Automated Specification-Based Graphical User Interface Testing

by

©Krista A. King

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the requirements for the degree of

M.Eng. in Computer Engineering

Department of Electrical and Computer Engineering

Memorial University

May 2015

St John's Newfoundland and Labrador

Abstract

The event-based nature of graphical user interfaces, or GUIs, increases the difficulty of testing software applications. Automating this process reduces not only the time to test, but also the work involved in preparing the tests to be run. However, some form of specification is required for this automation.

TUIL, or Testable User Interface Language, is a lightweight prototype specification language based on XML that was created to specify desktop, widget-based application GUIs. TUIDE, or Testable User Interface Development Environment, is a prototype Java application that is used to handle the testing automation processes, as well as mock-up TUIL specifications.

A small Java application, containing four dialogs, is created to prove that the TUIL language and TUIDE processing can not only find errors within faulty versions of it, but also generate reasonably accurate mock-ups of the dialog GUIs. The findings from this testing are positive.

Acknowledgements

I would sincerely like to thank my supervisor, Dr. Dennis K. Peters, for all his support and guidance during the course of this research and the writing on it. I would also like to thank the Faculty of Engineering and Applied Sciences of Memorial University.

This thesis is dedicated to my parents, who have supported me throughout my life and graduate degree. It is also in loving memory to my mother, who was not able to see the end result.

Contents

List of Tables

List of Figures

List of Listings

List of Algorithms

List of Abbreviations

Abbreviations	Description	Page
GUI	Graphical User Interface	1
UI	User Interface	1
XML	eXtensible Markip Language	14
PATHS	Planning Assisted Tester for grapHical user interface Systems	7
FSM	Finite-State Machine	9
VFSM	Variable Finite-State Machine	9
MDL	MASTERMIND Dialogue Language	10
CSM	Concurrent State Machine	10
SMV	Symbolic Model Checking	10
FSA	Finite State Automata	10
HFSM	Hierarchical Finite-State Machine	11
CIS	Complete Interaction Sequences	11
VIEWS	Vendor Independent Extensible Windowing System	14
Haste	High-level Automated System Test Environment	11
XAML	eXtensible Application Markup Language	15
WPF	Windows Presentation Foundation	16
UPF	Universal Presentation Framework	16
XUL	XML User Interface Language	16
XBL	eXtensible Bindings Language	16
XIML	eXtensible Interface Modelling Language	16
TUIL	Testable User Interface Language	22
AUT	Application Under Test	19
TUIDE	Testable User Interface Development Environment	65
JAXB	Java Architecture for XML Binding	48
XJC	JAXB Binding Compiler	48
UTC	Coordinated Universal Time	61

...

Abbreviations	**Description**	**Page**
GMT	Greenwich Mean Time	61
JAR	Java ARchive File	66
DART	Daily Automated Regression Tester	7

Chapter 1

Introduction

Since the first software programs were developed, determining a specification of proper program behaviour has been a challenge. This specification is extremely important, not only when developing the implementation for the program, but also during verification and testing. It is the specification that lets the tester know if a particular behaviour has been properly implemented. In particular, the User Interface (UI), or Graphical User Interface (GUI), behaviour of the software application is important to get right, as this is the portion of the program that the end user will see, thus if the UI behaviour is implemented incorrectly it will be more readily discovered by the user.

In many cases the program specification will be generated in human readable text and images, i.e. documents. These are important, as the developers must be able to read and understand the desired program behaviour and appearance in order to implement it. As well, using these documents as a basis, the tester will know how to test each part of the program. However, with respect to automated testing processes and user interfaces, these documents fall short of being viable options to express correct program/GUI behaviour. While they are readily understood by humans, a computer would have difficulty with such a specification. While it is relatively simple to create a method for the computer to read in or parse the specified behaviour, the difficulty arises when the computer attempts to use said behaviour in order to perform an automated task, such as test case generation. It must have some prior knowledge of what each type of behaviour means in order to piece them together to get the behaviour of the whole GUI or part of it. This brings to mind the question: What kind of GUI specification language will provide system engineers, software developers, and testers with the information they require for their work, but will also enable automation of the testing process to occur?

The following are some of the factors that must be taken into consideration in choosing a specification notation for GUI software:

- How can the language be simplified to describe only what is needed? Meaning the language should not describe every detail possible about the GUI, but rather abstract out those pieces of information that the loss of which will not degrade the effectiveness of the language itself in accomplishing the rest of the factors mentioned here.
- How will it handle GUI actions/events?
- How will it show behavioural relationships between GUI components?
- How can it be used to generate test cases?
- How can it be used to generate test oracles?
- How can it be used to generate mockup GUIs?

1.1 Motivation

Graphical user interfaces are essential to software engineering today, as they are the first thing a user will see when starting an application. Thus, ensuring that these GUIs behave accordingly, and without error, is of the utmost importance to the software companies. If a GUI is not tested, with the purposes of determining if any faults exist, then it cannot have any claim of operating properly.

To properly test the GUI, one must first create test cases for it and run these against the application itself. But how will the tester know what is correct behaviour? The answer to this is simple: a proper specification for the GUI. By having such a specification, it is possible to not only have a method of comparison for allowed behaviour, but it could also be used to generate the test cases and oracles automatically. However, the specification must not over-specify the GUI or be so heavyweight that it is difficult to use. Over-specifying involves including too many aspects of the individual GUI components that have no effect on the required behaviour. Required behaviour changes depending on the application, thus it is up to the specifier to determine what should be included within the specification. An example would be to specify the size of a push button whose required behaviour is to open a dialog when pressed/released. The size of the button has no bearing on the behaviour itself, and thus does not need to be included within the button's specification.

1.2 Research Goals

A goal of this research is to develop or find a GUI specification notation that can be brought to a level where it is possible to accomplish all of the following:

- Automatically generating test cases
- Automatically generating oracles
- Generate mock-up GUIs

In order to do this, the specification must be able to effectively specify the interactions between the user and the GUI components, between GUI components themselves, and how the GUI will appear.

To restrict this research to feasible limits, the GUI being specified is restricted in the following ways:

- Only standard widget based GUIs are investigated.
- No visualization controls or components, such as 2-D or 3-D graphics, are looked at.
- Many complex GUI components, such as Menus, Scroll Bars, etc, are ignored. It is not that these can never be included, but that they are deemed advanced components, and thus, not part of the base language capabilities.
- Only statically allocated lists, whose contents are known ahead of time, are allowed. This ignores all lists that are generated and updated during the execution of the application. The reason for this is to simplify the ability to generate test cases, perform oracles, and generate mock-ups.
- Only those events that are provided in the specification are looked at during test case generation and generation/application of the oracle.
- The programming language used for manipulating the GUI specifications to perform the other requirements of this research is Java.

1.3 Outline

The following chapters describe the goals, research, implementation, and accomplishments of this thesis:

- Chapter two - Related research.
- Chapter three - Methodology used to accomplish the goals of this thesis.
- Chapters four to seven - Discussion of the prototype tools developed.
- Chapter eight - Examination of an example application using the tools and concepts developed.
- Chapter nine - Conclusions and future work.

Chapter 2

Related Work

This chapter reviews some of the methods used to specify and test GUIs that have been found during the course of this research. To mention all related research within the areas defined in this chapter would be difficult due to the sheer volume. Instead, we focus on a subset of related work to better discuss the differences and similarities to our own.

2.1 GUI Ripper, GUITAR, DART, and PATHS

Of particular note, is work by Memon et al. [1–8], who have done extensive work over the years in all aspects of the testing process. Emphasis has been placed on the different automation tools developed to simplify the amount of time required to perform each step. In [7, 8] an event-flow model is discussed, which is used to model the behaviour of a GUI. Events are modeled as operators, that take advantage of the hierarchical nature of a GUI's structure, where preconditions are matched with effects. Preconditions represent what state the GUI must be in for the event to occur, while effects represent the state changes made to the GUI after the event has been applied. These operators allow for a graph to be developed and used for different purposes. In [7] a tool known as GUI Ripper is introduced. Its purpose is to automatically reverse engineer an application by traversing its GUI, opening each window in turn, and saving its information, including controls and properties, as well as also saving extra information about events. Any windows missed by the tool are added in manually by a test designer. Both sets of information are used to generate the event-flow graphs required. They can then be used to automatically generate test cases. The first implementation of it is designed around Microsoft Windows based GUIs, however, there is also a Java Ripper tool, designed to perform the same tasks in a Java application. The model provided by the GUI Ripper is then made use of by two other tools credited to Memon's research:

- GUITAR - GUI testing framework used to automate the process.
- Daily Automated Regression Tester (DART) - Performs nightly testing on GUI applications.

In [4], Memon et al. discuss the automatic test case generation capabilities of their Planning Assisted Tester for grapHical user interface Systems (PATHS) tool. This tool uses the event-flow modeling graphs mentioned above to actually generate the test cases, a set of initial and goal states are provided to the PATHS tool, which already contains the graph(s). The tool then uses AI path generating algorithms in order to determine the set of hierarchical plans, or event sequences, which represent the tests to be applied to the AUT. In [2] Memon et al delve into oracle generation from their modeling language using PATHS, while in [6] the best form of applying said oracles is discussed.

Memon's work is quite extensive, and covers the majority of the work we will be undertaking within this research:

- GUI modeling
- Test case generation
- Oracles

What we take from this are the ideas:

1. That modeling a GUI requires placing pre- and postconditions on GUI properties, with respect to event behaviour.
2. That making use of the GUI hierarchy can help when automating processes.
3. Of combining the majority of the tools or processes within the same application. This can also be called an integrated environment.

Our research, however, differs from some of the concepts provided above. While the modeling of a GUI in Memon et al's work uses pre- and postconditions for events, our specification language will not contain these directly. Instead it will be the responsibility of the test case generation process to ensure events are only applied in situations where they make sense. As well, our research does not use ripping to generate the specification for a GUI. Instead, it centers on the idea that the specification of a GUI should come first so that all behaviour can be finalized before any implementation is done. Thus, the specification is a manual process, rather than the GUI Ripper, which is mostly automatic.

2.2 FSM Based Work

It is a very common practice within industry to model a GUI as a finite-state machine (FSM), sometimes in its directed graph form due to the event-based nature of GUIs. We only examine a subset of FSM approaches here.

In [9], Campbell and Veanes describe a state exploration algorithm that uses multiple state grouping functions to group related states within a FSM model, for the purpose of reducing the state space. The reason for this is to help guide the algorithm to generate tests for states that are interesting. The Spec Explorer tool is used to demonstrate use of the algorithm.

In [10], Chow describes how test cases can be generated from software modeled as an FSM by use of automata theory, and then be applied against the FSM to verify responses. The testing strategy investigated has several characteristics:

1. the FSM of the control structure is the only part tested
2. an 'executable' prototype is not required
3. any test cases generated are guaranteed to find faults in the control structure, given the following assumptions about the control structure FSM:
    - it is completely specified,
    - it is minimal,
    - its initial start state is fixed,
    - every state is reachable.

Shehady and Siewiorek [11] describe modeling UIs with Variable Finite-State Machines (VFSM), which are designed to provide more advanced transition and output functions to describe the UI properly. The 'Variable' portion of the name derives from the use of global variables, that are defined for a VFSM, and are modified a finite number of times by the transitions made in response to a test input sequence. The global variables help determine what the next state in the transition should be, as a single variable can allow or deny different transitions based on its current value. A VFSM can be converted to a FSM for test cases to be generated using the partial W method, also called the W_p method. This process yields fewer states than a direct translation from the UI to a FSM.

In [12] Stirewalt discusses translation of specifications written in the MASTERMIND Dialogue Language (MDL) into an interpreter by way of Concurrent State Machines (CSM). The authors make use of Symbolic Model Verifier (SMV) to perform model checking. They also used SMV to verify different strategies for intermachine communication between the CSMs, as trade-off issues existed between strategies. The communication method chosen was a combination of prioritized scheduling and peeking at parent states.

In [13] Belli makes use of deterministic Finite State Automata (FSA) and regular expressions to accomplish the tasks of testing the GUI modeled as an FSM. It is assumed in Belli's approach that the application should be tolerant of incorrect user interactions, as in there are no user errors. Instead, faults are grouped into two categories:

1. Specification - The behaviour in the specification is wrong.
2. Implementation - The implemented behaviour does not match the expected behaviour from the specification.

Paiva et al. discuss a method which makes use of Hierarchical Finite-State Machines (HFSM) [14]. First the GUI is modeled in Spec#, then converted to a FSM using the Spec Explorer tool. The HFSM is created from this FSM, and is used to reduce the size by exploiting knowledge of hierarchical GUIs. The reduced FSM is then used as input to the Spec Explorer to generate testing procedures.

In [15] White models the GUI as a FSM, then generates tests from it based on Complete Interaction Sequences (CIS), which are sequences of user interactions on GUI objects and cooperative methods. Reductions to the complexity of the FSM, while the CIS is being applied to it, were made where feasible.

Apfelbaum and Schroeder describe a behavioural model of GUIs based on state-machines [16], where events and responses are state transitions, and windows are either states, if simplistic, or entire sub-models if they are complex. Goals to be achieved by the testing, as well as usage patterns of how the software is normally operated should be determined ahead of time, to help with generating appropriate tests.

Modeling a GUI as a directional FSM, where the edges of the state-machine contain information about the event, is key to developing test cases for our research. This will be described in more detail within chapter 5.

2.3 JUnit Based Approaches

In [17] Erickson et al. introduce the tool Haste, built upon the ideas of JUnit, and designed for general system testing of software. While the concepts used by Haste are language independent, the first iteration is in Java. There are three main elements of Haste:

- The testing framework used by Haste: Story (Single test), Step (Part of a Story) and StoryBook (Suite of Story objects).

- Narcitecture - How Haste gains internal state information from GUI objects, and represents the test code executed within the same address space as the AUT.

- Pilots and Droids - Pilots simplify access to complex GUI objects, while Droids allow the application to be manipulated through programmatic calls.

References are made within the article to other GUI testing toolkits, such as JFCUnit [18], Jemmy [19], Robot [20], Abbot [21], and Marathon [22], that are mainly used for manipulating the GUI under test.

The idea of using a toolkit in order to manipulate a GUI during runtime is extremely important for our research into oracles. The reason for this is that it will provide a means by which to apply any action, from the current test case, against the GUI under test. As well, placing test code within the AUT itself, in order to leverage access to information and for GUI manipulation is key to some of our research. These concepts will be described in more detail in chapter 6.

While the above ideas are similar between our research and that of Haste, there are also many differences. The StoryBook, Story, and Step objects are Java classes extending similar JUnit classes. Haste makes use of these to create test sets to be applied to the AUT based upon requirements. In contrast, our test cases are written using elements of the specification language we created, and are generated from a specification of the GUI. Thus, the test cases exist external to the Java code developed to run them against the AUT.

