

Acknowledgements

We take this opportunity to thank and acknowledge the co-operation and support – both moral and technical extended by people who helped us in our endeavor to complete the thesis. We shall always cherish our association with them.

We would like to thank Mr. Pär Pålhed, our Project Supervisor at Ericsson who has been a constant source of advice, encouragement and above all for the continuous support he has provided throughout the thesis work.

A Special thanks to Mr. Antonio Cicchetti, our Project Supervisor at Mälardalen University, for taking his time to listen to our ideas, guiding us well in writing the thesis report and providing us his valuable feedback.

We would also like to thank Mr. Hakan Fredriksson, Mr. Ebrahim Amirkhani and all other test team members of the project. Their assistance and camaraderie helped us to tide over the difficulties encountered.

A special mention to Mr. Michael Lidén and Mr. Athanasios Karapantelakis from Conformiq for their training and support in learning the tool.

Last but not the least; we would like to thank our student group from Tata Consultancy Services at Mälardalen University, who stood by us in both good and bad times, and made it a wonderful journey throughout our Master's program.

Abbreviations

AH	Authentication Header
API	Application Programming Interface
CSV	Comma Separated Value
CPP	Cello Packet Platform
CPU	Central Processing Unit
DC	Design Configuration
EFSM	Extended Finite State Machine
ESP	Encapsulated Security Payload
FSM	Finite State Machine
GNU	GNU's Not UNIX
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IKE	Internet Key Exchange
IPSec	IP Security
IUT	Implementation under Test
JAR	Java Archive File
MBT	Model Based Testing
MC	Marcov Chain
MDE	Model driven Engineering
MDPE	Model Driven Performance Engineering
MTL	Motes Transition Language
OOPS	Object Oriented Programming Specification
QML	Qtronic Modeling Language
QTP	Quick Test Professional
SA	Security Association

SAD	Security Association Database
SPD	Security Policy Database
SUT	System under Test
SysML	System Modeling Language
TTCN-3	Test and Test Control Notation version 3
TC	Test Case
TCL	Tool Command Language
UI	User Interface
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Terminology

Functionality
Functionality can be defined as the supported operations, capabilities and utilities provided by the software.

Usability
Usability is the quality property of software that defines how easy the software can be used. Some usability attributes of the software can be its learnability, efficiency, error Handling and debugging capability.

Flexibility
Flexibility is also the quality attribute of software that defines its ability to change or adapt easily in response to different configurations and system requirements.

FSM
Finite state machine is used to model behavior of the SUT by means of graph. It has finite number of states. Each of its nodes represents a state and arcs which represents the transitions from current state to next one, which is performed by means of certain actions.

EFSM
Extended Finite State Machine is an enhanced version of FSM also used to model behavior of SUT. In EFSM, each transition might be associated with the Boolean expression known as Guard Condition. If this guard is evaluated to true then only transition is fired, some actions are performed and SUT changes its state from current to next.

Marcov Chains
A Markov chain is also used to model behavior of SUT where next state depends only on the current state. There is a transition that causes the changes state, and the transition probabilities associated with various state-changes. These sets of all states and transition probabilities collectively called as Markov chain.

Test Harness
A test harness is a collection of tools and test data configured to automate the test execution process. It consists of Test execution engine and the Test script repository. It performs the task of calling functions with supplied parameters and prints out and performs analysis on the results. The test harness is glue to the SUT, which is tested using an automation framework.

1. Introduction

For decades, billions of dollars are lost due to software errors. Sometimes they affect lives also. They are results of poor software quality. Software testing has become one of the important and crucial factors in Software Development Life Cycle. As per researcher's estimate, on an average approximately 50-60% of total development time is dedicated for Software testing in Industries [16]. As the needs of developing new kinds of software related products increase, the demand for releasing them in to market with efficient quality and additional advanced features at cheaper cost increases in order to withstand the competition. This results in designing of more and more complex systems in lesser amount of time. More complex the system is, even more is the effort required to test it. A single and simple change in the system may increase the time and effort of testing the system proportionally. Also with the increasing demand in software products, customers expect more reliable, efficient and a quality software product that contains advanced features and functionality. The competition between many companies forces the manufacturer to deliver the product with above prerequisites within a short period of time. This leads to a short period of testing time. There comes the need of test automation. Automation of testing not only reduces the effort and time but also the cost incurred as testing needs to be done regressively when meeting tight project schedules.

The traditional manual preparation of test cases is done by analyzing the functionality and working of the system. But in automation techniques, the testing process involves usage of tools and scripts to generate test cases and perform testing in a quick and efficient manner. One good automation technique is Model Based Testing. It is nothing but the application of model based designing to software testing process. This process involves development of a model that describes the test cases, test data and the system under test execution environment and using that model test cases are generated. The main advantage of using MBT is that the time require for modeling the behavior is less than manual test case writing and execution. Also it generates wide range of test cases which more often may not be derived manually. MBT is basically used for functional black-box testing. Functional requirements of a system that describe the behavior of the System under Test can be easily modeled as the functional aspects of the system involve specific sequence of actions. Whereas the non-functional requirements require analysis models defining both the structure and behavior of the analyzed system, resource requirements, branch probabilities, and details about factors due to contention of resources [17]. For the same reason it is difficult to have a generic way of modeling non-functional requirements of a system. Though MBT is used to some extent for robustness testing, there is no full-pledged established way of checking it for other non-functional attributes such as performance, security etc. It is still an area under development.

1.1 Organization of the thesis

This Master's Thesis has the following outline. Chapter 2 provides background information for several key areas related to this Master's Thesis. Chapter 3 presents the problem formulation. Related information on use of MBT for testing non-functional requirements is provided in Chapter 4. Chapter 5 gives an overview of some MBT tools. Section 6 provides a comparative study on the tools discussed in Chapter 5. In Chapter 7, case study on a telecom system for testing Non-Functional requirements using MBT tools is presented. The discussions on the results are presented in Chapter 8.

2. Background

This chapter provides the background information regarding our thesis work. Section 2.1 discusses about black-box and white box testing and introduces Model-based testing. The consecutive sections discusses about the modes, benefits and existing limitations of Model Based Testing. Section 2.6 gives an overview of current MBT process in Ericson.

2.1 Testing (Black-box and White Box)

Software testing and fault detection activities are mostly inexactly and inadequately understood, especially in the complex systems. But they are very much crucial for the success of the project and if in case it is a product, it is necessary to ensure better quality [18]. Though manual testing may yield many defects in a software application, it is a laborious and time consuming task. Automating the tasks may reduce the time to more extent. Once the tests are automated, they can be run quickly and instant results are generated.

In the scope of test automation there exits two prominent methodologies, Black-Box and White-Box techniques [19]. Black box testing refers to the testing of the response of the system in terms of its behaviour. Black box testing involves exploration of various possible inputs and the corresponding possible outputs from them. It does not bother about the internal implementation of the system or the process adapted for the functionality. White-Box testing on the other side looks under the cover of the system, taking into consideration the way the system is implemented, internal data structures and possible logic involved in the coding etc.

The difference lies in the areas on which both the testing methodologies focus. Black-box testing focuses on the outcome irrespective of the inside processes involved to achieve those outcomes. A White-box testing concern with the details and the testing perspective is complete only if the sum of all the parts contributes to the system as whole [20].

One of the variants of black box testing is the Model Based testing. As the time preceded Model based Testing is used for adapting to White Box methodology also. The next sections briefly discuss about what Model-based Testing is and how it can be used for efficient generation of test cases.

2.2 What is Model Based Testing?

More often complexity is resolved by considering abstraction. Same is the case in software testing. Considering the software testing at an abstract level not only helps in reducing the complexity but also be useful to generate efficient test cases in short period of time. Model based testing is one technique that lets the tester to consider the System under Test in an abstract manner. The behaviour of the whole SUT is modelled to generate test cases. According to Wikipedia , MBT is defined as "***software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test (SUT)*** " [1].

Model based testing is a process used to generate abstract test cases by using the abstract formal models of the SUT. Later these abstract test cases are converted into executable test cases and the test cases thus generated are automated. The following are the four important stages of MBT application in industry.

2.2.1 Designing the test model

The abstract behaviour of the entire system is modelled by using standard modelling languages such as UML, SysML etc. Most commonly used methods are Finite State Machines. But they are large and abstract for modelling real systems. So Extended Finite State Machines are generally used. For storing data EFSMs have the facility of adding state variables and also they facilitate the guard conditions and actions that are required to update state variables during transactions. The transactions are mapped with the requirements to ensure traceability and later used to generate test cases.

2.2.2 Selection of test generation criteria

A huge number of test cases may be generated usually based on the designed model. But prioritization of the test cases is needed in order to structure the testing process and reduce test effort. Standard test generation criterion such as boundary value analysis [2], equivalence class partitioning, cycle coverage etc are generally used. But the underlying factor is the selection of any criteria that covers all the requirements.

2.2.3 Test Generation

This is an automatic process done by using some of the available MBT tools in market. Using the designed model, high level abstract test sequences are generated by the tool that uses some branch traversal algorithms. These abstract test sequences are independent of the language that is used to write the test cases and of the SUT environment [2]. Each test sequence contains the input parameters, required actions and expected outputs. Since they are at abstract level, they cannot be directly executed in the SUT. Modifications are required to be done in the abstract test sequences in order to automate. Abstract test cases are translated into executable test cases by mapping the data values in the data model to real values in the SUT [21]. This can be done by using an intermediate platform called SUT adapter which is used to covert and automate the abstract test cases. The generated test cases from the test model are structured and organized into multiple test suites and can be saved in repositories such as HP Test Director and IBM Rational Quality Manager repository etc.

2.2.4 Test Execution

The generated and modified test cases are executed in a standard and automated test environment or even run manually on SUT, and the results are compared with that of expected results. If there are failed test cases, they are verified whether it is a fault or bug in SUT or due to construction of behavior during modeling.

2.3 Modes of Model Based Testing

The timing related to test case generation and test case execution gives rise to another dimension of MBT which is referred to as the modes of MBT. There are two modes of MBT, Offline and Online. Online testing is the process in which both the test generation and test execution processes are combined and used at a time, where the instant result is used for pruning the MBT process. In Offline testing mode, test execution is followed by test generation phase. The test generation phase generates an artifact called test suite, which is later interpreted by the test execution phase of MBT [22].The test cases are generated before they are run on the SUT in offline mode.

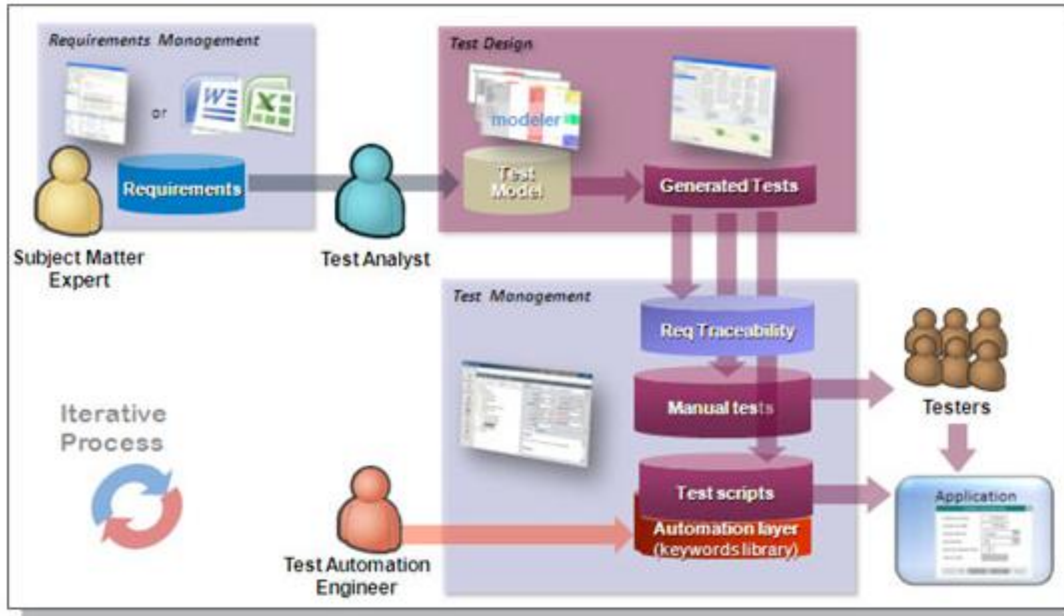

Figure 1: MBT Process

The online mode is useful for identifying the non-deterministic paths of SUT whereas in offline mode it is difficult as the test cases does not involve creating sequences, but are created as graphs and trees. But the advantage of having offline mode is that it helps for efficient management of test cases which further is useful for executing same test cases in different kind of environments at different times [23].

2.4 Scope of Model Based Testing

Currently MBT is used mostly in system and acceptance testing phases of software development life cycle. It may not be efficiently used for unit and component testing of systems. It is a time consuming effort and moreover a repetitive task as the functional aspects of the individual entities can be covered when the whole system or sub-system is modeled in the later phases.

2.5 Benefits of Model Based Testing

The main benefit of MBT is to generate wide range of test cases in short span of time. Even though modeling takes considerable amount of time, it will always be less than deriving the test cases manually. Moreover if there is a change in any of the requirements, it is a heavier and painful task to reflect the similar changes in test cases or scripts written manually. But in MBT, it is easier to modify the model than to modify the test suite. Apart from that, MBT form of testing is less ad-hoc and adapts systematic coverage of test cases providing a complete control on test coverage. It is easy to detect flaws in modeling at earlier stages based on the analysis of test cases that are derived. MBT also facilitates easy traceability by considering the matching of test cases with the requirements.

MBT helps to establish a systematic way of testing. It sticks to main goal of testing, i.e., finding faults in SUT. But the number of faults detected depends upon the way in which the system behavior is modeled. A wrong understanding of requirements may lead to incorrect modeling of the behavior and thus leads to wrong faults which can be at times confusing. It is like adding new faults rather than finding some

existing ones. Factors such as test selection criteria chosen from the experience and expertise of the Tester's knowledge with SUT drives modeling and test generation.

2.6 MBT in Ericsson

In the previous section a brief overview of MBT was given in four stages: Test model design, Selection of Test Criteria, Test Case generation and execution of test cases. A general and detailed MBT working process of Ericsson is given in this section.

In Ericsson, currently Conformiq Qtronic tool is used for MBT. It is an eclipse-based interactive workbench for automated test design. Qtronic is used in Ericsson to generate test suite for functional black-box testing. A SUT is identified initially. All the input documents such as customer requirement specifications, Implementation proposals, functional description documents and functional specifications are read and understood by the test model designers initially for building the models. It is very much important to ensure the compliance of the details in documents with the working system of SUT because the only input for test modeling is the specific set of documents mentioned above. This is a primary prerequisite for any MBT approach.

Once the requirements are clearly understood, they are modeled incrementally using Qtronic. Qtronic uses the combination of UML State Transition diagram and an action language which is a variant of action java (similar to java and c#) as the modeling language. It is called as QML. Conformiq Modeler is used to draw the UML state Transition diagram. The use of diagrams is optional. Whole model can be developed just by writing the QML code. But usage of diagrams is recommended for easy understanding of the functionality of SUT.

Using the model, a set of abstract test sequences is generated. These test sequences can be stored and managed using a test management tool such as HP Quality Center. At least one scripting backend has to be set up in order to generate the test cases. These backbends are specific to the test design configuration. A valid scripting backend JAR file should be provided. Some backbends such as HTML, Perl, TCL and TTCN-3 are provided with the Qtronic itself. Any of these backend needs to be pre - configured in order to generate test cases in the desired format.

The generated test sequences or test cases are then reviewed by the experts. If there are any inconsistencies in the test cases or bugs, they are verified again in the model and are modified accordingly. If any modification is done to the model, the test cases have to be regenerated to reflect the changes. This is the process of test case refinement. The test cases should be regenerated even if there is a change in any of the Qtronic settings or any test design configuration.

Once a satisfactory model is achieved, the test cases can be exported through the active configured scripting backbends. The exported test cases are used in the test harness in order to convert them to executable test scripts. These scripts are executed on the SUT using a test automation framework and the results are captured. These test results are also stored and managed in the test management tool HP Quality Center. The whole existing process is depicted in Figure 2 [24]. Another brief version of process is given in Figure 3.

