AUTOMATED SOFTWARE TESTING USING COVERING ARRAYS

by

CHINMAY P. JAYASWAL

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2006

ACKNOWLEDGEMENTS

I am grateful to Dr.Lei for helping me tremendously in my work and for giving me the opportunity to work under him. I am thankful to Dr.Kung and Dr.Fegaras for serving as committee members for my defense. I would also like to thank Arun Ramani, Keerthika Koteeswaran and all the members of the Software Engineering Research Group led by Dr.Lei who helped me during my work.

I would like to acknowledge all my friends who encouraged and motivated me throughout my work.

Last but not least, I would like to mention my family, especially my dad and mom, Prakashbhai and Purnima Jayaswal. They have always been the source of inspiration and strength for me. I have reached where I am today because of their constant moral support throughout my studies, and I know they will serve as the foundation of my future success, both personally and professionally.

November 15, 2006

ABSTRACT

AUTOMATED SOFTWARE TESTING USING
COVERING ARRAYS

Publication No. ______

Chinmay P. Jayaswal, M.S.
The University of Texas at Arlington, 2006

Supervising Professor: Dr. Jeff Yu Lei

Modern society is increasingly dependent on the quality of software systems. Software testing is a widely used approach to ensure software quality. Since exhaustive testing is impractical due to resource constraints, it is necessary to strike a balance between test efforts and quality assurance. Interaction testing is one approach to marrying these two qualities. It characterizes the system under test by a set of parameters and the respective test values (domain size) for each parameter. Instead of testing all possible combinations of values for all the parameters, interaction testing constructs a covering array as a test set to cover all the t-way combinations (i.e., combinations involving t parameters, where t is referred to as the degree of interaction and is usually small). Each combination of values in a set of parameters represents a possible interaction among those parameters. The rationale of interaction testing is that not every interaction contributes to every fault, and many faults can be exposed by the interactions among a small number of parameters. Empirical studies have shown that interaction testing can significantly reduce the number of tests while still detecting faults effectively.

This thesis mainly describes the GUI of an interaction testing tool called FireEye. FireEye constructs covering arrays that provide multi-way coverage for up to 6-way testing. We focus on the design and implementation of the GUI for FireEye. The GUI is developed using Java Swing. Software testing demands a great deal of time and money, so it is necessary to save these resources wherever possible. One way to meet time and resource constraints is to develop a user friendly GUI that can save the tester some time while also being able to rapidly generate test cases. Various goals such as ease of use, interactivity and portability were kept in mind while designing the GUI paradigm (model of interaction) for FireEye. Other features of FireEyes GUI included the following: decreasing the time necessary for the tester to analyze the test configuration, minimal hardware requirements, easy installation, and no high end software requirements.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

Many software systems today are built using various components. Often, system faults are caused by unexpected interactions among these components. One solution to remove any such faults from a system is software testing. Testing is a process that requires a great deal of time and resources. It is widely recognized in the computer science community that testing consumes approximately 50% of the total cost of developing new software. Furthermore, the cost of testing new hardware and safety critical systems is even higher. Inadequate testing can lead to catastrophic consequences. Testing is an important but expensive part of the software and hardware development process. To thoroughly test a large software or hardware system, many combinations of possible inputs must be tried and the expected behavior of the system must be verified against the systems requirements. However, the size of a test suite required to test all possible interaction combinations could be prohibitive in even a moderately sized project. Therefore, it is necessary to decrease the set of test configurations by selectively testing only a subset of this test configuration.

One approach to software testing is pairwise testing. Pairwise testing helps in detecting faults caused by interactions among two parameters. However, it is not necessary that faults are only caused by the interaction between two parameters. There are chances that faults can be caused by the interaction of more than two parameters. In fact, a study conducted by NIST has shown that about 95% of actual faults involved up to 4-way interactions in the software studied. Therefore, it is necessary to test interactions between more than two parameters.

A strategy that tests interactions among more than two parameters is t-way testing. T-way testing, where the value of t is usually small and is referred to as the degree of interaction, requires that for any t parameters, every combination of their values should be covered by at least one test. T-way testing guarantees that all t-way combinations are tested together. The main principle behind it is that not every parameter is responsible for every fault in a system, and many faults can be exposed by interactions involving only a few parameters.

One such tool that implements the t-way or the multi-way testing is FireEye. This thesis focuses on developing an interactive user interface for FireEye that can be easy to use for the tester and at the same time provide him with the test suite in an easy-to-read format.

1.2 Structure of Thesis

The thesis is structured as follows: Chapter 1 gives a brief overview of the thesis work. Chapter 2 deals with the background and related work for t-way testing. Chapter 3 gives a detailed explanation about pairwise and multiway testing. Chapter 4 explains the IPO algorithm in a nutshell and the IPOG algorithm in detail. Chapter 5 deals with the information FireEye tool, the design and implementation of the GUI for FireEye as well as various experimental statistics carried out on FireEye. Lastly, Chapter 6 concludes with the goals achieved during my work.

CHAPTER 2

RELATED WORK

A reasonable amount of work has been done on t-way testing in the past, but most of it focused on pairwise or 2-way testing. Various tools are available which implement these approaches. What follows is a brief overview on such work previously carried out or work which is still in progress.

2.1 Orthogonal Arrays

The case study of testing AT & T PMX/ StarMail using OATS [4], mentions a tool called OATS (Orthogonal Array Test System), which was used to generate orthogonal arrays. In this case, the method is described as Robust Testing. The paper discuses how a large number of system configurations and short testing intervals led to the first application of orthogonal arrays to achieve pairwise testing. Testing using conventional methods required a lot of resources and the number of tests were too large for the time available. However, an analysis after the execution of the robust testing suite showed that 12% of the faults found using this method would have been missed by the original test plan. Furthermore, if there is a huge test plan, 10% of the test faults found would be in the test cases, which would have to be dropped due to time constraints.

2.2 AETG and CATS

AETG (Automatic Efficient Test Generator) developed at BellCore and CATS (Constrained Test Generator) developed at Bell Labs uses the Combinatorial Testing approach. Out of the two, AETG seem to generate fewer test cases then CAT. The reason being that CATS doesnt have notion of explicit constraints or hierarchy that AETG has. Instead, it uses multiple relations to express constraints and that requires more tests then necessary. AETG is a combinatorial design algorithm and follows computational approach. Combinatorial designs are mathematical constructions widely used in medical and industrial research to construct efficient statistical experiments. It uses a greedy strategy in selecting the test cases. As it uses computational approach, it involves explicitly enumerating all possible combinations. Moreover, computational approaches can be applied to an arbitrary system configuration. It creates many different candidate test cases and selects from the one that covers the greatest number of new combinations. AETG generates only pairwise (2-way) or triple (3-way) test coverage where as CATS generates test sets for n-way combinations.

2.3 TSUNAMI

Ted [5] presents an algorithm for generating tests for single stuck line faults using a combination of algebraic processing and conventional path oriented search. The algorithm used is named as Tsunami. This method uses the algebraic methods to determine the complete set of input set of assignments which will propagate an error signal through a gate in a path to a primary output. In algebraic methods, test sets are derived from covering arrays without performing any explicit enumeration of the combinations to be covered. Due to this reason, algebraic approaches are not affected by combinatorial explosion and are fast in execution. Though, the use of algebraic approaches is restricted due to few limitations like fixed domain size i.e each parameter should have same number of values.

2.4 IBM's Intelligent Test Case Handler

IBMs Intelligent Test Case Handler also known as ITCH tool, uses the sophisticated combinatorial algorithms to construct test suites. It enables the user to generate small test suites with strong coverage properties, choose regression suites and perform other useful operations for the creation of systematic software test plans. Though, it only supports 2,3 and 4 way testing.

CHAPTER 3

UNDERSTANDING PAIRWISE AND MULTIWAY TESTING

3.1 Pairwise Testing

Pairwise (2-way testing) is a specification-based criteria in which every combination of valid values of each pair of input parameters should be covered by at least one test case. To understand Pairwise testing, consider the following system.

