

**Validation of Ontology Based Test Case Generation
for Graphical User Interface**

Thesis Report



A THESIS PRESENTED
TO

**FACULTY OF BASIC & APPLIED
SCIENCES**

DEPARTEMENT OF COMPUTER SCIENCE & SOFTWARE ENGINEERING

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE
DEGREE OF
MS IN SOFTWARE ENGINEERING

BY
HAJRANASEER
307-FBAS/MISE/F09

Department of Computer Science & Software Engineering
Faculty of Basic and Applied Sciences
International Islamic University, H-10,
Islamabad

(May 2012)



Accession No TH-9617

MA / MSC
005-72
HIV

- 1 - Validation data
2. Data Entry

DATA ENTERED

Amg 31/05/13

FINAL APPROVAL

Subject: External Approval Of The Research Thesis “Validation Of Ontology Based Test Case Generation For Graphical User Interface”

It is certified that we have read the thesis submitted by **Miss Hajra Naseer**; registration number **307-FBAS/MSSE/F09**. It is our assessment that this thesis is of sufficient standard for warranting its acceptance by International Islamic University, Islamabad for the degree of **MS in Software Engineering**.

Examination Committee

External Examiner

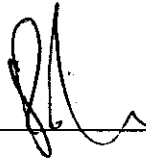
Dr Muhammad Ramzan

Assistant Professor

Department of Software Engineering,

Foundation University Institute of
Engineering & Management Sciences,

Rawalpindi



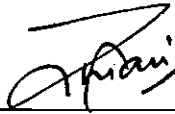
Internal Examiner

Dr. Zunera Jalil

Assistant Professor

Department of CS & SE,

International Islamic University Islamabad



Supervisor

Dr Abdul Rauf

Assistant Professor

Department of CS & SE, IIUI

International Islamic University Islamabad



Co-Supervisor

Miss Salma Imtiaz

Assistant Professor

Department of CS & SE, IIUI

International Islamic University Islamabad



DEDICATION

This thesis is dedicated to

MY PARENTS

*I am most indebted to my parents,
whose affection has been the source of encouragement,
and
whose prayers have always been key to my success.*

Hajra Naseer

Hajra Naseer
07-FBAS/MSSE/F09

A dissertation submitted to
Department of Computer Science & Software Engineering,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad
as a partial fulfillment of the requirement
for awarding the degree of
MS in Software Engineering (MSSE)

DECLARATION

I hereby declare that this Thesis "Validation of Ontology Based Test Case Generation for Graphical User Interface" neither as a whole nor as a part has been copied out from any source. It is further declared that I have done this research with the accompanied report entirely on the basis of my personal efforts, under the proficient guidance of my supervisor **Dr Abdul Rauf** and co-supervisor **Miss Salma Imtiaz**.

If any part of the system is proved to be copied out from any source or found to be reproduction of any project from any of the training institute or educational institutions, I shall stand by the consequences.

No portion of the research work presented in this thesis report has been submitted in support of any other degree or qualification of this or any other university or institute of learning.

Hajra Naseer

Hajra Naseer
307-FBAS/MSSE/F09

ACKNOWLEDGEMENT

Many thanks to Almighty Allah; The Merciful, The Beneficent and The source of all Knowledge; and His Holy Prophet Muhammad (S.A.W) whose blessings have enabled me to perceive and pursuit higher ideas of life, Who Have given me the courage, insight and knowledge to complete this thesis.

I am also grateful to my parents for their continued moral support during my research work. I would here like to acknowledge my supervisor Dr. Abdul Rauf and my co-supervisor Miss Salma Imtiaz as well whose precious guidance made me able to complete my research and who helped a lot during the documentation of this research work.

I would like to acknowledge International Islamic University Islamabad and its teachers as well for giving us an insight to look into things in a professional way. Last, but by not the least, I would like to acknowledge my friends for their moral support.

For errors and inadequacies in this research work, I accept the responsibility.

Hajra Naseer

Hajra Naseer
307-FBAS/MSSE/F09

THESIS IN BRIEF

- THESIS TITLE:** Validation of Ontology Based Test Case Generation for Graphical User Interface
- OBJECTIVE:** To generate test cases for graphical user interface application based on Ontology meanwhile validating “GUI Testing Framework”
- UNDERTAKEN BY:** **Hajra Naseer**
307-FBAS/MSSE/F09

Student of MS in Software Engineering

Department of Computer Science & Software Engineering,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.
- SUPERVISED BY:** **Dr Abdul Rauf**
Assistant Professor
Department of Computer Science & Software Engineering,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.
- CO-SUPERVISOR:** **Miss Salma Imtiaz**
Assistant Professor
Department of Computer Science & Software Engineering,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.
- START DATE:** April 22nd, 2011.
- COMPLETION DATE:** December 28th, 2011.

ABSTRACT

Software Testing is an important activity during the Software Development Life Cycle. Much of an organization's time and money is spent on it. Test case generation (TCG) is an important activity during this process. The techniques which are being used for TCG in Command Line Interface can't be used in GUI.

TCG for GUI is a least focused area. GUI testing is knowledge intensive in nature. It requires complete information about the components involved, interaction among them, their sequences, the context in which they were used etc.

One of the means to formally represent knowledge is through ontology. It represents domain information as a set of concepts and relationship among them. Through this domain knowledge is separated from application knowledge and makes it easy to communicate it among people and systems, makes its reuse possible. Ontology based GUI testing is a new branch of testing. It has been introduced in last few years and is in its initial stages. The work which has been proposed has neither been validated before. Ontology has the potential to be used for test case generation. An experiment has been conducted to validate a framework.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION	1
1.1. INTRODUCTION	2
1.2. MOTIVATION AND CHALLENGES	2
1.3. BACKGROUND	3
1.4. PROBLEM DOMAIN	4
1.5. PROPOSED SOLUTION	4
1.6. THESIS OUTLINE	7
CHAPTER 2: LITERATURE SURVEY	8
2.1 INTRODUCTION.....	9
2.2 SOFTWARE TESTING	9
2.2.1 WHAT IS TEST CASE GENERATION.....	10
2.2.2 TEST CASE GENERATION TECHNIQUES	10
2.2.3 LIMITATIONS	12
2.3 ONTOLOGY	13
2.3.1 RELATED WORK.....	13
2.3.2 BENEFITS OF ONTOLOGY	- 16-
2.4 ONTOLOGY BASED TEST CASE GENERATION	- 16-
2.5 SUMMARY	- 17-
CHAPTER 3: METHODOLOGY	- 18-
3.1 INTRODUCTION.....	- 19-
3.2 EXPERIMENTAL DESIGN	- 19-
3.3 PROBLEM DEFINITION.....	- 22-
3.4 RESEARCH QUESTION(S).....	- 22-
3.5 PROPOSED SOLUTION	- 24-
CHAPTER 4: RESULTS	- 35-
4.1 INTRODUCTION.....	- 36-
4.2 COVERAGE	- 36-
4.2. EFFICIENCY	- 36-
4.3. COMPARISON BETWEEN COVERAGE & TEST EFFICIENCY	- 37-
4.4. DISCUSSION	- 37-
CHAPTER 5: CONCLUSION.....	- 44-
5.1 CONCLUSION.....	- 44-
5.2 FUTURE WORK	- 44-
ABBREVIATIONS	- 45-
REFERENCES	- 46-

LIST OF FIGURES

FIGURE 1: GUI TESTING FRAMEWORK	5
FIGURE 2: PDCA CYCLE	9
FIGURE 3: THE DEFECT TESTING PROCESS	10
FIGURE 4: AN ONTOLOGY EXPRESSIVENESS SPECTRUM	14
FIGURE 5: EXPERIMENT SETUP	- 26 -
FIGURE 6: LEVEL 1 GRAPH REPRESENTATION OF NOTEPAD APPLICATION	- 26 -
FIGURE 7: LEVEL 2 GRAPH REPRESENTATION OF NOTEPAD APPLICATION	- 26 -
FIGURE 8: FILE MENU EXPANDED	- 27 -
FIGURE 9: EDIT MENU EXPANDED	- 27 -
FIGURE 10: CLASS HIERARCHY OF NOTEPAD APPLICATION	- 28 -
FIGURE 11: PROPERTIES	- 29 -
FIGURE 12: RULES ADDED	- 30 -
FIGURE 13: OVERVIEW OF EVENTS	- 30 -
FIGURE 14: EVENTS THAT FALL UNDER LEVEL ONE OF APPLICATION	- 31 -
FIGURE 15: EVENTS THAT FALL UNDER LEVEL TWO OF APPLICATION	- 31 -
FIGURE 16: EVENTS THAT FALL UNDER LEVEL THREE OF APPLICATION	- 31 -
FIGURE 17: EFG ELABORATED	- 32 -
FIGURE 18: FILE MENU OPTIONS	- 32 -
FIGURE 19: EDIT MENU OPTIONS	- 33 -
FIGURE 20: SHORTCUT OPTIONS	- 33 -
FIGURE 21: VALUE PARTITION	- 34 -
FIGURE 22: ARC TYPES	- 34 -
FIGURE 23: RESULTS WITH 50 TEST CASES	- 39 -
FIGURE 24: RESULTS WITH 75 TEST CASES	- 39 -
FIGURE 25: RESULTS WITH 100 TEST CASES	- 40 -
FIGURE 26: RESULTS WITH 130 TEST CASES	- 40 -
FIGURE 27: COMPARISON OF COVERAGE	- 41 -
FIGURE 28: COMPARISON OF COVERAGE ACCORDING TO LEVELS	- 41 -
FIGURE 29: COMPARISON OF TEST EFFICIENCY	- 42 -
FIGURE 30: COMPARISON OF TEST EFFICIENCY ACCORDING TO LEVELS	- 42 -
FIGURE 31: COMPARISON OF COVERAGE & EFFICIENCY	- 43 -

LIST OF TABLES

TABLE 1: EXPERIMENTAL DESIGN	- 21 -
TABLE 2: COMPARISON OF COVERAGE ACCORDING TO LEVELS	- 36 -
TABLE 3: COMPARISON OF TEST EFFICIENCY ACCORDING TO LEVELS	- 37 -
TABLE 4: COMPARISON OF COVERAGE VS TEST EFFICIENCY	- 37 -

CHAPTER 1: INTRODUCTION

1.1. INTRODUCTION

Software Testing is an important activity during Software Development Life Cycle *SDLC*. Much of an organization's time and money is spent on it. To manage the cost of manual testing process and increase its reliability automatic testing was a way out. Test case generation (TCG) is an important activity during this automatic process.

The manual process of testing has problems associated with it. Resource consumption is more, it is hard to memorize which states or functions of the system have been tested and which are not, reliability of manually tested system is lesser etc [2]. So there was a need to automate this process.

The origin of Ontology is from Metaphysics; the branch of Philosophy; as a study of existence. It is currently being used in many disciplines like Artificial Intelligence *AI*, Software Engineering *SE*, biomedical informatics, system engineering, library sciences and many more. In knowledge representation community agreed upon and highly cited definition of ontology is proposed by Gruber in 1993 as "Ontology is formal, explicit specification of a shared conceptualization. '*Conceptualization*' refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. '*Explicit*' means that the type of concepts used, and the constraints on their use are explicitly defined. '*Formal*' refers to the fact that the ontology should be machine readable. '*Shared*' reflects that ontology should capture consensual knowledge accepted by the communities". [7].

1.2. MOTIVATION AND CHALLENGES

In automatic testing, Test Case Generation *TCG* is an activity which is of great importance. Automatically generated test cases are time consuming to generate, maintain & evaluate [2, 14]. The techniques which are being used for TCG in Command Line Interface *CLI* can't be used in Graphical User Interface *GUI*. GUI applications are different in nature as they are event driven. There are not only the components of an application to be tested but also its graphics, and the user interacts with those through events. Ultimately testing such applications is knowledge intensive and context

dependent as well. The main steps in GUI testing are: to reverse-engineer an Event Flow Graph *EFG* from a GUI, generate and execute test cases, and then apply the oracle [18].