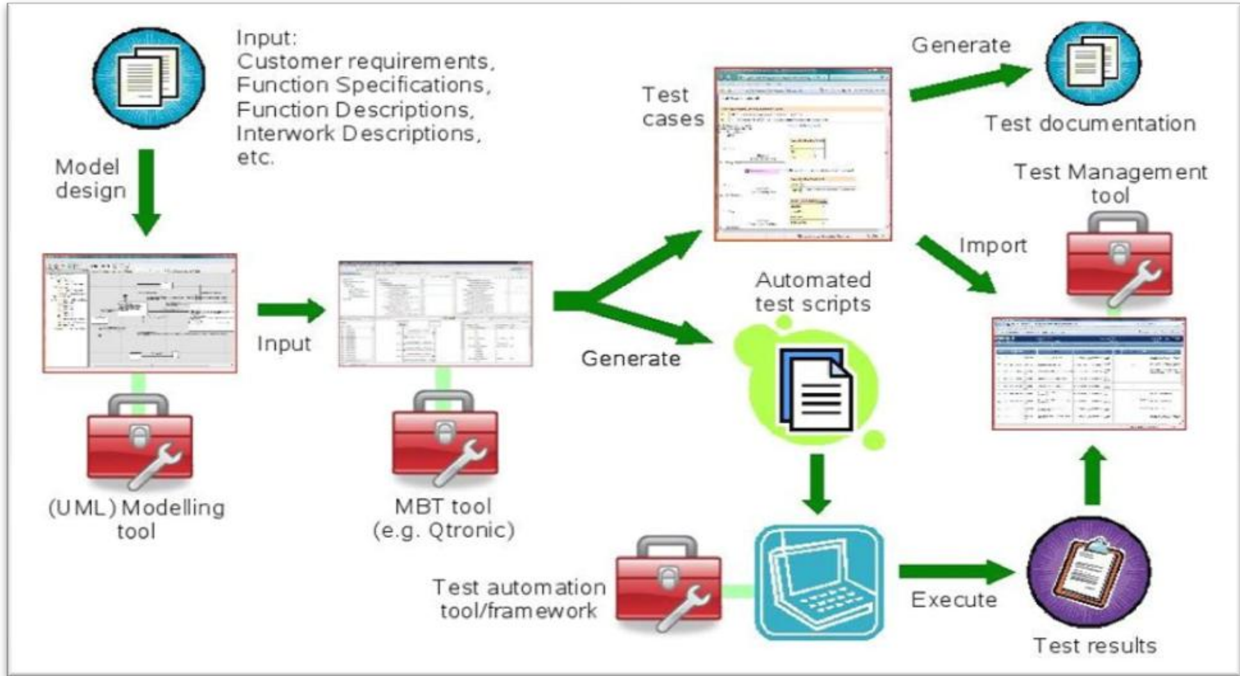

Figure 2: MBT process in Ericsson

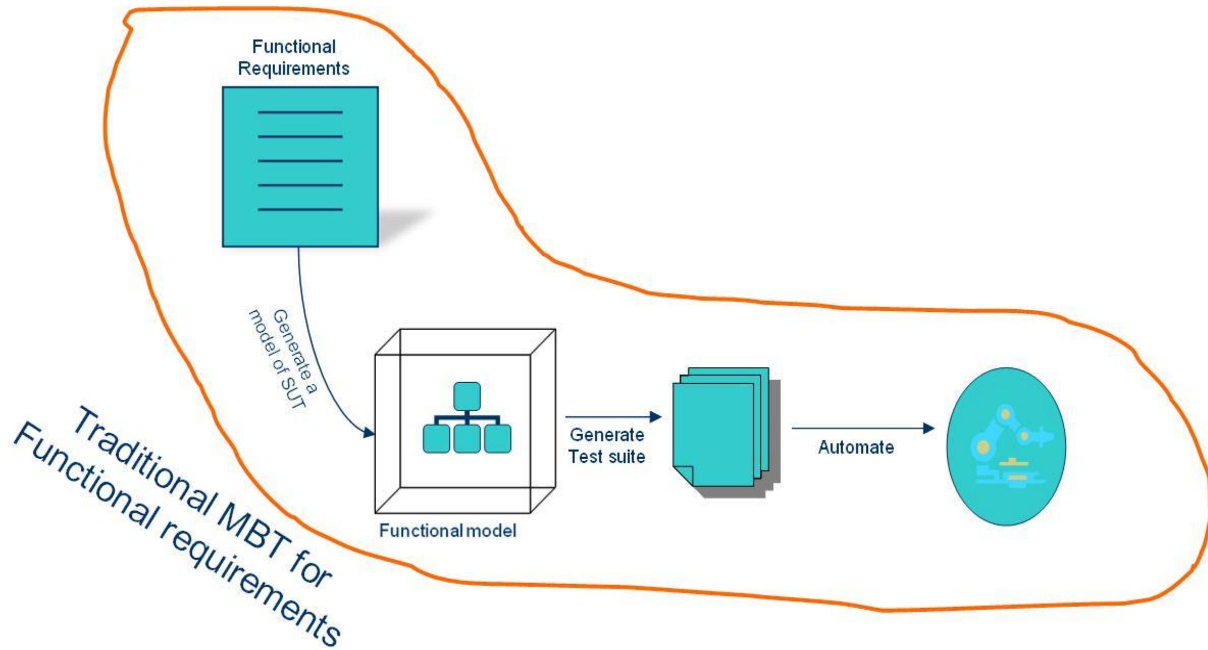

Figure 3: MBT process in Ericsson for Modeling Functional Requirements

3. Thesis

This section introduces the problem statement of our thesis work. Then it divides the problem into several sub-problems and finally tries to offer a solution to the bigger problem by solving the smaller ones individually.

3.1 Problem Definition

This thesis work is done in Ericsson. It has started adapting Model Based Testing for functional testing in some of the software divisions of their products. They currently use a tool named Qtronic from Conformiq for functional verification at system level.

In general MBT is not easily adaptable for modelling non-functional requirements of the system. Any new approach for modelling non-functional requirements can be a cost-beneficial and time-saving effort. This thesis is an attempt for finding such an approach.

A wide range of MBT tools are available in market currently, but the suitability and adaptability of the tools to the Ericsson's current test environment has to be determined. Such a process involves taking into consideration many factors such as cost, ease of adaptability, amount of extra effort needed to educate on tools etc. A summary of those related factors for some tools will be helpful in carrying out a decision by the authorities regarding which of the tools can be used.

3.2 Goals

This thesis is mainly about exploration of possibility of applying MBT techniques for non-functional testing in Ericsson, using various MBT tools like Conformiq Qtronic, Elvior Motes and ModelJUnit etc. One goal of the thesis is to do a comparative study of the tools based on some of the factors such as Functionality, Usability, Flexibility, Performance and Pricing in general. Another goal of the thesis is to try and devise a new efficient way of using MBT techniques in modeling and testing the non-functional requirements.

In order to streamline the study, only performance aspect is taken into consideration. A new process is proposed based on the current research carried out in MBT in regular and with respect to the way of working in Ericsson specifically. This proposal is briefly described in the thesis. Based on the proposal, a SUT is identified from the CPP product of Ericsson. The process is applied on the SUT and test cases are generated using different tools. Once if it is found that this proposal works, the same model is used to generate test cases using different available MBT tools in the market, and then a bench marking is done based on several performance metrics of the tools such as ease of automation, test case generation time, cost, limitations, work-arounds, implementation of test harness and availability of easy documentation etc.

3.3 Limitations

- Exploring all categories of non-functional requirements for modeling is a huge task. Instead, we chose upon the performance related specifications of the SUT to be modeled as they are of higher priority in any telecommunication system.

- Generic UML models cannot be exported to all tools. It is a time-consuming task to model separately the same SUT behavior in different tools in order to prototype. When benchmarking the tools, we

are only considering aspects such as Functionality, Usability and Flexibility. The Performance factor of the tools is considered only for the case studies which we have conducted on two tools only, Conformiq Qtronic and ModelJUnit.

- Within the given time limit of thesis, the work is confined to only the generation of test sequences by modeling the behavior of the SUT. The test execution part is out of the scope of this thesis.

4. MBT For Non-Functional Requirements Testing

Model based testing is basically used for functional black box testing. To some extent it is used to test robustness specifications such as invalid inputs and security related specifications such as authorization and authentication also. Black box testing does only consider the behavior of the SUT and discard the internal structural specification of the system, where as in white box testing the internal structure of the SUT is considered during generation of tests. Ideally efficient test cases are generated if the white box model of SUT is considered.

For testing of non-functional requirements related to attributes such as performance, MBT is not widely used. It is still an area under research. Researchers have been working on the specific ways to use MBT techniques in testing non-functional specifications especially performance testing. In this section some of the difficulties in adapting MBT techniques for performance testing are discussed.

As said earlier, MBT is currently used for functional requirements testing. The general behavior of the SUT can be described by its functional requirements. Hence it is easy to model functional requirements, whereas it is a different case for non-functional requirements. Apart from models describing the functional behavior, other aspects such as analysis models depicting the implementation structures, additional resource requirements for the models etc are required for modeling non-functional requirements. Moreover, since modeling deals with higher level of abstraction, it is difficult to provide reasonable estimates for non-functional properties of the SUT. In that way it is difficult to provide generic solution because it is difficult to specify the non-functional properties of the system at a given point of time.

4.1 Problems in modeling Non-functional Requirements

The problems in testing non-functional characteristics of a system using MBT techniques are discussed earlier. In this section we discuss about the basic issues in modeling of non-functional requirements to generate test cases in context to Ericsson's testing process and based on that propose a solution for modeling them.

The primary obstacle comes in segregation of testing functional and non-functional characteristics. As mentioned earlier, it is a complicated effort to observe behavioral patterns in non-functional requirements of the system. It is closer to understanding of internal implementation of structures, external resources and factors affecting them and branch probabilities etc. One solution can be modeling the system by combining functional and non-functional aspects of the SUT. This can lead to a new type of testing process in the organization. The idea of combining them together for modeling sprout from the concept of Model Driven Performance Engineering [25].

MDPE is the technique derived by Mathias Fritzsche and Jendrik Johannes for Model Driven Development. They observed that iterative Performance analysis during modeling of a system, instead of addressing during the implementation of actual system and make it run, contributed in reducing the total turnaround time of the project and was helpful for on-time delivery with good quality. For this purpose they proposed MDPE as an extension of Model driven Engineering, by combining performance engineering with it. At every phase of incremental development of the model of the system, they

started considering the performance. Based on MDPE, in this thesis we propose a similar process for modeling non-functional requirements using existing functional model developed for MBT.

Secondarily, some of the characteristics testing in Ericsson require additional testing equipment and also manual intervention since it is in the manufacturing of telecom equipment. Those characteristics are difficult to automate even if some can be included in the models. Such characteristics can be tested separately and manually. The idea behind including them in the model even if it is difficult to automate is to generate some special abstract test sequences that may not be thought of while designing the test cases manually.

Performance related characteristics are chosen to be modeled in this thesis to streamline the study. Other related non-functional attributes such as robustness, security etc can be modeled by using standard MBT techniques and also rather it is a common practice.

4.2 New Proposed method for Non-Functional Requirements Verification Using MBT

Inspired by MDPE, the proposed solution is the inclusion of non-functional requirements (or rather performance characteristics) in the existing functional model of the SUT. This reduces the effort of modeling the non-functional requirements separately. Before including them, logical grouping of the requirements should be done based on the similarity among the requirements. Then a feasibility analysis is done about including these grouped requirements in the existing functional model. This can be done either by including them in the existing states or by creating new states depending on the type of transformations that are required. Once a stabilized model is developed, the test cases are generated.

If the chosen requirements could not be logically grouped, or distinct or dependent on any other factors such as requirement of additional tools or involvement of physical interferences etc, those requirements can be modeled separately or tested manually based on the ease of testing. A brief overview of the new process included in the existing process is depicted in the Figure 4.

4.3 Pros and Cons of the New Process

The following are some of the factors on which some of the standard MBT benefits are dependent upon. This section describes the effect of new process on those factors in order to retain the usual benefits.

Time

The time to model is proportional to the total turnaround time of the project. In case of a development project, the modeling of behavior for MBT is developed in parallel to the design and implementation and then the test cases are generated and automated. This reduces the total testing time of the project. Since we are including the non-functional requirements in the existing model and we are not writing any model from scratch, this initially reduces time and effort in basic modeling. Assuming that the functional model already exists for maintenance, the time required for initial modeling is not considered. Only the time required for additional inclusion of non-functional requirements is taken into consideration. Since the new inclusion generates test cases automatically by tool, it will be obviously less than the manual generation time for those test cases. That accounts for the reduction in total turn-around time of the maintenance project.

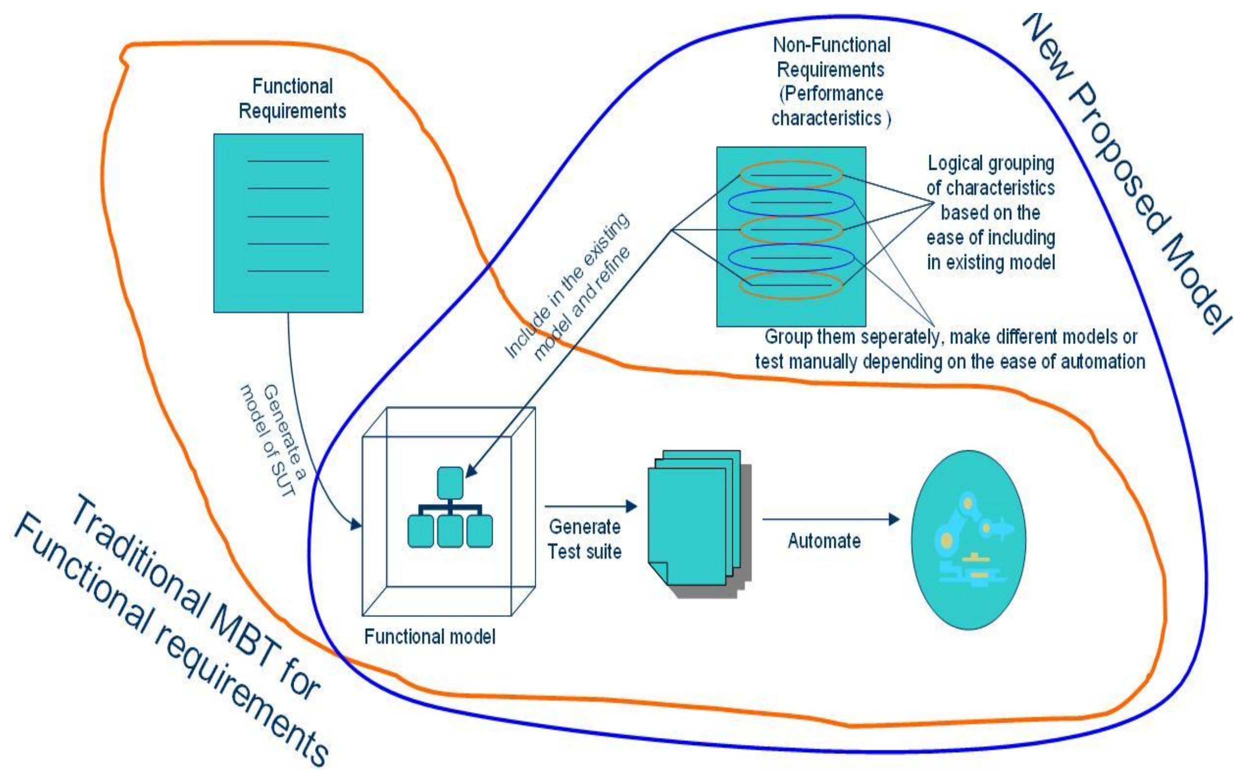

Figure 4: New Process flow for Verification of Non-Functional Requirement

If the time invested in modeling the system behavior is more than the usual manual analysis and generation of test cases, then it is a hindrance to the overall project time rather than acting as a benefit. In the given process, if the characteristics are totally distinct and cannot be grouped, the time invested in modeling all of them separately may be sometimes more than the manual testing. This process cannot be used in such a scenario.

Quality

In MBT, quality can be measured in terms of the coverage that is done in the model of the SUT behavior. Initially, in the functional model particular coverage criteria can be chosen based on the tool that provides the facility. For example the deeper the look-ahead depth in Qtronic more is the number of more is the coverage and hence more number of test cases are generated.

In the case of this method the approach that followed for modeling to some extent determines the coverage. So quality in this context is independent of the proposed method but purely based on the way the behavior is modeled. Same is applicable for non-functional requirements also.

One of the major advantages of using MBT is to derive key test sequences that are unexpected or missed during manual analysis. Some of the state combinations and branch traversals can give new set of test cases which were non-traditional but are still important.

By including the non-functional requirements in the existing functional model, the chances of new branch iterations will be more and this may result in generation of additional test cases that may not be

usually derived from manual analysis of the requirements. This adds to the better quality of testing in lesser period of time by increasing the coverage.

Cost

The cost incurred in adapting the basic MBT for functional requirements remains unaffected by introduction of this new process for non-functional requirements. Basically, the cost of a testing project is affected by the factors such as the tool selected for the MBT, number of resources working on the new tool and the number of man hours spent on modeling. By introducing the above process the only factor that is affected is number of man hours spent in modeling. But as usual the cost benefits that arose for general functional MBT are applicable for this process too as this process is not introducing any new tools, but only a new method.

5. Tools Overview

This section provides the basic overview of the several tools studied as part of the thesis goal. It is required explore on different tools available for MBT in the market and do a comparative study on the tools based on the factors such as Functionality, Usability, Flexibility, Performance etc. The performance factor of the tools Qtronic and Model JUnit is discussed in Chapters 7 and 8 where the case-studies are briefly discussed. Following sections give a brief overview on the work-flow of different MBT tools like Conformiq Qtronic, Model JUnit, and Elvior Motes etc.

5.1 Qtronic

Qtronic is the automated test design tool from Conformiq, which automates functional tests of the System and Software. It facilitates the System Tester by automatic generation of human readable test cases from the expected system behavior of the System under Test (SUT). A high-level system model is given as input to the tool. The tool generates a set of test cases calculated mathematically from the given model. The test cases later can be exported to customizable formats such as TCL, TTCN-3, and HTML etc.

5.1.1 Workflow

The following diagram shows the workflow of Qtronic tool.

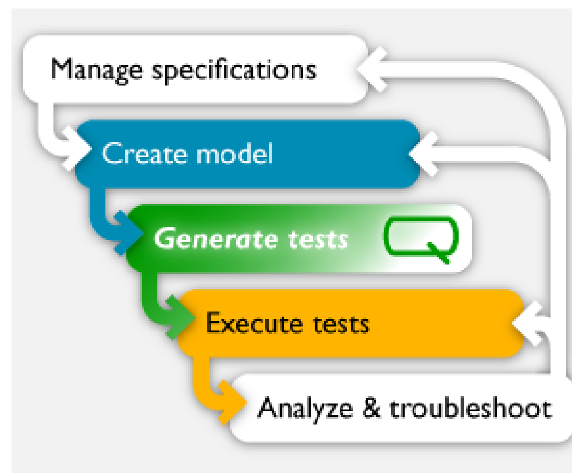

Figure 5: Workflow of Qtronic

5.1.1.1 Modeling SUT

The first step as in every MBT tool is a clear understanding of specifications. The experience and knowledge about SUT behavior of a tester should help him to visualize the transitions based on the understanding of specifications. The next step is to model those specifications. Today widely used modeling tool is UML 2 whose syntax is based on diagrams. UML supports 13 different types of diagrams such as use case diagrams, class diagrams, sequence diagrams and state transition diagrams etc. These diagrams are basically categorized as structural and behavioral diagrams. Since MBT is mainly based on

the consideration of whole system behavior, Qtronic uses state machine diagrams, class diagrams and component diagrams depending upon the platform used for modeling as input. The output generated test cases can be visualized in the form of message sequence diagrams. There is no specific diagrammatic representation of data in order to manipulate the actions. Hence a programming language usage was necessary. UML2 facilitates the textual notation of the actions in the transitions through a language called action language. Conformiq uses the modeling language called QML (Qtronic Modeling Language) which is a combination of state transition diagrams and an action language which is a variant of Java.