Table 3.1. Parameters and its values for a system

Parameter	Values
A	A1, A2, A3
B	B1, B2, B3
C	C1, C2

For parameters A and B the test sets are {A1,B1}, {A1,B2}, {A1,B3},{A2,B1},{A2,B2}, {A2,B3},{A3,B1},{A3,B2},{A3,B3}. For all the three parameters, a considerable size of pairwise tests exists. Below are a few extracts from the test configurations:

{A1,B1,C1},{A1,B1,C2},{A1,B2,C1},{A1,B2,C2}, {A1,B3,C1},{A1,B3,C2}

{A2,B1,C1},{A2,B1,C2}, {A2,B2,C1},{A2,B2,C2},{A2,B3,C1},{A2,B3,C2}

The IPO (In-Parameter-Order) strategy is used for Pairwise testing. This strategy generates a Pairwise test set for the first two parameters. It then extends the test set to generate a Pairwise test set for the first three parameters and continues to do so for each additional parameter.

3.2 Multi-way Testing

T-way Testing (or Multi-way testing) is a type of interaction testing which requires that for each t-way combination of input parameters of a system, every combination of valid values of these t parameters must be covered by at least one test case. It involves selecting test scenarios in such a manner that it covers all the t-wise interactions between the parameters and the values of a given system. For example, consider a system that must function on three operating systems, two browsers, three printers and two communication protocols.

Table 3.2. Software system under test

Operating System	Browser	Printer	Protocol
Windows	Explorer	IBM	Ethernet
Linux	Mozilla	HP	Token Ring
Solaris		Epson	

Although there are 3 * 3 * 2 * 2 = 36 possible test configurations, just nine tests as shown below in figure 1.2 cover all the 2-way interaction between different parameters of the system. Following table 3.1 describes the test suite for the system.

Table 3.3. Set of test cases for a system

Operating System	Browser	Printer	Protocol
Windows	Expolrer	Epson	TokenRing
Windows	Explorer	Epson	Token Ring
Windows	Mozilla	HP	Ethernet
Windows	Explorer	IBM	Ethernet
Linux	Mozilla	Epson	Token Ring
Linux	Explorer	HP	Ethernet
Linux	Mozilla	IBM	Token Ring
Solaris	Explorer	Epson	Ethernet
Solaris	Mozilla	HP	Token Ring
Solaris	Explorer	Epson	Ethernet

A subset of combinations that covers all t-way interactions at least once is called a covering array (CA) of strength t. A study of actual faults conducted by NIST has shown that about 95% of faults involved 4-way interactions in the software being investigated.

Moreover, almost all faults could be detected by 6-way interaction testing in the types of software that were investigated. Thus, test suites based on covering arrays of strength up to t = 6 are needed. Because it is quite difficult to manage, execute and evaluate a test involving a very large number of parameters and test runs, automated tools are needed.

CHAPTER 4

THE STRATEGY

FireEye uses the IPOG strategy, which is a generalization of the IPO (In-Parameter-Order) from pairwise testing to multi-way testing. In multiway testing the number of combinations grows exponentially as the degree of interaction increases and therefore, the requirement for time and space is more noticeable as compared to pairwise testing.

4.1 IPO Overview

IPO is a test generation strategy used for Pairwise (2-way) testing. For a system with two or more input parameters, the IPO strategy first generates a pairwise test set for the first two parameters. It then continues to extend the test set to generate a pairwise test set for the first three parameters and continues to do so for each additional parameter until all the parameters of the system are covered.

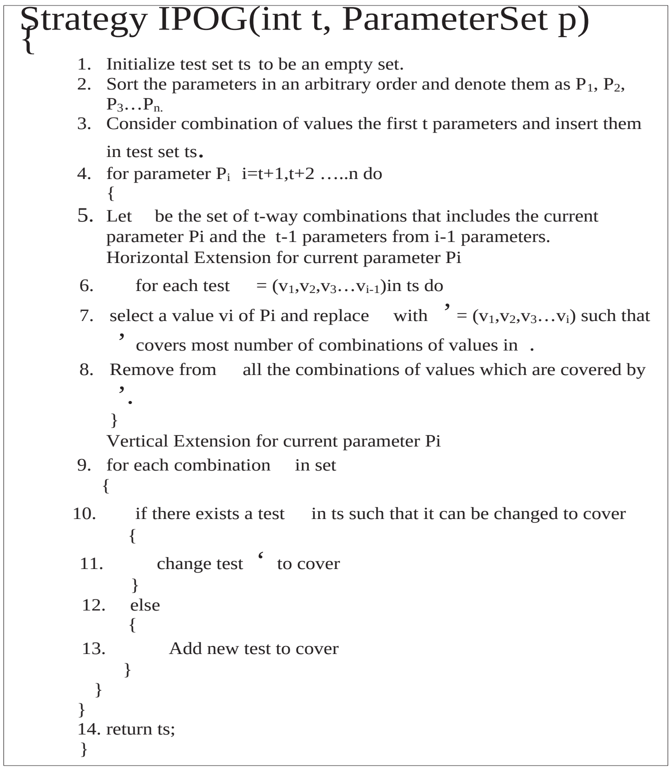
IPO follows two steps to extend the test when additional parameters are added:

1) Horizontal Growth, which extends each additional test by adding one value of the new parameter

2) Vertical Growth, which adds new tests if required after the completion of Horizontal growth.

4.2 IPOG strategy

Multi-way testing has a high demand of time and space requirements as compared to pairwise testing because the number of combinations increases exponentially as the degree of interaction increases. As a solution, the IPOG strategy is introduced as a generalization of the IPO strategy from pairwise testing to multi-way testing.



```
Strategy IPOG(int t, ParameterSet p)
{
    1.  Initialize test set ts to be an empty set.
    2.  Sort the parameters in an arbitrary order and denote them as $P_1$, $P_2$,
        $P_3 \ldots P_n$.
    3.  Consider combination of values the first t parameters and insert them
        in test set ts.
    4.  for parameter $P_i$  i=t+1,t+2 …..n do
        {
    5.  Let    be the set of t-way combinations that includes the current
        parameter Pi and the  t-1 parameters from i-1 parameters.
        Horizontal Extension for current parameter Pi
    6.      for each test     = $(v_1,v_2,v_3 \ldots v_{i-1})$in ts do
    7.   select a value vi of Pi and replace     with  ’ = $(v_1,v_2,v_3 \ldots v_i)$ such that
          ’  covers most number of combinations of values in  .
    8.   Remove from     all the combinations of values which are covered by
          ’.
         }
        Vertical Extension for current parameter Pi
    9.  for each combination    in set
       {
   10.      if there exists a test     in ts such that it can be changed to cover
             {
   11.          change test  ‘  to cover
             }
   12.      else
             {
   13.            Add new test to cover
            }
      }
}
14. return ts;
}
```

Figure 4.1. Algorithm for IPOG Strategy.

The framework of the IPOG strategy can be illustrated as follows: For a system with at least t or more parameters, the IPOG strategy constructs a t-way test set configuration for the first t parameters. Then it extends the test set to construct a t-way test set for the t+1 parameters, after which it continues to extend the test set until a t-way test set has been constructed for all the parameters of the system.

Like IPO, IPOG too follows the same steps for the extension of the test set.

1) Horizontal Growth, which extends each additional test by adding one value of the new parameter

2) Vertical Growth, which adds new tests if required after the completion of Horizontal growth.

Figure 4.1 illustrates the framework for the IPOG strategy. The inputs to the algorithm are the degree of interaction t and the set of parameters ps. The output is a t-way test set for all the parameters in the system. The above framework is explained as follows: Consider a system having four parameters

A: A1, A2

B: B1, B2

C: C1, C2

D: D1, D2, D3.

First, the test set ts, which is used to store the resulting test set is set to null (line 1), and the parameters are sorted in an arbitrary manner (line 2). Next, t parameters are selected and their every combination is added to test set ts and thus, the first ts test set is ready (line 3).