Several challenges are pertinent testing a graphical interface.

1. The number of possible states in which a system can be is enormous, so the number of interactions among the states and their sequences may be large as well as different. So it is difficult to validate a state when test cases are executed at each step [3, 14, 16].
2. When a change occurs in the sequence of states then the regression testing becomes a problem because the two version of same software are not in accordance with each other [14, 16].
3. Coverage of test cases in GUI is also a problem as in CLI it is based on the amount and type of code which has been tested. In GUI we can't determine the coverage of test cases only by the part of code being tested, but also the number of possible states in which that code has been tested [3, 14, 16].
4. Number of events and interaction among them is very huge [3, 5, 14, 15].
5. "An incorrect GUI state can lead to an unexpected screen, making further execution of the test case useless since events in the test case may not match the corresponding GUI components on the screen" [14, 16]. Due to these challenges test case generation for GUI testing is least focused area [3].

1.3. BACKGROUND

GUI testing is *knowledge intensive* in nature. It includes components and operations which are of different kinds and complicated as well, there is a complex relationships among these components and a lot of abundant testers' experience is required [10]. While testing a GUI application one has to keep in mind underline *business logic* of the application, the *nature of the components* involved, their *properties* which can be observed like color, title etc. for determining whether a test case was pass or fail. All these activities require knowledge to be managed thus ontology best fits here [19].

GUI testing is *context dependent* and requires knowledge of the application under test. One has to remember the states which have already been tested as well their sequence. Much of the underlying functionality in a GUI application is un-documented; there are *several paths* through which a user can achieve a task [19]. It is of great importance in GUI to maintain context of events; information of events, their interactions, their sequences etc. Sequence of events is also important in TCG which is context dependent. To maintain this context we need knowledge. Ontology thus makes it possible to manage this information.

1.4. PROBLEM DOMAIN

Among one of the reasons why I have focused on ontology to generate test cases for a GUI is that through this *knowledge related to an application domain can be separated from the business logic*. It also provides standard specification of concepts in a specific domain. Thus test cases which are generic in nature can be generated for a domain irrespective of the application in which they are used ultimately making *reuse of knowledge* possible.

It is but natural that applications evolve with the passage of time; new components are added or the existing ones are modified. Test cases break if there is even a slightest change in the GUI thus *regression testing* becomes a problem. Test cases generated for an application adds in our knowledge base i.e. ontology. For newer versions; test are generated reusing the existing ontology. The problem of re-testing an application is thus solved. Knowledge created in this whole process can be managed and reused in future.

1.5. PROPOSED SOLUTION

Even though ontology has been an active area of research since the last decade, there has not been any ontology reported in the literature which deals with the software engineering field. The authors of [20] have developed ontology for general process of web based application testing. Ontology oriented GUI testing is a new area of research in software testing [10].

There is initial level work with regard to generating test cases for GUI using ontology. A framework titled as “GUI Testing Framework” has been proposed by the authors of [5] but they have not validated it. Figure 1 shows this framework. The authors state that in the process of defect identification of GUI’s; due to their event driven nature longer test sequence is better than shorter sequences. The approach is based on EFG and incorporates ontology for this purpose. Ontology can be used to represent concepts and relations and as the number of concepts keep on increasing with the passage of time, so the proposed ontology for this work must also be evolving ultimately.

To the best of our knowledge the authors Han Li, Feng Chen et al have first introduced in their paper [10] the concept of GUI testing based on ontology. They mentioned the elements required for developing GUI ontology and also gave their definition. These elements are concepts, instances, relations and property. In GUI testing there is a great role of tester experience as in what sequence they interact with GUI components. So they have contributed in maintaining this through developing ontology.

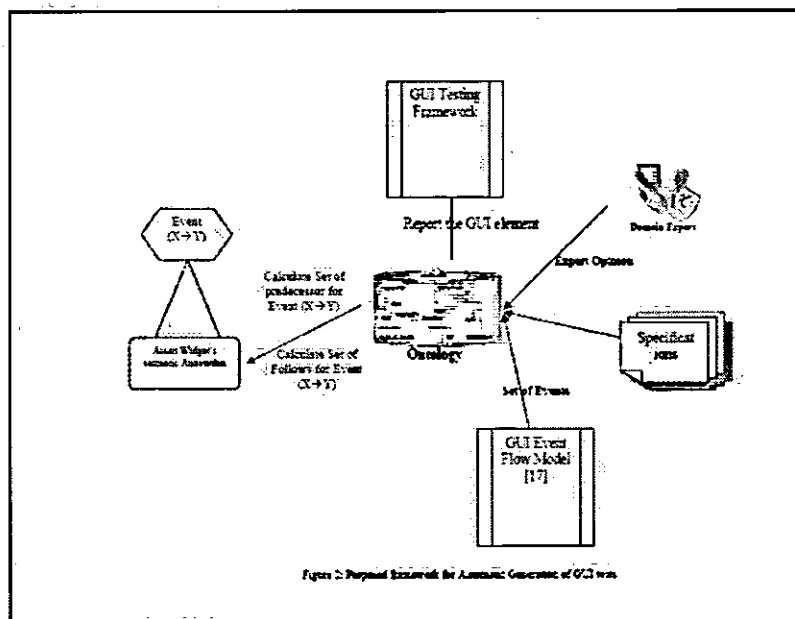


Figure 1: GUI Testing Framework

I will be using ontology to generate test cases for a GUI application. For this purpose we will use EFG.

EFG [17] is a model developed for GUI of a particular application. It represents all possible sequences of events which a user can execute while interacting with a graphical application. It represents dynamic behavior of a GUI. Nodes correspond to events and edges correspond to event sequences or a relationship of flow of events. They are cyclic in nature such that an event may be executed multiple times in a single session with an application.

EIG is based on the events and interaction among them [15]. Reverse engineering techniques are used to obtain EIG but some limitations are associated with it. It does not contain any state-based relationships. Secondly its coverage requires a large number of test cases. EFG or EIG are used for test coverage and not for test case generation. They can be used for TCG but this has not been done.

The approach I have proposed for test case generation for GUI is different from the current prevailing techniques. Currently test cases generated for GUIs are based on modeling techniques, planning techniques [14, 16] adapted from AI, user interactions with the application is saved in the form of widgets and later on they might be compared for testing. There is no such work in which test cases for GUI are generated based on ontology.

An experiment has been conducted to validate the proposed framework. At first an application will be developed. Initially test cases will be generated based on the application and later on ontology will be developed (tool based) depending on these generated test cases. For testing the application multiple times or in regression testing then test cases should be generated automatically based on ontology. To measure the coverage of the test cases it will be done on the event count. Validation of the generated test cases will be the last step carried out.

Version 2 of Web Ontology Language (OWL) OWL2 will be used for the ontology development. It supports classes, instances, properties and data values which are stored in semantic web document. An open source software Protégé™ version 4.1 will be used to construct domain models and knowledge-based applications with ontologies. It is an open source software and provides two ways for modeling ontologies i.e. (1) Frame-based modeling. (2) Modeling ontology using OWL.

1.6. THESIS OUTLINE

In Chapter 2 deals with the literature survey conducted for the thesis. Chapter 3 focuses on the research methodology. I have given a detail description of the procedure and tool used for ontology development as well as test case generation. Chapter 4 deals with the results. I have mentioned the graphical representation of our results as well. At last we concluded our work and gave an insight to the future work in chapter 5.

CHAPTER 2: LITERATURE SURVEY

2.1 INTRODUCTION

This chapter outlines and describes the process of software testing, what is test case generation. It also presents an overview of various software test case generation techniques. It also describes what ontology is and what benefits it offers?

2.2 SOFTWARE TESTING

Software testing is an activity in which the developed software application/system is executed with the intention to find defects in it. Wikipedia defines this activity as “Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).” [22]

Four phases are included in software testing process. These are **Plan, Do, Check and Act**; which is known as **PDCA Cycle** [24]. Testing process includes the following steps: *test plan* (strategy, plan, testbed), *test development* (procedures, test scenarios, test cases, test datasets, test scripts), *test execution* (execution and reporting of error(s) found), *test reporting* (metrics, effort), *test result analysis* (defect reporting), *defect retesting* (known as resolution testing), *regression testing* and *test closure* (lessons learned, results, logs, documents) [22].

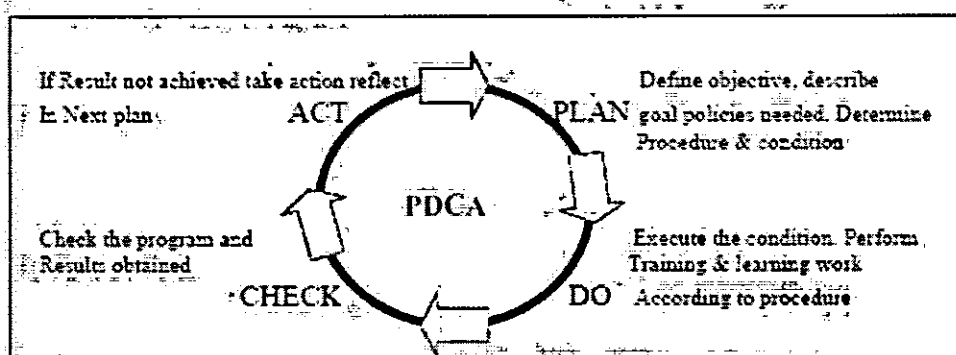


Figure 2: PDCA Cycle

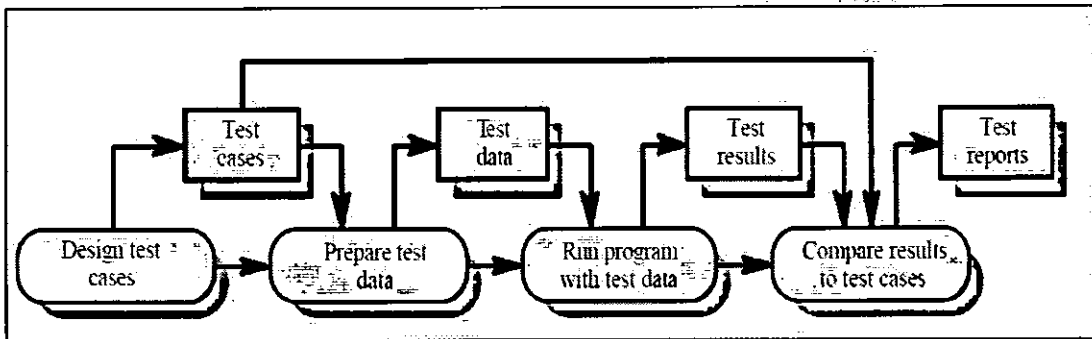


FIGURE 3: The Defect Testing Process

The above figure shows defect testing process [23].

2.2.1 What Is Test Case Generation

Test cases can be defined as “A test case is a description of a test, independent of the way a given system is designed” [2]. We can generate test cases directly from the system requirements or from use cases. The benefit of using system specifications and design for this process is that test cases can be created early in SDLC and are available for use even before the actual system is developed [2].

2.2.2 Test Case Generation Techniques

Techniques which are being used for Test Case Generation in Graphical User Interface are as follows:

Capture/Replay technique/tool [3]. In the first mode they capture the mouse coordinates of user actions in the form of test case. In the second mode these recorded test cases are replayed automatically. There is a problem associated with it that the test cases break when there is a slight change in the GUI.

Planning technique [14, 16] adapted from AI. It is based on states mainly initial and final; called as “GUI tasks”; for the generation of test cases. The motivation of this approach lies in the fact that the users of an application are more concerned with the goal they want to achieve by interacting with the software. It is easier to design test cases keeping in mind these goals rather to specify sequences of GUI actions [16]. List of operators represent events. The sequences of events; also called as plans; become test

cases for the GUI. The authors proposed a tool for test case generation called as PATHS (Planning Assisted Tester for graphical user interface Systems).