2.4 Formal Methods

In [23] Peters and Parnas describe how software documented by way of tabular notations may be used to generate test oracles, which would then present results when run against the software application in question. Documentation using tabular expressions can clearly denote intended software behaviour. Peters et al. have also developed a prototype Eclipse plugin to make it possible to write formal specifications, using tabular expressions as a base, and leveraging new technologies, within the Eclipse IDE [24].

Weyuker describes, in [25], various methods to determine if a program is displaying correct behaviour when no oracle is assumed to exist. Such programs fall into one of three categories:

1. Programs written to determine the answer to a problem.

2. Programs generating too much output to verify easily.

3. Programs where the tester has a misconception about correct behaviour.

For the first two types of programs, there are several possible ways to attempt verification of correct behaviour. One is to use a pseudo-oracle, which is a program written in conjunction with the original, but by an independent programmer, that is designed to meet the same specification. The same test data is then provided to both programs and a comparison of results is performed. If there is a difference, then it warrants a check of how the difference occurred. The second method, is to use a simpler set of data for testing, then extrapolate out to more complex data. The third method is to accept plausible results even though they may be wrong. The problems existing with these techniques are that pseudo-oracles are not always practical, depending on the program, simpler data may behave correctly, but the more complex data is usually more error-prone, and lastly, accepting plausible results can allow incorrect results through.

For the third type of program, problems with misconceptions can be mitigated by using one or more testers that are not part of the development team. Another way to avoid this problem, is to use better specifications that are put in place before implementation occurs.

Automated oracles are not always required during testing, depending on the method of testing being employed and the application being tested. In our research, however, oracles are a necessary part of the testing process as they determine whether or not the application being tested conforms to the originally specified behaviour. Typically they are generated separately from the test cases, however, our research will differ from this in that our generated test cases will also contain our generated oracles. Similar to Weyuker's research, the idea of the specification being developed before implementation occurs is desirable. Please see chapter 6 for more details.

2.5 XML Based Work

In recent years, XML [26, 27], or eXtensible Markup Language, has emerged as a language that may be used for specifying or modeling UIs. As well, in many cases, the purpose of the XML-based language is to simply define the UI for an application or website for display purposes, rather than to determine correct behaviour.

In [28] Bishop and Horspool present the VIEWS system, which has a XML specification language for modeling the GUI and a layer to handle dynamic GUI behaviour. The event handling is handled separately using either event loop capabilities or callbacks. In [29] Bishop discusses the merits of using Mirrors, a C# .NET toolkit that makes heavy use of reflection [30] to generate a GUI from an XML specification. Reflection is a mechanism for a program to determine the types and properties associated with objects (e.g., controls and forms) at runtime, and to invoke events against them. This allows the specification to remain independent of the implementation and platform.

Xin describes the development of a GUI Toolkit based on XML specifications [31]. The specification has both static behaviour, in terms of widget definitions and layout, as well as dynamic behaviour, events, and transformation by way of two XSLT documents that transform the XML specification into XHTML or Java.

Abdel Salam et al. [32,33] discuss the development of an XML based specification language geared for testing Java Swing applications, along with tools to generate the test scripts, generate application code to run the test scripts against the AUT and log expected results, and finally a visual editor to generate a specification containing only the GUI structure from an existing AUT.

In [34] Tubishat et al. describe using XML-based language files to store the current state of a particular GUI, for use with certain actions such as undo, redo, etc, and for comparison with other saved states to determine if regression testing is required. By limiting the amount of information stored to these files, it can reduce the number of possible states for a GUI, and decrease complexity of processing them.

XAML, or eXtensible Application Markup Language, is Microsoft's XML application that can be used to specify the UI for .NET applications, by providing a means to describe the interface components separately from the actual application code [35, 36]. This means both programmers and designers can work on the same application, at the same time, with limited interference with one another. XAML has multiple assemblies, which contain the base object definitions for the language, which map directly to .NET objects.

eFace is a Java solution based on Microsoft's XAML/WPF [37]. The code used for eFace, UPF (Universal Presentation Framework) is compatible with WPF.

XUL, or XML User Interface Language, is Mozilla's XML based UI specification language designed to build cross platform applications [38]. It is used extensively within Firefox, which is Mozilla's web browser. It makes use of several technologies:

- XBL - Provides bindings for XUL objects, allowing extensions to define new content, event handlers, as well as new properties and attributes.
- Overlays - Describe extra capabilities for XUL applications for customization or extension of existing XUL code.
- XPCOM/XPConnect - Allows XUL applications to interface with external libraries.

XMLTalk is an XML based specification language, that is used to specify user interfaces for Java/Swing applications [39, 40]. It leverages key components of both the Java and SmallTalk languages to be able to automatically generate the UI using the Model-View-Controller pattern. The part that makes XMLTalk different, is that it specifies the Model side of the application, as well as the View.

XForms [41] a W3C funded development of forms technology for web based applications based off XML.

XIML is an XML language for universal specifications of interaction and knowledge data for user interfaces [42]. Some of its capabilities are to enable a single interaction specification to be loaded on multiple platforms, as well as to provide a means by which to store interaction data from a user interface or set of user interfaces.

Most of these languages are geared towards the render or display of the UI during runtime, and thus describe the look, as well as the dynamic event behaviour, of the UI components. This is not what we are attempting to accomplish with our own research. Instead, any specification language described in the subsequent chapters should simply define the GUI and its event behaviour, but have no built-in dynamic behaviour capabilities nor the ability to automatically render the GUI during execution.

As well, the XIML specification language, while providing a mechanism to define a user interface, does not itself provide a means by which the UI is rendered or displayed on the target machine's screen. While our own research will also attempt to not specify exactly how a GUI should be rendered to screen, it will provide some details as to how it should look and feel.

Abdel Salam et al's work lies in a similar realm of research to part of what we are attempting to accomplish here. However, the following differences exist:

- Our research is based on the idea of starting with a specification, and then the AUT is implemented from this.
- The structure of our language will be more detailed with respect to the controls.
- Our research will not use automatically generated scripts/code to apply test cases and handle their results.

Tubishat et al's research [34] is similar to ours in that our work will also attempt to reduce the amount of information that is saved for our specified controls. However, the purpose for doing this is different, as our research is geared towards specification of the GUI, as well as automating the testing process.

All of the aforementioned differences will be covered more fully in chapters 4 and 5.

Chapter 3

Methodology

As stated in the introduction, the main goals of this research include:

- Development of a specification language.
- Automatic generation of test cases based on the specification language.
- Automatic generation/application of oracles using the generated test cases.
- Automatic generation of mock-up GUIs based on the specification.

All tool development for this research will be in the Java programming language. The reasons for this are that it is:

- a high level, object-oriented language,
- easily portable,
- widely used within industry.

The specification language developed must be light-weight, easily parsed by Java, and easily extensible if required. It is desired that the language should have components available to be able to describe test cases as well. This will allow common capabilities to be shared between the two sides. For the purposes of this research, the language itself is a prototype or example language. Not all capabilities of a GUI are covered, and limitations are put in place to reduce the complexity of processing the language.

The most important question to answer here is whether or not our prototype language is good enough. The following are some of the language's attributes:

1. That it can be used to generate test cases that actually expose problems in implemented Applications Under Test (AUTs).

2. That it is descriptive enough to allow a mock-up of it to be automatically generated, for checking validity of specified layout and appearance.

Test case generation is a method whereby the specification is taken as input and the result is a set of test cases. The test case generation process for this research uses a graph in order to get from input to output. Limitations are put in place to ensure that the complexity of the graph is reduced. Graphs, or models of GUIs, can have what is called a state space explosion problem, as mentioned in [9], [14], [43], and [44], in relation to FSMs. It means that the number of possible states that can exist within a GUI may be quite large, or even infinite, thus increasing the time and memory required to process its graph representation into a usable form for testing. As well, in [45], Ganov et al. discuss placing constraints on some event sequences and minimizing the set of usable event sequences in order to reduce the set of tests to work with. This article references an issue with text input, where it significantly increases the number of test sequences, as text can have a large number of possible combinations. The limitations mentioned above are implemented by selecting arbitrary sequences of events to apply, thus reducing the number of states that the graph contains.

Generation of oracles involves determining what the expected states of the GUI are based on the specification of it. Application of oracles is a pass/fail process, where the test cases generated are applied to an **A**pplication **U**nder **T**est (AUT). The results of applying each test case to the AUT are captured and compared against the generated oracle, giving a pass or fail designation depending. For this research, the oracle will take in both a set of test cases and an executable of the AUT itself. Each test case is then run against the AUT in order.

The comparison between actual result and expected result should occur after each step of the test case, instead of at the end of the test case. This will allow for more information when a failure is found. The expected result is taken from the generated oracle, that is part of the test case, and comprises of the combination of all control and container states at that particular point within the test case. The actual result is the combination of control and container states taken from the AUT at the same point within the test case. A state is the value of all properties for a control or container, and differs based on the type of that control or container. The properties for these are defined by the developed prototype language mentioned above. For example, if the test case calls for a check-box to be checked, but in the AUT itself the check-box is not checked, then the comparison between the two results will fail.

The purpose of mock-up GUIs, with respect to this research, is to simply provide a way of giving the specification writer a general idea of what the resulting GUI would look like. However, limitations will exist based on the specification language itself, which is only a prototype. For example, while the mock-up will be able to let the specifier know where a button should possibly be located within its container, it will not provide any way to denote colour, size, etc.

The above descriptions are further explained within the subsequent chapters.

Chapter 4

TUIL

TUIL, or **T**estable **U**ser **I**nterface **L**anguage, is an application of XML that specifies the behaviour of the UI of a software program, in a form that can be used for automating the testing process and generating mock-up GUIs to confirm the proper layout of controls within each container.

4.1 Why Use XML?

XML is mainly used to store and transfer data between different applications. The reasons why this language was chosen as a basis for TUIL are given below [31].

- It provides us with a way to define our own elements and types, that make use of XML's built in constructs and data types.
- The language can be defined within XML Schema files, that can be used to validate any XML documents based on them.
- XML is an open standard, it is platform independent, and supported by a

multitude of different programming languages and applications. This is especially important to GUI specifications because this allows the specification to be ported to different programming languages for processing and usage, rather than being tied down to a single language.

4.2 How TUIL is Different

As mentioned in section 2.5, there are many XML based languages used to describe user interfaces that currently exist within industry or research communities. Some of these include:

- XAML [35, 36] - XAML, or eXtensible Application Markup Language, is Microsoft's XML application that can be used to specify the UI for .NET applications by providing a means to describe the interface components separately from the actual .NET application code.
- eFace [37] - An XAML solution designed for Java applications.
- XUL [38] - Mozilla's XML based application UI specification language.
- XMLTalk [39,40] - An XML based specification language, that is used to specify user interfaces for Java/Swing applications.
- xForms [41] - A W3C funded development of forms technology for web based applications based off XML.
- GUI Toolkit [31] - An XML based GUI toolkit for development of GUIs. It displays these using the Java platform.

- VIEWS [28] - Has an XML specification language for modeling the GUI and a layer to handle dynamic GUI behaviour.
- XIML [42] - Specifies the user interface and any relations that exist between components in XML. One of XIML's goals is to have one specification that can be displayed in multiple platforms and devices.

TUIL differs from the above languages in the following ways:

- It does not provide a means to automatically render the GUI and the defined event behaviour.
- Unlike XIML, TUIL is not capable of dynamic display and management of the interface.
- TUIL does not contain dynamic event behaviour, but rather specifies what is correct behaviour for the application. Only when the application is implemented will the event behaviour actually work.
- Not only does the language provide a way to describe a GUI, but it can also describe test cases for a GUI.

A GUI is a state based system, that can move from one state to another by way of an event. Events will be covered in a later section, but Figure 4.1 shows an example GUI state graph. It is implied in this state graph that each button must be visible and enabled for each event to occur. For Button B to be clicked by a mouse, it must be visible and enabled within State A. This is similar for the other buttons and State B.

Due to the fact that a GUI is inherently state based, TUIL must be able to both specify it, as well as describe the current state. A state contains properties

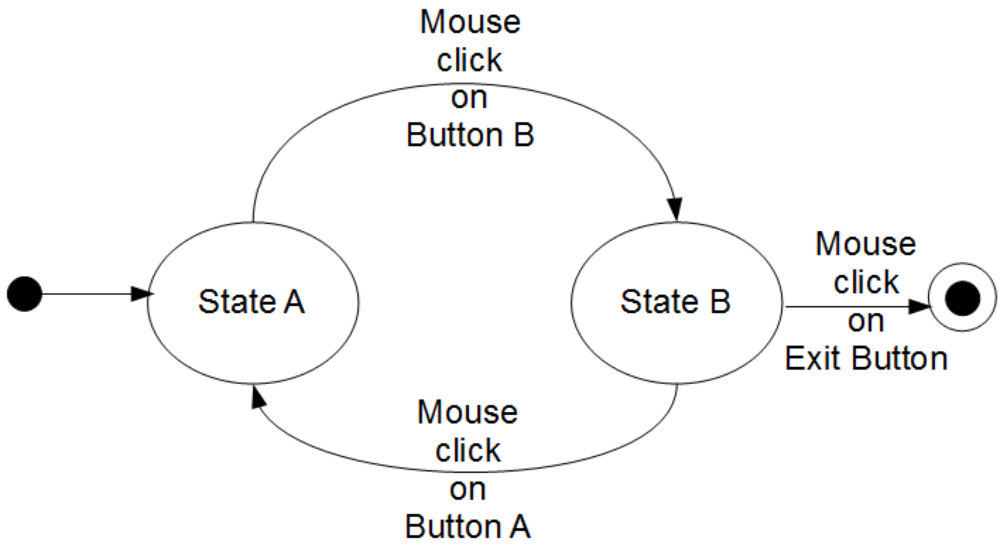


Figure 4.1: An Example GUI State Graph

that directly link back to some visual representation of the GUI component they are associated with. As such, different properties are required based on the component.

If we think of the state in this manner, we can divide the properties into two sets:

- Common visual properties represent those that are common across all GUI components. These include, but are not limited to, layout, size, location, and whether the component is visible.
- Component specific properties represent those that are specific to a particular GUI component or a set of components. For example, radio buttons and check-boxes have a property that denotes whether they are selected. As well, a text box may appear disabled when its read only property is turned on, and prevent a user from entering text.

The component specific properties can be used in two important ways within a GUI application:

1. To be saved as the application's configured settings, either in their raw form or as a more usable form for the application. These can be loaded again the next time the application starts.

2. To change the state of other GUI components. For example, selecting a check-box that in turn enables a text box for user input.

Common visual properties such as layout, whether the component is enabled, has focus, and is visible are required by our language. Layout allows us to orient the components within containers, and will be essential for mock-up creation, as discussed in chapter 7. Enabled, focused, and visibility can have an effect on event processing during test case generation, which is further discussed in chapter 5.