Here is a sample of Qtronic model which contains the action code and corresponding state machine. The action code is written in some file called 'Sample.java'.

Table 1 : Sample QML code

```
record Msg{ }

system {

Inbound in: Msg ;

Outbound out: Msg ;

}
class NewClass extends StateMachine {}

void main ( )
{
    NewObject = new  NewClass( ) ;

NewObject.start( );
}
```

The first line signifies the message type that is equivalent to record which is a mix of data types which can encapsulate both data types and functions. The next section defines the external interface of the system. The system block contains the declarations of the Inbound and Outbound interfaces or ports. The identifiers in and out are the names of the interfaces. The data that falls after the semicolon are the records that are allowed through those ports. The next block is a definition of class whose name is *NewClass*. This class inherits the concept of being a state machine. This class doest have any content as of now. The next block is the main block from where actual program starts. In the main block only a new instance of the class *NewClass, NewObject* is created. Using the *start ()* action the *NewObject* is being executed inside the model.

This model is still incomplete. Here we need to associate a state machine to the code. The corresponding state machine is described in Figure 6 which can be drawn by using the Qtronic modeler. It shows the functionality that is running inside the state. The associating link between the code and state transition diagram is done through a test design configuration which will be discussed later in this section. The important rule for the association is that the state machine name should be same as the class name. This links the state chart to the class. In this case the xmi file is named as '*Sample.xmi*'.

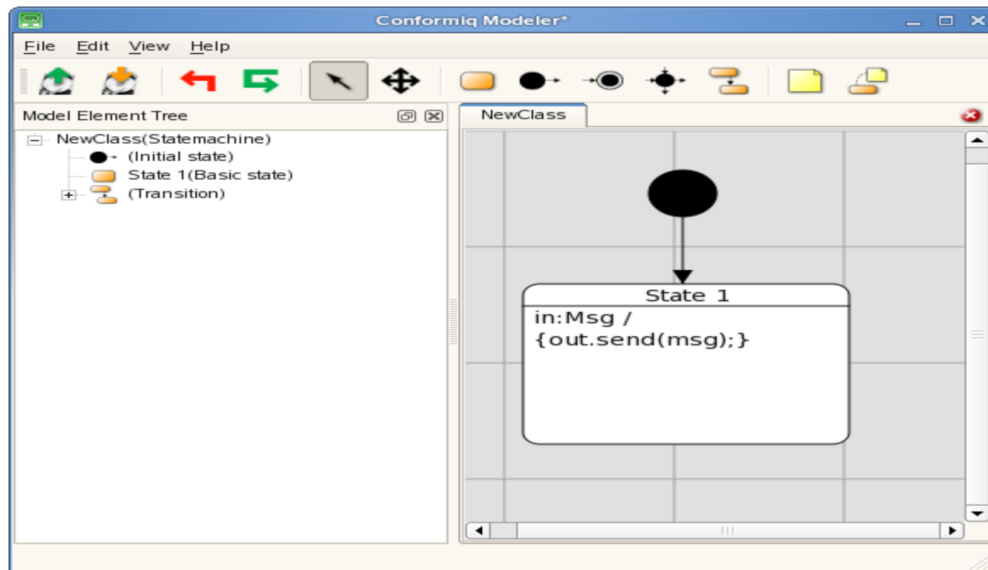

Figure 6 : Modeler Provided with Qtronic

The black circle above represents the 'initial state' of the machine. The arrow represents the transition to state *'State 1'*. The code inside the state is the internal transition logic within the state. It represents the activity being trigged to be done when the system is in this state *'State 1'*. '*in:Msg*' denotes the triggering of action when the message *'Msg'* is arrived inside thought the port *'in'*. The code in the braces represents the action to be taken after the trigger. '/' is used to separate the trigger and action. The line *'out: send (msg)'* represents the action. *'msg'* is the default variable used to handle the message in that state. This contains the record of the type *Msg*. The line represents sending of the message outside through the port *'out'*.

To summarize the functionality, the system first creates an instance of class New Class which is the object New Object. The object is executed and goes from initial state to *'State 1'*. In this state the objects wait for incoming message through port *'in'*. Once the message is received it sends the message through the *'out'* port and the object still remains in the state.

5.1.1.2 Test Configuration and Generation

Here is the sequence of steps for test configuration and test generation process in Qtronic.

1. After installing Qtronic plug-in in Eclipse, switch to Qtronic perspective by selecting Window > Open Perspective. Now select Qtronic.

2. Select New > Qtronic Project. This will also create a Test Design Configuration (DC). The DC contains the settings for coverage criteria selected and configured scripter plug-ins.

3. Import the model files *Sample. java* and *Sample.xmi* into the project.

4. Click 'Load model files to Computation Server' to load the model files into computation server.

5. Generate test cases by clicking 'Generate Test Cases from Model'. This generates the test case with sequence numbering in the 'Test case List' window. By clicking on each of the test cases a sequence diagram of the test case is seen in 'Test case' window.

Test cases can also be rendered by configuring a scripting back-end. Script back-end can be configured by selecting the test design configuration (DC) and then clicking New > Scripting Back-end. Once it is configured, click on 'Render All Test Cases' to render in the format specified by the scripting back-end.

5.2 Motes

Motes [5] [9] [10] [11] [12] is a MBT tool that generates TTCN-3 test-cases from the Implementation under Test (IUT), whose behavior is modeled as Extended Finite State Machine and by use of some supportive TTCN-3 files. These files consist of Test Data, Test Data types, Context Variables and Interface specification (Message Types and Communication Ports) of the IUT. The generated TTCN-3 test cases can be executed using any TTCN-3 supportive tool to check the IUT behavior.

5.2.1 Workflow

The principle work flow of the Motes which is used to generate the test cases from the IUT behavior is as follows:

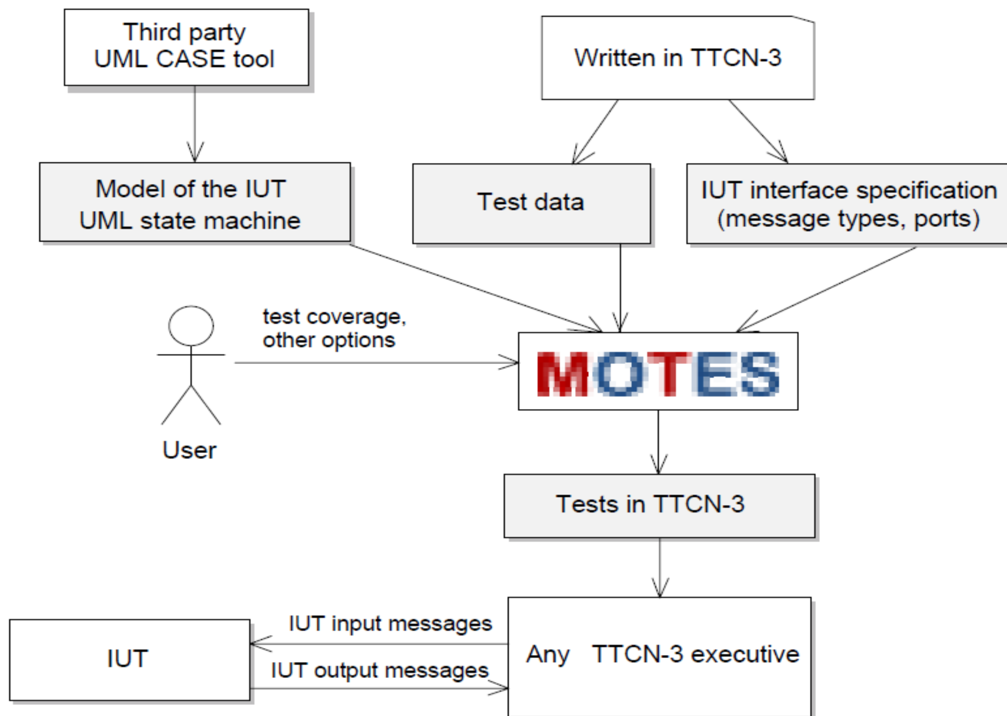

Figure 7: Workflow in Motes [12]

5.2.1.1 Modeling IUT

The model of the IUT which defines its functional testable behavior consists of the EFSM Model, Context variables, Communication Message Types definitions and Communications Ports Definition.

EFSM Model

It defines the behavior of the IUT by means of the UML state machines. Motes don't have own modeler for developing EFSM models. Motes currently supports EFSM models from third party UML case tools such as Artisan Studio and Gentelware Poseidon. In these tools IUT behaviors are modeled as flat state machines. Parallel and Hierarchical state machines are currently not supported. These models are then exported in XMI 2.1 format. These XMI files are used by Motes as one of its inputs for test generation. In the next versions, Motes can accept XMI files from other UML case tools also. According to the UML specification, EFSMs do not have any formally specified action language for representing guard conditions, input events and actions on the transitions. To define the transitions in the UML state machines a formal transition language known as Motes Transition Language is used.

Context Variable

The context variables are the variables which are defined in the TTCN-3 files and are used in EFSM models. These variables are used to define the state information as well as to control the state transitions in EFSM model.

Interface Specifications

These specifications define the interface of the IUT with the test environment. It consists of the Input/output ports and their respective definitions and is declared in TTCN-3 files. The port definition consists of the port behavior and message types that accept while communicating with the external world through the system's boundary. The port behavior comprises of the direction in which port operates, such as Input, Output or both and its message types. Message type defines the data types accepted by the port.

5.2.1.2 Prepare Test Data

This test data defines the stimuli which are supplied to IUT model to generate TTCN-3 test cases. It is prepared manually by the user in the form of TTCN-3 files. These files consist of TTCN-3 templates filled with the respective data for the corresponding message type accepted by the IUT ports.

5.2.1.3 Importing Models and Test data

Models and the associated Test data which are prepared in the previous steps are imported to the Motes tool to generate the TTCN-3 test cases. Next, the user must define a new resource set or re-use existing ones. Importing of the models and test data to Motes is performed through resource set. A resource set performs the task of linking together the relevant input resources required by the test generator. User is required to modify resource set accordingly, whenever there is any change in the model or /Test data.

5.2.1.4 Defining Test Coverage/Goal

Motes use the test coverage specified by the user on the model structure elements (transitions and states) to generate the test cases. The coverage criteria provided by Motes are as follows:

- Selected Elements (states/transitions)
- All Transitions

- All N-Transition Sequences

Selected Elements (selected states and transitions): A coverage criterion is used to define test coverage for the list of EFSM transitions and states. It is possible to create ordered and unordered sets of coverage elements and to define how many times each coverage element or a subset of the elements should be covered. It is possible to cover complex test scenarios using ordered sets of coverage items. There are two types specified for this purpose: set and list. Set consists of unordered test elements, whereas list consists of ordered set of test elements.

All Transitions: This test coverage defines that the test generator should find the test case that covers all of the transitions in the EFSM at least once.

All N-transition sequences: It is a test coverage criterion that allows some long and exhaustive test cases which cover all subsequent transition sequences of n transitions in EFSM model to be created. MOTES allows N to be 2 or 3.

5.2.1.5 Selecting Test Generation Engine

Motes support two engines for the test case generation:

- Model checking engine
- Reactive planning tester engine

Model Checking Engine: It is the off-line test generation engine that supports test generation of the deterministic IUT. It utilizes the UPPAL CORA Model checker to identify the test sequence from the IUT model. Generated test cases always comprise of the sequence of events to test the modeled IUT behavior. It is operated in two modes.

A. ***Iterative:*** It is greedy mode of test case generation, which is operated in iterations to identify the sub-optimal test sequence. This mode is preferred to use when there is a constraint of low memory.

B. ***Non-Iterative:*** It is the optimal mode in which the engine identifies the whole test sequence of minimum length required to achieve the set test goals.

Reactive Planning Tester Engine: This engine operates in both on-line and off-line mode. It generates the test cases for deterministic and non- deterministic IUT.

5.2.1.6 Generating the test cases

Once the above steps are completed, user proceeds for the test generation and the test generator does the rest. The TTCN-3 test cases or reactive planning tester TTCN-3 code are generated under the current resource set on the basis of selection of the test generation engine. The generated TTCN-3 files can be imported to any supportive TTCN-3 test tool and run against the IUT.

5.3 MaTeLo

Markov Test Logic (MaTeLo) [4] is an off line model-based testing tool provided by All4Tec [4]. MaTeLo has its own modeler and the tests are exported in textual notation. MaTeLo uses Markov Chain models (Refer Terminology) and therefore focuses on test control oriented systems. MaTeLo facilitates exporting formats for automatic and manual test execution. It is only supported in Windows platform.

MaTeLo is divided into two separate programs, Usage Model Editor and Testor. The Usage Model Editor uses the Markov Chain Usage model for modeling notation. The usage model can be viewed as a finite state machine extended with probability numbers. It is not extended with programming languages but does accept variables, Scilab/Scicos functions, and Matlab/Simulink**Error! Reference source not found.** ransfer functions, extending the model for simulating expected results. While the use of these variables and functions limits the model's complexity, they are not as effective as programming languages. The lack of a programming language extension sets limitations on present data flow models. MaTeLo accepts many kinds of models as inputs via the MaTeLo converter. These inputs are important for the reuse of existing models. While MaTeLo only provides deterministic models, these can include asynchronous inputs.

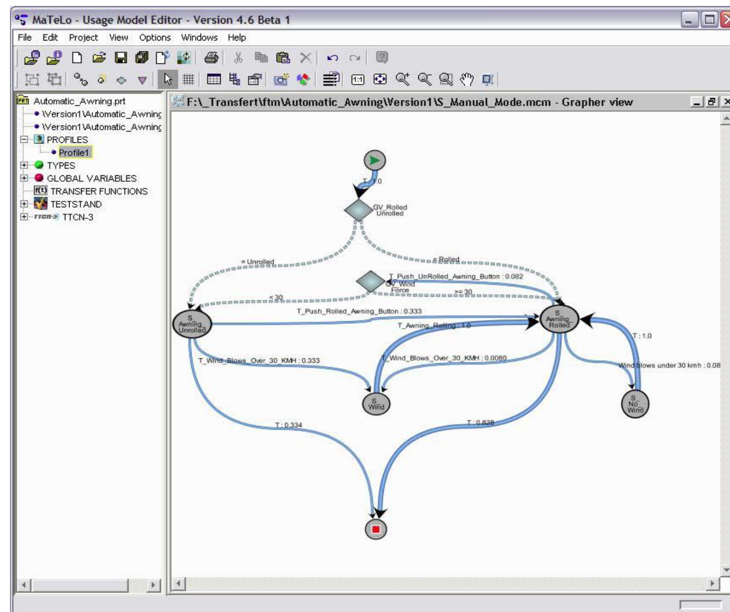

Figure 8: MaTeLo Modeler Window

Figure 9: MaTeLo Testor Window

MaTeLo Testor takes the model as input and generates a test suite. Testor validates the model before usage. Validation means checking modeling errors like unattainable states. Test generation facilitates the user with the selection of any of the configurable options: random, boundary value testing, most probable route, state coverage and transition coverage. MaTeLo also provides time limits.

MaTeLo cannot execute tests itself. However, it is possible to export test suites into HTML, TTCN-3 or Test Stand formats. TTCN-3 and Test Stand are used for automatic test execution, while HTML can be used as documentation for manual testing. Test Stand is a test management tool created by National Instruments. During the testing process, Report Management is used for making a report of test campaign monitoring. Report Management has the additional feature of presenting pleasing graphical figures of the testing process.

Probability numbers is the main idea behind the Markov Chain. MaTeLo is therefore a good choice for control-oriented testing.

5.4 ModelJUnit

ModelJUnit [2] [14] is a set of open source libraries that consists of a set of JAVA classes for model based testing. The libraries are designed by Dr.Mark Utting. It aims at generation of test sequences from the FSM/EFSM models written in JAVA and measures different model coverage metrics. Model JUnit has the features to automate both test generation and test execution.

5.4.1 Workflow

5.4.1.1 Modeling SUT

In ModelJUnit, FSM model is written as a JAVA class that implements various interfaces defined in the library such as

- **FsmModel:** It is the basic interface which every (E) FSM model class must implement for model based test generation.
- **TimedFsmModel:** It is a special interface that extends the functionality of *FsmModel* interface and builds the FSM that uses the ModelJUnit timing framework.

The model class created using interfaces shown above must have the following methods:

1. *Object* getState()

This method returns the current state of the model, which typically is an Object. It performs the task of mapping the internal state of the EFSM model to the actual visible state represented in the model by the model designer.

2. *void* reset(*boolean*)

This method performs the task of resetting the SUT to the initial state or creating new instance of SUT class. It is used typically in online testing, where we need to reset SUT to the initial state. Default value of Boolean parameter is 'true'. The Boolean parameter with value 'false' is used when operations in SUT are less responsive or when we want to generate only the sequences with the current EFSM model.

3. *@Action* void name()

These types of methods are used to define the actions in the EFSM model. These actions change the state of the SUT. There can be more than one action methods defined in one EFSM model. It requires only the *@Action* annotations and no parameters. Each action method can exist with or without guard. This guard decides the enabling/disabling of the action to be called during test sequence generation. In case of no guard, the library provides the default guard that always remains true.