Genetic Algorithms Test cases for GUI can also be created by mimicking usage of the application by a novice user [4, 16]. It assumes that the novice user takes on longer path when interacting with a GUI application as compared to an expert user. This approach is dependent on expert's input in a way that s/he has to generate the initial sequence of GUI events manually. Genetic techniques are later on used for modifying and extending the sequence and generating longer sequences [16].

Directed Graph Models In order to reduce manual work, several new systematic techniques based on graph models of the GUI have recently been developed. They are based on EFG and Event Interaction Graph *EIG* [4].

Network Centrality Measures is adapted from network analysis-based approach for GUI testing [17]. This is a new area of research in GUI testing. The authors state that their approach is able to identify both the events and their sequences in the GUI which are of the most importance. The authors have stated that a comparative study needs to be done to compare the performance of various network measures with existing techniques for ranking GUI events so that the input EFG becomes more enriched with more information that can lead to more interesting results.

Event Flow Graph (EFG) [17] is a specific model of the GUI for a particular application, representing all possible sequences of events that a user can execute on that GUI. It represents dynamic behavior of a GUI. Nodes correspond to events and edges correspond to event sequences or event-flow relationship between two events. They are cyclic in nature such that an event may be executed more than one time during a session with an application.

Event Interaction Graph (EIG) which is based on the events and interaction among them [15]. Reverse engineering techniques are used to obtain EIG but some limitations are associated with it. It does not contain any state-based relationships. Secondly its

coverage requires a large number of test cases. EFG or EIG are used for test coverage and not for test case generation. They can be used for TCG but this has not been done.

Ethar Alaska, Atif Memon, et al gave the concept of using EFG and network analysis technique for test case generation specific to graphical applications [17]. They have used betweenness centrality score to rank events and not for generating test case. Generating test cases and later on prioritizing them using the results obtained from ranking of events has to be done.

2.2.3 Limitations

Some of the limitations associated with the already prevailing techniques are as follows.

1. Number of possible states involved in GUI application is large and interactions among them is very complex, so validation is difficult [3, 14, 16].
2. Regression testing is problematic due to change in sequence of states. Test cases break when there is even a slighter change in GUI interface of an application. [14, 16].
3. Number of events and interaction among them is very huge [3, 5, 14, and 15].
4. Coverage of test cases is difficult [3, 14, and 16].
5. More GUI testing experience is required [10]
6. Semiautomatic GUI ontology construction and the extraction of test case generation rules [10]
7. Test case generation and prioritization has to be done using the ranking of events obtained from EFG [17].
8. Complete specification of ontology needs to be provided. Also extensive experimentation is required to verify the results [5].

2.3 ONTOLOGY

Ontology is the mean for capturing domain knowledge in a generic way that provides a commonly agreed understanding of a domain [5]. “An ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them.” [6]

Wikipedia defines ontology as “In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain and may be used to describe the domain” [8].

Ontology is a complex multi-disciplinary field. It uses the concepts from information organization, natural language processing, information extraction, artificial intelligence, knowledge representation and acquisition etc [7].

Ontologies are commonly used on the World Wide Web (WWW). These ontologies range from large taxonomic categorization e.g. Yahoo! to categorizations of products for sale and their features e.g. Amazon.com [6].

2.3.1 Related Work

Ontologies can be classified according to their expressiveness. There are several levels of expressiveness [26, 28]:

- *Controlled Vocabulary* lists the terms
- *Treasures* gives relationships between different terms
- *Informal Taxonomy* describes explicit hierarchy without strict inheritance.
- *Formal Taxonomy* entails strict inheritance.
- *Frames* which is based on properties that are inherited by subclasses and instances.
- *Value Restrictions* is based on restricted/fixed property values e.g. data types.

- *General Logic Constraints* logical or mathematical formulas may be used to constraint values from other properties.
- *First-Order Logic Constraints* are very expressive ontologies which allow first order logic constraints between terms and more detailed relationships among them. e.g. disjoint classes, disjoint coverings, inverse relationships, part-whole relationships. etc.

Following figures shows the “*Ontology Expressiveness Spectrum*” [26]

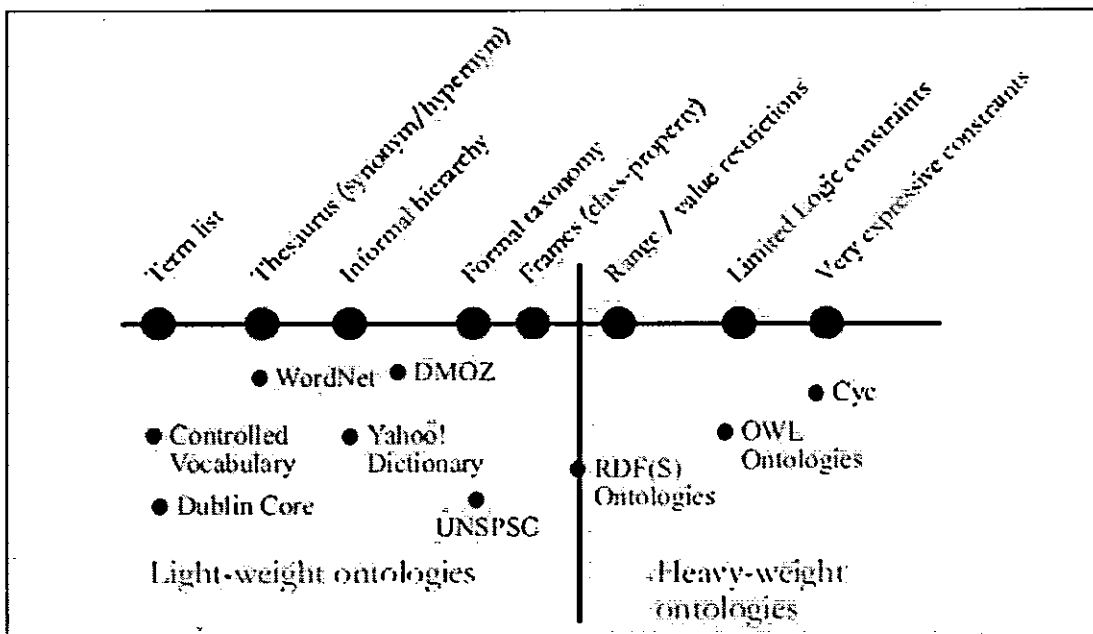


Figure 4: An Ontology Expressiveness Spectrum

Constructing ontologies can be achieved in two ways; domain dependent or generic [27]. Classes in ontology can be constructed in many ways [6] i.e. **Top Down**: from generalization to specification, **Middle Out**: from the most important concepts to generalization and specialization, and **Bottom Up**: from specification to generalization. Ontologies are commonly used on the World Wide Web (WWW). The concept of “*Semantic Web*” has gained popularity in the last few years. The term has been introduced by Tim Berners-Lee, the inventor of WWW. A semantic web is an extension of World Wide Web (WWW) which provides much more automated services by

presenting information in a machine readable form. Ontologies provide common and shared domain concepts which are a key ingredients of semantic web [6, 7, 26].

Harvey Siy and Yan Wu in their paper [9] proposed ontology for experiments that belong to software engineering field which can be used to validate experimental designs. The ontology is based on the experiments that are documented in the empirical software engineering body of literature.

Authors of [28] have proposed a novel approach to generate ontologies for the generation of fault injection test cases and failure detection. Dependability assessment increases our confidence in a system to be obtained, even though this does not assure correct operation under all circumstances. Ontology has been created for every element of Service Oriented Architecture (SOA) which is further used to trigger test cases.

Automatically generating ontologies in Semantic Web is an active area of research. Authors of [29] have proposed an approach which creates and renews the ontology automatically which are related with the keywords provided by the user.

The authors Chang-Shing Lee et al in their paper have used the concept of fuzzy ontology for Chinese news summarization. “The fuzzy ontology is an extension of the domain ontology that is more suitable to describe the domain knowledge for solving the uncertainty reasoning problems.” [30].

Ontology has been used in the field of requirement engineering. The authors Leonid Kof et al. have used ontology for requirement validation [31]. They first constructed application model ontology and then they checked its consistency with message sequence charts those who were relevant to the domain. Then a comparison is made among the models attained from requirements document and generic domain ontology. Through this comparison analyst can uncover information missing from the requirements documents.

The authors of [32] have used ontology for automatic speech recognition and its generation when an agent based application is to be used by a user. The authors of [33] have used ontology for testing robustness of web services. They have proposed a novel

approach to generate test data for robustness using OWL-S (Web Ontology Language for Services).

2.3.2 Benefits of Ontology

Ontology has a main advantage that it aids in common shared understanding of knowledge among the people and software systems [5, 6, 7] and effectively communicates this knowledge [5, 7]. It also provides ease of maintenance and updation. Through its use separation of domain and application knowledge is possible [6] and domain knowledge can be re-used [5, 6, 7]. It improves information organization, management & understanding [7].

Some of the advantages offered by ontology are mentioned as under.

- Shared understanding of knowledge among people & s/w systems [5, 6, 7, 30].
- Effective communication [5, 7].
- Reuse of domain knowledge [5, 6, 7].
- Ease of maintenance and updation.
- Improves information organization, management & understanding [7].
- Systems interoperability [7].
- Separation of domain and application knowledge [6].
- Explicit stated domain assumptions [6].

2.4 ONTOLOGY BASED TEST CASE GENERATION

To the best of our knowledge Han Li, Feng Chen et al. in their paper titled “An Ontology-based Approach for GUI Testing” [10] first ever introduced the new branch of testing i.e. ontology-based GUI testing. They have taken the knowledge intensive features of GUI testing into account. There are some short comings in their work. More experience is required regarding graphical testing so that a complete set of rules for test case generation can be achieved which is explicit in nature as well. Abdul Rauf, Sajid Anwar, et al. In their paper titled “Ontology Driven Semantic Annotation Based GUI Testing” gave the concept of using semantic annotation for GUI testing [5]. They

presented an approach to automate the test case generation process for GUI testing. It is based on semantic annotation and ontology and used the concepts from GetFollows algorithm as well. They also claimed that their proposed ontology could be used to group events based on functionality and hence reduces the manual effort required to do so.

An annotation can be defined as *meta tag*. It can be used to specify additional information to the code being used. They do not have direct influence on the program semantics, but affect the semantics of the program which is being executed. Semantics refer to the study of meanings, usually in language. Semantic annotation is a fundamental knowledge which is being used for the development of intelligent contents and also in their usage.

Limitations of the paper are that complete specification of ontology needs to be provided. Secondly extensive experimentation is required to verify the results.

2.5 SUMMARY

Ontology based test case generation is a new branch of testing and is in its initial stages. Han Li, Feng Chen et al. have first introduced in their paper [10] the concept of GUI testing based on ontology. Later on the authors have presented a framework incorporating ontology into test case generation for graphical user interface applications.

CHAPTER 3: METHODOLOGY

3.1 INTRODUCTION

Ontology based test case generation for graphical user interface applications is a new branch of testing. Very little work has been done in this area. Authors of [5] proposed a framework to generate test cases for a GUI enabled application based on ontology. I have validated their proposed framework using experimentation as a mechanism.

We have constructed ontology for File and Edit menus of Notepad application. Options *New*, *Open*, *Save*, *SaveAs* and *Exit* are covered under File menu. In Edit menu options *Undo*, *Cut*, *Copy*, *Paste*, *Delete*, *Select All* and *Time/Date* are covered. Protégé™ version 4.1 has been used to construct domain models with ontologies. It is an open source software and provides two ways for modeling ontologies i.e. (1) Frame-based modeling. (2) Modeling ontology using OWL. OWL stands for Web Ontology Language and supports classes, instances, properties and data values stored in semantic web document. OWL2 has been used for the ontology development in our scenario. Ontology had been constructed and later on added information of properties and rules to this. Event flow graph of the application is extracted and later on test cases were generated using this.

3.2 EXPERIMENTAL DESIGN

The motivation behind using experiment as a validation tool is that we can create a controlled environment in which an application can be tested. Secondly experiments are suitable for validation of applications or techniques. We will start the experiment by falsifying the null hypothesis.

