Chapter 6 Coverage Optimization Based GUI Test Framework
Every development organization is eager to ensure the maximum quality in its products, but testing a graphical user interface comprehensively, is still a lurid, as GUI testing has proved to be a labor-intensive effort. As have been mentioned earlier that most important advancement in automation of manual GUI testing process is to model GUI elements and interaction among these widgets (Normally referred as events). Event-flow graph (EFG) is comparatively an unsullied and positive addition to handle automation of GUI testing. In this chapter, we are presenting a framework based on coverage optimization techniques that we have evaluated in previous two chapters. Along with coverage optimization techniques based on evolutionary algorithms, ontology based test data generation and test oracle development are the major parts of this GUI test framework. Our framework works in three steps as have been shown in figure 6.1.

This ontology theoretically works on the foundation led by semantics of feasible actions (events) and then annotations can be used to generate the test cases and work as an oracle for verification of the output of testing effort.

By annotation process, the tester indicates what GUI elements are important in terms of the following: First, which values can a GUI element hold (i.e., a new set of values or a range), and thus should be tested; second, what constraints should be met by a GUI element at a given time (i.e., validation rules), and thus should be validated. The result of this process is a set of annotated GUI elements which will be helpful during the test case auto-generation process in order to identify the elements that represent a variation point, and the constraints that have to be met for a particular element or set of elements. From now on, this set will be called Annotation Test Case.
6.1 Oracles Development

A test oracle is an instrument to assess the tangible outcome of a test case either as pass or fail by producing an anticipated end result for an input and checking the actual results against this projected result [161], as shown in Figure 6.2. Conventionally the development of oracles has proved to be hard and expensive in software testing [161][162][163][164] Efforts to replace manual test oracles with automated and partially automated oracles involve specification based oracle development, program simulations or a trustworthy implementation use [161]. Several researchers recommend using domain-specific, model-based oracles [100][165][166][167].

![Figure 6.2 General Framework for Oracle](image)

6.2 Ontology Development

Because of ease and suppleness provided by graphical user interfaces (GUIs), they are becoming most vital modules of software systems. On the other hand, a lot of research work is being carried out in software testing field but subfield of GUI testing is still not getting its due attention. Freedom offered by GUI can be presumed by the fact that a user can access a particular component in a software system by following multiple itineraries of events. This freedom stimulates the interest of end user in software system but becomes a nuisance for
testers of the application. Large numbers of permutations of events and complex event interactions of GUIs present new challenges for this kind of testing.

Ontology defines the basic terms and relations constituting the vocabulary of a specific domain area as well as the rules concerning that specific domain. Ontologies have been applied to describe a variety of knowledge domains [168]. Ontology is a formal explicit description of concepts in a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions on these concepts [169]. The ontology is the means for capturing domain knowledge in a generic way that provides a commonly agreed understanding of a domain. The solitary purpose behind building up ontologies is to share widespread understanding of the structure of information among people or software agents [170]. Knowledge gathered through different ontologies may be reused and shared within communities or applications. A growing interest on the establishment of ontologies has been observed for the most different knowledge domains. This work presents an ontology of GUI testing, which has been developed to support test case generation and oracle development on basis of the domain knowledge.

Annotation of textual or graphical documents relating to software systems is a common and important software engineering activity. Computerized development tools incorporating annotation have become available in recent years. They are used in diverse areas such as annotating source code to explain design rationale [171]. In GUI testing, the annotation process is the process by which the tester indicates what GUI elements are important in terms of the values GUI element holding (i.e., a new set of values or a range), and constraints that GUI element has to meet at a given time (i.e., validation rules) [172].

Semantics is the study of explanation of symbols as used by group of people surrounded by scrupulous circumstances and contexts. Semantic Annotation is a fundamental knowledge being used for the development and usage of intelligent contents. A broad range of different software domains are using semantic annotation for intelligence oriented products and processes. Semantics-based fact retrieval is one fundamental use of semantic annotations [173]. Annotations are being used to reveal the design decisions and rationale behind these design decisions, although these decisions are normally documented ones [174]. An
annotator is defined as an analysis agent that can be written to process each entity of a certain type independently [175].

An imperative attribute of GUI systems is that their behavior is very much dependent on the context in which they are being used. [176]. Besides the functionality of a GUI element, response of GUI element to an event may be different depending on the perspective established by preceding events and their execution order [176]. Another important fact about event driven nature of GUI is that longer test sequences are better than shorter sequences in identifying defects. In [6], authors have presented an algorithm to find out the follows of an event. This algorithm helps to determine the subsequent events following an event. Authors also have classified these events depending upon their functionality. This classification is based on domain knowledge, but is currently being done manually [176].