Properties, such as colour, location, size, shape, font, etc, are not included in the TUIL language, even though they are used to define the look-and-feel of the GUI. These properties affect how the GUI looks, however, they do not directly affect how events would be processed nor how mock-ups are generated. As well, these properties will bloat the TUIL language and defeat its original purpose of being simplified and light-weight, with as much abstracted out as possible. To this end, properties such as these are left to the discretion of the implementing programmer to define.

The following lists the GUI components currently specified by TUIL with reference to the most important of each one's component specific properties. Emphasis is placed on those properties that relate to the two ways they can be used within the GUI. Other properties may exist, depending on the control, however these relate more to display and layout, and are not discussed here. Please see Table 4.1 for the associated images of each one:

- Push Button - In its typical form, which TUIL uses, this component actually has only visual properties, with the ability to fire events. The simplest way to determine if performing an action on the button actually occurred, is to check whether the specified event fired.

- Radio Button - The specific property for a radio button is whether the component is selected or not.

- Radio Button Group - A radio button group is a grouping of radio buttons either in a vertical or horizontal orientation, where only one of them can be selected at a time. The radio button group is not a control within the TUIL language, but rather it is denoted by an attribute of the button called a button group. This attribute is always empty for a push button. The orientation is determined by the layout mechanism that is discussed in a later section.

- Check-box Button - Similar to the radio button, a check-box's specific property is whether it is checked or not.

- Text Box - A text box accepts text entered by the user, though the form of the text may sometimes be tailored, as in, for example, only allowing numbers or expecting a password. Regardless of what the text box is used for, its most important specific properties are the current text contained within it, along with whether or not it is read only and/or multi-line.

- Label - A label is typically associated with another control, or is used to convey information to the user. This may either be statically defined text or text that is updated based on what the user is doing. As such, a label's specific property is defined by its currently contained text.

- Slider - A slider is a control that contains a range of possible values, and allows the user to select one of these. Each value is given an index, where the first index is zero. Thus, its specific property set contains the allowed range of values and the index of the currently selected value, along with the display orientation, which is either vertical or horizontal.

- Combo Box - A typical combo box contains a set of items, that are usually represented as strings, and displays the currently selected item. When the drop down is displayed, it will either show a subset of the full list or the full list, depending on the number of items. The specific properties for a combo-box are a combination of the set of items currently in the list and the index of the currently selected item.

- List Box - A list box is similar to a combo box, except that it will display as many of its item set as possible given its dimensions. Scroll bars can appear as necessary if the list is too long. There are two properties of a list-box that can affect its specific property set: whether it is multiple selection or single selection, and whether it has multiple columns or not. In its base form, the specific property set for a list-box is the same as the combo-box above. When multiple columns are included, the set of items in the list is changed to be the set of rows, that contain multiple values. When multiple selection is included, the currently selected item index is changed to the currently selected indexes.

- Group Box - The group box is meant to group controls with similar purpose together, so that they can be displayed in a logical manner to the user. Similar to the radio button group, however, it is not a true control in the same manner as the others. Instead its only specific property is a set of control ids denoting what controls are to be displayed within it. The ids will be discussed later in the chapter.

- Container - A container is the only component within the TUIL language that is allowed to have child controls. As such, its specifc property set is the combination of all the child control property set, plus a layout mechanism.

Based on the above, TUIL documents a combined view of the UI, bringing together the following two aspects:

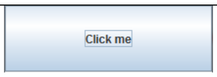
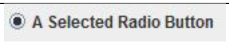
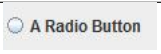
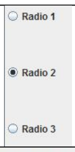
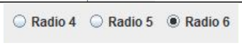
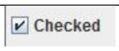
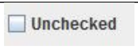
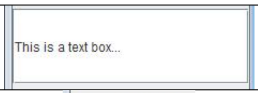
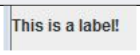
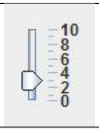
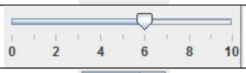
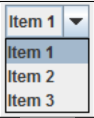
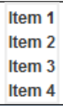
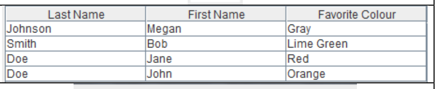
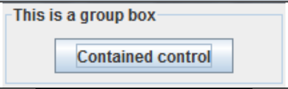
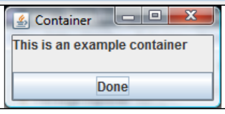
- GUI state - As defined previously, and abstracted as much as possible.
- Event Behaviour - Required to show interactions between different parts of the application. This is abstracted as much as possible. An event defines what happens when a GUI changes state, and is further discussed in sections 4.3.2 and 4.3.5.

4.3 The Schema Files

The general purpose of the TUIL specification language is to describe a Finite-State Machine. With this in mind, the language is structured as a way to define state nodes interconnected by state transitions. Please see the supplementary files for the full TUIL schema.

The language is broken out into several modules in order to group the Specification and Test Case aspects separately. They both, however, make use of common language

Table 4.1: Typical Control Images

Control	Image
Push Button	
Selected Radio Button	
Unselected Radio Button	
Vertical Radio Button Group	
Horizontal Radio Button Group	
Checked Checkbox	
Unchecked Checkbox	
Text Box	
Label	
Vertical Slider	
Horizontal Slider	
Combo Box	
List Box	
Multi-column List	
Group Box	
Container	 This is an example container

Last Name	First Name	Favorite Colour
Johnson	Megan	Gray
Smith	Bob	Lime Green
Doe	Jane	Red
Doe	John	Orange

components. The rest of this section describes key aspects of the TUIL language, and how they relate back to our FSM structure.

4.3.1 Unique Identification

The **gui_id_type** is a restriction on the base string type, where you cannot have whitespaces and there is a maximum of 64 characters. We use string as the base type so that the maximum number of unique possible values is increased. This type is used to differentiate GUI component objects and to aid with event processing. There are two uses for this type:

- **id** - The name of the current object.
- **parent_id** - The name of the current object's parent container object.

Within the Specification module the identifiers are used to create a tree or hierarchy of connections between UI components, to denote the parent to child relationships. At the root or top of this, we have the main container object for the application UI, which is denoted using a **parent_id** of 'NULL' or simply an empty string, meaning it does not have a parent container. All other containers and child controls will have their **parent_id** attribute filled out with the container that they are displayed in, or which owns them. The **id** attribute is always filled out for each component, and should be unique in the following ways:

- Each child control should have a unique identifier within its parent container. This is so that it can easily be distinguished from the other child controls within the same container. As well, this allows for the re-use of child ids within different parent containers, as the two controls are seen as unique.

- Each parent container should have a unique identifier within the entire specification. This is to distinguish different containers from one another so that it is easier to find them during test case generation and oracle application.

For example, to specify that a container owns a button, we would give the button an **id** of *'ButtonOne'* and a **parent_id** of *'ContainerOne'*. In turn, the container would then have an **id** value of *'ContainerOne'*.

The Test Case module uses only the **id** value, however, as it organizes the containers and control objects in a different way, thus negating the need for the **parent_id** to be used directly.

4.3.2 Shared Event Properties

There are some event elements and attributes that are shared by the Specification and Test Case modules. These are comprised of:

- Actions - Encompasses all actions available within the language, which are limited to the most commonly used actions. Please see section 4.3.5 where events are discussed in more detail.

- **event_object** - Denotes what object is currently being affected by the action or reaction (See section 4.3.5). It includes both the **id** and **parent_id** attributes, and allows the **id** attribute to be 'NULL' or empty string. Using one of these values means the component within which the event is defined is the one being affected.

4.3.3 TUIL's Container Object

tuil_container describes a container object, or an object that can contain child controls, and is the main element used to specify a UI. For the purpose of this research, a **tuil_container** will be restricted to the UI containers window/frame and dialog, since they are widget based, as discussed in section 1.2. To minimize the issues with finding what XML document contains what **tuil_container**, it is good practice to only specify one per file, and name the file accordingly.

Most container objects will have child controls, such as buttons, combo boxes, etc., and even though for the TUIL language the visual aspects of the UI components have been abstracted out as much as possible, we still need a way to denote control layout within the container. To account for this, we use the **children** element, which is of the **child_group_type**. It groups **child** elements in a particular **orientation**, either vertically or horizontally. However, these groups can also be nested within each other to allow for more advanced layouts. By doing this, we can ignore such things as extent and location. For an example of how the layout works on the TUIL specification side, see Listing 4.1. To see what this specification would look like as an actual dialog, please see Figure 4.2.

Listing 4.1: TUIL Specification of an Example Dialog

```
<?xml version="1.0" encoding="UTF-8"?>
<tuil_container id="TUILExampleDialog" parent_id="NULL" type="dialog"
    is_modal="true" use_default_button_placement="true">
    <is_visible>true</is_visible>
    <is_enabled>true</is_enabled>
    <has_focus>true</has_focus>
    <text>TUIL Example Dialog</text>
    <children orientation="vertical">
        <child_group orientation="horizontal">
            <button id="checkThis" parent_id="TUILExampleDialog"
                is_default="false">
                <is_visible>true</is_visible>
                <is_enabled>true</is_enabled>
                <has_focus>true</has_focus>
                <text>Check this</text>
                <type>checkbox</type>
                <is_checked>true</is_checked>
            </button>
            <child_group orientation="vertical">
                <button id="checkFirst"
                    parent_id="TUILExampleDialog" is_default="false">
                    <is_visible>true</is_visible>
                    <is_enabled>true</is_enabled>
                    <has_focus>false</has_focus>
                    <text>Check this first!</text>
                    <type>checkbox</type>
                    <is_checked>false</is_checked>
                </button>
                <button id="checkSecond"
                    parent_id="TUILExampleDialog"   is_default="false">
                    <is_visible>true</is_visible>
                    <is_enabled>true</is_enabled>
                    <has_focus>false</has_focus>
                    <text>Check this second!</text>
                    <type>checkbox</type>
                    <is_checked>false</is_checked>
                </button>
            </child_group>
        </child_group>
        <button id="ID_OK" parent_id="TUILExampleDialog"
            is_default="true">
            <is_visible>true</is_visible>
            <is_enabled>true</is_enabled>
            <has_focus>false</has_focus>
            <text>OK</text>
            <type>push</type>
        </button>
    </children>
</tuil_container>
```

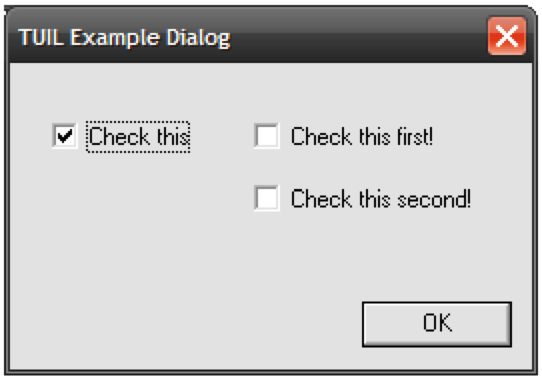


Figure 4.2: TUIL Example Dialog

4.3.4 TUIL's Base Object

tuil_elem is used to describe the most common attributes of all UI components, including the **tuil_container**. In Listing 4.1, the following XML elements are all part of the base element type:

- **is_visible** - Whether a particular component is visible or not.
- **is_enabled** - Denotes whether a component is enabled or disabled. Not as applicable for **tuil_container**.
- **has_focus** - Is true for whichever component has the focus. With respect to the **tuil_container** components, it defines the container that currently has focus or that contains the currently focused control. This can become important when multiple containers are open at the same time.
- **text** - Describes the textual information that is displayed, whether this is a title, with respect to containers, or the text displayed in a push button. For text boxes and labels this actually a control specific property.

4.3.5 Specifying Event Behaviour

Event behaviour is very important for any language that attempts to specify a GUI. But what is an event? In terms of a GUI, it is what causes the GUI to move from one state to another. In [46], Lee describes an event as actions performed on GUI components, such as clicking a button on the screen or pressing a key, which the software should respond to. The GUI code catches these events and ensures that the required behaviour occurs.

However, for the purposes of the TUIL language, this type of event is not descriptive enough. How can we define GUI behaviour if the only thing we know about an event is the action that causes it to occur? To do this properly, we also need to know what the software response is with respect to the GUI components. Thus, we define an event differently by breaking it down into two parts:

- A single action - What occurs to cause the GUI to change state. An action can either be external or internal. External actions are initiated through mouse clicks, keyboard input, etc. Internal actions are generated from the application code, for example to update a control based on a timer. Application code is defined as any code within the application that is not part of the GUI code itself.

- One or more reactions - What happens to the GUI when the corresponding action is applied. There must always be at least one reaction defined for an event. The reason for this is that an event will only be used when its action can be successfully applied to the associated GUI component. This is described in more detail in section 5.3.

Unlike the XML languages mentioned in section 4.2, TUIL is designed to be lightweight, thus we do not provide the set of all possible actions and reactions that can occur within a GUI. Those included in the language can be combined into events that are of importance to the GUI in question, and most relevant to its behaviour. Thus, only those that are specified are included in the test case/oracle generation process. Those not specified are essentially ignored, and any behaviour for these is allowed. It is left up to the implementer, programming language, and operating system, within which the application is run, to define the behaviour for these ignored events.

It is recognized that by not specifying some events we are taking a risk of these events interfering with our defined behaviour, thus polluting the results when applying oracles. However, we have tried to mitigate this as much as possible, by providing a list of limitations for what types of programs we will cover, as per section 1.2. As well, our test cases/oracles will only encompass what events were specified, thus we will not be interacting with the GUI in any other way. It is more likely that internal actions from the application code would cause unknown behaviour.

In TUIL, an event is described using the **event** element, that contains one **action_pair** element and one or more **reaction_pair** elements. These are defined as follows:

- **action_pair**:
  - **action** element - Describes the type of action that is being applied.
  - **event_object** element - The GUI component the action is applied to.
- **reaction_pair**:
  - **reaction** element - Describes the type of reaction that should be applied.
  - **event_object** element - The GUI component the reaction should be applied to.

The **event_object** element, used by both **action_pair** and **reaction_pair**, provides the associated GUI component's unique identifiers, as discussed in section 4.3.2. For descriptions of the actions and reactions possible within TUIL, please see Appendix A.

To handle the instance where a **tuil_container** is updated by the application code, which was defined earlier, TUIL provides the following:

- **code_action** is how the **tuil_container** is either initialized or updated from the program code itself.
- **container_updated** is the result or reaction that occurs when a **tuil_container** is initialized or updated by the program code.

The original definition of event given earlier would describe a function similar to B below.

$$B : state \times event \rightarrow state'$$

The state mentioned in B will be discussed in further detail in chapter 5. Here it will simply encompass the state of the GUI.