Hypothesis

1. The proposed ontology is able to group all number of events with respect to testing.
2. The proposed ontology is able to generate test cases that can detect maximum faults in the application.
3. The proposed ontology is extendable to support future development.

Null Hypothesis:

1. The proposed ontology is not able to group all number of events with respect to testing.
2. The proposed ontology is not able to generate test cases that can detect maximum faults in the application.
3. The proposed ontology is not extendable to support future development.

Treatment: The framework to generate test cases through ontology.

Experiment Operation: The experiment has been executed in following steps.

1. Ontology for *File menu* and *Edit menu* of Notepad application was created.
- 1) Generated rules for the ontology.
- 2) Event Flow Graph of the respective ontology was created.
- 3) Test cases were generated based on the EFG.
- 4) The results of the execution were analyzed using performance measures coverage and efficiency used for evaluation.

Experiment objects: The test case generation will act as an experiment object.

Experiment subjects: Students of undergrad level will be the subjects to perform this experiment.

Experimental Design: One-Shot Case Study

Experimental Steps:

1) Problem identification. 2) Formulate hypothesis. 3) Ontology Development. 4) Test Case Generation. 5) Ontology development based on already generated test cases. 6) Execution of the application. 7) Verify the results by applying it on unknown applications.

Independent Variable: Framework for Automatic Generation of GUI tests by [5] will be independent variable during ontology based test case generation.

Dependent Variable: The dependent variables in our experiment are the number of events being grouped, the number of test cases being generated, whether the interaction supported among the events will be one way or will it also work beyond it. Metrics used to detect these attributes are coverage and test efficiency.

Control Variable: Educational experience of students and background.

Random Variable: Cultural background, ethnicity etc.

Internal Validity: Students have some attributes associated which vary from one person to another like their skills, background, educational experience, performance etc. The selection of students will be from the same domain having same education experience and background. This will increase our confidence on results attained because they will not vary due to educational experience but due to the individual skills or fatigue etc but will be based on the technique used. If we pick students from different educational level and experience then our results can vary.

External Validity: The experiment will be recursive. We will conduct it multiple times on a number of applications, with larger number of subjects so that we can generalize the results gathered from our experiment. Repeating the experiment at different sessions ensures that the results are due to our technique used rather due to the fatigue of continuous being involved in it.

Table 1: Experimental Design

Steps	Procedure		Aim
Step 1	ONTOLOGY CONSTRUCTION	Three weeks of instruction on ontology construction and rule development	Students may get an idea what an ontology is, how we construct it, how can we apply rules to it etc.
Step 2	TEST CASE GENERATION AND EVENT FLOW GRAPH	One week of instruction on test case generation and event flow graph.	Students may get an idea how test cases are generated for an application, what are different terms used in this process, etc. They may also get an idea about what is event flow graph, how can we construct it for an application etc.
Step 3	FRAMWORK TO GENERATE ONTOLOGY BASED TEST CASE GENERATION FOR GUI	One week of instruction on how to generate test cases for a GUI application based on ontology.	To influence the dependant variable i.e. Coverage and Test Efficiency
Step 4	ONTOLOGY BASED TEST CASE GENERATION FOR NOTEPAD APPLIACION		Measure the degree of change on the dependant variable. E.g. how many events can be grouped based on ontology, how many test cases can be generated using ontology

3.3 PROBLEM DEFINITION

GUI based applications provide interfaces to their users which are complex in nature and the user interacts with those through events. The number of events involved and the interaction among them is very large. Regression testing is also difficult for a graphical application; test cases break when there is a slighter change in sequence of states.

GUI testing is a complex task and knowledge intensive as well. Complicated & different kinds of components are involved in a GUI application and there is a complex relationship among those components. A lot of abundant testers' experience is required to accomplish this task.

GUI applications have most of the components in common like text boxes, menus, buttons etc. By making a general ontology for these test cases we can achieve its reusability independent of the application knowledge. Test cases for a GUI are specific to the application for which they have been generated. Our aim of research is to generate ontology of graphical user interface test cases which are not application specific and can be reused no matter what the application is.

3.4 RESEARCH QUESTION(S)

Q1. How does the proposed ontology provide automation for grouping events?

Ontology has been created keeping in mind the grouping of events. We created class hierarchy of events according to their levels. For example events which correspond to level one of Notepad application are FileMenuClicked, EditMenuClicked, MaximizeButtonClicked etc. At level two we have events FileMenuExpanded, EditMenuExpanded, WindowMaximized respectively. Events of the application are grouped according to the levels in which they fall. This provides ease in generating test cases; when there is need of specific ones then only those test cases will be generated which correspond to desired level and not all.

Q2. How does the proposed ontology detect sufficient number of faults?

While adding rules to the ontology we experienced a problem. Generating rules which involve interaction of events up till level 3 is manageable. As we keep on increasing the number of events in interaction rules become complex. I have mentioned two examples each having four or five events interacting with each other:

Following is among one of the rules created for “*SaveChangesDialogBox*”.

((hasFileContents only NonEmpty) and (isFileSaved only UntitledFile)) and ((hasSelectedMenu only New) or (hasShortcutKey only Ctrl+N))

As we can see four events are involved in the interaction i.e.

- Whether the contents of file are empty or not?
- Whether the file has already been saved or is it a new one?
- Which menu was selected to reach this event?
- Does the event have any shortcut key associated to navigate among events to reach this level?

Similar is the case with rule belonging to event “*OpenDialogBox*” at Level 2

((hasFileContents only NonEmpty) and (isFileSaved only UntitledFile)) and ((hasSelectedMenu only Open) or (hasShortcutKey only Ctrl+O))) and (previousMenu only Don'tSave)

As we can see five events are involved in the interaction i.e.

- Whether the contents of file are empty or not?
- Whether the file has already been saved or is it a new one?
- Which menu was selected to reach this event?

- Does the event have any shortcut key associated to navigate among events to reach this level?
- What menu was selected at previous level?

Q3. How can the ontology be extended for the modified version of the application?

Components which are used in any graphical application are universal e.g. text boxes, menus, buttons etc. The test cases are created based on the interaction among the events and not on their sequence. We can write “*file contents are not empty AND file is not saved*” as “*file is not saved AND file contents are not empty*”. Both the sentences have same meaning keeping in view only the events. There are only two events involved; one is about the file contents and the other is about existence of file.

Following is an example which illustrates the rule stated above means both are same.

((hasFileContents only NonEmpty) and (isFileSaved only UntitledFile))

⇒

((isFileSaved only UntitledFile) and (hasFileContents only NonEmpty))

General ontology of test cases can be reused independent of the application for which it was created. The shortcomings which are in an old ontology are inherited in the new one e.g. creating rules in which more than five events are involved etc.

3.5 PROPOSED SOLUTION

Several techniques for GUI test case generation exist. Most of them are based on events and interaction among them like state-based techniques use finite state automata; and directed graph models use event flow graph for this process.

An experiment has been conducted to validate the framework. Figure 2 shows the layout in which we carried out the steps of our experiment. First of all we developed ontology for a simple GUI application i.e. Notepad using Protégé 4.1. Later on we added rules to the ontology and then created test cases from this ontology. As a last step we analyzed the results obtained from the experiment.

Figure 3 shows the graphical representation of notepad application and also the levels of events. User can create entities and their class hierarchy. Figure 4 shows the class hierarchy of notepad application. The hierarchy is a tree like structure and *Thing* is the root class/super class of all the classes. Next we add object and data properties to the entities. Figure 5 shows the properties tab expanded. There are certain characteristics associated to properties like functional, inverse functional, transitive etc. User has to specify these for the added properties. Then we add rules to the ontology. Figure 6 shows the procedure for adding rules to an event.

Protégé provides several functionalities to its users like creating new ontology, exporting already existing ontology, merging two ontologies etc. It also creates an EFG of the ontology. Figure 7 shows the EFG of “*NoAction*” event which has been expanded on the basis of certain criteria e.g. *has individual*, *has subclass*, *hasFileContents* and many more which have been elaborated in Figure 8.

We generated multiple set of test cases from our ontology, which start from fifty and goes on till one hundred and thirty i.e. 50, 75, 100 and 130. To calculate the performance I have used the metrics of coverage and test efficiency. The formulas for these metrics are mentioned below.

$$\text{Coverage} = \frac{\text{Total number of covered paths}}{\text{Total number of test cases}}$$

$$\text{Efficiency} = \frac{1 - \text{Total number of covered paths}}{\text{Total number of test cases}}$$