Table 4.1. Sample System

A	B	C
A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B1	C1
A2	B1	C2
A2	B2	C1
A2	B2	C2

As shown in Table 4.1, the 3-way test set is shown for the first three parameters. Let P_i be the current parameter that is being covered. Now we compute the set π of combinations that must be covered in order to cover P_i. (line 5). In order to cover D, we need to cover all the 3-way combinations of the following groups (A,B,D), (A,C,D) and (B,C,D).

Next, the combinations in set π are covered by the Horizontal and Vertical Growth.

Horizontal Growth: (Lines 6 8)

Table 4.2. Horizontal Growth

A	B	C	D
A1	B1	C1	D1
A1	B1	C2	D2
A1	B2	C1	D3
A1	B2	C2	D1
A2	B1	C1	D2
A2	B1	C2	D3
A2	B2	C1	D1
A2	B2	C2	D2

This step covers all the remaining uncovered combinations either by changing the existing test set or by adding a new test set. Note that when we change the existing test set it means only the dont care values can be changed. A dont care is an entity, whose value can be changed without affecting the coverage of the test set.

Table 4.3. Vertical Growth

A	B	C	D
A1	B1	C1	D1
A1	B1	C2	D2
A1	B2	C1	D3
A1	B2	C2	D1
A2	B1	C1	D2
A2	B1	C2	D3
A2	B2	C1	D1
A2	B2	C2	D2
A2	B1	C2	D1
A1	B2	C1	D2
A1	B1	C2	D3
A2	B2	C1	D3
dc	B1	C1	D3
dc	B2	C2	D3

Vertical Growth: (Lines 9 to 12)

σ is the t-way combination involving the parameters P$_k$1, P$_k$2, P$_k t$. The existing test τ can be changed to cover σ if and only if the value of the parameters in the set is either the same as in σ or is a dont care condition. If no existing test can be changed to cover σ, a new test needs to be added in which the value of P$_k$ is assigned the same value as in σ, and the other parameters are assigned the dont care values.

CHAPTER 5

FIREEYE: AN INTERACTION TESTING TOOL

5.1 FireEye Overview

As mentioned before, a common source of system faults is unexpected interactions between system components. A system tester generally faces the constraints of time and money, and testing all possible configurations for the system within any reasonable allotment of resources is not possible. For each system test configuration, a system test suite must be executed. Changing between configurations normally requires additional effort. Therefore, it is necessary to reduce the size of the system test configuration, which in turn can help to effectively manage the available resources.

FireEye is one such tool. It is a t-way testing tool with an interactive user interface that supports multi-way interaction testing for non-homogenous arbitrary configurations. This tool implements the IPOG (In-Parameter-Order) strategy and is written in Java. The front-end for this tool is written using Java Swings. Effort has been made to make this tool as user friendly as possible for the tester.

5.2 Design of FireEye GUI

Testing being one of the most time consuming phases of the software development life cycle, it is important for a testing tool to be easy to use for the tester. Therefore, it is necessary to have an, interactive and user friendly GUI for any testing tool. GUI design is an important adjunct to any application. Its goal is to enhance the usability of the underlying logical design of a stored program.

The GUI paradigm (model of interaction) for the FireEye tool has been designed with the following design goals:

- Easy to learn and use

- Decreases the time for a tester to analyze any test case configuration
- Laymen operationsMinimal skills required to use it
- Interactive
- Readable Output
- Easily configurable
- Robust
- Platform Independent
- Minimal software requirements
- Easy installation
- No special hardware requirements

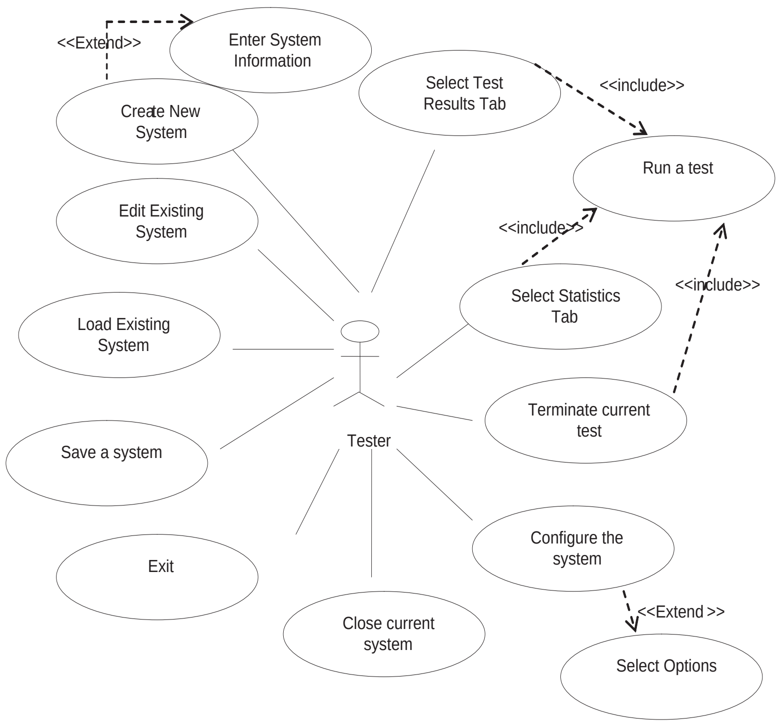


Figure 5.1. Use Case Diagram for FireEye GUI.

5.2.1 Use Case Diagrams

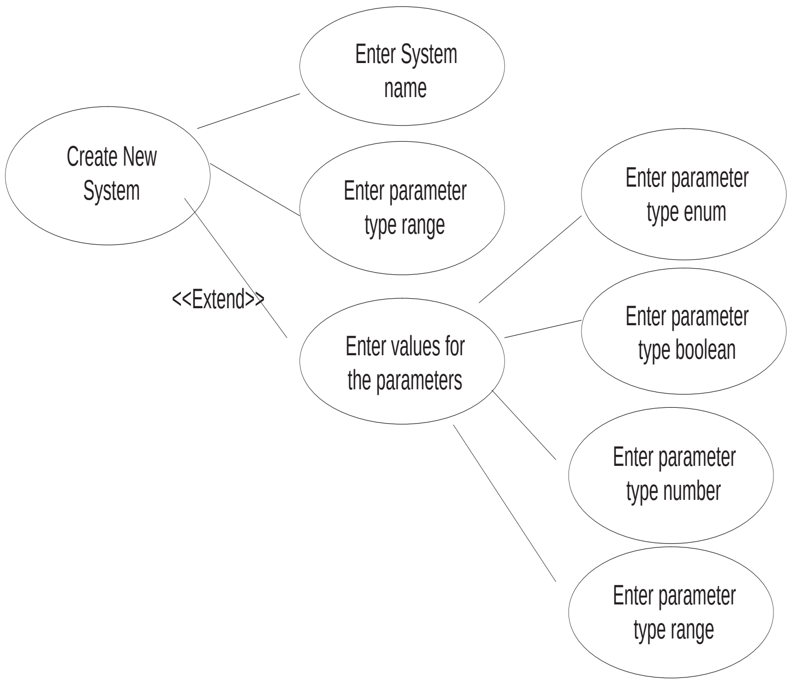

Enter System name
Create New System
Enter parameter type range
Enter parameter type enum
<<Extend>>
Enter values for the parameters
Enter parameter type boolean
Enter parameter type number
Enter parameter type range

Figure 5.2. Extended Use Case Diagram for Creating New System.

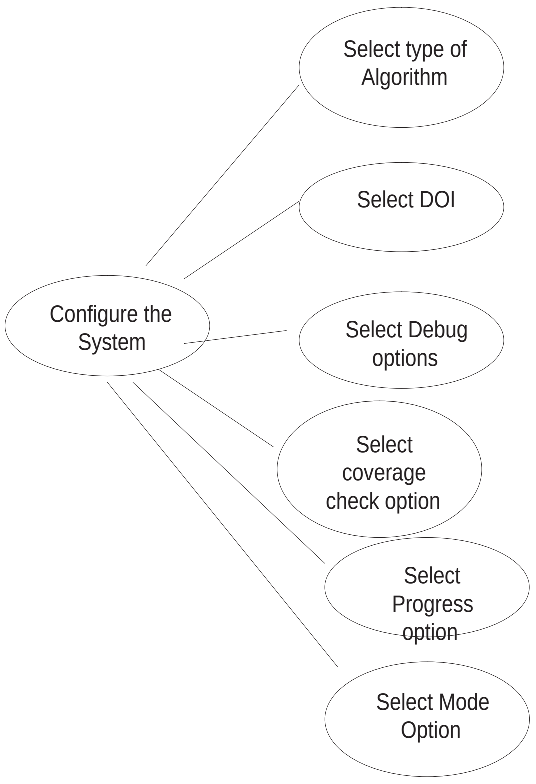


Figure 5.3. Extended Use Case Diagram for Configuring the System.