Using the TUIL version of **event**, we can define **event** as follows:

$$\mathbf{event} = (\mathbf{action\_object}, (\mathbf{reaction\_object})^*)$$
$$\mathbf{action\_object} = (\mathbf{action}, \mathbf{event\_object})$$
$$\mathbf{reaction\_object} = (\mathbf{reaction}, \mathbf{event\_object})$$

We can rearrange function B above in to the form of a Mealy machine [47], $B1$, using our **action_object** and one or more **reaction_object**.

$$B1 : state \times \mathbf{action\_object} \rightarrow state \times (\mathbf{reaction\_object})^*$$

An example of an external event would be clicking the OK button on a message pop-up dialog, causing the pop-up to close. This would be defined, using function $B1$ and our definition of event, as:

$$state\ s1 = dialog\ displayed$$
$$event = (Ok\ button\ clicked,\ (dialog\ closes))$$
$$B1(s1,\ Ok\ button\ clicked) = s2, \text{ where } state\ s2 = (s1,\ dialog\ closes)$$

Listing 4.2 gives the specification of the TUIL Example Dialog again, but this time with the **event** elements specified as well. The FSM for the specification is given in Figure 4.3.

Listing 4.2: TUIL Specification of an Example Dialog with Events Provided

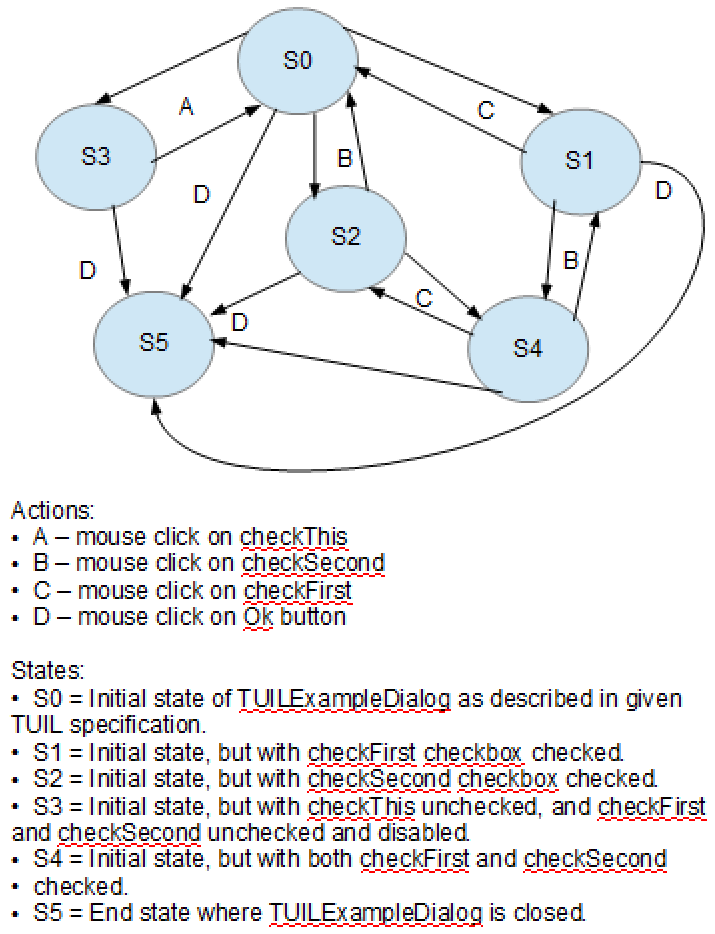
```
<?xml version="1.0" encoding="UTF-8"?>
<tuil_container id="TUILExampleDialog" parent_id="NULL" type="dialog"
    is_modal="true" use_default_button_placement="true">
    <is_visible>true</is_visible>
    <is_enabled>true</is_enabled>
    <has_focus>true</has_focus>
    <text>TUIL Example Dialog</text>
    <children orientation="vertical">
        <child_group orientation="horizontal">
            <button id="checkThis" parent_id="TUILExampleDialog"
                is_default="false">
                <is_visible>true</is_visible>
                <is_enabled>true</is_enabled>
                <has_focus>true</has_focus>
                <text>Check this</text>
                <type>checkbox</type>
                <is_checked>true</is_checked>
                <event>
                    <action_pair>
                        <action>
                            <mouse_action>mouse_click</mouse_action>
                        </action>
                        <event_object id="NULL" parent_id="TUILExampleDialog"/>
                    </action_pair>
                    <reaction_pair>
                        <reaction>
                            <button_reaction>toggle_check</button_reaction>
                        </reaction>
                        <event_object id="NULL" parent_id="TUILExampleDialog"/>
                    </reaction_pair>
                </event>
                <event>
                    <action_pair>
                        <action>
                            <button_action>toggle_check_on</button_action>
                        </action>
                        <event_object id="NULL" parent_id="TUILExampleDialog"/>
                    </action_pair>
                    <reaction_pair>
                        <reaction>
                            <base_reaction>enable</base_reaction>
                        </reaction>
                        <event_object id="checkFirst" parent_id="TUILExampleDialog"/>
                    </reaction_pair>
                    <reaction_pair>
                        <reaction>
                            <base_reaction>enable</base_reaction>
                        </reaction>
                        <event_object id="checkSecond" parent_id="TUILExampleDialog"/>
                    </reaction_pair>
                </event>
                <event>
                    <action_pair>
                        <action>
                            <button_action>toggle_check_off</button_action>
                        </action>
                        <event_object id="NULL" parent_id="TUILExampleDialog"/>
                    </action_pair>
                    <reaction_pair>
                        <reaction>
                            <base_reaction>disable</base_reaction>
                        </reaction>
                        <event_object id="checkFirst" parent_id="TUILExampleDialog"/>
                    </reaction_pair>
                    <reaction_pair>
                        <reaction>
                            <base_reaction>disable</base_reaction>
                        </reaction>
                        <event_object id="checkSecond" parent_id="TUILExampleDialog"/>
                    </reaction_pair>
                </event>
            </button>
            <child_group orientation="vertical">
                <button id="checkFirst" parent_id="TUILExampleDialog"
                    is_default="false">
                    <is_visible>true</is_visible>
                    <is_enabled>true</is_enabled>
                    <has_focus>false</has_focus>
                    <text>Check this first!</text>
                    <type>checkbox</type>
                    <is_checked>false</is_checked>
                    <event>
                        <action_pair>
                            <action>
                                <mouse_action>mouse_click</mouse_action>
                            </action>
                            <event_object id="NULL" parent_id="TUILExampleDialog"/>
                        </action_pair>
                        <reaction_pair>
                            <reaction>
                                <button_reaction>toggle_check</button_reaction>
                            </reaction>
                            <event_object id="NULL" parent_id="TUILExampleDialog"/>
                        </reaction_pair>
                    </event>
                </button>
                <button id="checkSecond" parent_id="TUILExampleDialog"
                    is_default="false">
                    <is_visible>true</is_visible>
                    <is_enabled>true</is_enabled>
                    <has_focus>false</has_focus>
                    <text>Check this second!</text>
                    <type>checkbox</type>
                    <is_checked>false</is_checked>
                    <event>
                        <action_pair>
                         <action>
                          <mouse_action>mouse_click</mouse_action>
                         </action>
                         <event_object id="NULL" parent_id="TUILExampleDialog"/>
                        </action_pair>
                        <reaction_pair>
```

```
                         <reaction>
                          <button_reaction>toggle_check</button_reaction>
                         </reaction>
                         <event_object id="NULL" parent_id="TUILExampleDialog"/>
                        </reaction_pair>
                    </event>
                </button>
            </child_group>
        </child_group>
        <button id="ID_OK" parent_id="TUILExampleDialog"
            is_default="true">
            <is_visible>true</is_visible>
            <is_enabled>true</is_enabled>
            <has_focus>false</has_focus>
            <text>OK</text>
            <type>push</type>
            <event>
                <action_pair>
                    <action>
                        <mouse_action>mouse_click</mouse_action>
                    </action>
                    <event_object id="NULL" parent_id="TUILExampleDialog"/>
                </action_pair>
                <reaction_pair>
                    <reaction>
                        <container_reaction>close_container</container_reaction>
                    </reaction>
                    <event_object id="TUILExampleDialog" parent_id="NULL"/>
                </reaction_pair>
            </event>
        </button>
    </children>
    <event>
    <action_pair>
        <action>
            <code_action>initialize_container</code_action>
        </action>
        <event_object id="NULL" parent_id="NULL"/>
    </action_pair>
    <reaction_pair>
        <reaction>
            <container_reaction>container_updated</container_reaction>
        </reaction>
        <event_object id="NULL" parent_id="NULL"/>
    </reaction_pair>
</event>
</tuil_container>
```

4.3.6 Test Cases

As described earlier, the TUIL specification language is designed to describe a FSM. The test case structure used in TUIL is that of a single path through a FSM.

Typically test cases take the form of a set of transitions to apply to the AUT.



Actions:

- A – mouse click on checkThis
- B – mouse click on checkSecond
- C – mouse click on checkFirst
- D – mouse click on Ok button

States:

- S0 = Initial state of TUILExampleDialog as described in given TUIL specification.
- S1 = Initial state, but with checkFirst checkbox checked.
- S2 = Initial state, but with checkSecond checkbox checked.
- S3 = Initial state, but with checkThis unchecked, and checkFirst and checkSecond unchecked and disabled.
- S4 = Initial state, but with both checkFirst and checkSecond
- checked.
- S5 = End state where TUILExampleDialog is closed.

Figure 4.3: TUIL Example Dialog FSM

However, TUIL defines the test case differently, in that it combines the oracles into the test case as well. Thus, instead of a set of transitions, we have both a sequence of expected states and the transitions between those states, hence a path.

Earlier we defined the change of state using the TUIL event definition, and the idea of one action and a set of reactions. If we consider a path in these terms, then a path is a sequence of states and transitions between states, that proceeds from a start state and finishes in an end state. Each transition is a single transition of the

Mealy machine *B1*, as defined in section 4.3.5.

4.3.7 Test Case Components

The main element to look at for test cases is **tuil_test_case**, which represents a single path. The components that make up the **tuil_test_case**, and how they relate to the definitions already discussed are:

- **edge** - The path is made up of a sequence of these elements. Each one contains two **node** elements and a single **action_pair**. It represents a single transition between states.
- **node** - Each **edge** contains both a start and an end **node**. They represent the states involved in the transition, and can potentially be equal.
- **action_pair** - The action that causes the state transition.

A **tc_node** type represents the state of a GUI, and provides a Boolean attribute that is set to true only if the **tc_node** equals the initial start state of the AUT. This allows for proper comparisons during the application of oracles. Both the start and end nodes mentioned earlier are of this type. We define the current state of the GUI in the following way:

> *GUI State* is the combination of all available Container States, where the number of containers is variable.

Note that the *GUI State* can be empty if all containers have been closed, including the root container. This represents when the application itself has ended.

The container object used here is the **tc_container** type, which contains a few attributes and the set of **tc_child** elements. Similar to that of the GUI state, the current state of a container is defined as:

> *Container State* is the combination of all control states within the container.

The manner in which the **container** elements and **tc_child** elements are organized is different from when the objects were defined within the original TUIL specification. Here each **container** and **tc_child** is matched with their **id** attribute to make a pair. All pairs are grouped into sets. This is different in that, within a TUIL specification, container objects are placed in their own files, whereas here a **container** element is part of a set contained within the **tc_node** element type. As well, each **tuil_container** knows both its **parent_id** attribute and **id** attribute, whereas here it is only the **id** attribute that is known. The reason for this is that we are less interested in which is the main **container**, and instead are interested in the set of containers as a whole. This is based on the assumption, that all **container** elements have unique **id** attributes within the entire specification, as mentioned in the section 4.3.1.

For examples of what the XML for a test case may look like, we will use the TUIL example specification provided earlier in Listing 4.2. To increase the readability of the examples, some text has been removed.

Listing 4.3: TUIL Test Case Toggle Check Example XML

```
<?xml version="1.0" encoding="UTF-8"?>
<tuil_test_case>
    <path>
        <edge>
            <start root_node="true">
                <entry>
                    <id>TUILExampleDialog</id>
                    <container type="dialog" is_modal="true">
```

```
                        <is_visible>true</is_visible>
                        <is_enabled>true</is_enabled>
                        <has_focus>true</has_focus>
                        <text>TUIL Example Dialog</text>
                        <entry>
                            <id>checkThis</id>
                            <tc_button>
                                <is_visible>true</is_visible>
                                <is_enabled>true</is_enabled>
                                <has_focus>true</has_focus>
                                <text>Check this</text>
                                <type>checkbox</type>
                                <is_checked>true</is_checked>
                            </tc_button>
                        </entry>
                                .
                                .
                                .
                    </container>
                </entry>
            </start>
            <end>
                <entry>
                    <id>TUILExampleDialog</id>
                    <container type="dialog" is_modal="true">
                        <is_visible>true</is_visible>
                        <is_enabled>true</is_enabled>
                        <has_focus>true</has_focus>
                        <text>TUIL Example Dialog</text>
                        <entry>
                            <id>checkThis</id>
                            <tc_button>
                                <is_visible>true</is_visible>
                                <is_enabled>true</is_enabled>
                                <has_focus>true</has_focus>
                                <text>Check this</text>
                                <type>checkbox</type>
                                <is_checked>false</is_checked>
                            </tc_button>
                        </entry>
                                .
                                .
                                .
                    </container>
                </entry>
            </end>
            <action_pair>
                <action>
                    <mouse_action>mouse_click</mouse_action>
                </action>
                <event_object id="checkThis" parent_id="TUILExampleDialog" />
            </action_pair>
        </edge>
    </path>
</tuil_test_case>
```

Listing 4.4: TUIL Test Case Close Container Example XML

```
<?xml version="1.0" encoding="UTF-8"?>
<tuil_test_case>
    <path>
        <edge>
            <start root_node="true">
                <entry>
                    <id>TUILExampleDialog</id>
                    <container type="dialog" is_modal="true">
                        <is_visible>true</is_visible>
                        <is_enabled>true</is_enabled>
                        <has_focus>true</has_focus>
                        <text>TUIL Example Dialog</text>
                                .
                                .
                                .
                    </container>
                </entry>
            </start>
            <end />
            <action_pair>
                <action>
                    <mouse_action>mouse_click</mouse_action>
                </action>
                <event_object id="ID_OK" parent_id="TUILExampleDialog" />
            </action_pair>
        </edge>
    </path>
</tuil_test_case>
```

How test cases are decided and generated, including design considerations, is further discussed in the next chapter.

Chapter 5

Test Case Generation

As stated previously, testing of a GUI is complicated. According to Memon [4],

> Even when tools are used to generate GUIs automatically, these tools themselves may contain errors that may manifest themselves in the generated GUI leading to software failures. Hence, testing of GUIs continues to remain an important aspect of software testing.

In other words, regardless of whether a GUI is created automatically or manually, the potential for introducing faults remains.

In chapter 4, we introduced the concepts of the TUIL language, and the abilities of the language to specify a GUI and its behaviour in a light-weight, platform independent manner. In this section, we introduce the process of automatic test case generation based on TUIL specifications, as well as a prototype tool to accomplish this.