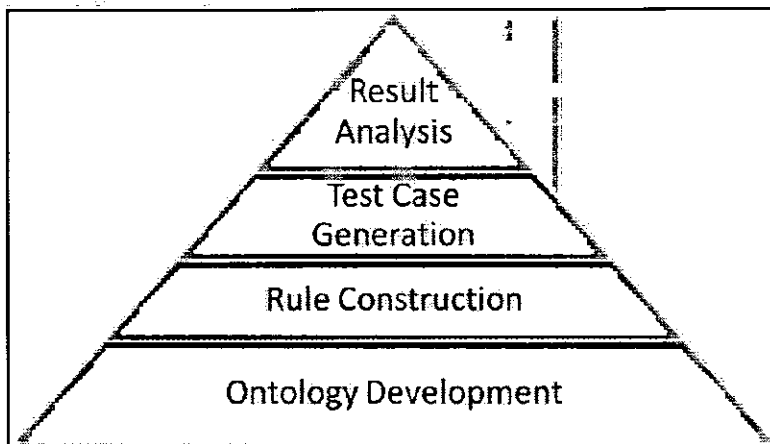


Figure 5: Experiment Setup

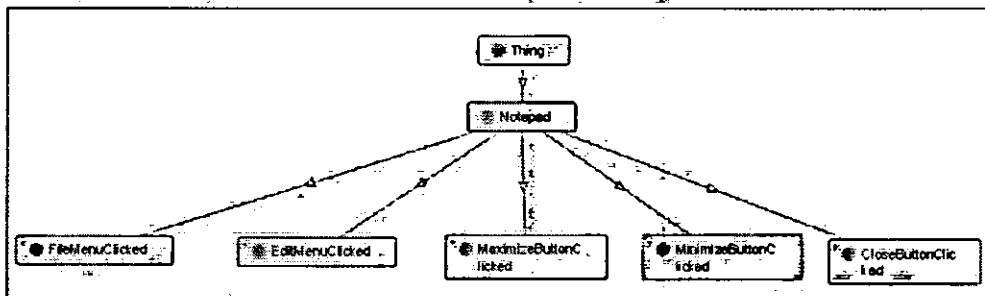


Figure 6: Level 1 Graph Representation of Notepad Application

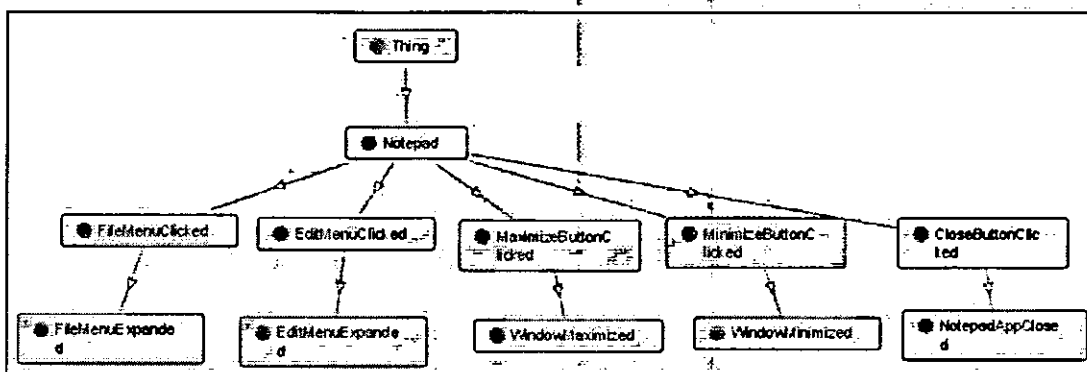


Figure 7: Level 2 Graph Representation of Notepad Application

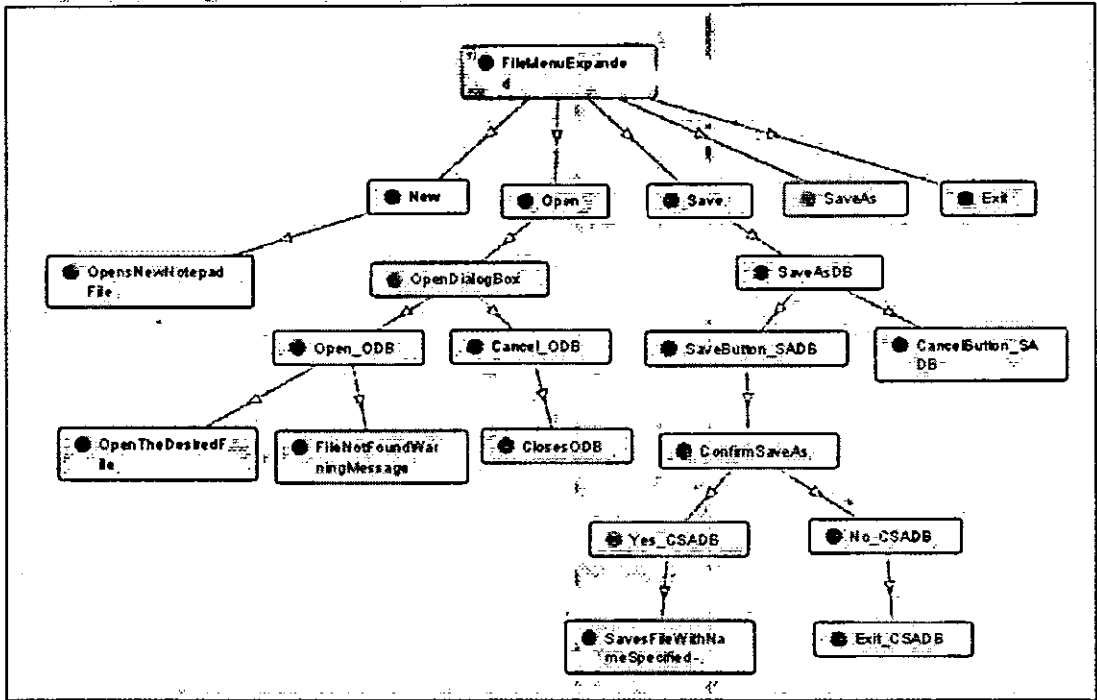


Figure 8: File Menu Expanded

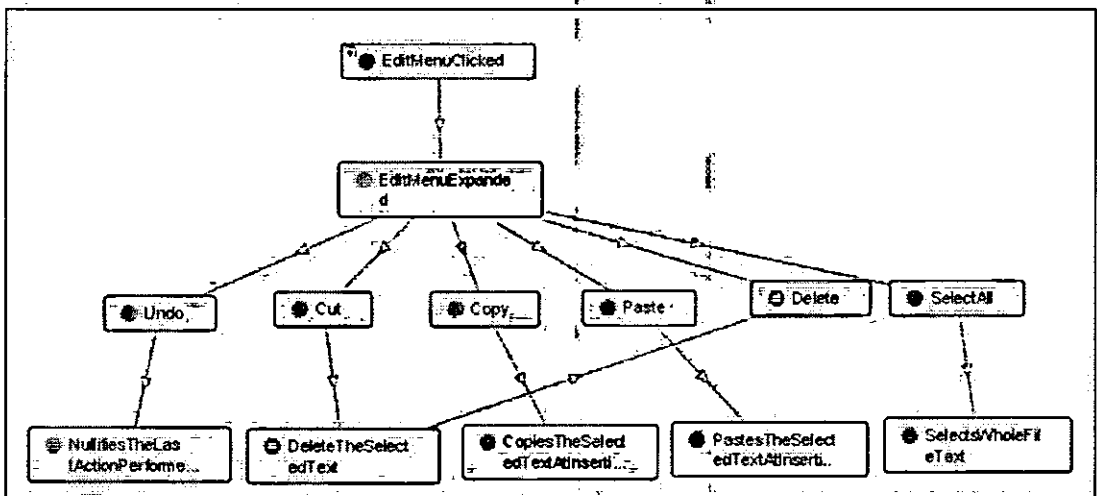


Figure 9: Edit Menu Expanded

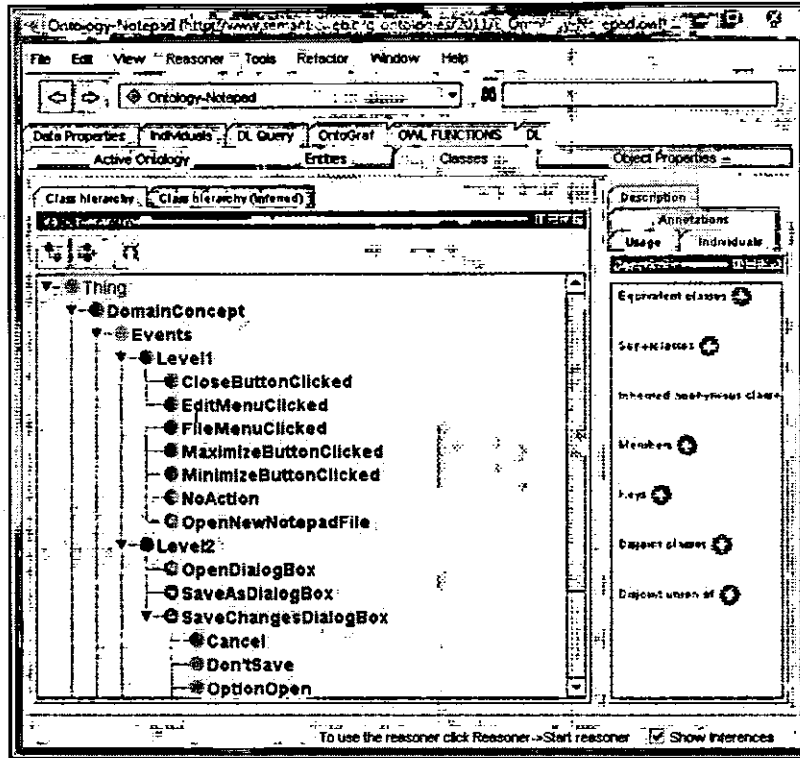


Figure 10: Class Hierarchy of Notepad Application

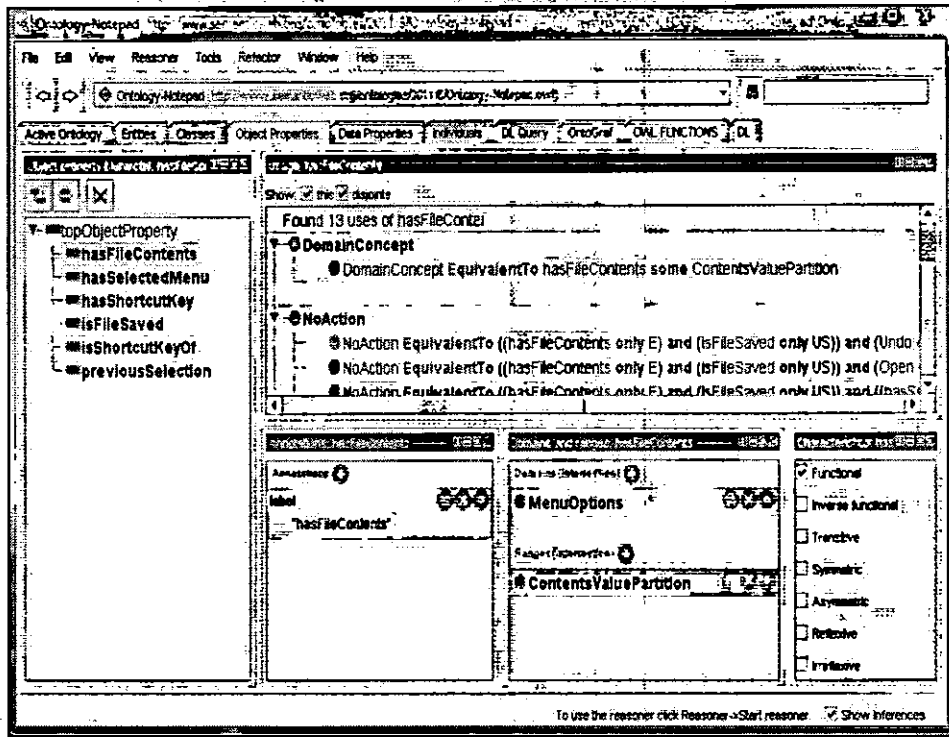


Figure 11: Properties

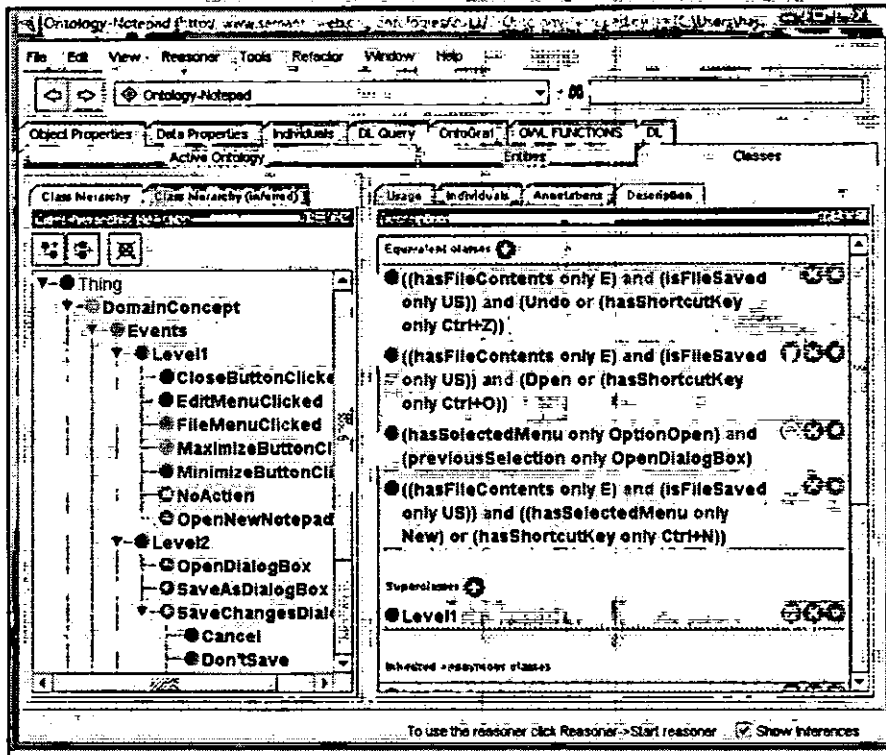


Figure 12: Rules Added

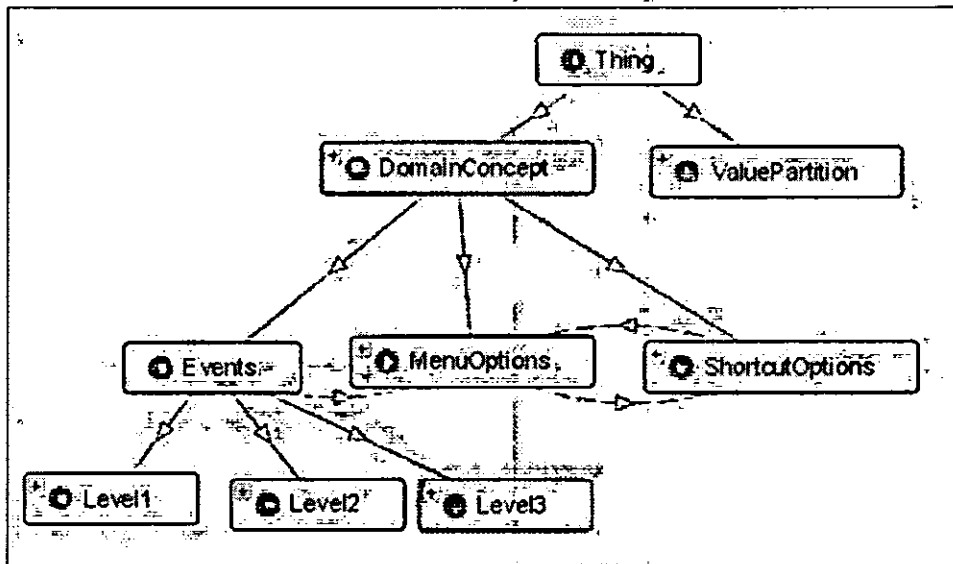


Figure 13: Overview of Events

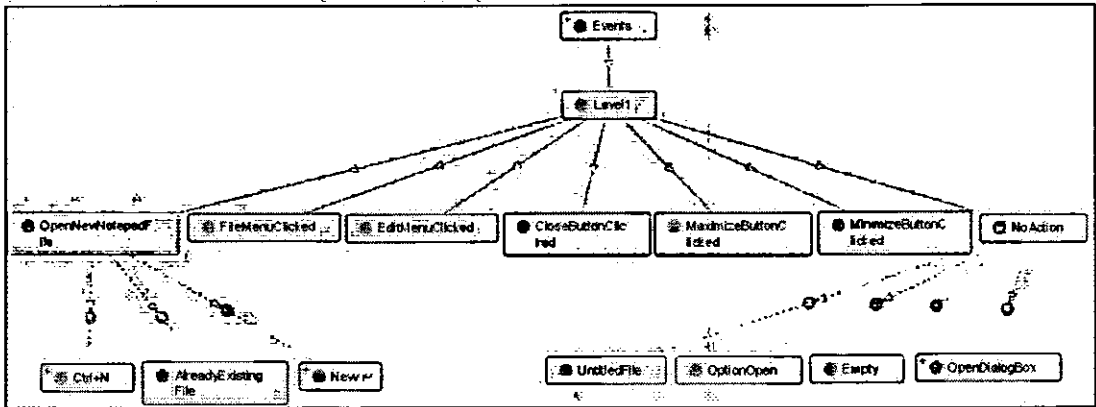


Figure 14: Events That Fall Under Level One of Application

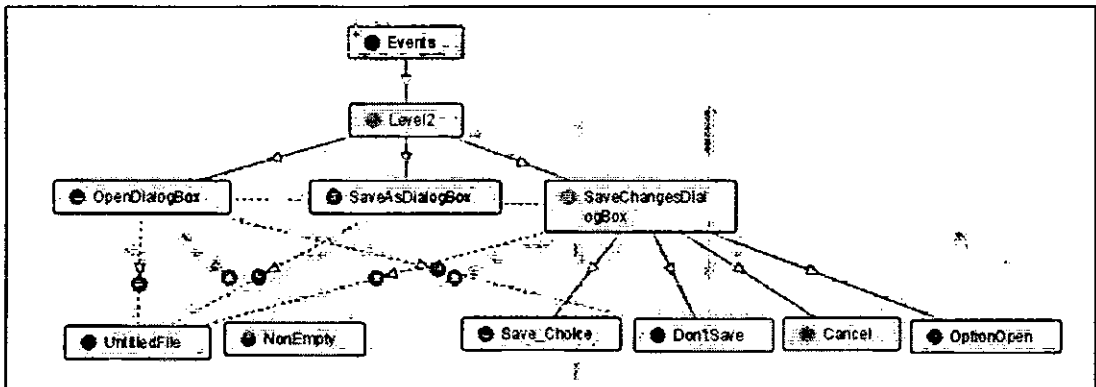


Figure 15: Events That Fall Under Level Two of Application

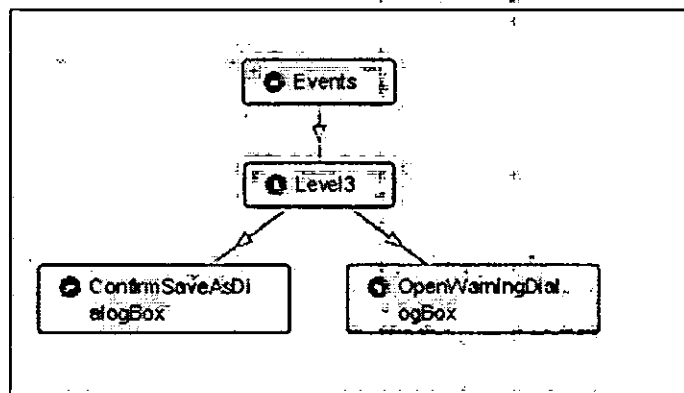


Figure 16: Events That Fall Under Level Three of Application

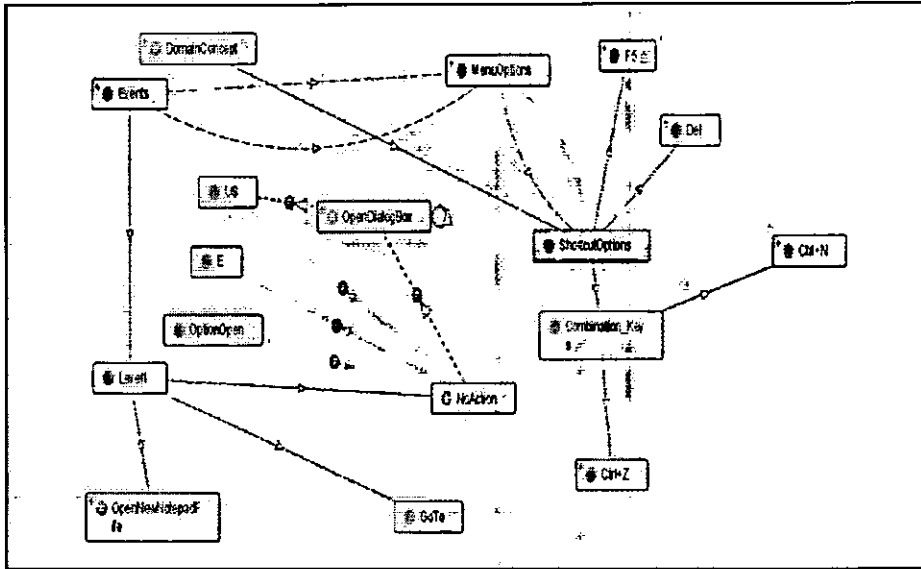


Figure 17: EFG Elaborated

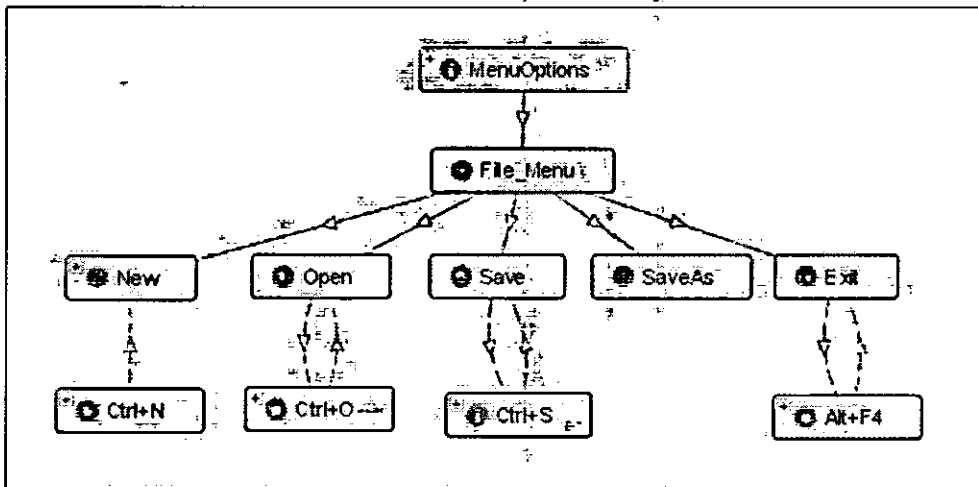


Figure 18: File Menu Options

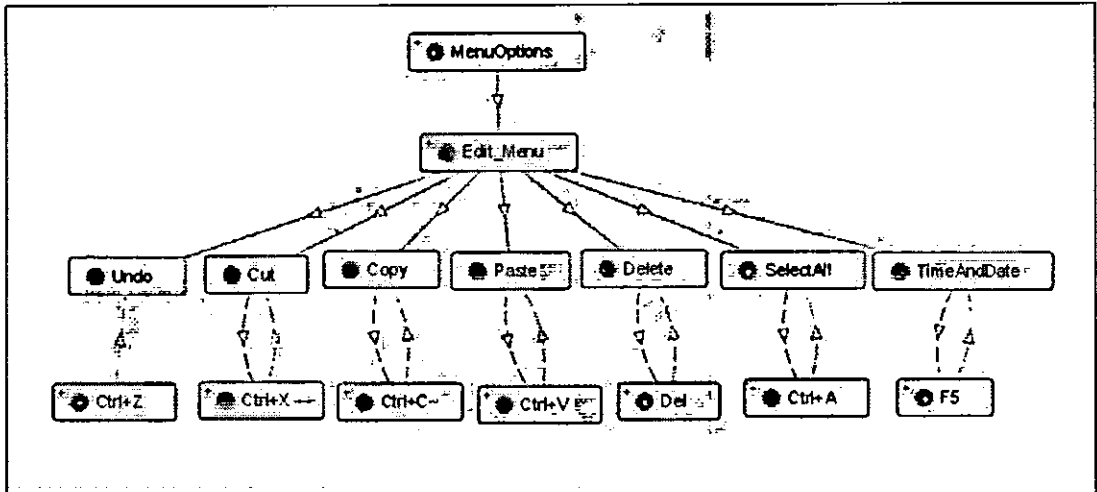


Figure 19: Edit Menu options

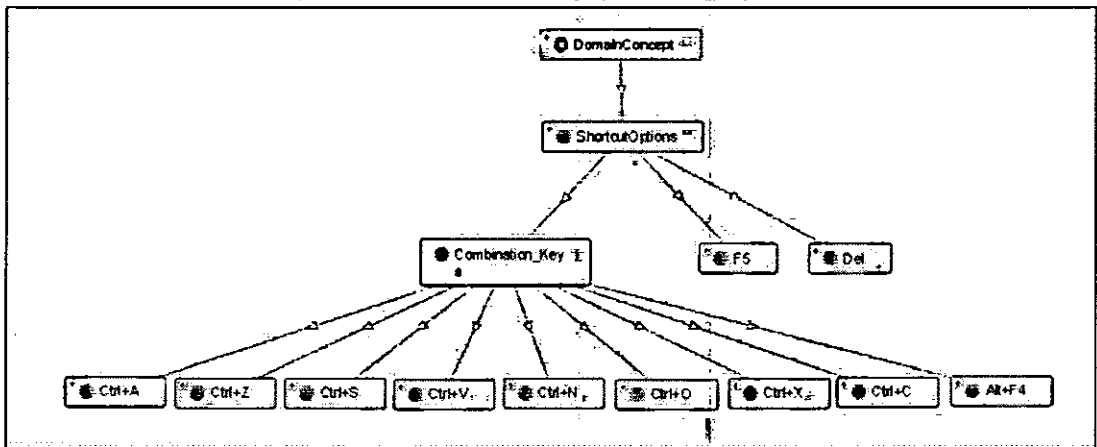


Figure 20: Shortcut Options

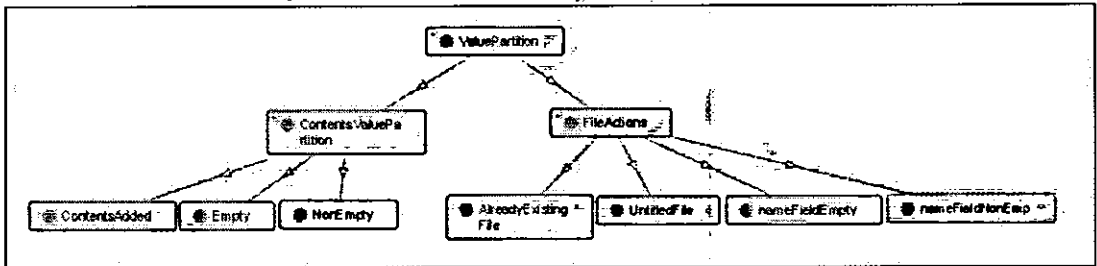


Figure 21: Value Partition

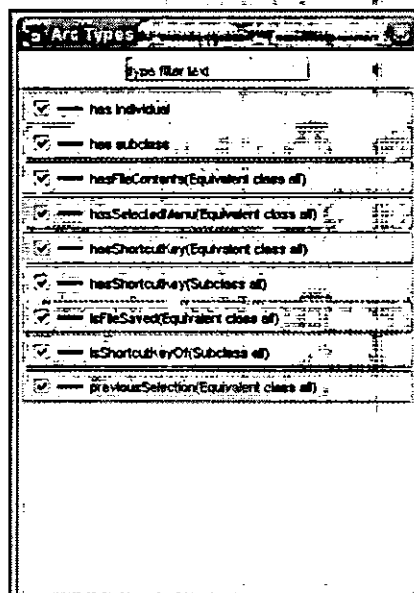


Figure 22: Arc Types

CHAPTER 4: RESULTS

4.1 INTRODUCTION

Set of test cases in our experiment has been created keeping in view that different types of events are covered each corresponding to different level of the application. E.g. FileMenuClicked or MaximizeButtonClicked are the events associated to the initial level of the application where as SaveAsClicked or SaveChangesDialogBox are the events which belong to level 3 of the application.

4.2 COVERAGE