5.2.2 Class Diagram

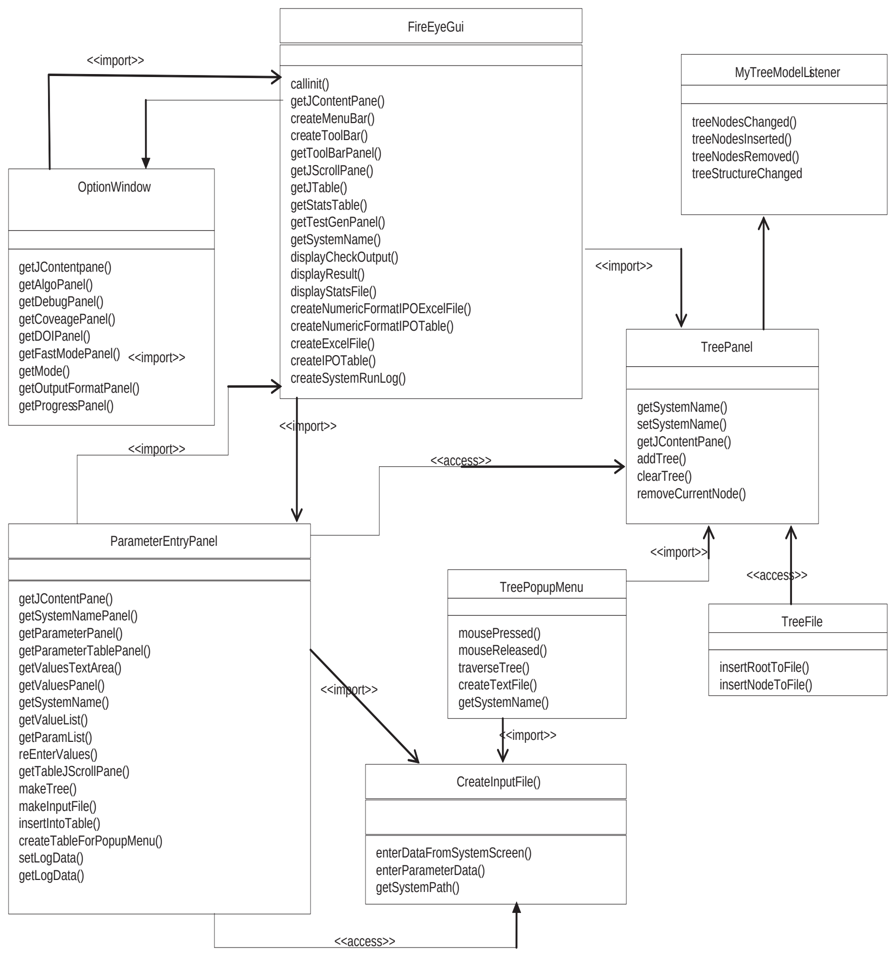


Figure 5.4. Class Diagram.

5.2.3 Technology Used

The GUI for FireEye is developed in Java Swing. Swing is a GUI toolkit for Java, a part of the Java Foundation Class (JFC). Swing uses the GUI widgets such as buttons, radio buttons, panels, text boxes, text fields, etc. The Swing widgets are more sophisticated GUI components compared to the earlier Abstract Windowing Toolkit (AWT). Because Swing is written in pure Java, it runs the same on all platforms unlike AWT, which is tied to the underlying platform's windowing system. Moreover, Swing supports a pluggable look and feel which means that we can get any supported look and feel on several platforms. Despite these strengths however, Swing suffers from one disadvantage of slowness in execution due to the lightweight components.

In short Swing can be summed up as a model that is a lightweight UI, loosely coupled, platform independent, component oriented, customizable, MVC GUI framework for the Java System.

A few important APIs used in this project are described below:

- javax.swing.table:

  Used to create the JTable for displaying the test configuration (output).

- javax.swing.tree:

  This API is used to create the hierarchical tree structure to display the system name, parameters and its corresponding values.

- jxl.write.Label, jxl.write.WritableSheet, jxl.write.WritableWorkbook;

  This jxl API is used for creating the Excel files for the output generated. It creates a new Excel file, writes data into each specified cell and saves it at the given location. The Excel file can be read similarly.

- java.io.BufferedReader, java.io.BufferedWriter, java.io.FileReader, java.io.FileWriter, java.io.File:

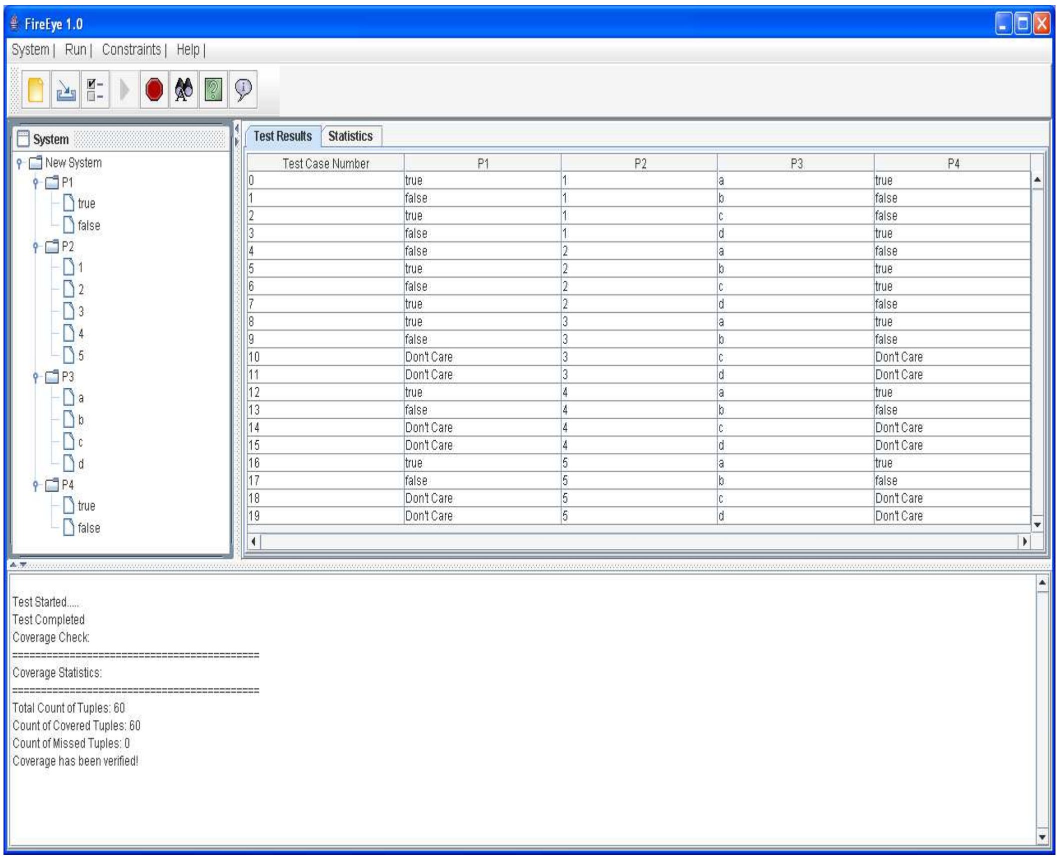
  This IO API is used to read and write into a file structure after creating a file handle from the java.io.File API. The files in this project are all in .txt format.