This method contains sequence of adapter code which tests the particular behavior of the SUT .The adapter code calls one or more SUT-defined methods and check the correctness of the results on the basis of the response from SUT. Testers have to specify the test data in methods called on SUT.

4. *boolean* nameGuard()

It defines the guard for the action methods defined in the model. This method always returns boolean. The method name must be same as the action name (for which guard is defined), with the added word 'Guard' at its end.

Table 2 : Sample ModelJUnit model code

```
public class FSM implements FsmModel {

private int state = 0 ; // 0 . . 1

public FSM( ) { state = 0 ; }

public String getState ( )

 { return String.valueOf (state) ; }

public void reset (boolean testing )

{ state = 0 ; }

public boolean action0Guard ( )

{ return state == 1 ; }

public @Action void action0 ( )

{ state = 0 ; }

public boolean action1Guard ( )

{ return state == 0 ; }

public @Action void action1 ( )

{ state = 1 ; }

public boolean actionNoneGuard ( )

{ return state != 1 ; }
```

```
public @Action void actionNone ( )

{ }

}
```

The java class is equivalent to the graphical notation depicted in Figure 10 below:

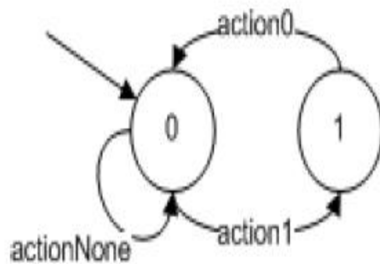

Figure 10: ModelJUnit Generated Graphical EFSM Model

5.4.1.2 Test Configuration

In order to perform testing on the SUT, testers have to specify following parameters:

- **Test Algorithm**

The test generation algorithm explores the FSM graph of the model, and generates the test sequences for it. Model JUnit provides varieties of the traversal algorithms such as

Random tester: This algorithm performs random walk around the generated FSM graph. During the walk-in, in each state it randomly selects one of the enabled out going transitions from the current state towards any next possible transition.

Greedy tester: It does a random walk around the FSM graph and during walk, in each state it gives fondness to the unexplored outgoing transitions. It is efficient as compared to Random Tester, in traversing the graph.

Look ahead tester: Look ahead tester algorithm works somewhat similar to Greedy Tester, but provides more refined options available, such as look ahead depth and several other parameters. Some of the parameters specify to look ahead in the several available transitions to identify the possibility of reaching the unexplored areas and then it tries to reach those states. One more parameter is '*Maximum Test Length*' that shall be set in order to control the search of graph for test sequences.

- **Test coverage**

Model JUnit defines the coverage metrics for measuring the coverage of the model designed by the tester. It is generally a good practice to use these metrics to check the efficiency and quality of the

model that has generated considerable amount of test sequences to cover the majority of the behavior of the SUT. These Coverage Matrices are:

State Coverage: This coverage shows the number of times each state of the model has been entered in the generated test sequences. A call to the reset action will increment the count for initial state. The count for target state will be incremented on the basis of transition. It shows the comparison details between the number of states covered and the total numbers of states defined in the model.

Transition Coverage: It defines the number of transitions that are covered in the generated test sequences. It also shows the comparison details on the number of executed transitions against total numbers of transitions defined in the model.

Action Coverage: Action coverage defines the number of distinct actions that are covered in the generated test sequences. It shows the comparison details on the number of distinct actions visited / Total numbers of actions defined in the model.

Transition Pair Coverage: This coverage shows the number of transition pairs that have been tested in the test sequences generated from the model. It shows the comparison details between the numbers of transition pairs tested and total numbers of Transition Pairs defined in the model.

5.4.1.3 Test Generation

It is last step that leads to generation of test sequence from the model and defined test configurations. There are two modes of test generation.

- **Online:** Model JUnit is usually used for on line testing, where the tests are executed on the SUT, as they are generated by the model. In on line testing, users have to integrate the ModelJUnit code with the JUnit API. So that each time when the user run your JUnit test suite, you will generate a suite of tests from your FSM/EFSM model of ModelJUnit. Also, the *@Action...* methods in the model class will include code to interact with the methods of SUT; it is required to check their corresponding return value, and the current status of the SUT. In this process, when user runs the JUnit tests, the corresponding model is used to generate a sequence of *@Action...* calls and test the SUT.

- **Offline:** In this mode only messages are generated by model execution. The generated message is the test sequence in from (Initial state, Action executed, Target State) .These generated messages could be saved in a file and used as a test script for later test execution (offline testing).

5.5 MBT Tigris

MBT Tigris [6] is an open source tool which has been designed in JAVA, for generating test sequence from the SUT models. These test sequences can be rendered in Java or Perl platforms. It can be operated in both GUI and command mode and supports generation of both online and offline test sequences. Tigris can be integrated with other tools that understand web-services and also it can support requirement traceability.

5.5.1 Work flow

The basis workflow that generates the test sequence for the SUT is depicted in the flow graph of Figue11.

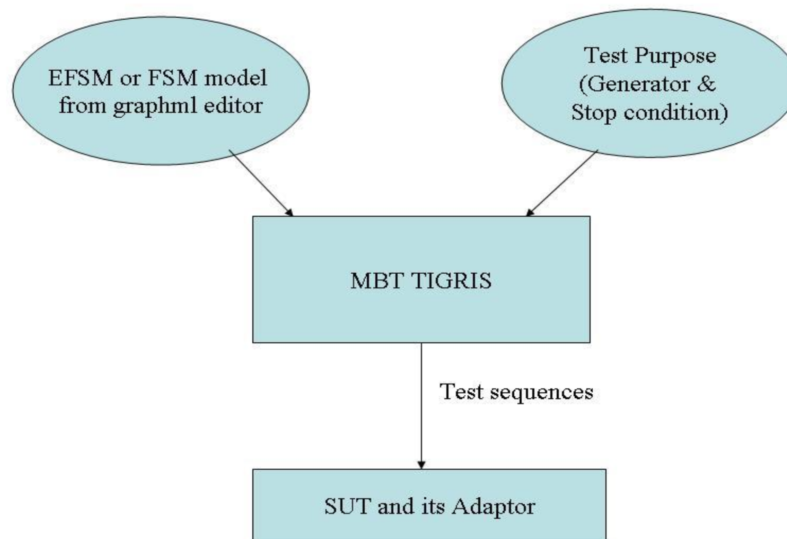

Figure 11: Workflow of MBT Tigris

5.5.1.1 Modeling SUT

The system behavior of the system is modeled as Finite State Machine or extended Finite State Machine. The tool doesn't have its own modeling editor. It depends on the open source graphical modeling language tools such as Yed. There are certain guidelines that the user has to follow to model SUT behavior:

1. FSM or EFSM model has to be the graph or digraph.
2. Every graph must have the vertex with label 'Start'. It defines the start point of the graph.
3. There shall be no Stop or Exit points in the Graph. Modeling doesn't imply to the UML specifications.
4. The model should use proper naming conventions defined in the online manual for the edges and vertex.
5. The generated graph must to save in *graphml* format.

5.5.1.2 Test Configuration

This is the next step after the modeling of the SUT. The tool uses the graphml file supplied by the graph editor. User specifies the following parameter for generating test sequence as:

1. Test Generator
2. Stop Condition

Test Generator

It specifies the possible test sequence generator algorithms. For a particular test suite user can select only one algorithm at a time. Possible test generators are:

Table 3: List of Generators for MBT Tigris

GENERATOR	DESCRIPTION
A_STAR	This algorithm works well with the small models to identify the shortest possible test sequence with complete coverage. If, it is employed in the large models the downside is that it takes lots of CPU computation time to generate test sequences.
RANDOM	It works well with the considerable larger model and generates test sequence on the basis of random selection of out edges from the existing vertex and then it repeat the process till the stop conditions are reached.
SHORTEST_NON_OPTIMIZED	This algorithm works intermediately between A_STAR and RANDOM, with generating short sequences from larger models.

Stop Condition

In this tool the model doesn't have stop or exit points. In order to provide a stop to the test sequence generation, stop condition is used. These are shown in the table below.

Table 4: List of Stop Criteria for MBT Tigris

STOP CONDITION	DESCRIPTION	USAGE VALUE
VERTEX_COVERAGE	It specifies the percentage of the vertices covered in the model	An Integer between 1-100
REACHED_EDGE	The edge at which the test sequence will stop	Label used to represent edge
NEVER	The test sequence generation shall never stop/Halt	N/A
REQUIREMENT_COVERAGE	Specification for the percentage count of the requirement covered in the generated test sequence	An Integer between 1-100
REACHED_VERTEX	The vertex at which the test sequence will stop	Label used to represent vertex
TEST_DURATION	The amount of time specified till the test generation will be performed	An Integer that specify time in seconds
REACHED_REQUIREMENT	It signifies that test generation will run until the specified requirement is reached	Requirement Tag is used to specify the requirement in the

		model.
TEST_LENGTH	It specifies the length of generated test sequences in terms of Edges/Vertices pair	An Integer
EDGE_COVERAGE	It specifies the percentage of the edges covered in the model	An Integer between 1-100

5.5.1.3 Test Generation

This is the last step in which use model and test purpose to generate the test sequences. These generated test sequences can also be rendered to the language such as Java or Perl.

6. Tools Comparison

In this chapter we attempt to compare the examined tools based on their functionality, usability, flexibility and pricing level and how well they can be integrated with the process suggested in the section. We are writing this data on the basis of the conducted case studies, tools exposure experiences and tools available literature [3][4][5][6][12][13][14].

6.1 Functionality and Usability

6.1.1 Graphical User Interface

- **Qtronic**

1. Qtronic is a GUI tool which is based on the eclipse-based framework and can be used on Windows, Linux and other Unix-like platforms such as Solaris, HP-UX etc. It can be either installed as a stand-alone application or as eclipse plug-in. This can handle multiple test generation projects in a single workspace and different types of test settings per single project.
2. Once the test sequences are generated, they can be viewed in both hierarchical lists and sequence diagrams.
3. The tool also contains additional UI features like interactive traceability matrix view, dependency matrix view, requirements coverage view and graphical mapping between state transition model charts and generated test cases.
4. Qtronic itself provides a graphical modeler called Conformiq Modeler, which is useful for drawing Finite State Machine models.

- **Motes**

1. Motes test generator is an eclipse plug-in with rich GUI features. In eclipse framework users have to create the test generator project. This project has predefined folder structure, which is shown in the figure 13 [12]

2. Its test project browser view includes a pre-defined structure of folders for the input and output artifacts required for the test generation. This feature provides proper organized structure for the content of test generator project.

3. Motes also provide facility for the set up of multiple test configurations per test project. It allows the flexibility of generating test cases for different scenarios and this feature is provided by means of directives. Each project can have one or more directives. These directives are present in the folder named “Resource set”. These directives are created for a particular resource set. There is a separate directive windows shown in the Figure 14 [12] provided for configuring test purpose, test generation engine and some other configuration attributes.

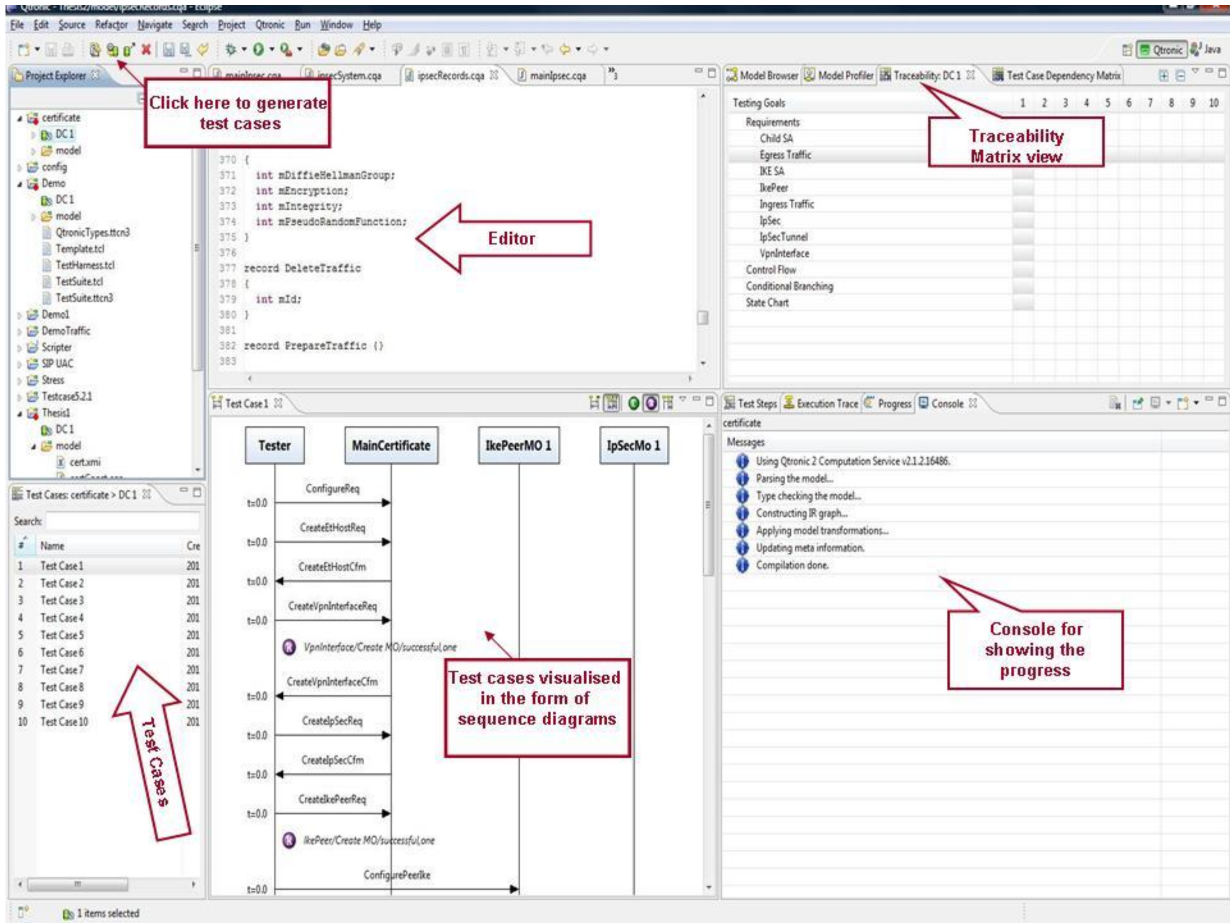

Figure 12: Qtronic GUI Main Window

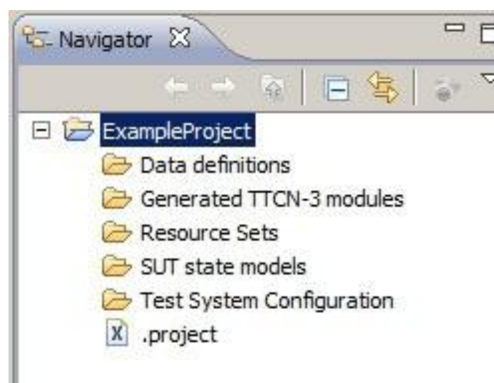

Figure 13 : Motes Project Structure

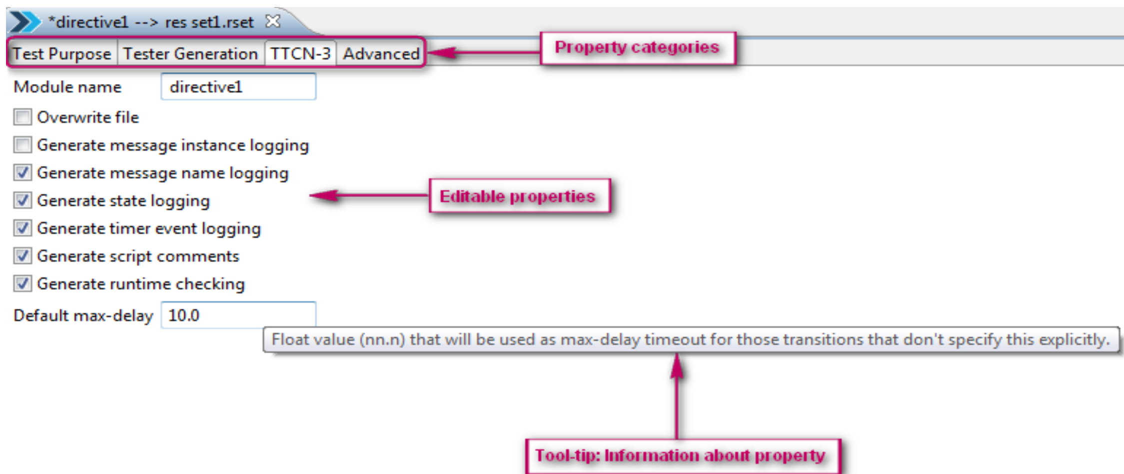

Figure 14: Motes Directive Definition Window

4. Motes provide the capability of single click generation of test cases. The test cases are generated per project and as per defined directive. The generated test cases are stored in the well defined folder named as "Generated TTCN-3 modules" shown in the Figure 15 [12].