First of all we calculated coverage for 50 test cases. The value obtained with this set of test cases was 0.71; results shown in Figure 23. With 75 test cases coverage was 0.79, and with test data of 100 cases 0.85 was the coverage attained. Figure 24 and Figure 25 shows these results respectively. With 130 test cases coverage was 0.9 which is shown in Figure 26. Figure 27 shows the combined version of coverage with different set of test cases. Later on a comparison has been drawn for coverage at each level of application; shown in Figure 28.

Number of Test Cases	Coverage		
	Level 1	Level 2	Level 3
50	0.19	0.23	0.29
75	0.21	0.27	0.31
100	0.23	0.29	0.33
130	0.25	0.3	0.35

Table 2: Comparison of Coverage According To Levels

4.2. EFFICIENCY

First of all we calculated test efficiency for 50 test cases. The value obtained with this set of test cases is 0.26; results shown in Figure 23. With 75 test cases efficiency was 0.20 and with test data of 100 cases 0.17 was the test efficiency attained; Figure 24 and Figure 25 show these results respectively. With 130 test cases test efficiency was 0.09 shown in

Figure 26. Figure 29 shows the combined version of efficiency with different set of test cases. Later on a comparison has been drawn for test efficiency at each level of application that is shown in Figure 30.

Table 3: Comparison of Test Efficiency According to Levels			
Number of Test Cases	Efficiency		
	Level 1	Level 2	Level 3
50	0.13	0.07	0.06
75	0.10	0.06	0.04
100	0.09	0.05	0.03
130	0.06	0.02	0.01

Table 3: Comparison of Test Efficiency According To Levels

4.3. COMPARISON BETWEEN COVERAGE & TEST EFFICIENCY

A comparison has been made between coverage and test efficiency for a range of test cases. Table 4 shows the results of the comparison which are graphically been represented in Figure 31.

Table 4: Comparison of Coverage Vs Test Efficiency		
Number of Test Cases	Coverage	Test Efficiency
50	0.71	0.26
75	0.79	0.20
100	0.85	0.17
130	0.90	0.09

Table 4: Comparison of Coverage Vs Test Efficiency

4.4. DISCUSSION