- Apart from the above given APIs there were many other APIs like java.io.*, javax.swing.event.* java.util.*, java.awt.*, java.awt.event.*, etc. which are used in this project.

5.2.4 Features of the GUI

5.2.4.1 Main Screen

Figure 5.5 shows the main screen after the launch of the program. The various components are numbered and are explained according to those numbers.



Test Case Number	P1	P2	P3	P4
0	true	1	a	true
1	false	1	b	false
2	true	1	c	false
3	false	1	d	true
4	false	2	a	false
5	true	2	b	true
6	false	2	c	true
7	true	2	d	false
8	true	3	a	true
9	false	3	b	false
10	Don't Care	3	c	Don't Care
11	Don't Care	3	d	Don't Care
12	true	4	a	true
13	false	4	b	false
14	Don't Care	4	c	Don't Care
15	Don't Care	4	d	Don't Care
16	true	5	a	true
17	false	5	b	false
18	Don't Care	5	c	Don't Care
19	Don't Care	5	d	Don't Care

```
Test Started.....
Test Completed
Coverage Check:
==========================================
Coverage Statistics:
==========================================
Total Count of Tuples: 60
Count of Covered Tuples: 60
Count of Missed Tuples: 0
Coverage has been verified!
```

Figure 5.5. Main Screen.

1. Tree Panel

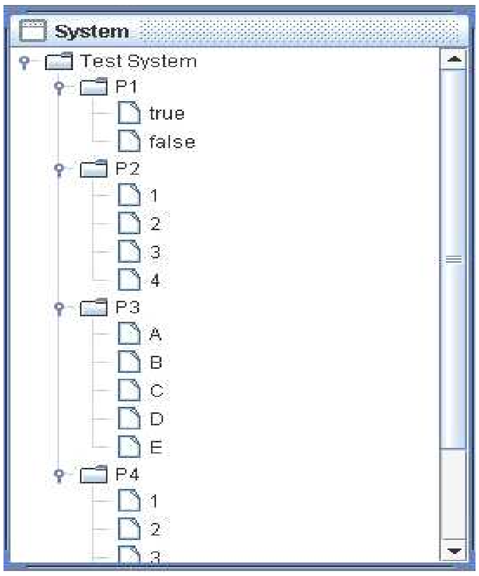


Figure 5.6. Tree Panel.

This panel, contained on the left hand side of the main frame, contains the hierarchical tree structure of the system. At the root is the system name followed by the parameter names whose child nodes are its corresponding values. The tree structure has a right clicking option that can add new values as well as parameters to the system. This tree structure displays the complete system after the new or existing system is loaded.

2. Menu Bar

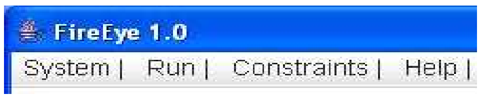


Figure 5.7. Menu Bar.

The menu Bar has the following options:

System

New: Opens a new window for creating new system

Load: Loads an existing system

Save: Saves the system

Create Fixed Domain System:

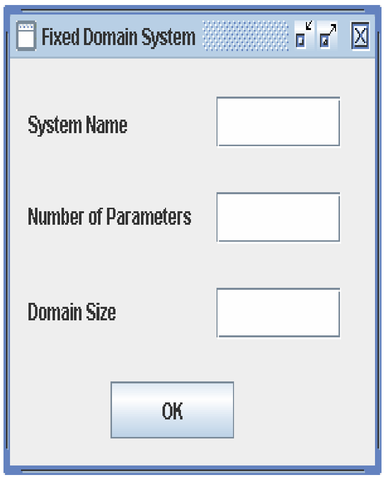


Figure 5.8. Fixed Domain System.

This option is used to create a system having the same number of values for all the parameters.

Close: Closes the current system

Exit: Exits from the program

Run

Options: Provides a panel to configure the system environment

Run Test: Runs the program for generating test cases

Terminate Test: Terminates the currently running test abruptly

Help

About: Displays the version of FireEye.

3. Tool Bar

Starting from the left, the icons are described as follows:

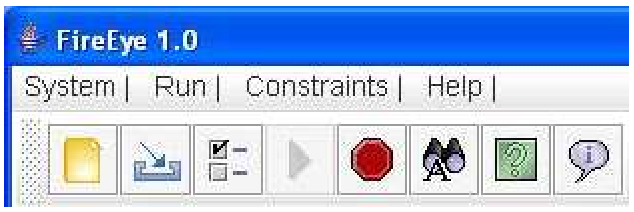


Figure 5.9. Tool Bar.

Figure 5.10. New.

Figure 5.11. Load.

Figure 5.12. Option.

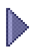


Figure 5.13. Run Test.

Tabbed Pane:

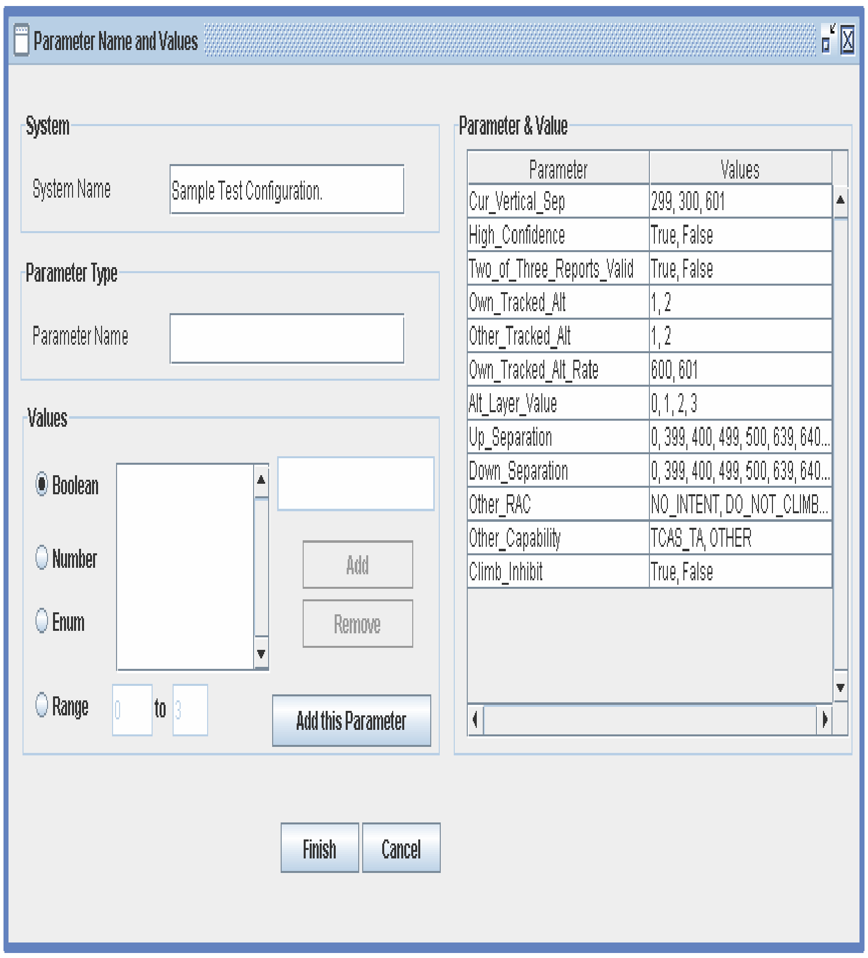
This window has a Test Results tabbed pane and Statistics tabbed pane. Both these panes extract values from the Excel file and print them in a tabular format.

4. Status Bar:

   This component displays the status of the operation that is being executed.

5.2.4.2 Parameter Entry Screen

This screen appears when the user clicks on the 'New' button or menu Item. This screen is used to enter the system information such as the System Name, Parameter Names and its Values. It has a table on the right side that contains the data that the user/tester can edit before actually creating a system. A system is created only after the tester clicks on the 'Finish' button.



System Name: Sample Test Configuration.