5.1 The TUIL Specification

As discussed in section 1.2, the main programming language used for manipulating and processing the specifications is Java.

To load the specification into Java for processing, we use the Java package JAXB [48,49] and XJC [50,51]. Please see Appendix B for how XJC is used to generate the tuil Java package, as well as one of the generated Java files.

Instances of the following JAXB variables are required:

- JAXBContext - How the XML/Java binding information, required to perform unmarshalling/marshalling, is managed [52].

- Unmarshaller - Parses the XML file contents into corresponding Java class instances, or throws an error if the XML files do not conform properly to the JAXBContext defined.

Please see section B.2 for examples of how these variables may be instantiated. The JAXBContext created is associated with the **tuil** package of Java classes that we generated using XJC. The Unmarshaller object is then created from this JAXB-Context, thus it is already setup to handle conversion from an XML file to these Java classes.

It is assumed that each XML file provided will yield a single **TuilContainer** object instance. This refers back to the TUIL section 4.3.3, where it is suggested, as good practice, to limit the contents of each XML file to a single **tuil_container** element. Thus, in order to process all XML files for a particular specification, there has to be one *unmarshal* call for each file. This will yield a complete set of **TuilContainer** instances to work with.

5.2 The Graph

In this research, a graph is defined as [53]

> A collection of vertices and edges that join pairs of vertices.

In many cases, vertices are called nodes or points, while edges are called arcs or lines, and may be directed or undirected. An undirected edge means that the edge may be traversed from either of its vertices, while a directed edge may only be traversed from one vertex. For the extent of this research, we will use nodes and edges/arcs to describe these.

A graph relates directly to GUIs, which is why FSMs or similar structures have been used to model them. The following aspects of a GUI are modeled by the graph:

- An edge is a transition from one node to another. When associating this with a GUI, a transition would typically be linked to an event on the GUI. However, since this research defines an event differently, as seen in section 4.3.5, an edge is instead associated with an action on the GUI.

- An edge links two nodes together within a graph. Within a GUI, an action transitions a GUI from one state to another. Therefore a node is associated with a GUI state.

- A GUI should be modeled as a directed graph, where the edges transition in one direction only.

With respect to test case generation, if the GUI is modeled as a directed graph, then as mentioned in the TUIL section 4.3.6, a test case is a single path through that graph, from the initial state to an end state determined by the goal of the test case itself. So a test case is made up of three parts:

- An initial state, which would typically represent when the GUI application has just been opened.
- An end or goal state, which could be as simple as a check-box being selected.
- A set of actions to move from initial state to end state.

5.2.1 annas Graphing Library

The test case generation process uses the February 21, 2010 release of the Java based graphing library annas, created by Sam Wilson [54]. This library provides a set of generic classes that can be extended to create an implementation specific to this research. The rest of this section introduces the aspects of the library used, and some of the inner workings thereof.

Three classes are required here:

- A node class to represent the GUI's state at a particular moment.
- An arc class to represent an action being applied to the GUI.
- A directed graph class that makes use of both the node and arc classes.

The node class created is called **TUILNode**. It is used to hold a set of containers ordered by keys having the following form '**parent_id+id**', where the two values are replaced by the two unique id attributes for each container. It allows access to the current state of each container or control for the purposes of manipulation or state checking.

In the annas library, an arc is an object that extends from one node to another. The node being extended from is called the *tail*, while the node the arc extends towards is called the *head*. As well, arcs are generally given a weight, which can be used in graphing algorithms for cost analysis of paths, etc. The arc class created is **TUILArc**, which extends annas's *ArcInterface*$< N >$, where the 'N' generic type parameter is replaced by **TUILNode**. A **TUILArc** not only encompasses the two **TUILNodes** it spans, but also the action that will be applied to the GUI, and a default arc weight of 1.

The annas class *DirectedGraph*$< N, A >$, where 'N' is the node type parameter and 'A' is the arc type parameter, is extended upon to create **TUILGraph**. 'N' and 'A' are replaced with **TUILNode** and **TUILArc** respectively. The underlying default methods are used as much as possible for manipulating the graph object, however, extra functionality was added as required.

One important piece of functionality is when adding an arc into the graph. *DirectedGraph* only provides a way to add a generic arc, defined only by its two nodes and a weight value. This default behaviour would prevent access to the action saved within the **TUILArc** class. Thus, since access to this information is needed, another *addArc* method was implemented, taking a **TUILArc** as its parameter. As well, a method was added to the **TUILGraph** to allow a search to be done for all **TUILArcs** where a container is closed. The **TUILNodes** associated with this arc are then returned to aid in test case generation. Another method that was added allowed a search to be done within the graph to determine if a **TUILNode** passed in was the head of at least one **TUILArc**.

The base graph class within the annas library uses a *protected Hashtable*$< N, MultiHashMap < N, A >>$ *nodeMap*. The *MultiHashMap* mentioned is also defined within the library, where the node type is the key into the *HashMap* and the arcs are the values.

All nodes must be added to this *Hashtable* regardless of whether or not they have arcs extending from them. If both nodes are not contained here, then the arc cannot be added between them. When an arc is added, the *tail* node is used as the key into the *Hashtable*, while the *MultiHashMap* portion has a key-value pair of (*head* node, arc). Each node could have multiple arcs coming out of it to other nodes, or back to itself if the arc is redundant. This makes determining what actions cause a state to change easier.

Due to the fact that the annas base implementation uses hashing data structures for storage, all **TUILNodes**, **TUILArcs**, and class types stored within them, must implement both base Java *Object* methods *equals* and *hashCode*. This is so that the keys and values can be compared properly with the structures.

Two other classes from the library are used as examples for classes here, however, these have more relevance in the section 5.4, where the actual test case generation process is described.

5.3 Generating a Graph

This section discusses how the test case generation process actually generates a graph. Several algorithms are used in order to generate a graph. Algorithm 1 provides the pseudo-code for the main method that begins the graph generation process.

In order to ensure easier access to required data during the generation process, the **TuilContainer** class is not used. Instead each **TuilContainer** is converted to a **ContainerState**, including all of its child controls, which are converted to a matching state class, eg. **TuilSlider** to **SliderState** (Please see Appendix C for one of the state Java classes). This particular conversion allows access to the child control information

Algorithm 1 Test Case Graph Generation Algorithm

```
function GENERATE(mapParsedSpec)
    if mapParsedSpec invalid then
        return FALSE
    end if
    for all tuilContainer in mapParsedSpec do
        if tuilContainer has a set of children then
            get all container events, if any
            if child events retrieved and event set is not empty then
                save event set for container
            end if
            create default container state
            save default container state into mapDefaultStates
        end if
    end for
    get root container key                    ▷ root has 'NULL' or '' parent_id value
    if root container key not found then
        display error message
        return FALSE
    end if
    graph ← new TUILGraph instance
    start ← new TUILNode instance
    n ← null
    isGen ← GenerateContainerBranch(rootContainerKey, start, n, n, n, graph)
    if isGen then
        save graph
    end if
    return isGen
end function
```

in a simpler manner, since instead of being linked in with layout information, they are stored in a set with their **id** attribute value as the key. All **ContainerState** objects are then stored in a set with their respective '**parent_id+id**' string key, as mentioned earlier in section 5.2.1.

During this conversion process, all **Event** objects from the **TuilContainer** and its child controls are broken out as a group and saved into a set using the same key as the **ContainerState** set. This is so that if one of these can be accessed properly, the other one can also be accessed. While doing this, the **EventObject** attributes are filled in properly to prevent any 'NULL' or empty string values. These values are typically used to say that the **Action** or **Reaction** is to be applied to the control or container the **Event** is specified within. However, since the **Event** is no longer associated here, the proper values must be used instead. When the child events are retrieved, any **toggle_check** reactions are saved into one of two lists, for either **toggle_check_on** or **toggle_check_off**. This makes it easier to apply the reactions during graph generation, as either one can be used based on whether the child control was toggled on or off by its associated action.

The root or main container object is identified by searching the set of **TuilContainers** for the one with a **parent_id** attribute value of 'NULL' or empty string. Not being able to determine a root container is an error condition, as the graphing process itself starts at the root container identified, and extends depth-wise from there in a recursive manner using the **GenerateContainerBranch** method (Please see Listing 5.1 for this method's signature). After the root container is dealt with, this method will only ever be called if an **open_container** reaction is encountered. The method takes as input, the following parameters:

- *key* - The key into the **ContainerState** set that represents the container to be worked on.

- *start* - The previous **TUILNode**, or the *tail* node of the **TUILArc** causing this container to be worked on. This parameter should never be *null*, as this would mean we have no way to attach this branch into the graph. For example, when beginning the graphing process, a new, empty **TUILNode** is created and passed in.

- *inProgress* - The TUILNode that represents a partial end state. This means reactions were applied to *start* before the **open_container** reaction was encountered, and thus, they also need to be taken into account. If this is not the case, then *null* is used.

- *connector* - The **TUILArc** causing a container to be opened and worked on cannot be added to the graph until its *head* node is also added. The *head* node being the one where the container is now displayed. For all cases where the connecting **TUILArc** is not required, *null* is used.

- *extraEvents* - A set of events, decomposed to more accessible objects, that do not originate from the container to be worked on, but which may also be applied. This is really only applicable if a non-modal container is opened from within a modal container, and thus the modal container's events are also applicable here. The decomposition of the **Event** object is discussed further below.

- *graph* - The **TUILGraph** as it exists so far.

Listing 5.1: GenerateContainerBranch Method Signature

```
public boolean GenerateContainerBranch(String key, TUILNode start, TUILNode
    inProgress, TUILArc connector, List<Pair<TUILArc, ArrayList<ReactionObject>>>
    extraEvents, TUILGraph graph);
```

Within the **GenerateContainerBranch** method, the work can be divided into three sections, as shown in Algorithm 2.

Algorithm 2 The GenerateContainerBranch Algorithm

```
function GENERATECONTAINERBRANCH(key, start, inProgress, connector, extraEvents, graph)
    setup initial state for branch
    setup event set to process
    apply event sequences
end function
```

During the initial node setup, the *start* node, provided as a parameter to the method, is first copied. Copying **TUILNode** instances is continued throughout the rest of the method and sub-methods, in order to ensure that any changes made will not affect the original **TUILNode**. The newly opened container about to be processed is then added into the copied **TUILNode** instance. If this new node has not been added into the graph yet, and the container being worked on is the root container, then this node is saved as the start node for the entire application. This allows access to the initial node when generating test cases.

Event setup is a bit more complex. The first step is to gather together the set of **Events** to be applied to the container being worked on. These are taken from two different sources:

1. Those that were specified for the container itself.
2. **Events** from other currently displayed non-modal containers. However, if the container being worked on is modal, then of these, only those **Events** that originate from the code behind the GUI are allowed, as user interactions are not.

This, possibly combined, set of **Events** is then broken down one-by-one into a single **TUILArc** and a set of **ReactionObject** type objects. The **TUILArc** will be the actual arc object used as an edge in the graph, however, in its current state it has no associated node objects. The **ReactionObject** class is used to encapsulate the reactions in a manner where it is easier to do checks for type.

However, the set of events is not fully realized at this point. Any existing items in the *extraEvents* parameter must also be taken into account. If there are any, they are added into the current set if they do not already exist within it. This will prevent duplicates from being processed.

Next the **initialize_container** event, if it exists within the event set, is applied. This must be applied first, before any event processing is performed, otherwise the container could be initialized multiple times. However, this step is more for future use, as the current implementation of graph generation does not save state for a container after it has been closed. For the moment, this step is responsible for ensuring that the default state, as defined by the specification, is applied to the container. Once the **initialize_container** event has been applied, it is removed from the set of events. If the node with the container initially loaded has already been seen, the method is exited. This is to prevent processing the same container, within the same overall application state, multiple times.

The set of event sequences is then generated from the event set. This is basically an array of sequences, where each element of a sequence represents the index of an event to apply, and each sequence represents a permutation of events. Exhaustive sequence generation, meaning all possible event permutations are included, is very memory intensive and time consuming. In order to ensure that the memory and time costs are reduced, both the sequence and permutation generation must be brought down to a manageable number. Permutations are used, as the only events evaluated for the GUI are those that are specified for it, thus, any sequence will simply be a reordering of events. The following describes, by way of an example, the arbitrary method developed in this research to accomplish this.

Say there are four events, named Event0, Event1, Event2, and Event3. They are assigned indexes 0, 1, 2 and 3 respectively.

The first sequence is:

$$\begin{pmatrix} 0 & 1 & 2 & 3 \end{pmatrix}$$

For the first set of sequences, the first sequence above is shifted to the left by one item, for the number of items that are left. Here, this means it is shifted three times. This yields the following set of sequences:

$$\begin{pmatrix} 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{pmatrix}$$

Next, the second set of sequences is generated by simply reversing the first four. This gives the following set of sequences:

$$\begin{pmatrix} 3 & 2 & 1 & 0 \\ 0 & 3 & 2 & 1 \\ 1 & 0 & 3 & 2 \\ 2 & 1 & 0 & 3 \end{pmatrix}$$

There are now two sets of four sequences. However, since this is a small subset to work with, a few more operations will be performed. For each sequence currently generated, the initial event is frozen in its current location. Then, the rest of the events in each sequence are shifted to the left by one each time. In this example, this operation will be performed twice. The following shows the result of this shifting for the first sequence only. The other sequences are also shifted, but the results will not be shown here.

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \end{pmatrix}$$

In total, the number of event sequences generated, S, is equal to $2E(E\text{-}1)$, where E is the number of events being processed.

If this equation is applied to the example above, the total number of sequences generated is 24, each containing 4 events.

Once the set of event sequences has been determined, they are applied to the container in order. For each sequence the following actions are performed:

- The container to apply the event against is retrieved using its key value. By default, this is the key of the container being worked on, however, it may also be the key for the modal parent container or other non-modal container, since events from these may be allowed.

- The event is evaluated to determine if it can currently be applied. For any event to be applied to a control, that control must exist in the event container's current state and cannot be disabled or hidden. If an event is to be applied to a container, then the container must exist within the current **TUILNode** state, and must not be disabled or hidden. It is normal if an event cannot be applied, as not all events will be applicable in all scenarios. Instead, it is simply ignored and the process moves on to the next event in the sequence. If it is applicable, then it is applied.

- If something fails during event application, the process fails. There are two possible outcomes to a successful event application:

  1. A new container needs to be opened from the current event container. A recursive call is made to **GenerateContainerBranch**, after setting up the *connector* **TUILArc** properly, and determining if the rest of the events in the sequence need to be passed to the method as the *extraEvents* set.
  2. No container was opened and there is a resulting **TUILNode**. This is saved into the current **TUILArc** as the *head* node, and both are saved into the *graph*. The *head* node is then assigned to the *tail* node for the next event application. Please see Algorithm 3 for an example of saving the data into the graph.

- Processing ends when all events that need to be applied have been, and all recursive calls have returned.