To accomplish a certain task user has to follow specific sequence of events associated with the application. Exhaustively testing any application is not possible. Assume that if we generate test cases for level two of the application they will always cover test cases which are for specific to level one. Hence a tester can skip generating test cases for level one. Similar is the case with level two test cases. Test cases generated for level three will

always cover test cases of level one as well as of level two. Two scenarios have been mentioned for accomplishing a task.

Scenario 1: Save An Empty Notepad File

Steps: Open notepad application then Click File menu / Ctrl+S. Click Save button

Scenario 2: Save a notepad file with some text in it. Then delete a portion from the contents

Steps: Open notepad application and type some text. Click File menu / Ctrl+S. SaveAs dialog box will appear. Specify name for file and click Save button. Select some text. Click Edit menu then click Cut or hit Ctrl+Z

Two scenarios have been mentioned here as an example which explain different tasks a user wants to achieve. In scenario 1 a user wants to save an empty notepad file then s/he has to follow certain steps. At first user has to open notepad application. It will open an untitled file for him/her. Then he must click the File menu or must hit the combination key Ctrl+S. Later on user must click the Save button. Since the file is to be saved for the first time, SaveAs dialog box will appear. Similarly in scenario 2 a user wants to save a file having some text typed in it and then cut some of the contents. To do so s/he must open notepad application, type some text in the file, save the file and later on select the text which s/he want to remove. The click cut option in edit menu.

It can be observed from the above two scenarios that some of the events are in common i.e. opening notepad application, clicking file menu and then saving file. The user has followed these steps. Assume that we want to create a test case for SaveAs dialog box. Then it will cover the events specific to level one of the application i.e. opening notepad application, clicking a menu etc.