In this chapter, we are presenting an approach to automate the test case generation process for GUI testing based on semantic annotation and ontology. Our approach uses the concepts from GetFollows algorithm [176], semantic annotations and ontology. Our proposed ontology can also be used to remove the manual effort required in grouping events based on functionality described in [176]. The results of our study show that by increasing event combination strength and controlling starting and ending positions of events, our test cases are able to detect a large number of faults, not detected by exhaustive test suites of short tests [176]. In [176], Memon et. al relates a GUI’s response with context and says that response of a GUI to an event may vary depending on the context established by preceding events and their execution order. In previous efforts, our work on coverage analysis using GA [169] and work on coverage analysis using PSO [30] have shown that without considering the strict ordering constraints, we can have very good coverage of GUI events.

A number of annotation approaches exit for producing semantic annotations. OntoAnnotate, a framework for the semantic web, includes tools for both manual and semi-automatic annotation of pages [177]. Not unlike Knuth’s literate programming, Decker et.al has used semantic annotation for embedding in the semantic tags of ordinary hyper text markup language (HTML) [178]. Knuth’s approach basically uses few semantically relevant and formal statements that are embedded in unstructured prose text. McMaster also believes that
defining GUI element invariants in annotations would make it possible to generate test cases that cover the invariant conditions [169].

A number of ontology modeling methods have been proposed in the literature. Knowledge Interchange Format, description logic, and object oriented modeling, such as in UML are among the most widely used traditional approaches [180]. XML supports customizability, extensibility, and simplicity. Due to these reasons, XML is most commonly being used as the format to represent ontology and as a format of agent communication languages. For these reasons, XML is used in our system to codify the ontology for computer processing. However, an XML representation of ontology is at a rather low level of abstraction. It does not support the validation of the ontology by domain experts [181]. In another work, Huo et al. investigated the development of ontology of testing as a support for a multi-agent software environment which tests web-based applications [168].

6.3 Ontology Driven Semantic Annotation Based GUI Testing

Coverage analysis using evolutionary algorithms like GA and PSO has shown that without considering the strict ordering constraints, we can have very good coverage of GUI events [12, 30]. In [6], one method of modeling a GUI for testing creates a representation of events within windows (or components) called an event-flow-graph (EFG) [6]. Memon et al. explained how a GUI’s response varies with the change in the context of its use. According to [5], the absolute position of the event within the sequence affects fault detection.

In this chapter, we have made an attempt to expose the opportunities of building a close relation between semantic annotation and ontology engineering. Annotation can help a lot in GUI testing from test case generation to oracle development as has been proposed by McMaster [179]. Adding semantics to these annotations can help in capturing the context of events. Concepts and relations are contained in the ontology and as concepts keeps on growing, so proposed ontology for this work must also be evolving with the passage of time.
Figure 6.3 GetFollows from GUI Event Flow Model [17]

Figure 6.4 Proposed framework for Automatic Generation of GUI tests
Proposed approach describes a GUI test case auto generation process based on ontology and the annotations relevant to the GUI elements. All the promising test cases are created automatically depending on the values defined during the annotation process [6].

As we can see in figure 6.3, GUI event flow model produces the list of follows of each event. Incorporating this follows set into ontology can produce the list of predecessors and follows of each event. Event flow graph is being used to build ontology as have been shown in figure 6.4. This GetFollows algorithm of this model helps ontology to grow with ordered list of events. Document specification can also be used as a useful tool in evolving this as well as expert opinion. For each GUI element, when GUI testing framework interacts, it reports the ontology name of the element and event. Ontology extracts the follows set and set of predecessors for each event. Ontology, annotates the widget on semantics basis. Relationship between semantic annotations with ontologies is the core of our proposed method.

For ontology implementation, OWL 2 has been used as ontology language. The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF (Resource Description Framework), and OWL 2 ontologies themselves are primarily exchanged as RDF documents. RDF is a representation format for meta data defined by the W3C. It is used for representing metadata for describing the semantics of information in a machine accessible way. Figure 6.5 shows an ontological implementation of GUI hierarchy. This implementation is based on the relationship that exists between different GUI widgets (objects). This relationship has been shown in form of a graph in figure 6.6.

Figure 6.5 Ontological Implementation of GUI Hierarchy
The RDF along with RDF graph and RDF vocabulary description language (RDF schema) could take a central part in this development, since RDF graph consists of concepts and relations. The relations denote the semantic associations between concepts and the RDF vocabulary description language (RDF schema) extends RDF to include the basic features needed to define ontologies. The most important reason that makes us adopt RDF graph in our work is the structure similarity between event flow graphs and RDF graph, as RDF graph consists of concepts and relations and event flow models provides elements and possible interactions between these elements. RDF is a simple language with a labeled directed graph as its underlying data structure and its only syntactic construct is the triple, which consists of three components, referred to as subject, predicate, and object [182]. A triple represents a single edge (labeled with the predicate) connecting two nodes (labeled with the subject and object); it describes a binary relationship between the subject and object via the predicate [182].

6.4 Summary

In preceding chapter, we have shown that coverage analysis using evolutionary algorithms without considering the strict ordering constraints; we can have excellent coverage of GUI events. In this chapter, we have tried to use test coverage optimization for building a GUI test framework. Semantic annotations based test case generation was suggested by more than one researcher in literature. In this chapter, we have made an effort to blend the concept of ontology development with semantic annotations to generate test cases.