Algorithm 3 How to Add the Nodes and Arc to a Graph

```
function AddInformationToGraph(graph, tailNode, headNode, arc)
    $arc.tail \leftarrow tailNode$
    $arc.head \leftarrow headNode$
    add tailNode to graph
    add headNode to graph
    add arc to graph
end function
```

When each event sequence is finished, the *tail* node is reset and the next event sequence is started. Once all sequences have been processed, if everything went well, i.e., no failure conditions were encountered, the graph generation process is said to have succeeded.

The *graph* is then saved for generating the test cases. This is explained in the next sub-section.

5.4 Generating Test Cases

Graph generation must have been successful, and at least one container must have been selected, for test cases to be generated properly. If a graph was generated, then a worker thread is started. This thread is responsible for generating the test cases and displaying a message box of the outcome. However, before any test cases can be generated from the graph, the directory structure is created.

The directory structure used is designed to ensure that if multiple test case sets are required, they can be generated without erasing or overwriting the previous set. This is done by way of time stamps. The main directory name has the form **'Test Cases %s UTC'**, where the %s is to be replaced by a date/time string. This date/time string is the current date/time converted into string form using the format *dd-MM-yyyy 'T'HHmmss*. The current date/time is in UTC or Coordinated Universal Time, which is synonymous with GMT or Greenwich Mean Time, hence the *UTC* at the end of the directory name. UTC time is used to prevent ambiguity with respect to when a set of test cases was generated. Inside of this main directory, sub-directories for each selected container will be created when that container has test cases generated for it.

Before this happens, however, the **TUILGraph** must be placed in a more usable form for test case generation, by running the single-source version of Djikstra's algorithm on it. This calculates the shortest path from a single start node to all other nodes in the graph. There are limitations to using this algorithm, as GUI events do not operate based on shortest paths. Thus, some events may not lie on shortest paths and be ignored. However, the algorithm will provide us with a working set of test cases to make use of.

To this end, a class which mimics most of the functionality of the annas library's *Dijkstra* class was created, and called **TUILDijkstra**. It performs the single-source version of the algorithm, instead of only calculating the shortest path(s) between two known nodes. This allows the shortest path from that start node to any other node in the graph to be determined. All shortest paths are determined based on the limitation that, if there are multiple **TUILArc's** having the same distance, one of these is randomly chosen.

The annas library also has a class *GraphPath*, that represents a single path through the main graph, and holds the information in the form of a smaller graph, with the start and end nodes defined. Similar to the *Dijkstra* class, this class cannot be used directly for the following reasons:

1. The class cannot be extended because, while it provides a means to specify the node and arc types, the type of the graph used cannot be overridden. This becomes troublesome, because as mentioned in the annas section 5.2.1, the ability to save a **TUILArc** object into the graph is required.

2. The class does not provide a method to generate a path from end to start, which is how the Dijkstra algorithm works here where each node has knowledge of the previous node before it in the graph. Instead, the *GraphPath* class generates its path from start to end.

Again, a class based heavily on the ideas and form of the *GraphPath* class is created and called **TUILGraphPath**. This class has the ability to copy itself, and to merge an existing **TUILGraphPath** into itself, as long as a connection between the two exists and both are finished. The connection is that the end node of one is the same as the start node of the other. Besides that, it makes use of the **TUILGraph** class for its graph, so that **TUILArc** objects can be added directly. As stated, the addition of nodes will go from end of the path to the start, to mimic the progression through the Dijkstra path.

Now that the two classes required for test case generation are introduced, how these are used to generate test case paths for each container selected is described. The following is done:

1. Determine the shortest path from the graph's start node to the one where the container is first opened. This requires checking for a transition where the first node does not have the container within it, but the second does. If the container is the root container identified during graph generation, then the start node itself is the entire shortest path here. A failure condition will occur if the container is not the root, but an open path could not be determined.

2. Using the open path found, all paths leading from this to an end node are determined. An end node is defined as one where the container is no longer displayed. This limits test case generation to container types that can be closed definitively, windows/frame and dialog-types. Only finished and valid paths are allowed in the final set, however, a failure will occur if no paths are found.

If each container selected for test case generation has a set of paths generated for it, this means test case paths were successfully found. Thus, the next step in the process is to export these to XML files using the classes discussed in section 4.3.7. To this end, the **TUILGraphPath** class is provided with a method which returns a **TuilTestCase** version of itself. To be able to accomplish this, the rest of the affiliated path variables are also equipped with methods similar to this:

- **TUILArc** - Converts to **TcEdge**. Must be provided the start node for comparison purposes.
- **TUILNode** - Converts to **TcNode**.
- All state object classes convert to corresponding **tuil** versions of themselves. Conversion is performed through the *ToTcObject* method, that returns a **tuil** object representing the current state object. For example, a **ButtonState** will return a **TcButton** object from this method.

However, before conversion takes place, each container's individual directory must be created. This takes the form of **parentId id**, where these placeholders are replaced by the string equivalent of the container's **parent_id** and **id** attributes respectively. Each directory contains a single XML file for each test case exported. The naming convention used for the XML file names is **%d.xml**, where %d represents a number, starting from zero, and with zero padding on the front to ensure proper sorting in the folder.

Exporting to the XML files is handled by JAXB marshalling. Some of the setup for this was discussed in section 5.1. The *Marshaller* object is used to take a Java class instance from the **tuil** package and export it in some form. Here exporting will be to an XML file.

If errors occur when exporting all of the **TuilTestCase** objects to file then a failure message is displayed. However, if everything worked properly, a success message is displayed on screen. The worker thread will be ended in either case in preparation for the next test case generation request.

5.5 TUIDE

TUIDE, or Testable User Interface Development Environment, is a prototype tool, written entirely in Java, that is used for performing all automation tasks involved with this research.

TUIDE itself has no support for generating TUIL specifications. Instead these must be written in an outside application, such as a text editor. However, it does have support for reading them in and processing them accordingly. Please see Appendix D for how to load a TUIL specification into TUIDE. TUIDE's specification parsing has been implemented based on the suggestion of a single **tuil_container** defined per XML file.

TUIDE's test case generation process, that was described in the previous sections, is initiated by the method outlined in Appendix E.

Automatic testing is unable to exist without test cases. They are one of the most important aspects. Several algorithms are given in this chapter that allow a TUIL specification to be turned into a set of test cases on one or more of the specified containers. These test cases contain both the set of actions to apply, as well as the state of the GUI before and after each action. This essentially combines the generation of test cases with the generation of oracles. The TUIDE test cases become more powerful than simple sets of actions using this combination.

Chapter 6

Oracles

This chapter presents the design concepts, processes, and algorithms used to apply the generated oracles against an AUT. In [25], Weyuker describes an oracle as the mechanism that checks the correctness of an application's response to a particular set of inputs.

Please see Appendix G for details on how to actually run the oracles within TUIDE itself. As noted there, certain steps must be completed before the oracles can be applied. They include:

1. A loaded set of test cases, that were previously generated from TUIDE. These also contain the generated oracles, as mentioned in chapters 3 and 5.

2. Selection of the runnable JAR file, or Java ARchive file, representing the implementation of the AUT. Appendix H contains a set of guidelines for implementing the GUI of the AUT for a successful oracle application process. The guidelines are somewhat restrictive in nature, and would be prohibitive in the case of adapting an existing application to the oracle process, unless the application is

very small. However, the benefits of using them far outweigh the restrictiveness, as it means the oracle process can run more smoothly, and be more accurate with result comparisons, as it gets its generated state information directly from the controls themselves.

TUI_Utils is a Java library designed to be used by both TUIDE and the AUT. It is built into a non-runnable JAR file, that is then added to the referenced libraries list for each application, however, not all parts of the code are usable by both of the applications. The AUT makes use of some packages that are not used by TUIDE, and vice versa. The following two sections describe the parts of the library that each uses.

6.1 TUIDE Oracles

The oracle functionality within TUIDE is actually simplified in comparison to that of the AUT. This is because TUIDE merely initiates each oracle, and is not designed to handle anything more in depth.

The oracle manager class makes use of a worker thread from which to initiate each oracle. This means that it will not hold up the entire TUIDE application in order to run, and once all oracles have been applied the thread exits. The algorithm used within this thread is defined in Algorithm 4.

As can be seen in the aforementioned algorithm, the thread creates both a main oracle results directory, as well as directories for each container encountered during iteration, to ensure that the results are organized. The main directory is timestamped with the current UTC timestamp to ensure that each oracle application run is unique from other runs. This folder is always created in the TUIDE application directory.

Algorithm 4 TUIDE's Oracle Thread Algorithm

```
function RUN
    $stop \leftarrow FALSE$
    $isRunFinished \leftarrow FALSE$
    $jarFile \leftarrow the\,runnable\,JAR\,file's\,canonical\,path$
    $cmdLineFormat \leftarrow 'javaw\ -jar\ \backslash'\%s\backslash'\ -oracle\ -testCase\ \backslash'\%s\backslash'\ -logFile\ \backslash'\%s\backslash''$
    $oracleDir \leftarrow CreateOracleDir(void)$
    $testCaseSet \leftarrow the\,test\,case\,map's\,entry\,set$
    $testCaseIterator \leftarrow null$
    setup the rest of the required variables
    while !stop do
        if $testCaseIterator = null$ or $testCaseIterator$ has no items left then
            if $containerIterator$ has next item then
                $entry \leftarrow the\,next\,containerIterator\,value$
                $testCaseIterator \leftarrow entry's\,value\,property's\,iterator$
                containerDir ← CreateContainerDir(oracleDir, entry's key property)
                if $containerDir = null$ then continue
                $containerDir \leftarrow containerDir + '\backslash\backslash'$
            else
                display completed message and exit while loop
            end if
        else
            $testCasePair \leftarrow next\,testCaseIterator\,value$
            $testCaseFile \leftarrow second\,value\,in\,testCasePair$
            $logFile \leftarrow GetFilenameWithoutExtension(testCaseFile)$
            if $logFile = null$ then continue
            $logFile \leftarrow containerDir + logFile + '.log'$
            if $jarFile = null$ then exit while loop
            cmdLine ← Format(cmdLineFormat, jarFile, testCaseFile, logFile)
            $jarProcess \leftarrow$ execution of cmdLine
            if $jarProcess = null$ then continue
            wait until AUT communications setup or 10 seconds pass
            wait until message received from AUT or 30 seconds pass
            examine any message received
            destroy jarProcess and clean up AUT communications channel
        end if
    end while
    $isRunFinished \leftarrow TRUE$
end function
```

Each container directory is created within this main directory, and provided with a name that uniquely denotes the container by using its **parent_id** and **id** properties.

Test cases are organized into groups, based on what container they were generated for. Each test case is used to generate two file paths:

- The file path for the test case to run.
- The file path for the results from running the test case against the AUT. This will be referred to as the log file from here-on-out. The file name for the log file is the same as the associated test case file, so that it is easier to distinguish what log file is associated with what test case for reviewing the final results.

When the command line is generated, in order to run the AUT JAR file, we pass both file names to it as parameters. These will be discussed further in the next section. The JAR file is run within its own javaw process to ensure that it has a separate set of memory to work with, rather than sharing TUIDE's.

TUIDE makes use of the OracleCommunication class within the TUI_Utils library. OracleCommunication allows String communication only, and provides an input queue and output queue, onto which Strings are stored depending on whether they were received or sent respectively. This class, on the TUIDE side, sets up a server connection to the local port 33333, on which it listens for the AUT to send messages to denote whether the oracle being applied has finished or whether an error state was encountered. First, however, the TUIDE thread waits to ensure the AUT has established a client connection before proceeding. This is provided with a timeout of ten seconds, after which the AUT JAR process will be shut down if no client connection was established. Once a connection is established, the thread waits for a finished or failure message to be sent from the AUT. This is also given a timeout, that is set to thirty seconds. At this stage, the thread will have the same response whether the time out occurred, the connection to the AUT was terminated, or the AUT sent us a valid response: the AUT JAR process is shut down and execution continues to the next test case, if any. The timeouts used here prevented indefinite lock ups of the oracle application process, and were chosen based on how well they worked for the application instance being tested (please see Chapter 8).

Once all of the oracles have been run, the user is notified that the oracles have completed and where the results can be found. This will happen regardless of how many oracles were completed successfully.

6.2 The AUT and Oracles

As stated previously, the AUT contains the majority of the oracle functionality, since it is the AUT that will apply the actions and gather the GUI states. As such, it makes use of more classes within the TUI_Utils library, the main class being the AUTHandler. This class must be instantiated before the view is up and running, and have its Start method called after this. This works very well within Model-View-Controller architectures.

Any command line arguments received by the AUT are passed to the AUTHandler for processing. There are three such arguments that this class is interested in:

- **-oracle** - Only when this argument is present, will the oracle functionality be enabled within the AUT. This allows the AUT to be built with the TUI_Utils library once, but depending on how the JAR file is instantiated at runtime, it may or may not run with the oracle code available.
- **-testCase** - The value associated with this argument is the file path for the test

case to apply against the AUT. The file is unmarshaled into a **TuilTestCase** object, using JAXB, to allow for easier access to its contained information.

- **-logFile** - The value that follows this argument is the file path for the log file that all oracle results will be written to. A FileWriter object is created for saving these results.

If either one of these arguments is missing from the command line, or either the **TuilTestCase** or FileWriter could not be created successfully, all further oracle processing is ended and a failure message, in the form of a string ”AUT FAILURE OCCURRED\r\n”, is sent back to the TUIDE application, using the OracleCommunications channel created.

The AUTHandler makes use of an internal thread to apply the oracle to the GUI. The pseudo-code for the thread's algorithm can be seen in Algorithm 5.

6.2.1 Applying Actions

In order for the AUTHandler to apply each action from the parsed test case to the AUT GUI, the code to do this must be run in a special way. Code that accesses or modifies the state of Swing objects must always be executed on the event dispatch thread [55]. Rather than use a SwingWorker [56] implementation however, the method SwingUtilities.invokeAndWait [57] is used. Listing 6.1 is an example of a code-snippet that would represent how to use the SwingUtilities.invokeAndWait method.