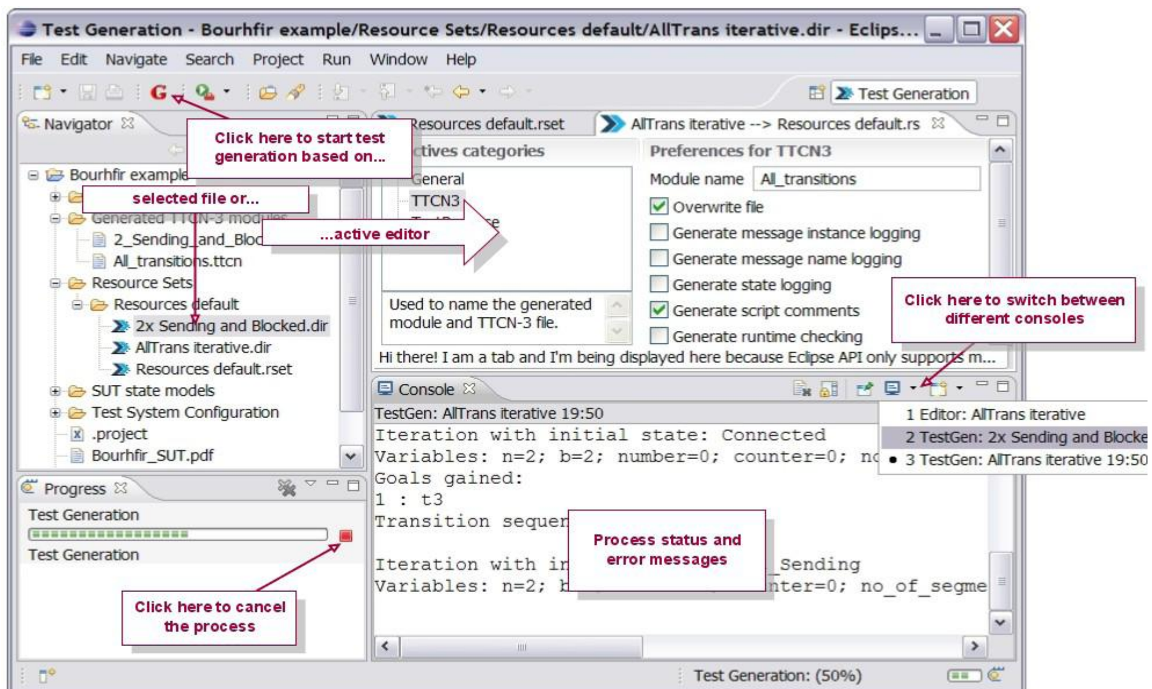

Figure 15 : Motes Main GUI Window

5. It facilitates well defined console view of the test generation progress. In console view, user can visualize the status of the test generation process in the form of transitions covered and amount of generated TTCN-3 test cases. It also displays information about the errors occurred during the test

generation. There is well structured error mechanism that also shows proper description of error in separate window as shown in figure 16 [12].

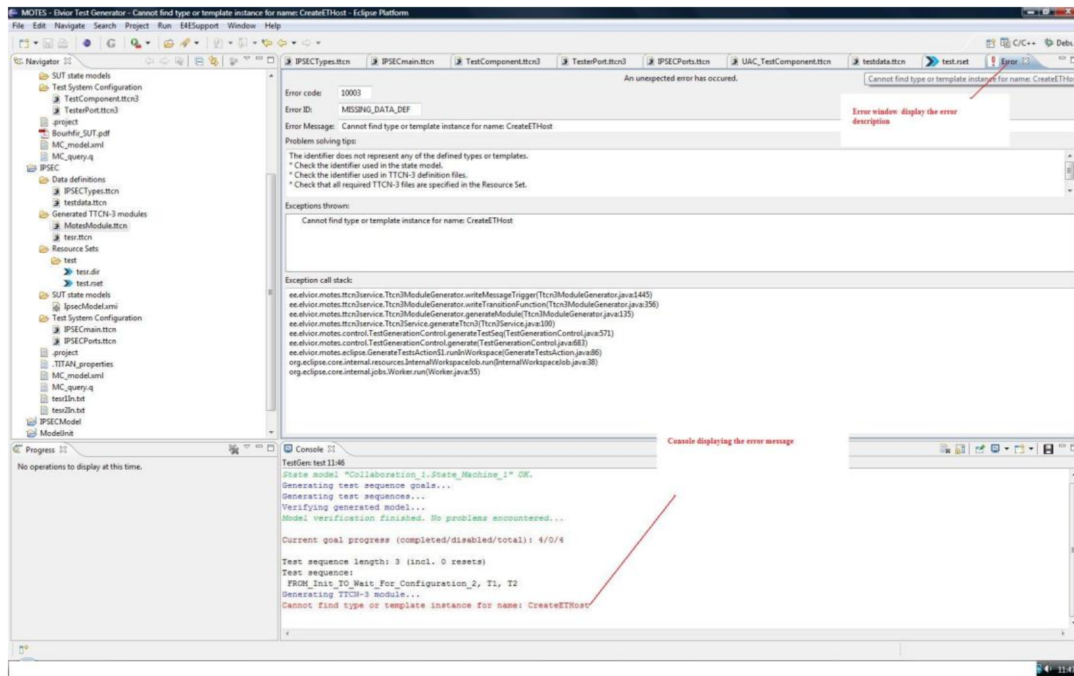

Figure 16 : Motes GUI showing Error Message

- ## ModelJUnit

ModelJUnit supports GUI features only in the test configuration and test generation, while there is no support provided for modelling activity. In order run the GUI, user can just double-click on the ModelJUnit .jar file, or put it in the *CLASSPATH*, and the following command is executed

java nz.ac.waikato.modeljunit.gui.Main

1. ModelJUnit allows user to generate the test sequences by one click and visualize the generated test sequences in both textual as well as graphical format shown in Figure 17.
2. It also provides separate window for configuration of test generation as shown in Figure 18. In this window user can select various test generation algorithms and configure several reporting parameters.
3. ModelJUnit facilitates the display of animated view of the generated FSM from the java code and also provides the option to select the particular state and analyze the state variables.

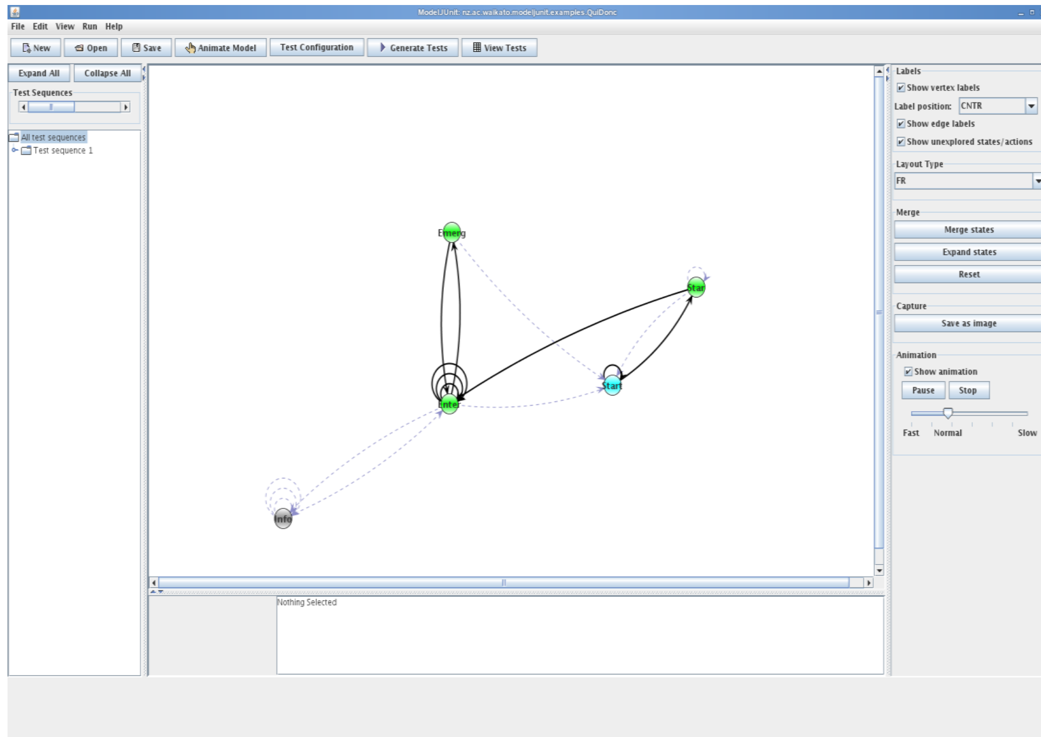

Figure 17: ModelJUnit GUI Main Window

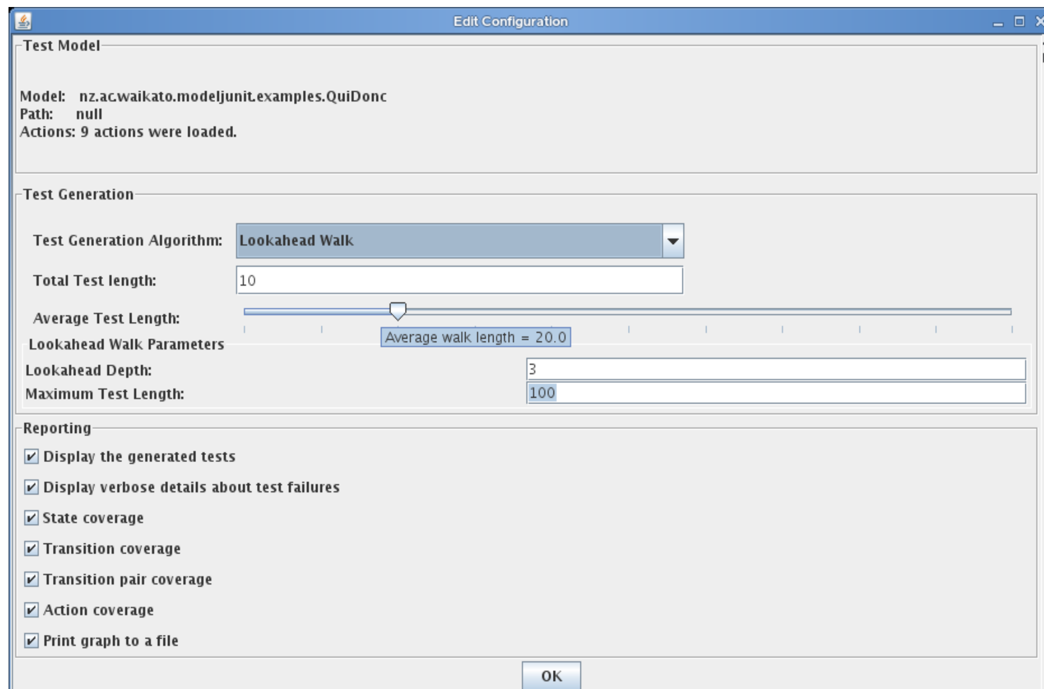

Figure 18: ModelJUnit GUI Test Configuration Window

- **MBT Tigris**

It is used in graphical mode by using the command "*gui*". The graphical part of Tigris (shown in Figure 19[6]) provides less support in test generation. It allows the user to supply xml file with the options that includes generator algorithm, stop conditions, models etc. User has to load XML file in the GUI window. Here are some of the limited features it provides:

1. The actual graphical model is displayed.
2. The tool supports only online mode of testing.
3. Tigris provides visualization of test steps in the model when the tests are run.
4. It offers the options to play pause and reload the tests.
5. It also shows current test coverage statistics.
6. The window also displays the current values of context variables.

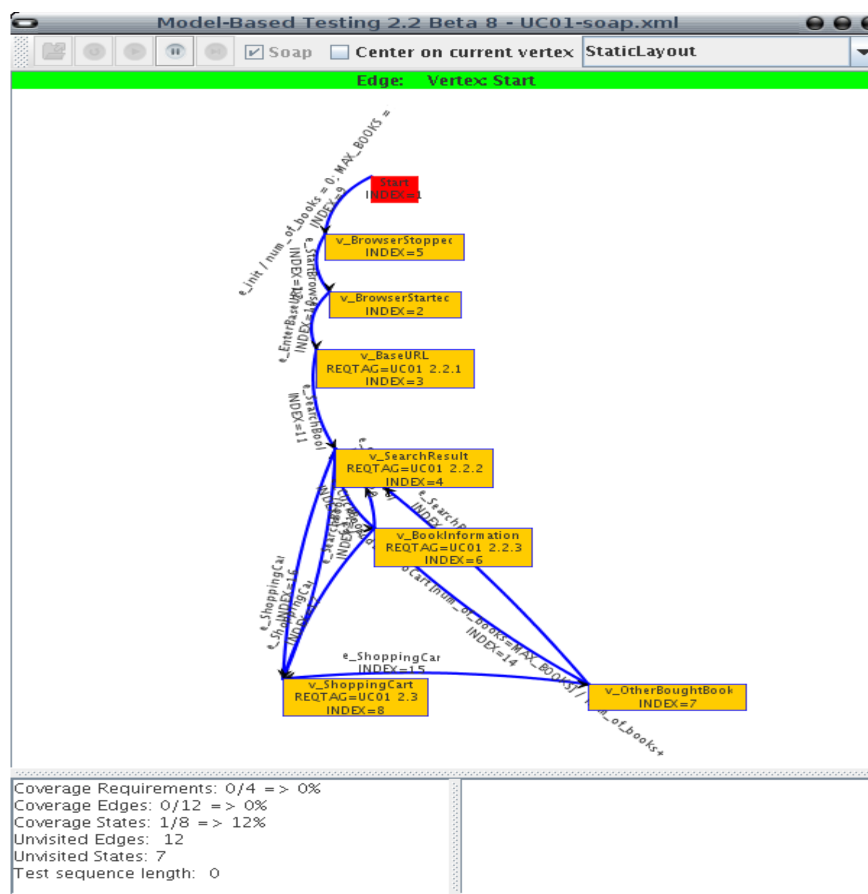

Figure 19: MBT Tigris Main Window

- **MaTeLo**

It has its own well thought out and matured GUI, which consists of following sets:

Usage Model Editor shown in the figure 20 [4] has good graphical user interface, allows the test engineer to design the test models, define test functions, associate system requirements with model and adjusting test profiles.

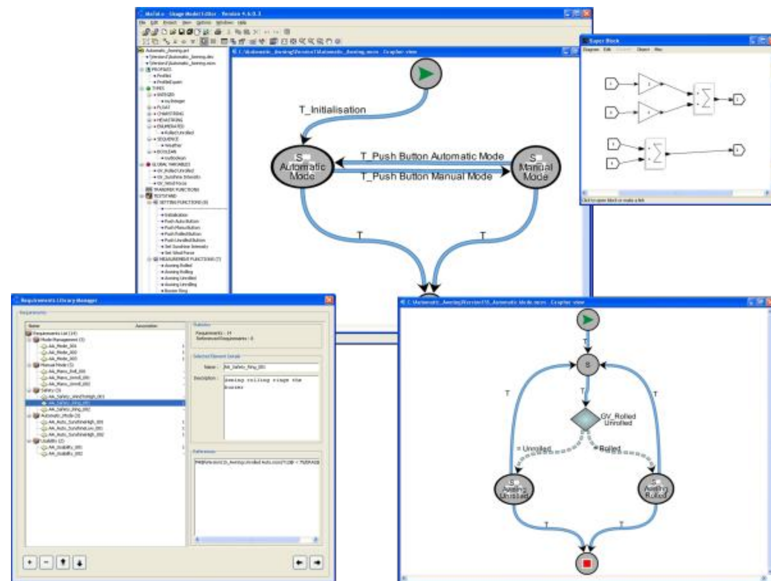

Figure 20: MaTeLo Modeler Windows

MaTeLo Testor provides powerful user interface for tasks such as configuring test strategy, test cases generation and checks the quality and the test coverage of the generated test suite as shown in Figure 21 [4].

Figure 21: MaTeLo Testor Window

6.1.2 Libraries

- Qtronic is available as a single integrated suite that comprises of Qtronic client, Qtronic computation Service and Conformiq Modeler. It doesn't require any other third party library. It runs in the JRE environment. Qtronic also provides the facility of customizing the scriptors.
- Tigris is a Java based open source tool. It is available as JAR file (mbt.jar) and its corresponding property file and it requires third party modeling tool named as Yed from Y Works for modeling SUT in the form of graphs or digraphs.
- ModelJUnit library is available as JAR file. It is required to add the JAR file to the class path. It also requires the "*graphviz*" tool to visualize the graphical model of the EFSM created by the Java code.
- Motes tool is available as eclipse plug-in currently supported to be used in Eclipse (version 3.2 or newer). It is the integration of the following libraries:
  - **Uppaal Cora** library is available as a separate executable with proprietary license. It is used by the Motes for the test sequence generation.
  - **RPT Synthesizer executable** is available to the user as part of the Motes plug-in. It is also used for the test sequence generation.
  - **Prolog executable** is available as the open source library of the prolog, which is required by Motes for its functions.
  - **Test Generator** is the actual test generator which generates the test sequence from the model and TTCN-3 supportive files.
- MaTeLo is available as integrated package which consists of mandatory components such as Usage Model Editor and Testor. An optional set of components are also available such as Java runtime, Adobe SVG Viewer, CVI Runtime and Scilab.

6.1.3 Design Rules

6.1.3.1 Modeling Strategy

- Qtronic uses a combination of UML state transition diagrams and an action language which is Java-like. The language is a variant of Java whose syntax is similar to Java and C#. The UML model and action language that it supports together is called QML (Qtronic Modeling language). The UML diagrams are optional but useful for better usability and understanding of the behavior of SUT. Qtronic provides support for object-orientation, including classes, inheritance, multiple threads, and asynchronous communication between model threads etc.
- In Motes, model includes only the UML state machine and MTL, a transition expression representation language. It allows only modeling of flat executing state machines. It also provides rich programming features of TTCN-3 in modeling.
- ModelJUnit uses the Java language constructs to represent the EFSM model. It only supports flat executing state machines. It also provides rich java features in modeling.
- Tigris requires graph representing EFSM. It has its own defined rules for defining the labels marked for the vertex and edges. Tigris allows merging of the several models. But it doesn't support modeling for hierarchical and parallel executing state machines. It also provides very limited programming features in modeling.
- MaTeLo uses MC concept in modeling. It doesn't provide modeling language with high programming capabilities.