Parameter Name:

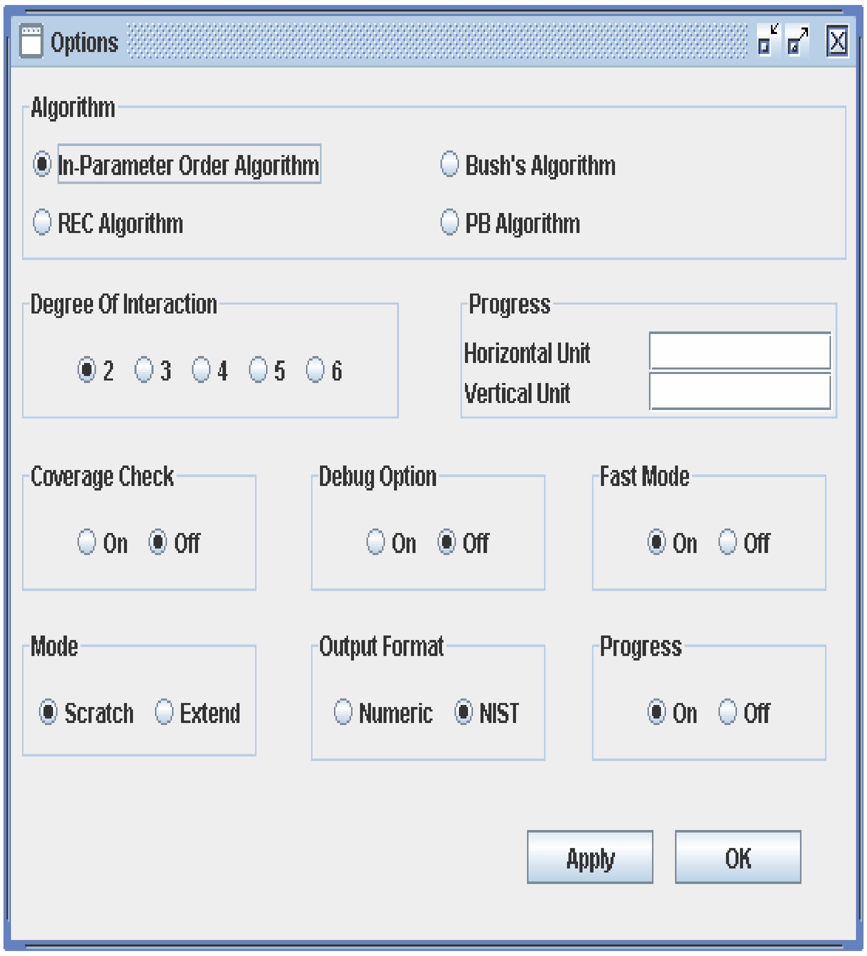
[✓ Boolean] Number Enum Range 0 to 3

Parameter	Values
Cur_Vertical_Sep	299, 300, 601
High_Confidence	True, False
Two_of_Three_Reports_Valid	True, False
Own_Tracked_Alt	1, 2
Other_Tracked_Alt	1, 2
Own_Tracked_Alt_Rate	600, 601
Alt_Layer_Value	0, 1, 2, 3
Up_Separation	0, 399, 400, 499, 500, 639, 640...
Down_Separation	0, 399, 400, 499, 500, 639, 640...
Other_RAC	NO_INTENT, DO_NOT_CLIMB...
Other_Capability	TCAS_TA, OTHER
Climb_Inhibit	True, False

Figure 5.14. Parameter Entry Screen.

5.2.4.3 Option Screen

This screen is used to modify the configuration of the test environment for the loaded system. The options include the following: changing the algorithm, the degree of interaction, Coverage check, Debug Option, Fast Mode, scratch mode or extended mode and output format.


Options

Algorithm
[✓ In-Parameter Order Algorithm] Bush's Algorithm
REC Algorithm PB Algorithm

Degree Of Interaction
[✓ 2] 3 4 5 6

Progress
Horizontal Unit
Vertical Unit

Coverage Check
On [✓ Off]

Debug Option
On [✓ Off]

Fast Mode
[✓ On] Off

Mode
[✓ Scratch] Extend

Output Format
Numeric [✓ NIST]

Progress
[✓ On] Off

Figure 5.15. Option Screen.

5.3 Experimental Results

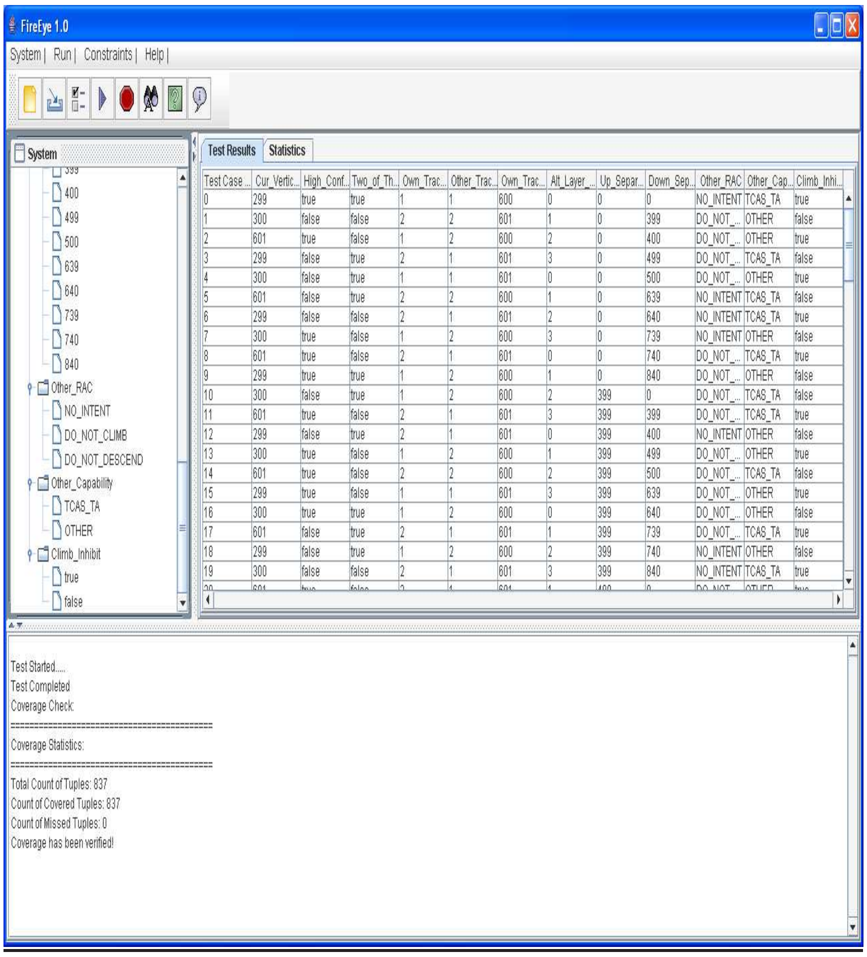
5.3.1 Experiment to find the test size and time for a given system on different executions

A sample input file as given below was considered for experimentation, and the size of the test configuration as well as the time the test took to complete was noted. The same system was tested for all the 2-6 ways repetitively.

```
[System]
Name: Sample Test Configuration.

[Parameter]
-- only compare with MINSEP and MAXALTDIFF
Cur_Vertical_Sep: 299, 300, 601

High_Confidence : TRUE, FALSE
Two_of_Three_Reports_Valid : TRUE, FALSE

-- Low and High, only compare with Other_Tracked_Alt
Own_Tracked_Alt: 1, 2
Other_Tracked_Alt : 1, 2

-- only compare with OLEV
Own_Tracked_Alt_Rate : 600, 601
Alt_Layer_Value : 0, 1, 2, 3

-- compare with each other (also see NOZCROSS) and with ALIM
Up_Separation : 0, 399, 400, 499, 500, 639, 640, 739, 740, 840
Down_Separation : 0, 399, 400, 499, 500, 639, 640, 739, 740, 840
Other_RAC : NO_INTENT, DO_NOT_CLIMB, DO_NOT_DESCEND
Other_Capability : TCAS_TA, OTHER
Climb_Inhibit : TRUE, FALSE

[Relation]

[Constraint]

[Misc]
```

Figure 5.16. Input File (.txt file).