Listing 6.1: How to Code SwingUtilities.invokeAndWait

```
try
{
  final checkbox = m_CheckBox;
  SwingUtilities.invokeAndWait(new Runnable()
  {
    public void run()
    {
      checkbox.setSelected(true);
    }
  });
}
catch(Exception ex) {}
```

Listing 6.2: ApplyAction Method

```
  public boolean ApplyAction(ActionPair p)
  {
    if(p == null || p.getAction() == null || p.getEventObject() == null)
    {
      return false;
    }

    String parent = p.getEventObject().getParentId();
    String child = p.getEventObject().getId();
    if(parent == null || child == null || child.isEmpty())
    {
      return false;
    }

    boolean isRoot = (parent.isEmpty() || parent.equals(new String("NULL")));

    ComponentFinder finder = new ComponentFinder(parent, child);
    Window w = finder.FindWindow();
    if(w != null)
    {
      if(!isRoot)
      {
        Object o = finder.FindComponent(w);
        if(o != null)
        {
          if(!(o instanceof TuideTitledBorder))
          {
            return ApplyAction(p, (Component)o);
          }
        }
      }
      else
      {
        return ApplyAction(p, w);
      }
    }
    return false;
  }
```

The invokeAndWait method places the code found inside the Runnable.run method onto the AWT event dispatch thread, thus enabling it to interact with the

Algorithm 5 AUTHandler Algorithm

```
function RUN
    $stop \leftarrow FALSE$
    $isRunFinished \leftarrow FALSE$
    $stepCount \leftarrow -1$
    $currentNode \leftarrow null$

    if $testCase = null$ then
        exit run function
    end if

    $edges \leftarrow testCase\, edge\, set$
    $edge \leftarrow null$
    $currentEdge \leftarrow 0$
    $edgeCount \leftarrow size\, of\, edges$
    $empty \leftarrow empty\, ActionPair$
    while !stop do
        if $currentEdge = edgeCount$ then
            send complete message to TUIDE
            exit while loop
        end if
        $edge \leftarrow edges.get(currentEdge)$
        increment currentEdge by one
        if $edge \neq null$ then
            $p \leftarrow null$
            if $stepCount \neq -1$ then
                $currentNode \leftarrow edge's\, end\, node$
                $p \leftarrow edge's\, ActionPair\, value$
            else
                $currentNode \leftarrow edge's\, start\, node$
                $p \leftarrow empty$
                $currentEdge \leftarrow 0$                ▷ reset to get end node next
            end if
            clear current StateHandler values
            save p into StateHandler
            write p to log file
            apply action to GUI
            sleep for 1 second
            retrieve GUI state
            if $log\, file\, valid$ then
                write state result to log file
            end if
            backup current StateHandler values for next iteration
        end if
        $stepCount \leftarrow stepCount + 1$
        **sleep** for 10 milliseconds
    **end while**

    **shutdown** communications channel to TUIDE
    **close** log file writer
    $isRunFinished \leftarrow TRUE$
**end function**
```

GUI properly. The method blocks the current thread from continuing until all pending AWT events generated are completed. Any variables declared outside the method that need to be used internally, must be declared final or assigned to a final temporary variable to be used. Exception handling must also be included in some manner, as the invokeAndWait method can throw either an InterruptedException or an InvocationTargetException.

The AUTHandler uses this to access the ActionHandler class, which is responsible for everything to do with the actions. The ActionHandler is provided with the ActionPair object, representing the action to apply. The EventObject, from the ActionPair, provides the names of the parent container and the control. This information is used, along with the ComponentFinder class from TUI_Utils, to find the object that the action needs to be applied to (See Listing 6.2 for the ApplyAction method code).

ComponentFinder works on the Window.getWindows() method, then iterates through to see if any of the current set of application Window objects has a name equal to the EventObject's **parent_id** value. If the Window is found, the controls contained within it are searched in a recursive manner to see if there is one with its name equal to the **id** value of the EventObject.

The root Window is considered to be a special case here, as there is no actual control to manipulate, but rather the Window is manipulated. As well, at the moment the TuideTitledBorder cannot be manipulated by an action, as it does not extend upon Component or JComponent like the other controls do.

Once the control/container is found, it is cast to an appropriate type, based on the type of action about to be performed. The control/container is first given focus before the action itself is applied. Several actions are more complex to apply than others:

- Key Press Actions - Makes use of the Robot class's keyPress and keyRelease methods to simulate pressing the key on said control. Part of the Java AWT package, Robot provides a means to manipulate the controls within the GUI through Java code. Through use of this class, we have the ability to click a button, select a check box, and even simulate pressing a key on the keyboard, as mentioned above.

- Key + Mouse Actions - These represent actions such as, Shift+Left Mouse click, etc. The Robot class is also used here. Please see Listing 6.3 for the code that is used to apply this type of action.

- Mouse Actions - Makes use of the Robot class as well. As with the Key + Mouse Action above, the mouseMove method must be called to ensure the mouse cursor is over the control before calling any subsequent functions. For a single click, mousePress is called followed by mouseRelease, with the appropriate button value passed in. To simulate a double click, mousePress and mouseRelease are called in order, and then this is repeated.

- Item actions - This includes both selecting an item, de-selecting an item, and selecting a specific item. Since the test cases do not provide an actual item index to use for selection, the ActionHandler must determine this itself. It uses a random index generator method found in TUI_Utils, which takes the count of items in the list object and returns a random index from this, in the range [0, count). If the action is to de-select an item, then items must already be selected, and the random index must be the index for one of these. If the action is instead a select item action, then there must be an index available to select.

- Code actions - Code actions are slightly different than a normal action, as they are usually called from background code itself. To simulate, a CodeEvent class is created, which is then fired off to all current IAUTEventListeners. The UpdateContainer or the InitializeContainer methods are used here, depending on the code action that needs to be performed. As discussed previously in section 5.3, the test case generation code action to **initialize_container** only uses the default settings for a container, thus similar should be implemented within the AUT overridden methods.

Listing 6.3: Code for Applying a Key+Mouse Action

```
if(action.getKeyMouse() != null)
{
  c.requestFocusInWindow();

  Robot robot = null;
  try
  {
    robot = new Robot();
  }
  catch (AWTException e)
  {
    robot = null;
  }

  if(robot == null)
  {
    return false;
  }

  Point pt = c.getLocationOnScreen();
  robot.mouseMove((int)(pt.x + c.getWidth()/2), (int)(pt.y + c.getHeight()/2));

  switch(action.getKeyMouse())
  {
    case CTRL_PLUS_MOUSE_CLICK:
      robot.keyPress(KeyEvent.VK_CONTROL);
      robot.mousePress(InputEvent.BUTTON1_MASK);
      robot.mouseRelease(InputEvent.BUTTON1_MASK);
      robot.keyRelease(KeyEvent.VK_CONTROL);
      break;

    case SHIFT_PLUS_MOUSE_CLICK:
      robot.keyPress(KeyEvent.VK_SHIFT);
      robot.mousePress(InputEvent.BUTTON1_MASK);
      robot.mouseRelease(InputEvent.BUTTON1_MASK);
      robot.keyRelease(KeyEvent.VK_SHIFT);
      break;

    default:
      return false;
  }
}
```

Once the action has been applied, the next step in the process is to request the GUI's state.

6.2.2 Getting Current GUI State

The current state of the GUI is retrieved only after the action has been applied, including a pause to give the event queue enough time to push our action through to the GUI controls themselves. State generation is done by way of an event, StateRequestEvent, and an event interface, IAUTEventListener. The IAUTEventListener provides interface methods each GUI container should implement in order to enable the state request process to work properly, namely OnStateRequest and CompileControls. As well, the ActionPair that was applied is saved for state generation, since in some cases it may be required.

As the state generation process will be accessing the GUI containers and their controls, it must also be wrapped in a SwingUtilities.invokeAndWait call. Listing 6.4 shows the method used to call the OnStateRequest event function for each container currently associated.

Listing 6.4: fireStateRequestEvent Method

```
protected synchronized TcNode fireStateRequestEvent()
{
  StateRequestEvent event = new StateRequestEvent(this);
  for(IAUTEventListener isrel : m_Listeners)
  {
    isrel.OnStateRequest(event);
  }
  return event.GetNode();
}
```

As each Window receives the OnStateRequest call, it in turn calls the CompileControls method, which adds all applicable controls into a List of Objects. As per Appendix H, one of the implementation guidelines discussed there states that if there are any JComboBoxes, JLists, or JSlider controls, then their JLabels must be paired with them, and the pair itself is saved into the List. This is to ensure the **text** element information is available. It then makes use of the StateHandler to add its set of controls into the StateRequestEvent object. See Listing 6.5 for an example of the implementation of the OnStateRequest method.

Listing 6.5: OnStateRequest Example Implementation

```
@Override
public void OnStateRequest(StateRequestEvent e)
{
  List<Object> controls = CompileControls();
  StateHandler.AddToCurrent(e, false, this, controls);
}
```

The AddToCurrent method converts the controls and the container to their associated TUIL test case objects, then adds them all into a TcNode.Entry. To convert the controls, the previous state or applied action type is sometimes required, especially when dealing with combo box, list, and slider objects. Listing 6.6 provides the method code for converting a JComboBox into its associated TUIL object, which requires knowing whether the action just applied was a select item action on said control. As well, it can be noted that the conversion does not take the focused state into account here, as seen by the line `c.setHasFocus(false);`. The focused state is instead handled outside of the individual conversion methods, and is only set if the conversion was successful and a focused control or container could be determined.

Listing 6.6: ConvertCombo Code

```
public TcContainer.Entry ConvertCombo(boolean selectItemAction, String label,
    JComboBox current)
{
  if(current != null)
  {
    TcContainer.Entry ret = new TcContainer.Entry();
    ret.setId(current.getName());

    TcCombo c = new TcCombo();
    c.setHasFocus(false);
    c.setIsEnabled(current.isEnabled());
    c.setIsVisible(current.isVisible());
    c.setText(label);
    c.setListShown(current.isPopupVisible());

    ItemList list = new ItemList();
    List<String> items = list.getItem();
    for(int i = 0; i < current.getItemCount(); ++i)
    {
      items.add((String)current.getItemAt(i));
    }
    c.setItems(list);

    int selectedItem = current.getSelectedIndex();
    c.setSelectedItem(BigInteger.valueOf(selectedItem));
    if(selectedItem != -1)
    {
      c.setSelectState(SelectionState.SINGLE_SELECTION);

      if(selectItemAction)
      {
        c.setItemChangeState(ItemChangeState.UPDATE_SINGLE);
      }
      else
      {
        c.setItemChangeState(ItemChangeState.NO_CHANGE);
      }
    }
    else
    {
      c.setSelectState(SelectionState.NO_SELECTION);
      c.setItemChangeState(ItemChangeState.NO_CHANGE);
    }
    ret.setTcChild(m_Factory.createTcCombo(c));
    return ret;
  }
  return null;
}
```

Once the event requests for all Window objects have been received, the state is compared to the expected result in the test case itself. This is described in the next section. However, when the result has been logged, the current state is saved so that it can be used for the next state request, if necessary.

6.2.3 Logging Results

This section describes how the code logs the oracle result from each step of a test case to a log file. The file structure was mentioned previously, so here the actual result process will be discussed.

However, before the state requested from the AUT is compared and the result logged, the action that was applied by the AUTHandler is first logged. This ensures that the action that was applied is known, along with what the result was based on this.

The first action applied is the empty action, which retrieves the first state of GUI. This is denoted as the start-up action and initial state. After this, each *Action* is denoted by what type of action was performed on what control in what container. The *State* line provides the state comparison result.

Comparisons are performed between the generated and expected TcNode instances, and then the comparisons tunnel down to the containers and their controls. Algorithm 6 shows the main comparison method, which is responsible for actually writing the result to the log file.

As can be seen, the CompareTcNodes method call returns a string result rather than a boolean. This is so that the result information written to the log file is as detailed as possible when a difference is found, denoting what failed and why. The current implementation works by returning only the first difference found. If more exist at this point in the test case, then they will not be noted in the log file. If everything was as expected, the following value is entered for the state:

Algorithm 6 LogFile WriteResultToFile Algorithm

```
function WRITERESULTTOFILE(TcNode exp, TcNode gen, int count, String value)
    if $writer = null$ then
        display message to user using invokeAndWait
        return $FALSE$
    end if
    $outputString \leftarrow null$
    if $exp = null$ and $gen = null$ and $value \neq null$ and $value$ $not$ $empty$ then
        $output \leftarrow value$
    else
        $result \leftarrow CompareTcNodes(exp, gen)$
        if $count = -1$ then
            $output \leftarrow initialStateString + result + lineEndString$
        else
            $output \leftarrow String.format(stateFormat, count + 1) + result + lineEndString$
        end if
    end if
    write output to $writer$
    flush $writer$
    if $an$ $exception$ $occurred$ $during$ $write$ then
        display message to user using invokeAndWait
        return $FALSE$
    else
        return $TRUE$
    end if
end function
```

```
\tPass: States identical!\r\n
```

The following are differences that would cause a failure result message to be logged:

- If the expected TcNode contained the root node for the AUT, but the actual TcNode did not.
- The opposite of the above, the root node was not expected, but it was found.
- If the expected TcNode was null, but the generated TcNode was not.
- The opposite of the above, where generated is null, but expected is not.
- If the number of expected TcNode.Entry objects is different from the number generated.
- If the comparison between the expected and generated TcNode.Entry objects fails. This can occur for several reasons, some of which include:
  - Similar null and number comparisons to above, but with respect to the TcContainer and TcContainer.Entry objects.
  - The expected base state values of an object are different from its generated base state values. This includes focused, enabled, visible, text, etc. Please see Listing 6.7 for the base state comparison method.
  - The expected control type specific values of an object are different from its generated specific value. An example of control specific properties is whether a text control is read only, or the selection state of a list is the same. The actual selected index values are never compared, as the test cases do not support this functionality. Please see Listing 6.8 for the TcButton object comparison.