6.1.3.2 Test Data Specification

- Qtronic automatically computes test inputs. The test inputs are computed on the basis of language constructs and supplied Qtronic keywords. QML provides support for basic data types, structural data with strings, numbers, and nested structures.
- ModelJUnit uses the data types supported by Java language to represent the test data. In this, test data is supplied externally by the user. Test generation is performed on the basis of supplied test data.
- MaTeLo requires test data to be supplied explicitly by the user. The system supports six basic data types (integer, float, hex string, char string, enumerated, Boolean) and only one complex type as Sequence.
- In Tigris test data is supplied as the part of test execution. During test generation no test data is required.
- Motes test data is supplied externally in form of TTCN-3 files. Test generation depends on the supplied test data.

6.1.3.3 Test Configuration Definition

- Qtronic offers wide range of options for configuring the test generation. It provides model driven coverage editor, where one can find lot of parameters. The tool can be configured with state coverage, transition coverage, boundary value analysis, branch coverage, atomic condition coverage, method coverage, statement coverage, parallel transition coverage, and all path settings.
- Motes on the whole provide three kinds of coverage strategies: All transitions, all N-sequences and selected elements. User can select one of the provided options to guide the test generator.
- MaTeLo provides two kinds of test suite configuration techniques: Coverage Indicators and Quality Indicators. Coverage Indicator provides various options such as States, Transitions, Model Items, Classes of Equivalences, Global Indicator, Requirements while Quality provides configurable parameters such as Reliability, Mean Time to Failure and Fault Intensity.
- Tigris provides test configuration by means of stop conditions that are defined in section 5.1.1.2.
- ModelJUnit library also provides variety of configurable options such as Action Coverage, Transition Coverage, State Coverage and Transition Pair Coverage.

6.1.3.4 Modeling Language

- Qtronic uses the action language named as QML .It is extension of Java and C# .It is proprietary to Conformiq and provides the OOPS features such as encapsulation, polymorphism etc along with some self defined keywords such as require, requirement .Modification is required in models while re-using model files in other MBT tools.
- Motes use TTCN-3 as scripting language. It is used to represent the test data, test configuration definitions and utility methods during modeling and uses most of the features available in the TTCN-3 library. Motes also use the MTL for defining the state transitions in the UML model.
- Tigris doesn't provide any specific modeling language. It allows defining basic programming constructs in the model such as variable declaration, assignment operation, arithmetic and logical operations.
- MaTeLo also doesn't provide any extensive programming language support in modeling .However it provides support for transfer functions used in test model, written in language such as Mathworks Simulink, Scilab Scicos or with its own context functions.
- ModelJUnit uses Java language to define the FSM/EFSM testing model of SUT.

6.1.4 Document Generation

- Qtronic provides support for automatic generation of test plans and test case documentation. It has extensible script backend facility that allows the rendering of generated test case in any format. The available HTML scripting backend allows rendering of generated test cases in HTML format.
- MaTeLo has option for generation of HTML documentation for the test cases.
- ModelJUnit doesn't have any documentation generator. But, its architecture is extensible enough to implement an adapter to render generated test sequence in any format. However it provides facility of rendering the generated test sequences to a text file.
- Tigris provides small support for document generation. It provides support for rendering the generated test sequence and SUT to a required text file. However test sequences can also be targeted in Java and Perl files.
- Motes don't support Test documentation generation.

6.1.5 Requirement Tracking

- Qtronic has the facility of requirement traceability by means of traceability matrix. It provides support for automatic generation of traceability information in the eclipse workbench. It shows the tabular representation of the requirements covered in generated test cases.
- Motes don't support requirement traceability mechanism. It has been done explicitly by means of requirement driven testing.
- MaTeLo owns a "Requirements library" allowing associations of requirements to model objects. The aim is to link the test model with the requirements and to ensure the requirements validation during the test campaign. The requirements library can be automatically imported from other requirement databases such as IBM Telelogic Doors © (Native Plug-In) or via files such as CSV (Geensys Reqtify, Microsoft Excel) or XML.
- ModelJUnit doesn't support implicit requirement tracking. It can be done explicitly by using the Java API. One of the possible ways could be defining the requirements as strings in Java files that define the model. These requirement strings can be rendered to console or any file during test sequence generation. In this way, requirements can be traced in ModelJUnit.
- MBT Tigris supports requirement tracking. It provides support for including the requirements in the model by means of "*REQTAG*" keyword. The requirements are tagged with this keyword in the edges and vertices of the model. It can trace by using the command line option "*requirements*". This option shows all the requirements tagged in the model.

6.2 Flexibility

6.2.1 Operating System

- Qtronic is available as a standalone application or an Eclipse plug-in that can be run on both Windows and Linux platforms. In fact Qtronic can be installed in any platform that supports Eclipse framework.
- Motes are available as Eclipse plug-in. It runs on every platform that support Eclipse, while it's supportive libraries operate on both Windows and Linux platforms.
- MaTeLo runs only on Windows.
- Tigris is Java based MBT tool which requires Java version 1.6 or higher. The supportive modeling tool

Yed also requires Java Runtime Environment. This tool as the whole, works on any platform that support JAVA 1.6 or higher.

- ModelJUnit is a java based library that runs on any platform that supports Java (version 1.6 or higher). Presently this library is available for Windows and Linux operating systems.

6.2.2 Tools Integration

- Qtronic facilitates compatibility with change management tools. It also import UML models designed in other UML modeling tools such as Rhapsody, Enterprise Architect and IBM RSD-RT. The tool also has facility to publish the generated test cases and sequences into other test management tools like HP quality Center.

- MBT-Tigris support integration with other tools that understands web services. In order to communicate with other tools, Tigris has command SOAP that enables itself to run as web service and communicate with other tools via that web service. The third party tool should recognize that MBT runs as the web service. It can be used to generate the test sequences and also other test management activities. One of the examples of integration with the third party tool can be the HP's Quick Test Professional, which has the capability to understand web service.

- Motes also supports integration with any tool that support TTCN-3 scripting platform. The integration is basically supported for test execution. There is no such support for integration with test management or test control tool. It also supports import of the UML state machine model from the third party UML CASE tools. For example *Message Magic* from Elvior can be easily integrated with Motes to ease the test automation. It also imports the models from UML case tools such as Poseidon and Artisan Studio.

- ModelJUnit is usually suitable for online testing of the Java Based SUT or SUT which has available JUnit framework for unit testing. It can be easily integrated with the tools that support JUnit framework for test execution. But, there is no support provided for the integration with test management tools.

- MaTeLo supports importing of UML models from tools such as Rational Rose. It also supports import of requirements from several requirement gathering tools such as IBM Telelogic Doors and requirements specified in the form of CSV and XML files. To automate the test execution, it has the facility of integration with several test execution tools such as National Instruments Test Stand, MBtech PROVEtech: TA, IBM Rational Functional Tester, HP QuickTest Professional and SeleniumHQ etc.

6.2.3 Script Languages

- Qtronic provides options for exporting abstract test cases in human readable format by means of scripting backend. Scripting backend is available for different formats such as HTML, Microsoft Word document, PERL, EXPECT. The scripting backend can be customized in any scripting platform by means of API provided by Conformiq.
- Motes allow exporting abstract test cases in TTCN-3 language. It also provides some configuration options while exporting the abstract test cases in TTCN-3 platform.

- MBT Tigris provides facility of exporting the abstract test cases in formats such as Perl, Java and Text files.
- MaTeLo generates the test cases in various execution platforms such as TTCN-3, XML and other customizable test scripts.
- ModelJUnit only generates the test sequences that can be displayed in console or rendered in any text file.

6.2.4 Adaptability

- Qtronic is currently adapted in the existing environment in Ericsson. Its ability to render test cases in various formats such as HTML, TTCN-3 etc and also to customizable formats makes it easily adaptable to most of the common testing environments.
- ModelJUnit is highly adaptable in online mode of testing. Whereas, it requires writing the adapter that is written in Java language. It connects the SUT with the models. It is required to write the methods that performs actions in SUT in *@Action* methods of model. In order to do online testing, the model configuration should be written in the JUnit test case format and it is executed to check the functionality of SUT. It can be adapted in the current test ecosystem as we have JCAT framework available that provide the methods to call actions of SUT. We can include that code in action methods and performs online testing.
- Motes can be used in the current test execution environment of Ericsson. It allows writing system adapter for different available testing tools. It allows writing adapter in different languages such as NModel, Message Magic etc.
- MBT Tigris requires writing adapter that connect SUT with MBT tool via web services. Then, it is difficult to adapt it in the current test execution environment.
- MaTeLo can also be adapted in current test execution environment as it supports generation of test cases in platforms such as TTCN-3, customizable test scripts and allows integration with several standard test execution tools listed in section above.

6.3 Pricing

6.3.1 Licensing and Support

- Qtronic is available with floating licenses and named-user licenses and offers piloting, training, coaching and technical support services.
- MBT Tigris is open source tool that requires open source GNU General Public License and required JAVA 1.6 that also available with free license. The supportive modeling tool names as Yed is open source tool that is available with GNU General Public License with all required features. Tigris provides support in the form of online free forums named as mbttigrisorg.freeforums.org, where users can report bugs, request for new features and also ask questions/query related to MBT-Tigris. In support, Tigris also provides on line wiki pages which gives information about the various articles such as available command line options, common terminology, several examples of using Tigris and information about using Tigris for Modeling and Test sequence generation.
- ModelJUnit is the open source Java library which doesn’t require any commercial license. There is no official support provided for ModelJUnit library.
- MaTeLo is available with floating licenses and named-user licenses. It offers wide range of support services which include paid services like training, test expert consulting, project work and technical assistance. Their free services support is available in form of wiki pages and forums.

- Motes Test Generator is available as free tool, which exist in beta version. MOTES test generator uses UPPAAL verification engine, which has a proprietary license. No official support is provided from Elvior. The required modeling tools like Artisan Studio and Gentelware Poseidon is available with evaluation and propriety licenses.

6.3.2 Education

- Qtronic provides well designed and structured manual. Conformiq also provides training and coaching services in Qtronic.
- Tigris doesn't provide any training to the users, but its demo models, wiki pages and on line manual provide required information about its functionality and usage for test sequence generation. It also contains information about the usage of commands, modeling guidelines and useful examples of using tools.
- There is no official training provided for ModelJUnit library. But online web documentation and example models are available for using the library.
- There is no separate manual exist for MaTeLo. But, there is online web documentation available that provides complete information. Separate MaTeLo training courses are available.
- For Motes, Elvior doesn't provide any training to the users, but it's on line manual, example models and tutorial provides required information about its functionality and usage of tool for test generation. The manual provides general information about the tool such as overview of test generation framework, inputs required by tool for test generation, using motes for test generation and other relevant information.

6.4 Support for Non Functional Testing

- Motes provide implicit support for non-functional testing (performance) of the reactive systems by means of certain features such as Time-Out for communication messages, support for introducing timer in the test model.
- Qtronic, ModelJUnit, MaTeLo and MBT Tigris implicitly provide support only for functional testing.

6.5 Applicability in current SUT Environment

- Qtronic tool is already used in System Functional testing.
- MaTeLo is capable to be used in functional testing. But it is not flexible enough to model complicated systems. It is typically used for use case modeling but not for modeling complete system behavior.
- ModelJUnit provides support for modeling telecom applications and supports required model characteristics.
- Motes are specially designed to test reactive systems. It is the most feasible tool, used for telecommunication system.
- MBT Tigris uses the basics of graph for modeling the SUT behavior. There are several other factors that make it less applicable such as lack of programming support, limit support for required model characteristics (non-determinism, timed, parallel, and hierarchical) etc.

7. Case Studies

This section describes the case studies we performed as part of our thesis work. Two tools Qtronic and Model JUnit have been selected for case study and the process proposed in the Section 4.2 was implemented for modeling non-functional specifications. We identified IPSec as SUT. Section 7.1 gives an introduction about the SUT and the consecutive sections describe our way of working with the two tools mentioned above. Chapter 8 gives a discussion on the observations that were made during modeling with these tools.

7.1 System Under Test : IP Security Protocol

System under Test [7] [8] [15] we have chosen is IPSec protocol implementation in Ericsson's CPP (shown in figure 22 [15]). IPSec is a well-known security protocol suite, with set of protocols and services providing a complete security solution for an IP in telecoms network, which is not too complicated and widely used.

The following are some of the terms that are associated with IPSec protocol:

Internet Key Exchange Protocol (IKE) - IKE is a security protocol which is used between two hosts (gateways or between gateway and host) for encoding authentication information and performing encryption of payload data. It performs the exchange of Security Association SA parameters and populates the Security Association Database (SAD) which is used later for exchanging cryptographic information for secure communications.

Security Associations - Security Association (SA) is a set of parameters used for specifying simplex "connection" that provides security services to the traffic carried by it. Security services are unified to the SA by specifying security protocol either as AH or ESP. It is required to establish two SAs for the protection of the traffic stream when both AH and ESP protection is applied to it. Thus, it is required to establish two SAs between two hosts or between two gateways or between gateway and a host for bi-directional secure communication.

Security Policy Database (SPD): It is the repository for security policies programmed in IPSec implementation. SPD is consulted during the flow of traffic across hosts or gateway implementing IPSec.

Security Association Database (SAD): It is the database that contains the entry for the SA parameter that is allowed to be used for secure communication in IPSec. The SA parameters specified are checked against the entry in the SAD.

IPSec is cryptographically-based security protocol that operates at IP Layer, designed to provide interoperable, high quality services including access control, protection against replays, connectionless integrity, data origin authentication, (a form of partial sequence integrity), confidentiality (encryption) and limited traffic flow confidentiality to Ipv4 and Ipv6 packets. These security services offer protection to IP and/or upper layer protocols.

It is implemented in a host or a security gateway environment such as routers, firewall or media gateways; provide protection to IP traffic that flows to them. The protection offered on the basis of requirements listed in a Security Policy Database (SPD), which is setup and maintained by the application operating in the host or by the user.

IPSec security services are operated by means of following components:

Security Protocols – The following protocols are used to provide traffic security:

- The IP Authentication Header (AH) provides services of data origin authentication, connectionless integrity and anti replay services.
- The Encapsulating Security Payload (ESP) protocol provide security services such as connectionless integrity, anti-replay service, limited traffic flow, confidentiality and data origin authentication.

These two protocols may be used to operate alone or in combination with each other. Each protocol type supports two operational modes: Transport mode and Tunnel mode. In transport mode the protocols provide primary protection to the payload data of the IP packets while in tunnel mode, the protocol provides protection to the tunneled IP packets (data + IP header).

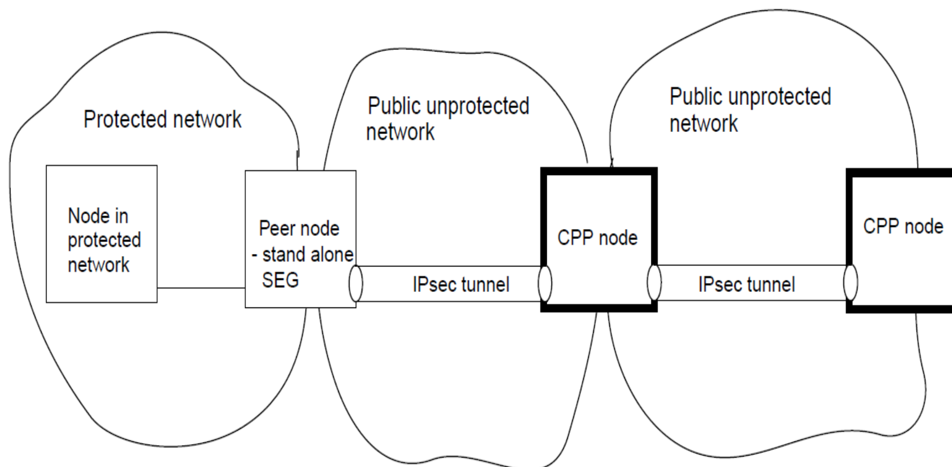

Figure 22: IPSEC Implementation in the System

IPSec provides its security services between one or more paths between the pair of hosts, between the security gateway and host or between the pair of security gateway by enabling the host or security gateway to select required security protocols, determine the algorithms for the security services, and set in any required cryptographic keys. To accomplish this, they must perform the following actions:

a. Multiple hosts (gateways or gateways and host) will agree upon which set of security protocols they use to provide security services, so that both hosts (gateways or gateways and host) must synchronize with each other.

b. Multiple hosts (gateways or gateways and host) also decide on selection of specific encryption algorithm to be used in encoding transmitted data between them.

c. Multiple hosts (gateways or gateways and host) also exchange cryptographic keys that are used to decode the data that has been encoded cryptographically.

d. Upon completion of the above task, each hosts (or gateway) must use the security protocols, encryption algorithm and cryptographic keys agreed upon to encode data and send it across the network.

7.2 Tools Selection

Though many tools are available in market for MBT, all are not well suitable for the model based testing of the SUT that we have considered in the telecommunication platform. It depends on many factors such as adapted testing methodology, adaptability into the existing environment, cost etc. Based on some of those factors (Refer Section 6.5) we have selected Qtronic, Motes and ModelJUnit for the case studies to be carried out as part of the thesis work.

MBT Tigris is mostly applicable and adaptable to Web Applications and it does lack in programming support. This is one reason why we didn't adapt it for the Case study. Similarly MaTeLo is also having the same problem of lack of programming support and it doesn't support the representation of complex data types, this tool is also excluded for case study. Moreover lack of extensive time for thesis work forced us to exclude Elvior Motes also. Since Qtronic has got readymade functional model to experiment on modeling non-functional specifications using the new proposed methodology, we have chosen it for case study. Also ModelJUnit code syntax is similar to that of QML code. So it was easy and less time taking for us to first develop the functional model and then incorporate non-functional requirements in ModelJUnit.

7.3 Case Study with ModelJUnit

This section demonstrates the applicability of the ModelJUnit library on the SUT IPSec and the feasibility of the methods mentioned in the section 4.2 using ModelJUnit.