Output:



Test Case ...	Cur_Vertic...	High_Conf...	Two_of_Th...	Own_Trac...	Other_Trac...	Own_Trac...	Alt_Layer_...	Up_Separ...	Down_Sep...	Other_RAC	Other_Cap...	Climb_Inhi...
0	299	true	true	1	1	600	0	0	0	NO_INTENT	TCAS_TA	true
1	300	false	false	2	2	601	1	0	399	DO_NOT_...	OTHER	false
2	601	true	false	1	2	600	2	0	400	DO_NOT_...	OTHER	true
3	299	false	true	2	1	601	3	0	499	DO_NOT_...	TCAS_TA	false
4	300	false	true	1	1	601	0	0	500	DO_NOT_...	OTHER	true
5	601	false	true	2	2	600	1	0	639	NO_INTENT	TCAS_TA	false
6	299	false	false	2	1	601	2	0	640	NO_INTENT	TCAS_TA	true
7	300	true	false	1	2	600	3	0	739	NO_INTENT	OTHER	false
8	601	true	false	2	1	601	0	0	740	DO_NOT_...	TCAS_TA	true
9	299	true	true	1	2	600	1	0	840	DO_NOT_...	OTHER	false
10	300	false	true	1	2	600	2	399	0	DO_NOT_...	TCAS_TA	false
11	601	true	false	2	1	601	3	399	399	DO_NOT_...	TCAS_TA	true
12	299	false	true	2	1	601	0	399	400	NO_INTENT	OTHER	false
13	300	true	false	1	2	600	1	399	499	DO_NOT_...	OTHER	true
14	601	true	false	2	2	600	2	399	500	DO_NOT_...	TCAS_TA	false
15	299	true	false	1	1	601	3	399	639	DO_NOT_...	OTHER	true
16	300	true	true	1	2	600	0	399	640	DO_NOT_...	OTHER	false
17	601	false	true	2	1	601	1	399	739	DO_NOT_...	TCAS_TA	true
18	299	false	true	1	2	600	2	399	740	NO_INTENT	OTHER	false
19	300	false	false	2	1	601	3	399	840	NO_INTENT	TCAS_TA	true

```
Test Started.....
Test Completed
Coverage Check:
================================================
Coverage Statistics:
================================================
Total Count of Tuples: 837
Count of Covered Tuples: 837
Count of Missed Tuples: 0
Coverage has been verified!
```

Figure 5.17. Output Table.

Statistics:

The statistics below display the size and time taken by the system for all the values for degree of interaction from 2 to 6 for three test runs.

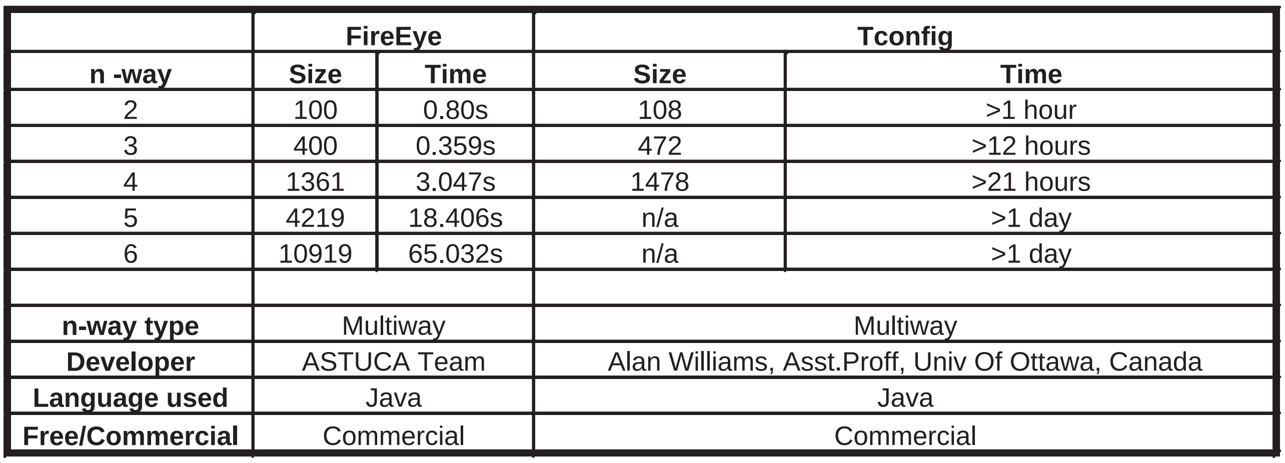
Here, all the results are collected using a laptop with 1.6GHz CPU and 1GB memory with a Windows operating system.

5.3.2 Comparison of FireEye with other publicly available tools

In this section, we compare the performance and efficiency of FireEye with other tools.

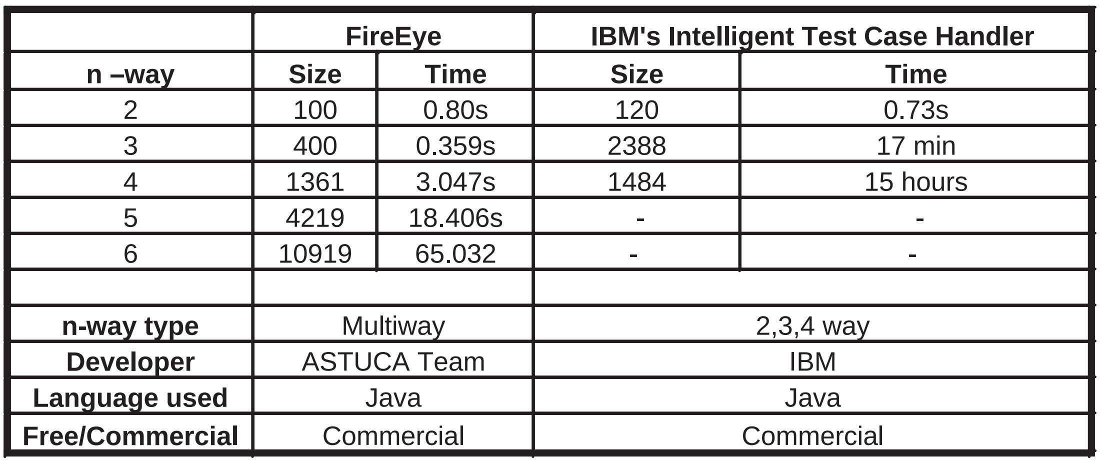
Table 5.1. Output after executing the sample input file

	FireEye	
n-way	Size	Time
2	100	0.79s
3	400	0.340s
4	1361	3.047s
5	4219	18.406s
6	10920	65.039s



	FireEye		Tconfig	
n -way	Size	Time	Size	Time
2	100	0.80s	108	>1 hour
3	400	0.359s	472	>12 hours
4	1361	3.047s	1478	>21 hours
5	4219	18.406s	n/a	>1 day
6	10919	65.032s	n/a	>1 day
n-way type	Multiway		Multiway	
Developer	ASTUCA Team		Alan Williams, Asst.Proff, Univ Of Ottawa, Canada	
Language used	Java		Java	
Free/Commercial	Commercial		Commercial	

Figure 5.18. Statistics for FireEye and TConfig.