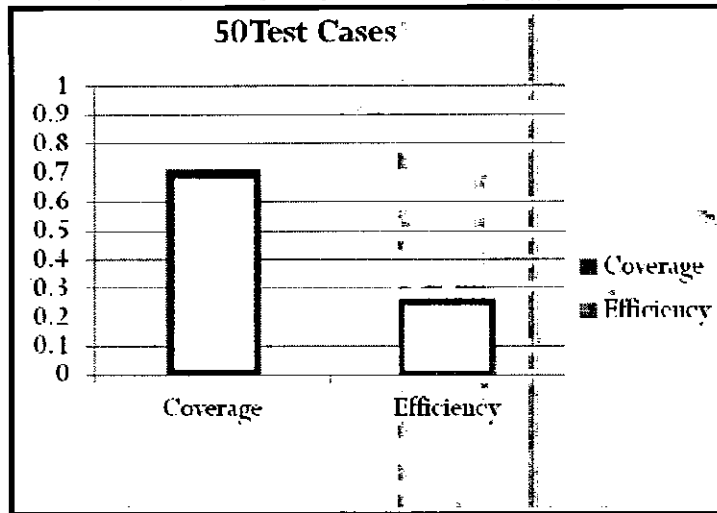


Figure 23: Results with 50 Test Cases

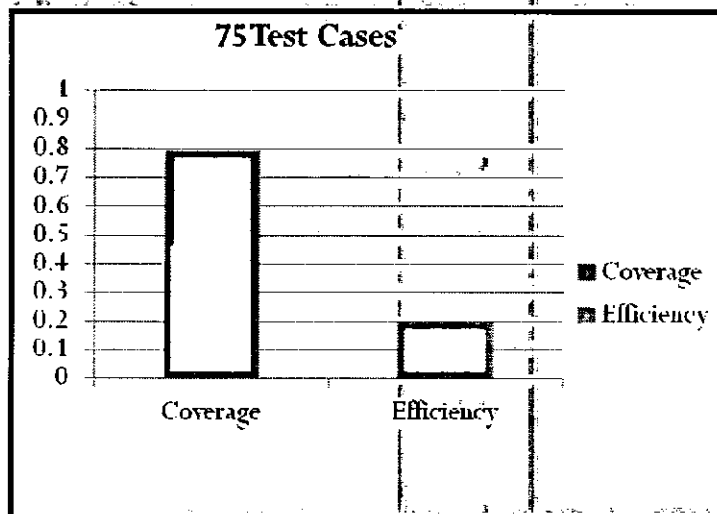


Figure 24: Results with 75 Test Cases

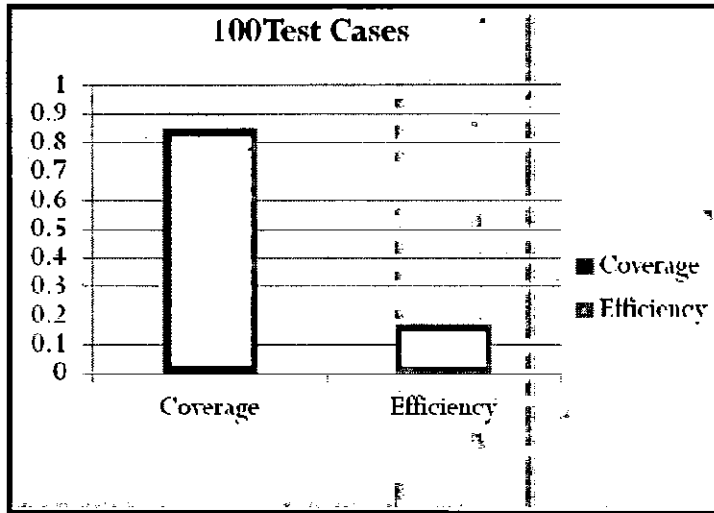


Figure 25: Results with 100 Test Cases

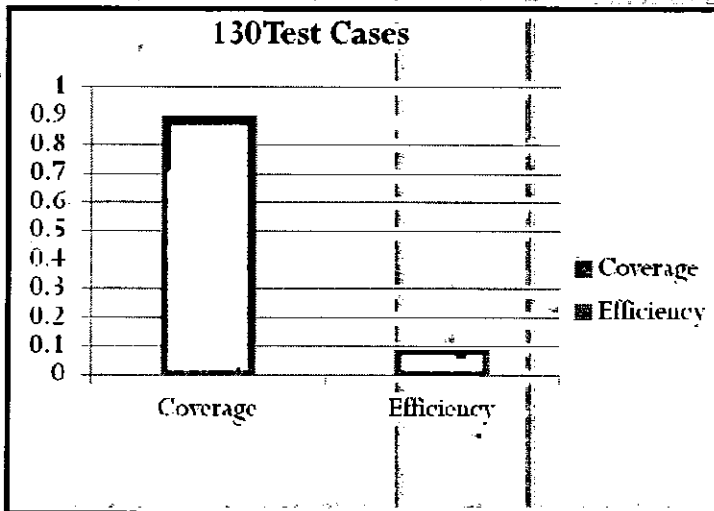


Figure 26: Results with 130 Test Cases

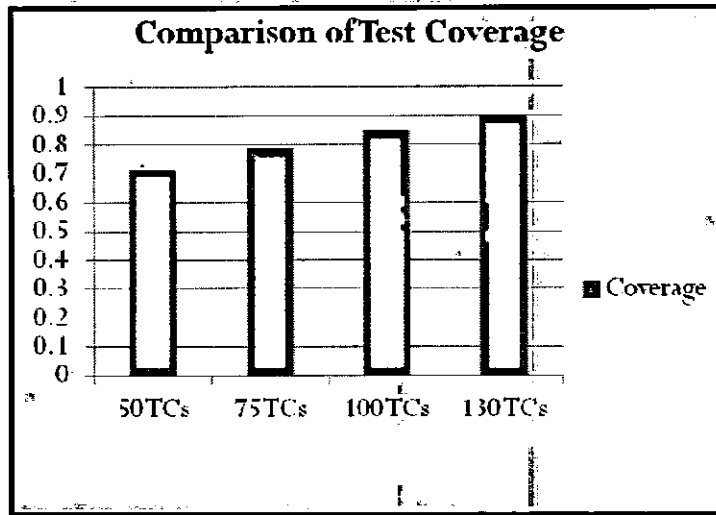


Figure 27: Comparison of Coverage

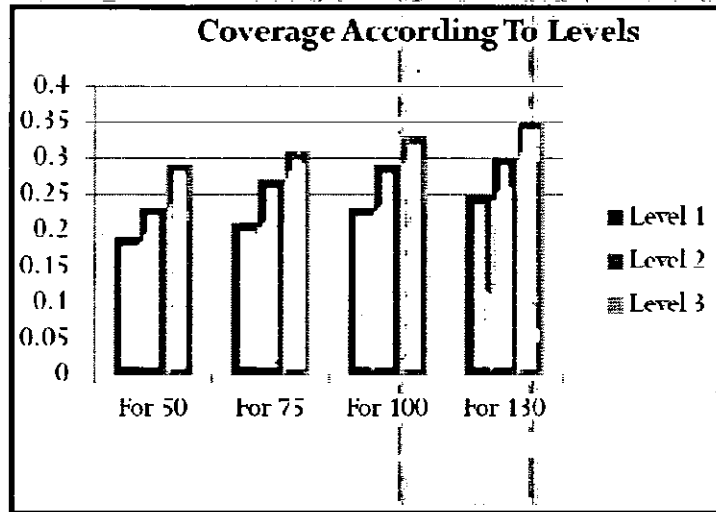


Figure 28: Comparison of Coverage According to Levels

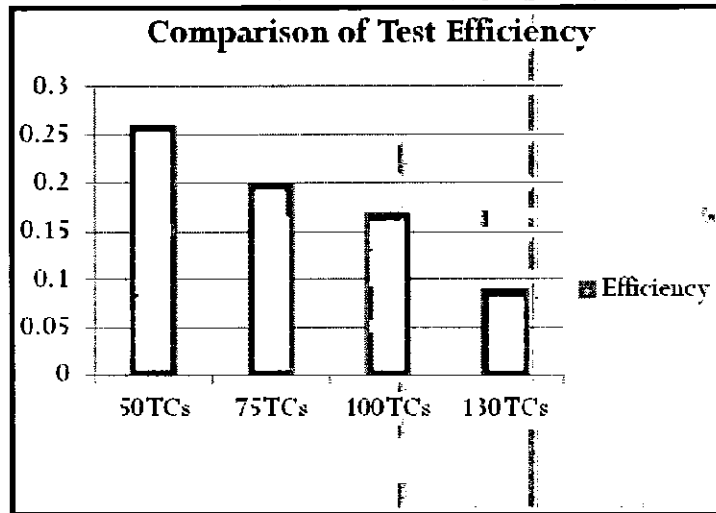


Figure 29: Comparison of Test Efficiency

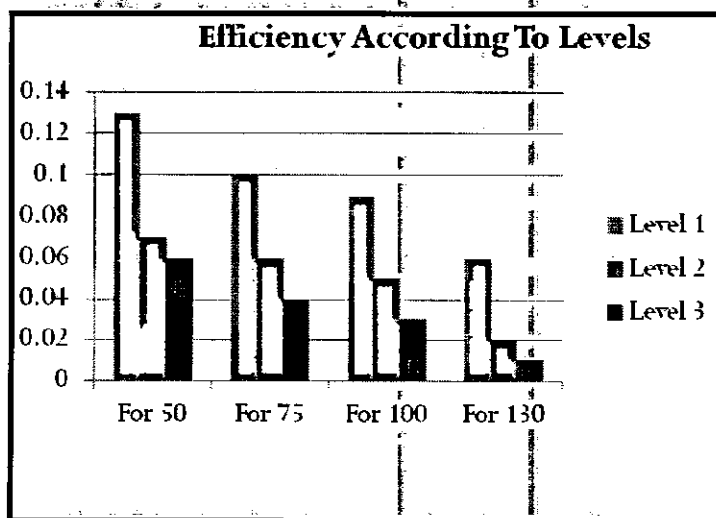


Figure 30: Comparison of Test Efficiency According To Levels

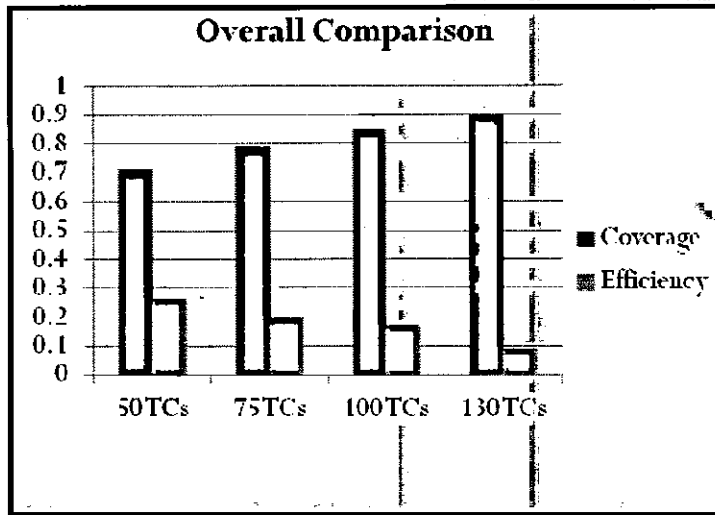


Figure 31: Comparison of Coverage & Efficiency

CHAPTER 5: CONCLUSION

5.1 CONCLUSION

In this research work a new approach for generating test cases for an application having graphical user interface has been presented. The technique is focused on using ontology as a field. An experiment has been conducted on an application i.e. “Notepad” to make the case simple. We generated multiple set of test cases using ontology so that we can gain maximum results. Since this approach is a novel one, more experimentation is required. Test cases for complex GUI application need to be made so that this technique can be improved.

5.2 FUTURE WORK

Managing rules is problematic when it exceeds four events involved in an interaction. In future work we will provide a solution to this problem.

ABBREVIATIONS

- AI Artificial Intelligence
- CLI Command Line Interface
- EFG Event Flow Graph
- EIG Event Interaction Graph
- GUI Graphical User Interface
- MBST Model Based Software Testing
- SDLC Software Development Life Cycle
- SE Software Engineering
- TCG Test Case Generation

REFERENCES

- [1] Jeff Tian; *“Software Quality Engineering, Testing, Quality Assurance and Quantifiable Improvement”*, IEEE Computer Society, 2005
- [2] M. Prasanna, S.N. Sivanandam, R.Venkatesan, R.Sundarrajan; *“A Survey On Automatic Test Case Generation”*, Academic Open Internet Journal, vol. 15, 2005.
- [3] Atif M. Memon; *“Employing user profiles to test a new version of a GUI component in its context of use”*, Springer, Software Quality Journal, vol. 14, pp. 359–377, 2006.
- [4] Xun Yuan, Atif M. Memon; *“Generating Event Sequence-Based Test Cases Using GUI Runtime State Feedback”*, IEEE Transactions On Software Engineering, vol. 36 (1), 2010.
- [5] Abdul Rauf, Sajid Anwar, Muhammad Ramzan, Shafiq ur Rehman, Arshad Ali Shahid; *“Ontology Driven Semantic Annotation Based GUI Testing”*, IEEE, 6th International Conference on Emerging Technologies (ICET), pp. 261-264, 2010.
- [6] Natalya F. Noy and Deborah L. McGuinness; *“Ontology Development 101: A Guide to Creating Your First Ontology”*,
- [7] Ding, Y., Foo, S; *“Ontology research and development, Part 1 - A review of ontology generation”*, Journal of Information Science, vol. 28 (2), pp. 123-136, 2002.
- [8] [http://en.wikipedia.org/wiki/Ontology_\(computer_science\)](http://en.wikipedia.org/wiki/Ontology_(computer_science))
- [9] Harvey Siy, Yan Wu; *“An Ontology to Support Empirical Studies in Software Engineering”*, IEEE, International Conference on Computing, Engineering and Information (ICC), pp. 12-15, 2009.
- [10] Han Li, Feng Chen, Hongji Yang, He Guo, William Cheng-Chung Chu and Yuansheng Yang; *“An Ontology-based Approach for GUI Testing”*, IEEE International Computer Software and Applications Conference (COMPSAC), 2009.
- [11] Gordon Fraser, Angelo Gargantini; *“An Evaluation of Specification Based Test Generation Techniques using Model Checkers”*, IEEE, Testing: Academic and Industrial Conference - Practice and Research Techniques (TAIC-PART), pp. 72-81, 2009.
- [12] Kenneth Kelley; *“Automated Test Case Generation from Correct and Complete System Requirements Models”*, IEEE, 2009.
- [13] http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library

- [14] Atif M. Memon, Martha E. Pollack, Mary Lou Soffa; "*Hierarchical GUI Test Case Generation Using Automated Planning*", IEEE Transactions On Software Engineering, vol. 27 (2), pp. 144-155, 2001.
- [15] Xun Yuan, Atif M. Memon; "*Iterative execution-feedback model-directed GUI testing*", ELSEVIER, Information and Software Technology, vol. 52, pp. 559-575, 2010.
- [16] Atif M. Memon, Martha E. Pollack, Mary Lou Soffa; "*Plan Generation for GUI Testing*", American Association for Artificial Intelligence, 1999.
- [17] Ethar Elsaka, Walaa Eldin Moustafa, Bao Nguyen, Atif Memon; "*Using Methods & Measures from Network Analysis for GUI Testing*", IEEE, Third International Conference on Software Testing, Verification, and Validation Workshops, pp. 240-246, 2010.
- [18] Atif M. Memon, Jayme Strecker; "*Accounting for Defect Characteristics in Evaluations of Testing Techniques*", ACM Transactions on Embedded Computing Systems, Vol. 9 (4), 2010.
- [19] Paul Gerrard; "*Testing GUI Applications*", EuroSTAR, pp. 24-28, 1997.
- [20] Qingning Huo, Hong Zhu; "*Developing A Software Testing Ontology in UML for A Software Growth Environment of Web-Based Applications*",
- [21] Xun Yuan, Myra B. Cohen, Atif M Memon; "*GUI Interaction Testing: Incorporating Event Context*" IEEE Transactions on Software Engineering, IEEE Computer Society, Los Alamitos, CA, USA, 2010.
- [22] http://en.wikipedia.org/wiki/Software_testing
- [23] Software Engineering 6th Edition, Ian Sommerville; 2000.
- [24] Pravin M. Kamde, V. D. Nandavadekar, R. G. Pawar; "*Value of Test Cases in Software Testing*" IEEE, 2006.
- [25] Craig Linn; "*A Metric Framework for Quantifying Semantic Reliability in Shared Ontology Environments*", Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), IEEE, 2004.
- [26] Jos de Bruijn; "Using Ontologies. Enabling Knowledge Sharing and Reuse on the Semantic Web", DERI Technical Report DERI-2003-10-29, October 2003.
- [27] Latifur Khan and Feng Luo' "*Ontology Construction for Information Selection*", Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02), IEEE, 2002.
- [28] Nik Looker, Binka Gwynne, Jie Xu, Malcolm Munro; "*An Ontology-Based Approach for Determining the Dependability of Service-Oriented Architectures*", Proceedings of the

- 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05), IEEE, 2005.
- [29] Joon Shim, Hongchul Lee; "*Automatic Ontology Generation Using Extended Search Keywords*", 4th International Conference on Next Generation Web Services Practices, IEEE, 2008.
- [30] Chang-Shing Lee, Zhi-Wei Jian, and Lin-Kai Huang; "*A Fuzzy Ontology and Its Application to News Summarization*", IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 35, NO. 5, OCTOBER 2005
- [31] Leonid Kof, Ricardo Gacitua, Mark Rouncefield, and Pete Sawyer; "*Ontology and Model Alignment as a Means for Requirements Validation*", IEEE Fourth International Conference on Semantic Computing, 2010.
- [32] Dario Bianchi and Agostino Poggi, "*Ontology Based Automatic Speech Recognition and Generation for Human-Agent Interaction*", Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'04), IEEE, 2004.
- [33] Luo Xu, Qiulu Yuan, Ji Wu, Chao Liu; "*Ontology-based Web Service Robustness Test Generation*", IEEE, 2009.