7.3.1 Objectives of case study

The IPSec protocol implementation in telecommunication system was used in the case study. The main objective was to generate test sequences according to the test purposes specified in the requirements document. IPSec functionality implementation as a whole was not tested in the case study, but only the part of Configuration Management of related objects and traffic flow in the system is done.

The following aspects are evaluated in the case study

- **General feasibility of tool to be used for functional testing of the IPSEC implementation and its services.**

The ability of the ModelJUnit library to test the functionality of the IPSec implementation in system is evaluated in terms of its test generation time, test coverage and parameter explosion.

- **Overall Adaptability of ModelJUnit for testing the non-functional requirements using the proposed method.**

The adaptability of the ModelJUnit library to integrate with the process discussed in the section 4.2. for verification of non-functional requirements is also evaluated in terms of its test generation time, test coverage.

The evaluation of these objectives is discussed in chapter 8.

Case study is performed using the same workflow mentioned in the section 5.4.1:

1. ModelJUnit library (version 2.0) is used along with JDK 1.6.
2. Eclipse Ganymede (version 3.4) is used as IDE for modeling the SUT.
3. EFSM model is created using the library defined by the ModelJUnit.
4. One of the three different test configurations is applied on the model.
5. Offline test sequences are generated and observed.
6. There is no case study-specific system adapter designed to perform the test execution of the generated test cases.

7.3.1.1 Modeling the SUT

The model was built using information from the given requirement documents and the functional model implemented in Qtronic. The main target was to create test sequences for the IPSec implementation in the satisfying the relevant test purposes in the test specification document. The SUT model is built to test particular functional requirements as well as corresponding characteristics requirements. The model of the SUT should be created on abstraction level and only testable features of the SUT will be present in the model.

1. **Test specification for IPSEC**

The scope of the case study includes 48 functional requirements and 4 non-functional requirements from the verification test specification [27] of the IPSec implementation.

The testing requirements defined are highly abstract. They may miss many important details that are designed for building test cases. The testing requirements are also reviewed, analyzed and interpreted by the test engineer or test designer who knows about the IPSec implementation specification. Review of testing requirements is necessary in order to put them into a JAVA language used for modeling in ModelJUnit.

2. **Modeling approach in case study**

Modeling in ModelJUnit is quite different from the way it is done in other MBT tools. There isn't any graphic tool to draw the model. Program code is used for modeling. This Java code generates the model to be more precise.

Two different code parts are needed in order to produce the model. First one is the Model Program which consists of actions, rules and data about states.

The actions, states are defined as discussed in section 5.4.1 and rules are defined by the programmer which defines the behaviour of the SUT in response to the particular actions and data. Test Data is supplied from outside the model by the User. This data is an important factor for generation of test sequences.

The model was constructed according to specifications defined in the test purposes. The IPSec implementation behavior that was required to be tested by the test purposes was modeled. During the modeling of IPSec implementation, the IPSec behavior is represented in abstraction by means of Java code constructs defined in the ModelJUnit library. The resulting model was the deterministic EFSM of the IPSec configuration management and traffic handling that includes the behavior that was required to be tested which is given as test purposes. The model has 8 states, 22 transitions and 20 actions. A section of the model with functional requirements is presented in Figure 23.

The model consists of 8 JAVA files. Of them only one file constitutes the main model while others are helper files. Total lines of Java code used for modeling are approximately 2600. It is difficult to measure the amount of time spent on building the complete model as it almost generates instantly. In this case study the modeling approach was iterative, which include building a model with small sets of requirements and increment it with more requirements.

Figure 23: Graphical model covering functional requirements of SUT

When done with modeling for the functional requirements, we included the non-functional requirements in the model. This is done by means of adding extra actions, which performs the task of incorporating the non-functional requirement with functional requirement in the same model. The resulting model has 8 states, 23 transitions and 23 actions. The total lines of Java code used for complete model are approximately 2900. Figure 24 depicts the full model that includes both functional and non-functional requirements.

The modeling also includes study of ModelJUnit API. Approximately total time spent on learning ModelJUnit API and modeling specification of IPSec comes to 25 days.

Figure 24: Graphical model covering both functional and Non-Functional requirements of SUT

3. **Test Configuration**

Every model needs a test configuration. The model generation takes place during exploration. Exploration is an operation by which ModelJUnit engine systematically discovers all possible states defined by the model, and the steps to transition from one state to another. Exploration can be run on machines and the attached configuration determines its behavior.

The code for test configuration is given in table 5.

Table 5 : Test Configuration code for ModelJUnit

```
Tester tester = new RandomTester(new IPSECMain ());

// The guards make this a more difficult graph to explore, but we can increase the default maximum search to
//complete the exploration.

GraphListener graph = tester.buildGraph(100000);
```

```
try {
graph.printGraphDot("IPSecModel.dot");
} catch (FileNotFoundException e) {
e.printStackTrace();
}
CoverageMetric trans = tester.addCoverageMetric(new TransitionCoverage());
CoverageMetric trpairs = tester.addCoverageMetric(new TransitionPairCoverage());
CoverageMetric states = tester.addCoverageMetric(new StateCoverage());
CoverageMetric actions = tester.addCoverageMetric(new ActionCoverage());
tester.addListener("verbose");
// this illustrates how to generate tests up to a given level of coverage.
int steps = 0;
while (actions.getPercentage() < 100) {
tester.generate();
steps++;
}
System.out.println("Generated " + steps + “steps.");
tester.printCoverage();
```

This test configuration defines the test generation algorithm as Random Tester and also other settings such as test coverage metrics such as Transition Coverage, State Coverage, Action Coverage and Transition Pair Coverage are also defined.

7.3.1.2 Test Generation

The test generation is performed by executing the main program that represents the model. Result of the execution shows the log trace introduced to check whether particular requirements are full-filled by the generated test sequence.

The resultant test sequences coverage values are shown in discussion section.

7.4 Case Study with Qtronic

In this section we present the case study of modeling non-functional requirements of the IPSec SUT, using Qtronic based on the method that has been proposed in the section 4.2. This case study adapts the first phase of the method in which we have ready-made existing functional model of the SUT behavior and we need to include the non-functional specifications in it.

We have taken the existing functional model of IPSEC and understood its functionality. Based on that, we have selected some of the performance related specifications that can be included in the existing model.

7.4.1 Objectives of case study

The IPSec protocol implementation in telecommunication system was used in this case study. A working and functional model was already implemented using Qtronic in Ericsson. The main objective was to generate test sequences for the performance specifications of the IPSEC system. Only few requirements such as creation/deletion of maximum number of objects, tunnels and Security Associations etc were modeled.

The following aspects are to be evaluated in the case study.

- **Explore the feasibility of using the existing functional model for efficient performance related specifications modeling with different approaches.**

Aspects such as using of existing states or creation of new states in the existing behavioral work flow of the system or creating separate models for specific requirements and then associating with the existing states have to be explored as per the type of requirement.

After identifying the appropriate method for different categories of specifications, the performance of the test case generation was evaluated based on the factors such as effect on test generation time, increase in effort in terms of increase in the number of lines of code and test coverage etc.

- **Based on the results, propose the possible workarounds within the existing limitations of the tool**.

There can be problems and issues while adapting the new method for modeling the non-functional specifications of the system. In order to resolve them, some work-arounds or any other efficient ways of handling them have to be identified. Also limitations, if exists, have to be published. These objectives are evaluated in chapter 8.

Following setup was done before proceeding to the modeling of non-functional requirements:

- Qtronic 2.1.1 was installed in SUSE Linux Enterprise Desktop 10 system. (Current version is 4.2.0 and it is named as Conformiq Designer).
- Existing Qtronic functional model for IPSec is imported.
- The existing configuration settings were not changed and the performance related attributes of the IPSec were identified from the documents.

7.4.1.1 Modeling the SUT

1. **Test Specifications for IPSEC**

The scope of the case study includes 4 test specifications which include non- functional requirements in the verification test specification document of the IPSec implementation.

The test specifications defined are highly abstract. They may miss many important details that are designed for building test cases. The test sequences generated by the Qtronic are reviewed, analyzed and interpreted by the test engineer who knows about the IPSec implementation specification.

2. **Modeling approach in case study**

Figure 25 shows the actual functional model that was developed for IPSec protocol using Qtronic.

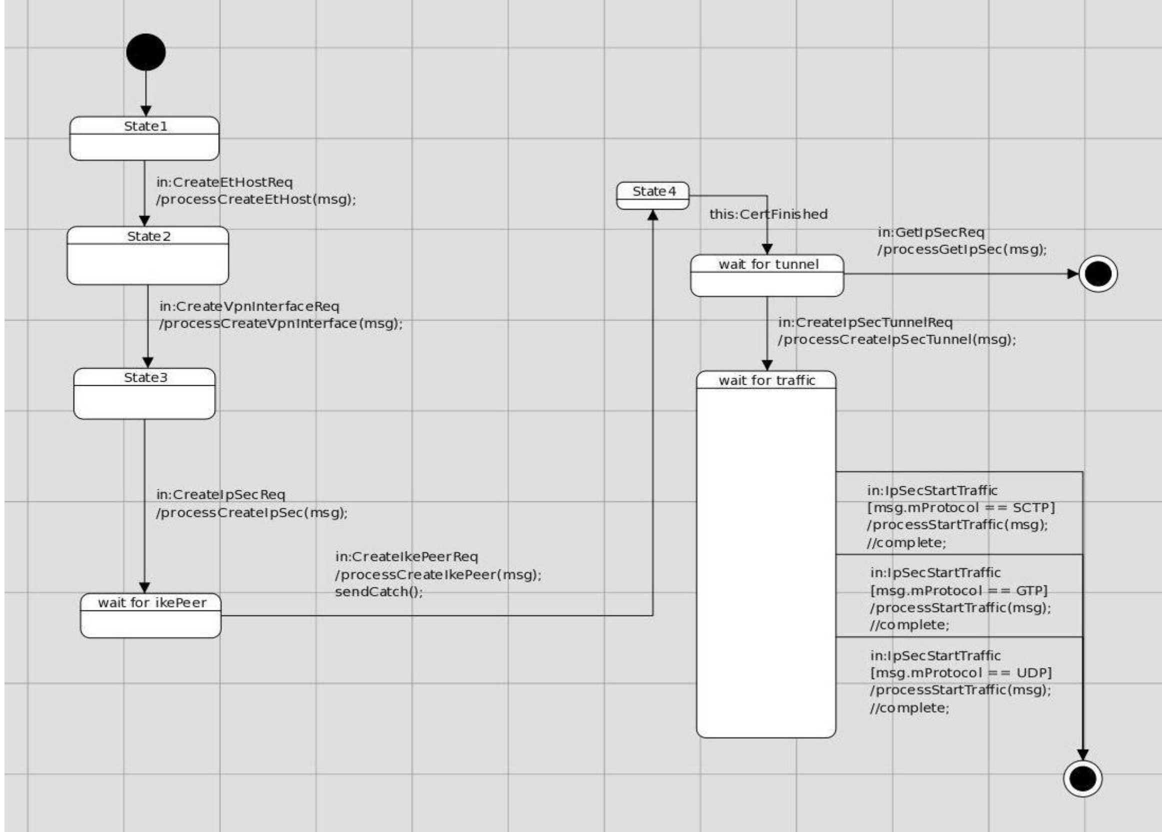

Figure 25: Existing Qtronic Functional Model

Before proceeding to model, it is to be noted that the new requirements which are introduced in the existing model can be distinguished from the existing ones by using the key word 'Performance' as prefix to the requirement statement.

Approach 1:

Once the setup is done, next step comes the modeling. Since most of the requirements involve testing of maximum number of objects creation, the states where the particular object is created are identified in the functional model. Initially we have started with a new transition to attain a new state separately and terminating it, in order not to disrupt the existing functional flow. Later it was found to be inappropriate since creation of another state doesn't really depict the exact behavior, instead adds to the complexity. The following [Figure 26] is the model which introduces new state (State 5) for adding the non-functional requirement.

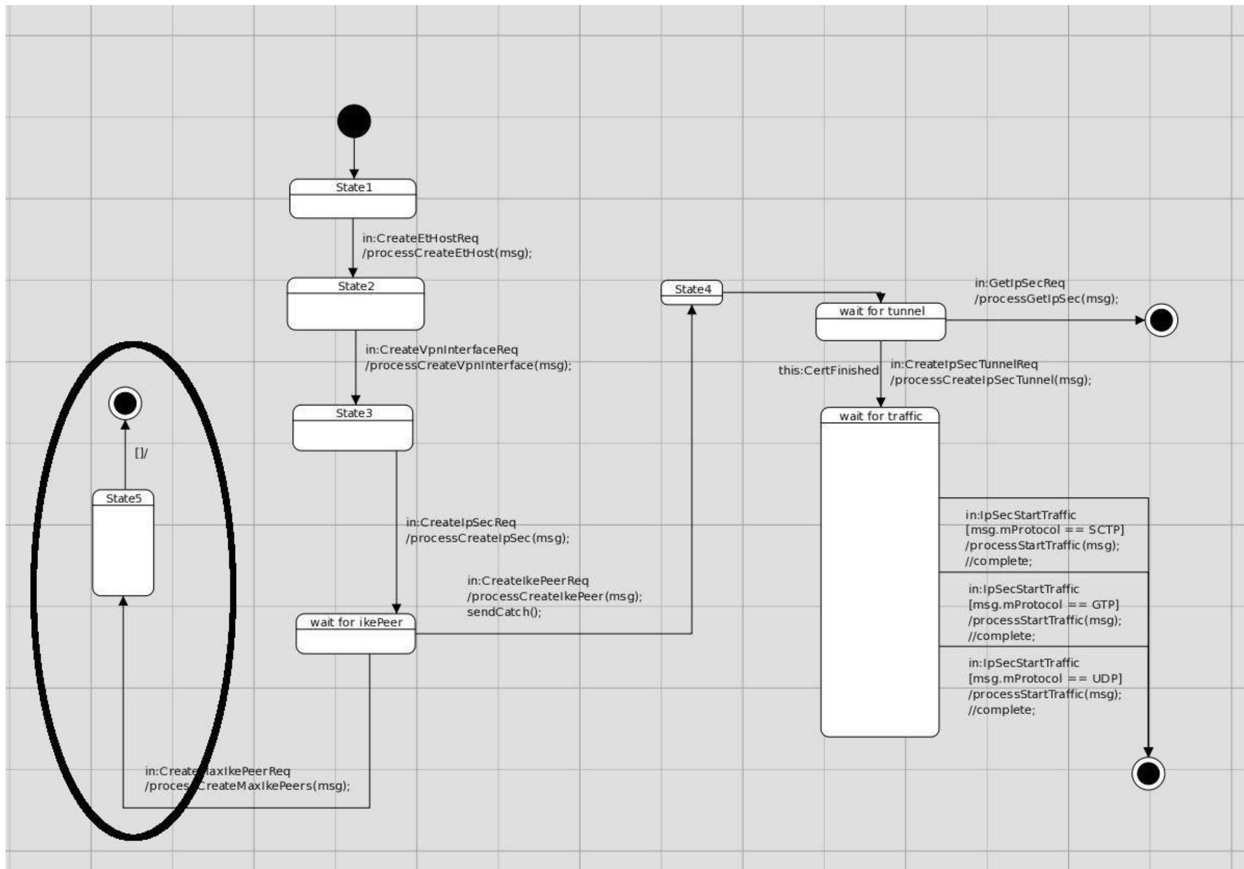

Figure 26 : Qtronic approach for NFR by introducing new state

Approach 2:

Another approach that was followed is to run the creation of objects in a loop using the existing messages in the state where a single object creation is done earlier. If the maximum number of the objects were created successfully, the last object can be used for further transitions in the existing functional model. But with the increase in the count on looping, it was observed that test generation time is also increasing proportionately. Moreover the size of the test case is also increasing and the steps are repetitive. This is effecting the abstraction which is the basic purpose of Model Based Testing.

Approach 3:

Our next approach was to explore the options in modifying the Qtronic scripter in a customized way. Qtronic provides a separate API for scripting backend which can be customized. This is used to render the test sequences in the user required notion. The QML record that is handled in the code for transition can be modified in the scripter. But this does not prevent us to use the loop in the model which again takes considerable amount of time.

Table 6: QML code for handling maximum number of objects creation

```
final int NFR_MAX_NR_OF_OBJECTS = 25;
    for (int i = 0; i < NFR_MAX_NR_OF_OBJECTS; i++)
    {
                this.mCurrentId++;
                        createMaxObjects(i, msg);
                sendMaxReplySignalCfm(msg.mCurrentId);
}
requirement "Performance/Max OBJECTS 25 Created/successful, one";
```

Approach 4:

One more approach of inclusion is to create a separate transition with a new method and message instead of using the existing methods and message which also proceed to the same state as the earlier one. This method was working with the generation of expected test cases fulfilling the new requirements also. This approach finally generates the required test sequences covering the newly added non-functional requirements. This was mentioned in the Figure 27.

Approach 5:

The untested approach which we thought could be beneficial was to handle the multiple creations of objects in Test Harness (the glue code used for automation). In the QML code we set the variable 'MAX_NUMBER_OF_OBJECTS' to 1 and run the loop only once to create one object and generate the test sequence. While modifying the Test harness, handle the variable to the maximum preferred value. This reduces the effort on the Qtronic modeling and also reduces the test generation time. But additional effort needs to be put in modifying the harness extensively if the number of such kind of requirements increases. One hazard of this approach can be increase in the size of harness.