	FireEye		IBM's Intelligent Test Case Handler	
n –way	Size	Time	Size	Time
2	100	0.80s	120	0.73s
3	400	0.359s	2388	17 min
4	1361	3.047s	1484	15 hours
5	4219	18.406s	-	-
6	10919	65.032	-	-
n-way type	Multiway		2,3,4 way	
Developer	ASTUCA Team		IBM	
Language used	Java		Java	
Free/Commercial	Commercial		Commercial	

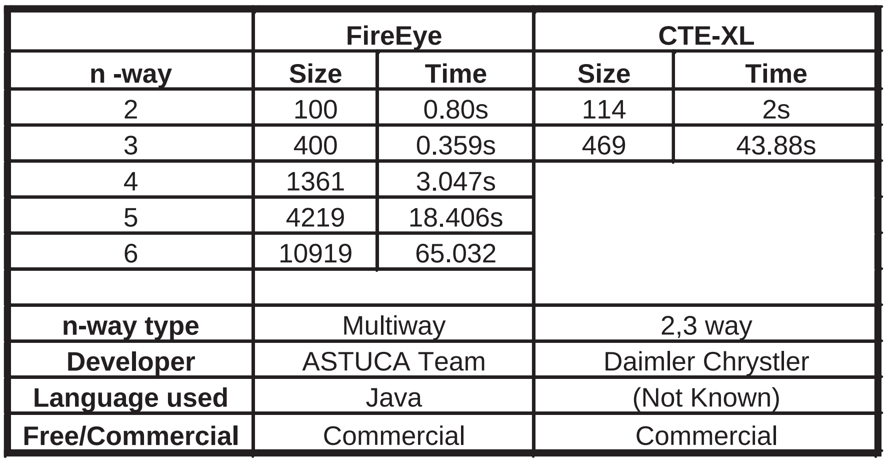
Figure 5.19. Statistics for FireEye and IBM's ITCH Tool.

	FireEye		Jenny	
N -way	**Size**	**Time**	**Size**	**Time**
2	100	0.80s	108	0.001s
3	400	0.359s	413	0.71s
4	1361	3.047s	1536	3.54s
5	4219	18.406s	4580	43.54s
6	10919	65.032	11625	7 min 50 s
n-way type	Multiway		Multiway	
Developer	ASTUCA Team		Bob Jenkins	
Language used	Java		C	
Free/Commercial	Commercial		Free	

Figure 5.20. Statistics for FireEye and Jenny.

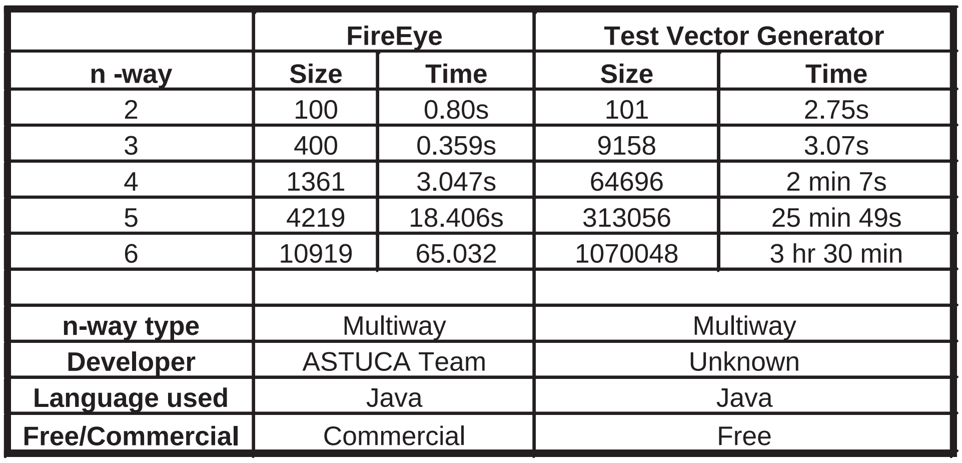
	FireEye		AllPairs	
n -way	**Size**	**Time**	**Size**	**Time**
2	100	0.80s	103	3.82s
3	400	0.359s		
4	1361	3.047s		
5	4219	18.406s		
6	10919	65.032		
n-way type	Multiway		2 - way	
Developer	ASTUCA Team		Satisfice	
Language used	Java		Perl	
Free/Commercial	Commercial		Commercial	

Figure 5.21. Statistics for FireEye and AllPairs.



	FireEye		CTE-XL	
n -way	Size	Time	Size	Time
2	100	0.80s	114	2s
3	400	0.359s	469	43.88s
4	1361	3.047s		
5	4219	18.406s		
6	10919	65.032		
n-way type	Multiway		2,3 way	
Developer	ASTUCA Team		Daimler Chrystler	
Language used	Java		(Not Known)	
Free/Commercial	Commercial		Commercial	

Figure 5.22. Statistics for FireEye and CTE-XL.



	FireEye		Test Vector Generator	
n -way	Size	Time	Size	Time
2	100	0.80s	101	2.75s
3	400	0.359s	9158	3.07s
4	1361	3.047s	64696	2 min 7s
5	4219	18.406s	313056	25 min 49s
6	10919	65.032	1070048	3 hr 30 min
n-way type	Multiway		Multiway	
Developer	ASTUCA Team		Unknown	
Language used	Java		Java	
Free/Commercial	Commercial		Free	

Figure 5.23. Statistics for FireEye and Test Vector Generator.

	FireEye		Smart Test R1.3	
N -way	**Size**	**Time**	**Size**	**Time**
2	100	0.80s	121	0.77s
3	400	0.359s	-	-
4	1361	3.047s	-	-
5	4219	18.406s	-	-
6	10919	65.032	-	-
n-way type	Multiway		2 way	
Developer	ASTUCA Team		SmartWare Technologies	
Language used	Java			
Free/Commercial	Commercial		Commercial	

Figure 5.24. Statistics for FireEye and Smart Test R1.3.

CHAPTER 6

CONCLUSION

We conclude that interaction testing is inevitable for any system that is made up of various components. Although pairwise testing can be considered useful, multi-way testing is necessary for a considerably large system. We came up with FireEye as the tool for t-way testing.

FireEye is envisioned as a complete multi-way testing tool implementing the IPO algorithm, which generates a minimized number of test sets quickly.

While the design and GUI implementation of FireEye draws on the work of its predecessors, its efficiency in combining test efforts with quality assurance surpasses past programs. The goal was to design a GUI that would be easy to use, interactive and portable while at the same time decreasing the time necessary for the tester to analyze the test configuration. FireEye's GUI also has minimal hardware requirements, no high-end software requirements and it is easy to install. We discussed the various features and uses of the GUI. The features such as the Tree Panel and the Parameter Entry Panel give a better view of the system to the tester. It helps him create a system that is free of errors and consequently, he can get a test set which is error free as well. The status bar keeps the tester informed regarding the status of the operation being executed. The output is shown in an organized tabular format, with the total number of configurations generated as well as with the statistics of the test run. The overall design of the GUI is such that it requires very few skills for its operation.

We discussed several methods to improve the performance of FireEye in terms of time and space required for test generation. The excellent performance and efficiency of FireEye is quite apparent from its comparison with other tools. FireEye generates a test set of optimum size in comparatively less time.

REFERENCES

[1] David M. Cohen, Siddhartha R. Dalal, Jesse Parelius, Gardner C. Patton, *The Combinatorial Design Approach to Automatic Test Generation.* IEEE Software: 1966.

[2] Y. Lei and K. C. , *In-parameter-order: a test generation strategy for pairwise testing*, Proceedings of the Intl. Conf. on Software Engineering ,(ICSE 99),1999, pp. 285-94, New York.

[3] R.McDaniel, J.D.McGregor *Testing the Polymorphic Interactions between Classes* Clemson University Dept. of Computer Science, Technical Report TR 94-103, Clemson SC USA, 1994.

[4] R.Brownlie, J.Prowse, and M.S.Phadke, *Robust Testing of AT & T PMX/StarMail using OATS* AT & T Technical Journal, Vol.71 No.3, May/June 1992.

[5] Ted Stanion, Debashis Bhattacharya," *TSUNAMI: A Path Oriented Scheme for Algebraic Test Generation*, Dept. of Electrical Engineering, Yale University, 15 Prospect St., New Haven, CT 06520

[6] L.J. White," *Regression Testing of GUI Event Interactions*, Proceedings of the International Conference on Software Maintenance, Monterey CA, Nov. 1996.

BIOGRAPHICAL STATEMENT

Chinmay P. Jayaswal was born on December 24, 1982, India. He received his Bachelor of Engineering in Information Technology in 2004 from U.V.Patel College of Engineering, India. In the Fall of 2004 he started his graduate studies in Computer Science and Engineering at the University of Texas at Arlington. He completed Masters in Computer Science and Engineering in December 2006 with Software Engineering as major.