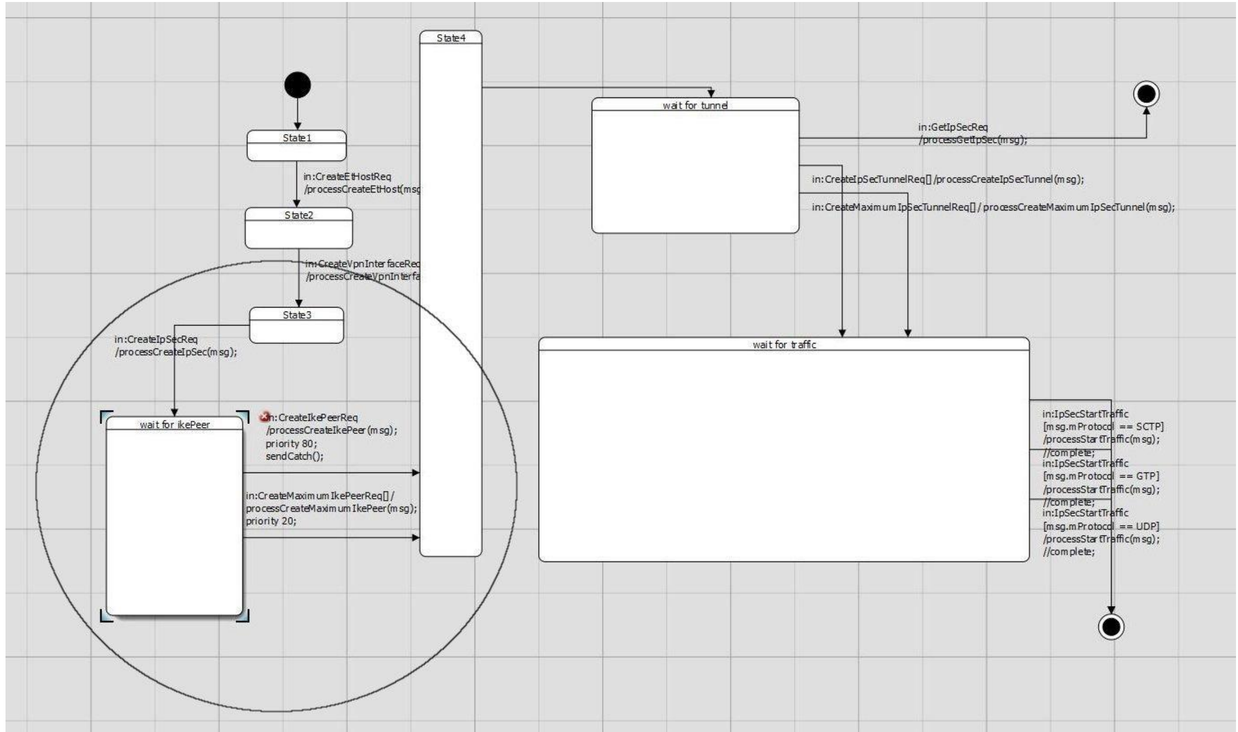

Figure 27: Addition of new transition approach in Qtronic

8. Discussion

This chapter devout to a discussion on the evaluation of the objectives of case study, approaches used in the above case studies and presents an overview of the observations made. Based on that, some conclusions can be drawn. The observations are based on some of the factors for each of the approaches we have discussed earlier, such as modeling time for additional requirements, test case generation time, changes in the models (in terms of lines of code, look-ahead depth etc), number of test cases.

8.1 Qtronic

In case of Qtronic, there is an existing functional model of IPSec which was used for including performance related specifications. Two of such models related to Certificate and Traffic were chosen. We have chosen around three requirements to model additionally.

Before analyzing the newly modeled requirement and the test cases that are thus generated, here are some of the observations related to the existing model.

Table 7 : Data Values for the Initial Functional Models in Qtronic

Model Parameters	Initial Data Values for Certificate Model	Initial Data Values for Traffic Model
Look Ahead Depth	Level1	Level1
Number of Generated Test Cases	8 TC with 156 steps	40 TC with 901
Test Generation time(in seconds)	28	286
Number of Requirements Considered in Modeling	72	72
Number of Requirements Covered	24	72

As per the proposed method for non-functional requirements modeling, a logical grouping of requirements should be done. Based on similar characteristics we have identified some requirements and grouped them. In this document we have provided the results of modeling one requirement from each group.

Requirement 1:

This requirement involves creation of maximum number of objects. In the functional model (both Certificate and Traffic Model) a single object creation is considered. We initially started with adding a new method in the Certificate Model, which creates the maximum number of objects (25) in a loop. But that lead to higher test generation time (32 seconds for 25 objects). If further the number of maximum

objects is increased, that proportionally increases generation time. The increase in test generation time is same even if we create a separate state for maximum number of objects creation or use the same state of object creation as in functional model for transition. This is related to the Approach 2 mentioned in the section 7.4.

In this case the optimum solution was to set the value of maximum number of objects to 1 (Approach 5) and generate test sequences as the test case generation time was not changed. Once the test sequence is generated during automation, changes in harness have to be made in such a way that the value of maximum number of objects is set to desired value.

Table 8 : Data values for Requirement 1 using Approaches 2 and 5 in Qtronic

Model Parameters	**Changed Data Values for Certificate Model** **Approach 2**	**Changed Data Values for Certificate Model** **Approach 5**
Look Ahead Depth	Level1	Level1
Number of Generated Test Cases	9 TC with 184 steps	9 TC with 184
Test Generation time(in seconds)	32	28
Number of Requirements Considered in Modeling	73	73
Number of Requirements Covered	25	25
Approximate new lines of code	80	80

Requirement 2:

This requirement involves checking of packet transmission delay under normal traffic load conditions. This requirement was modeled in Traffic model. The requirement involves sending 120000 packets per second of fixed size and the delay shouldn't be more than a specified time. In this case, since including the huge number for number of packets increase test generation time, in the model we have only considered 1 packet and the Maximum number will be handled in the harness as we did in the previous requirement (Approach 5). Since the time measurement is done through another tool, the timing details are excluded from the model. The only change observed from the original model is the test generation time. With the inclusion of these two requirements the test generation time increased by 13 seconds. This is modeled by adding a separate case in the existing model. This consideration added to two separate new states.

Table 9 : Data values for Requirement 2 using approach 5 in Qtronic

Model Parameters	**Changed Data Values for Traffic Model** **Approach 5**
Look Ahead Depth	Level1
Number of Generated Test Cases	42 TC with 943 steps
Test Generation time(in seconds)	299
Number of Requirements Considered in Modeling	74
Number of Requirements Covered	74
Approximate new lines of code	35

Requirement 3:

The third requirement involves measuring the time taken for a sequence of steps involved in configuration. The sequence is almost similar to the one that is included in the functional model. This requirement is added in the existing certificate model by considering as a special case. Only additional step is to wait for the receiving of a signal that marks the stop time. The signal sending is included in the model. There are two options of measuring the timing in this context. One is to include timing related variables in the model and the other is to handle it in harness (Approach 5). We have chosen the later to make the model as simple and as abstract as possible.

Interestingly, the model didn't cover all the scenarios using the look-ahead depth level 1. It generated the same test cases as it generated for original functional model with an additional test case that didn't cover the newly added requirement. The additional special case that was included in the new model was not considered by Qtronic with look-ahead depth level 1. The number of test cases is increased when the look-ahead depth was raised to level 2. Also the newly modeled requirement is covered when the look-ahead depth has been changed to 2.

Table 10 : Data values of Requirement 3 with different look-ahead depths

Model Parameters	**Changed Data Values for Certificate Model with look-ahead depth level 1**	**Changed Data Values for Certificate Model with look-ahead depth level 2**
Number of Generated Test Cases	9 TC with 174 steps	16 TC with 349 steps

Test Generation time(in seconds)	31	277
Number of Requirements Considered in Modeling	74	74
Number of Requirements Covered	24	26
Approximate new lines of code	90	90

From the above observations mostly it is evident that using Qtronic, performance related requirements can be included in the existing functional model without increasing the complexity much. But at the same time effort should be put more in modifying harness. With the increase in number of requirements, the efforts will also increase proportionally. In the approaches mentioned in case study, the best approach suitable with the existing environmental facilities is to include non-functional requirements in existing functional model, make the model as simple as possible and then handle the rest in harness.

Here are some of the problems we faced during modeling with Qtronic.

1) Qtronic does not provide the option to choose the type of algorithm. Thus produces different test cases at different times.

2) There is no separate feature for providing test data externally.

3) There is no facility for importing models from several real times profiles like Marte.

4) Some of the basic Java IDE features can be provided may be by a separate plug-in. It is difficult to debug the methods and variable definitions.

5) The debugging and error traceability features are not very much efficient.

6) It doesn't support iterations for higher count.

8.2 ModelJUnit

Unlike Qtronic, there is no functional model ready in hand in ModelJUnit to work with modeling of non-functional requirements. According to new proposed model, we need to have an existing functional model in place to include performance related specifications. As mentioned in the case study, using the specifications given for Qtronic modeling we have developed a new functional model in ModelJUnit. Once a stabilized functional model is achieved we included some performance related requirements in it. The requirements can be broadly classified into two categories. One related to the FSM model and the other one related to Timed FSM model.

The greatest advantage of using ModelJUnit probably is the test sequence generation time for FSM models. It takes a few seconds to generate the test sequences which are very much negligible when compared to that of Qtronic. Even addition of new requirements doesn't show much variation in test

sequence generation time. Since the algorithm used in test sequence generation is 'Random Tester' the concept of look-ahead depth doesn't come into picture.

Here are some of the configuration details of the functional model developed in ModelJUnit.

Table 11 : Data Values of the initial functional model in ModelJUnit

Algorithm	Random Tester
Action Coverage	20/20
State Coverage	8/8
Transition Coverage	20/20
Transition Pair Coverage	42/92
Number of Test Sequence Generated	348

Here we present the results and discussion about two requirements, each one belonging to different categories mentioned above.

Requirement 1:

This requirement involves creation of maximum number of objects and is done in FSM. In Qtronic we excluded the maximum number and modeled with one object creation leaving the handling to harness in order to reduce the test generation time. Since in ModelJUnit, the test generation is taking less time, the requirement is included by considering the maximum number (Approach 2 in Qtronic). Additional action function along with the existing guard condition is used for the new requirement. No additional state is included. Even though the time for test generation increased it was in few seconds. But the number of test sequences increased drastically, because increase in number of objects increased branch iterations proportionally.

Table 12 : Data Values of Requirement 1 modeling in ModelJUnit

Algorithm	Random Tester
Action Coverage	21/21
State Coverage	8/8
Transition Coverage	21/21
Transition Pair Coverage	57/104
Number of Test Sequences Generated	1279
Approximate Increase in Code(In lines)	35

Requirement 2:

This requirement is related to the checking of packet transmission delay under normal traffic load conditions. The requirement involves sending 1200000 packets per second of fixed size and the delay shouldn't be more than a specified time. The behavior was modeled as new Timed FSM that required changes in the existing functional model also. This resulted in reduction of number of actions and transitions. But no additional state is included. Since there is an involvement of huge number the time taken for test case generation is high (approximately 30 mins). In this case also it is better to handle the number to one and handle it in harness (Approach 5 in Qtronic).

Table 13 : Data Values of Requirement 2 modeling in ModelJUnit

Algorithm	Random Tester
Action Coverage	17/17
State Coverage	8/8
Transition Coverage	18/18
Transition Pair Coverage	28/39
Number of Test Sequences Generated	165
Approximate Increase in Code(In lines)	15

From the above results it is evident that the advantage of using this tool over Qtronic is its capacity of generating test sequences in shorter time. Moreover there is no need of separate modeling in this tool which reduces the effort of changes in model along with the changes in code for an additional requirement. Since handling of high number is a problem in both of the tools, it is always better to handle such variables in harness (Approach 5 in Qtronic).

Here are some of the problems we faced during modeling with ModelJUnit.

1) In ModelJUnit Non-deterministic models cannot be handled efficiently even in harness.
2) ModelJUnit cannot handle hierarchical states and sub-states.
3) There is no support for the import of UML models from other tools and integration with other tools.
4) There is no provision of graphical vision of test sequences.
5) Only a minimal number of coverage criteria (4) are provided.

9. Conclusion

Given the time limit of this thesis work, only two tools were tested for case studies. From the analysis, it was observed that the new method of inclusion of non-functional requirements in the functional model will work well and is applicable in most of the cases with similar type of test environments and testing methodologies. As per the original MDPE, it was beneficial if we have a parallel performance evaluation during the development itself. But in the testing scenario in Ericsson, the choices are two. One is to consider the performance related specifications for modeling along with the functional requirements. Second one is related to the availability of readymade functional model.

Of the many approaches that were proposed, the modeling effort was optimal in the cases when the actual data handling is considered for handling in harness and the abstract behavior for modeling. This keeps the model simple, easily modifiable and short.

Though the efficiency of a tool is evaluated based on many factors, it was observed that mostly the test generation time, number of lines addition in the code, efforts for re-modeling, adaptability to existing framework and quality of test cases are the main determining factors for generation of test sequences.

Though Qtronic has given satisfying results with the given constraints, ModelJUnit has also got additional benefits in modeling. Especially the characteristics such as open source nature, less test sequence generation time, less effort in modeling make ModelJUnit also to be one of the suggestible options for the existing testing framework and environment.

10. Future Work

The following are some of the recommendations for future work.

1) This thesis work was limited only to the part of test sequence generation. Equally important amount of effort should be spent in designing of the harness and checking for the compatibility with the adapters and existing testing framework. One more reason is the consideration of data handling much in harness as useful approach. This approach is experimentally not validated.

2) Since the thesis work is mostly based upon the performance related requirements modeling, other related attributes such as security, portability, scalability and robustness etc can also be verified in future.

3) Case studies can also be performed in other MBT tools such as Elvior Motes, Smartesting, and Spec explorer.

11. References

[1] MBT on Wikipedia,http://en.wikipedia.org/wiki/Model-based_testing

[2] M. Utting and B. Legeard. Practical model-based testing: a tools approach. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2006

[3] Conformiq Qtronic Homepage,http://www.conformiq.com

[4] All4Tec MaTeLo Homepage,http://www.all4tec.net

[5] Elvior Motes Homepage,http://www.elvior.ee/motes

[6] MBT Tigris Homepage,http://mbt.tigris.org/

[7] RFC-Internet Security Protocol,http://www.ietf.org/rfc/rfc2401.txt

[8] General Information on Internet Security Protocol, http://www.tcpipguide.com/free/t_IPSecGeneralOperationComponentsandProtocols

[9] Ernits, Juhan-P.; Kull, Andres; Raiend, Kullo; Vain, Jüri. Generating tests from EFSM models using guided model checking and iterated search refinement. In: Formal Approaches to Software Testing and Runtime Verification : First Combined International Workshops FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006, Revised Selected Papers: Havelund, K., et al.. Berlin: Springer, 2006, (Lecture Notes in Computer Science; 4262), 85 - 99.

[10] Ernits, Juhan P.; Kull, Andres; Raiend, Kullo; Vain, Jüri. Generating TTCN-3 test cases from EFSM models of reactive software using model checking . In: Informatik 2006 - Informatik für Menschen : Proceedings: Beiträge der 36.Jahrestagung der Gesellschaft für Informatik e.V.(GI), 2.bis 6.Oktober 2006 in Dresden. (eds.) Hochberger, Ch.; Liskowsky, R.. Bonn: Köllen, 2006, (Lecture Notes in Informatics; P-94), 241 – 248.

[11] Vain, J.: Raiend, K.; Kull, A.; Ernits, J. Synthesis of test purpose directed reactive planning tester for nondeterministic systems. In: ASE'07 : 2007 ACM/IEEE International Conference on Automated Software Engineering, Atlanta, Georgia, November 5-9, 2007, proceedings: 22nd IEEE/ACM International Conference on Automated Software Engineering. ACM Press, 2007, 363 - 372.

[12] Elvior Motes User Manual, http://www.elvior.com/files/TestGenerator.zip

[13] Qtronic User Manual,http://www.conformiq.com/downloads/Conformiq4dot2dot0Manual.pdf

[14] ModelJUnit Home Page,http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit/

[15] Ericsson specific document for Functional Specification for the IPSEC System Function in CPP

[16] O. Taipale, K. Smolander, and H. Kälviäinen. Finding and ranking research directions for software testing. Lecture notes in computer science, pages 39-48, 2005.

[17] Fritzsche, M., Johannes, J., Zschaler, S., Zherebtsov,A., Terekhov, A.: Application of tracing techniques in model-driven performance engineering. In: ECMDAFA 4th workshop on traceability. (2008)

[18] Victor R. Basili , Richard W. Selby, Comparing the effectiveness of software testing strategies, IEEE Transactions on Software Engineering, v.13 n.12, p.1278-1296, December 1, 1987

[19] B. Beizer, Van Nostrand Reinhold, Software Testing Techniques, 2nd edition, 1990.

[20] Black –box vs. White-box testing: Choosing the Right Approach to deliver Quality Applications ,www.testplant.com/download_files/BB_vs_WB_Testing.pdf

[21] Elfriede Dustin, Thom Garrett, and Bernie Gauf, Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Addison- Wesley Professional, 2009)

[22] C. Campbell, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Model-based testing of object-oriented reactive systems with Spec Explorer. Tech. Rep. MSR-TR-2005-59, Microsoft Research, 2005.

[23] M. Utting, A. Pretschner, and B. Legeard. A Taxonomy of Model-Based Testing. Department of Computer Science, The University of Waikato, New Zealand, Tech. Rep, 4, 2006

[24] Ericsson Internal Document, https://ericoll.internal.ericsson.com/sites/FTP_DN/pmt/ModelBasedTest/Wikis/Home.aspx

[25] Mathias Fritzsche, Jendrik Johannes, “Putting Performance Engineering into Model-Driven Engineering: Model-Driven Performance Engineering”, MoDeVVa’07 at the Model Driven Engineering Languages and Systems, ISBN 2-7261-1294 3, 2007, pp. 77–87, (selected to appear in Springer-Verlag LNCS format).

[26] Ericsson specific document for Functional Specification for the IPSEC System Function in CPP.

[27] Ericsson Specific document for Verification Specification for the IPSEC System Function in CPP.