Listing 6.7: Comparison Between TcBase objects

```
  protected String CompareTcBase(TcBase ours, TcBase theirs)
  {
    String ret;

    String ourText = ours.getText();
    String theirsText = theirs.getText();
    if ((ourText != null && theirsText != null && ourText.equals(theirsText)) || (
        ourText == null && theirsText == null))
    {
      if (ours.isHasFocus() == theirs.isHasFocus())
      {
        if (ours.isIsEnabled() == theirs.isIsEnabled())
        {
          if (ours.isIsVisible() == theirs.isIsVisible())
          {
            ret = m_OK;
          }
          else
          {
            ret = String.format(new String("Visibility state not same, expected (%s)
                vs generated (%s)"), ours.isIsVisible(), theirs.isIsVisible());
          }
        }
        else
        {
          ret = String.format(new String("Enabled state not same, expected (%s) vs
              generated (%s)"), ours.isIsEnabled(), theirs.isIsEnabled());
        }
      }
      else
      {
        ret = String.format(new String("Focused state not same, expected (%s) vs
            generated (%s)"), ours.isHasFocus(), theirs.isHasFocus());
      }
    }
    else
    {
      ret = String.format(new String("Text value not same, expected (%s) vs generated
           (%s)"), ourText, theirsText);
    }
    return ret;
  }
```

Listing 6.8: Comparison Between TcButton objects

```
if (oursC == TcButton.class)
{
  TcButton b1 = (TcButton) oursE.getValue();
  TcButton b2 = (TcButton) theirsE.getValue();

  String localResult = CompareTcBase(b1, b2);
  if(localResult.equals(m_OK))
  {
    if (b1.getType() != b2.getType())
    {
      result = String.format(new String("Button Type state not same, expected (%s) vs
           generated (%s)"), b1.getType().toString(), b2.getType().toString());
    }
    else if (b1.isIsChecked() != b2.isIsChecked())
    {
      result = String.format(new String("IsChecked state not same, expected (%s) vs
          generated (%s)"), b1.isIsChecked(), b2.isIsChecked());
    }
    else
    {
      String b1group = b1.getButtonGroup();
      String b2group = b2.getButtonGroup();

      // Basically if we're comparing "null" and "empty" together, these essentially
          mean the same thing:
      // there is no button group.
      boolean b1groupEmpty = (b1group != null && b1group.equals(new String("")));
      boolean b2groupEmpty = (b2group != null && b2group.equals(new String("")));
      if((b1group == null && b2group != null && !b2groupEmpty) || (b1group != null &&
           b2group == null && !b1groupEmpty) || (b1group != null && b2group != null
          && !b1group.equals(b2group)))
      {
        result = String.format(new String("ButtonGroup state not same, expected (%s)
            vs generated (%s)"), b1group, b2group);
      }
    }
  }
  else
  {
    result = localResult;
  }
}
```

The following Listing 6.9 shows an example of what a possible log file could look like, including some failure results:

Listing 6.9: Example Log File

```
Action: Start-up
Initial State:
  Failure: Root node expected!

Action: mouse_click on ID_OPEN_DLG_BUTTON within container ID_MAIN_DLG
State 1:
  Pass: States identical!

Action: return_key pressed on ID_CHECK_ONE within container ID_DLG
State 2:
  Pass: States identical!

Action: mouse_click on ID_CHECK_TWO within container ID_DLG
State 3:
  Pass: States identical!

Action: select_item on ID_LIST within container ID_DLG
State 4:
  Pass: States identical!

Action: mouse_click on ID_OK within container ID_DLG
State 5:
```

```
Failure: The Node was expected to contain 0 containers, instead contained 1
    containers!
```

Chapter 7

Mock-up GUIs

This chapter describes the process by which a GUI mock-up can be generated based on a TUIL specification. Mock-up can be defined as follows [58]:

A usually full-sized scale model of a structure, used for demonstration, study, or testing.

With respect to GUI mock-ups within the TUIDE application, a mock-up is exactly this, simply a model of what the GUI could look like when implemented. There are two main reasons for generating a mock-up for a TUIL specification:

1. To ensure that the TUIL language is both expressive and precise enough to be useful. Can it properly describe the layout and appearance of a GUI?

2. To provide the specification designer the ability to check their specification, to determine if the GUI container being specified appears in the manner expected: Are the controls in the correct locations? Orientations? Are the properties, such as text and list contents, displayed correctly?

As a mock-up is for layout and appearance purposes only, the following design considerations are used:

- The underlying event system within the specification will be ignored.
- All controls will be enabled and visible, so that the designer can see all of them at once.

7.1 Generating a Mock-up

The mock-up generation process is illustrated using an example specification consisting of one main root window, and two child dialogs. Since the purpose is for generating a mock-up, and not test cases, no events have been included in the specification's design. Please see Appendix F for how to load the container selection dialog for mock-up generation.

The designer is only able to select a single container from the list to mock-up. The reason behind this, is that if more than one container is selected, the screen may become too cluttered. By restricting this to only one extra window, the designer is better able to focus their attention on each mock-up properly.

The first part of the mock-up process is to take the **TuilContainer** object associated with the container selected, and create the equivalent JSwing component from it. The following is the **ContainerType** options and their equivalent JSwing components:

- **window** ≡ JFrame
- **dialog** ≡ JDialog

There is a design consideration that has been used when creating the JFrame component, and which has been documented in the **container_type** XML type declaration. That is, where JFrame components are not modal by default, the property available for **TuilContainer** objects, **is_modal**, must always be *false*. We do this so that we can adhere to the default JSwing functionality as much as possible. If this property is ever *true*, it is ignored during mock-up generation.

Once the main container component has been created, a recursive process is started to fill out the JSwing container with all of the child controls. The goal here, is to create the JPanel that holds them and then add this into the container. The recursive process is called on the original set of **children**, which is basically a **child_group_type**. See section 4.3.3 for more details on this. The main thing to note, is that it is a way to group child controls in a particular orientation, and that these groups can be nested to increase the complexity of the layout. As well, it directly corresponds to the JAXB stub class **ChildGroupType**.

Each **ChildGroupType** is associated with its own JPanel, and its own GridBagLayout and set of GridBagConstraints [59]. Then each of the controls it contains is looped through and added into the JPanel based on when they are encountered. If a **ChildGroupType** is found, then a recursive call is made to generate the JPanel representing this group, which is then added into our current JPanel at the proper location based on the GridBagConstraints. Special considerations are made for the following:

- Button group - A set of current **TuideButtonGroup** objects is saved for the duration of the mock-up process, as each button group is encountered. This is so that when JRadioButtons or JCheckBoxes are found that are associated with a button group, they can be added to that group properly, even if they reside in different **ChildGroupType** objects.

- JButton - When a JButton is marked as a default button, the placement of it is in the bottom-right corner.

- JCombo, JList/JTable, JSlider - These components typically have a label associated with them. The **text** property of the TUIL specification for these types of controls is directly associated to a JLabel component. Since there are now two controls instead of one, a sub-JPanel is created to contain them. Then this JPanel is added into the current one.

- TuilGroupBox - This control is simply converted into a TuideTitledBorder and added as the border for the current JPanel.

- JTable/JList - All JTable objects are associated with a JScrollPane, which is set to enable the vertical and horizontal scrollbars as needed. This is helpful in case the number of items is very large, and thus would cause problems with the layout.

- JTextField/JTextArea - We are forced to give the text components a minimum size for the layout in order to get them to display. Otherwise, they will be shrunk to a small rectangle where no text can be entered into it.

- JTextArea - Requires a JScrollPane in case the text is too large for the text area displayed.

The mock-up generation process does a reasonable job at laying out the controls, in order to provide an idea of what the specification describes. The following subsection will show examples of how well the process works.

7.2 Mock-up from a Specification

This section compares an example specification to the mock-up generated from it. Listings 7.1 to 7.3 match up with Figures 7.1 to 7.3 respectively.

Listing 7.1: MainWindow.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<tuil_container id="MAIN_WINDOW" parent_id="" type="window" is_modal="true"
    use_default_button_placement="false"
        xmlns:xs="http:\www.w3.org\2001\XMLSchema.xsd" xmlns:tuil="../../TUIL/
            Specification/TUILContainer.xsd">
  <has_focus>true</has_focus>
  <is_enabled>true</is_enabled>
  <is_visible>true</is_visible>
  <text>The Main Window</text>

  <children orientation="vertical">
    <button name="first_button" id="FIRST_BUTTON" parent_id="MAIN_WINDOW" is_default=
        "false">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>First Button</text>
      <type>push</type>
    </button>

    <button name="second_button" id="SECOND_BUTTON" parent_id="MAIN_WINDOW"
        is_default="false">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Second Button</text>
      <type>push</type>
    </button>

    <button name="exit_button" id="EXIT_BUTTON" parent_id="MAIN_WINDOW">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Exit</text>
      <type>push</type>
    </button>
  </children>
</tuil_container>
```

Listing 7.2: FirstDialog.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<tuil_container id="FIRST_DIALOG" parent_id="MAIN_WINDOW" type="dialog" is_modal="
    true" use_default_button_placement="true"
        xmlns:xs="http:\www.w3.org\2001\XMLSchema.xsd" xmlns:tuil="../../TUIL/
            Specification/TUILContainer.xsd">
  <has_focus>true</has_focus>
  <is_enabled>true</is_enabled>
```

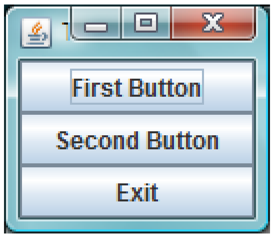


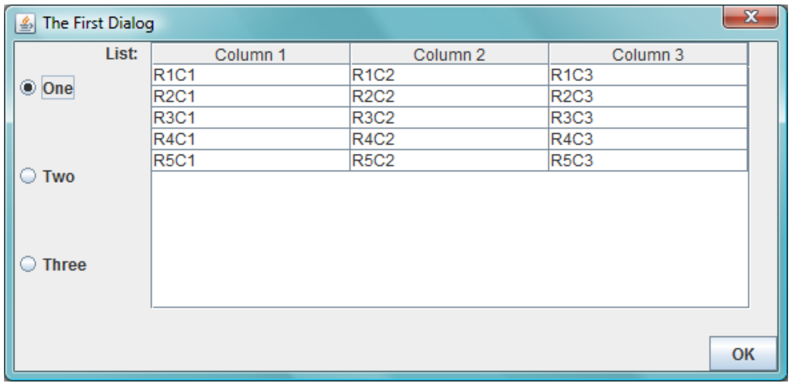
Figure 7.1: Mock-up of Main Window

```
<is_visible>true</is_visible>
<text>The First Dialog</text>

<children orientation="vertical">
  <child_group orientation="horizontal">
    <child_group orientation="vertical">
      <button name="first_button" id="FIRST_RADIO" parent_id="FIRST_DIALOG"
          is_default="false">
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>One</text>
        <type>radio</type>
        <is_checked>true</is_checked>
        <text_horiz_align>right</text_horiz_align>
        <button_group>ID_MAIN_CHOICE_GROUP</button_group>
      </button>

      <button name="second_button"  id="SECOND_RADIO" parent_id="FIRST_DIALOG"
          is_default="false">
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Two</text>
        <type>radio</type>
        <is_checked>false</is_checked>
        <text_horiz_align>right</text_horiz_align>
        <button_group>ID_MAIN_CHOICE_GROUP</button_group>
      </button>

      <button name="third_button" id="THIRD_RADIO" parent_id="FIRST_DIALOG"
          is_default="false">
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Three</text>
        <type>radio</type>
        <is_checked>false</is_checked>
        <text_horiz_align>right</text_horiz_align>
        <button_group>ID_MAIN_CHOICE_GROUP</button_group>
      </button>
    </child_group>

    <list_box name="big_listbox" id="BIG_LIST" parent_id="FIRST_DIALOG">
      <has_focus>false</has_focus>
        <is_enabled>false</is_enabled>
```

```
          <is_visible>true</is_visible>
          <text>List:</text>
          <multicolumn>true</multicolumn>
          <select_type>multiple</select_type>
          <rows>
            <row>
              <item>R1C1</item>
              <item>R1C2</item>
              <item>R1C3</item>
            </row>

            <row>
              <item>R2C1</item>
              <item>R2C2</item>
              <item>R2C3</item>
            </row>

            <row>
              <item>R3C1</item>
              <item>R3C2</item>
              <item>R3C3</item>
            </row>

            <row>
              <item>R4C1</item>
              <item>R4C2</item>
              <item>R4C3</item>
            </row>

            <row>
              <item>R5C1</item>
              <item>R5C2</item>
              <item>R5C3</item>
            </row>
          </rows>
          <columns>
            <item>Column 1</item>
            <item>Column 2</item>
            <item>Column 3</item>
          </columns>
          <count>5</count>
          <select_state>no_selection</select_state>
          <initial_selection>-1</initial_selection>
      </list_box>
    </child_group>

    <button name="ok_button" id="IDOK" parent_id="FIRST_DIALOG" is_default="true">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>OK</text>
      <type>push</type>
    </button>
  </children>
</tuil_container>
```

Listing 7.3: SecondDialog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```



Column 1	Column 2	Column 3
R1C1	R1C2	R1C3
R2C1	R2C2	R2C3
R3C1	R3C2	R3C3
R4C1	R4C2	R4C3
R5C1	R5C2	R5C3

Figure 7.2: Mock-up of First Dialog

```
<tuil_container id="SECOND_DIALOG" parent_id="MAIN_WINDOW" type="dialog" is_modal="
    true" use_default_button_placement="true"
        xmlns:xs="http:\www.w3.org\2001\XMLSchema.xsd" xmlns:tuil="../../TUIL/
            Specification/TUILContainer.xsd">
  <has_focus>true</has_focus>
  <is_enabled>true</is_enabled>
  <is_visible>true</is_visible>
  <text>The Second Dialog</text>

  <children orientation="vertical">
    <child_group orientation="horizontal">
      <child_group orientation="vertical">
        <button name="first_check" id="FIRST_CHECK" parent_id="SECOND_DIALOG"
            is_default="false">
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>One</text>
          <type>checkbox</type>
          <is_checked>false</is_checked>
          <text_horiz_align>right</text_horiz_align>
        </button>

        <button name="second_check" id="SECOND_CHECK" parent_id="SECOND_DIALOG"
            is_default="false">
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Two</text>
          <type>checkbox</type>
          <is_checked>false</is_checked>
          <text_horiz_align>right</text_horiz_align>
        </button>

        <button name="third_check" id="THIRD_CHECK" parent_id="SECOND_DIALOG"
            is_default="false">
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Three</text>
          <type>checkbox</type>
          <is_checked>false</is_checked>
          <text_horiz_align>right</text_horiz_align>
        </button>
      </child_group>

      <child_group orientation="vertical">
        <child_group orientation="horizontal">
          <label name="textbox_label" id="TEXTBOX_LABEL" parent_id="SECOND_DIALOG">
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Enter Text Here:</text>
            <text_vert_align>center</text_vert_align>
            <text_horiz_align>left</text_horiz_align>
            <text_bounds_interaction>none</text_bounds_interaction>
          </label>

          <text_box name="enter_textbox" id="ENTER_TEXTBOX" parent_id="SECOND_DIALOG"
              >
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <is_multiline>false</is_multiline>
            <is_readonly>false</is_readonly>
            <text>Testing...</text>
          </text_box>
        </child_group>

        <combo name="first_combo" id="FIRST_COMBO" parent_id="SECOND_DIALOG">
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Select Item:</text>
          <drop_arrow_side>left</drop_arrow_side>
          <is_sorted>false</is_sorted>
          <select_state>single_selection</select_state>
          <initial_selection>0</initial_selection>
          <count>5</count>
          <items>
            <item>Item 1</item>
            <item>Item 2</item>
            <item>Item 3</item>
            <item>Item 4</item>
            <item>Item 5</item>
          </items>
        </combo>
      </child_group>
    </child_group>

    <button name="ok_button" id="IDOK" parent_id="SECOND_DIALOG" is_default="true">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>OK</text>
      <type>push</type>
    </button>
  </children>
</tuil_container>
```

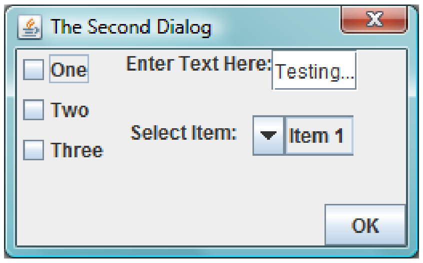


Figure 7.3: Mock-up of Second Dialog

The results of the comparisons are:

1. There are some issues with spacing, sizing, and alignment, but these are minimal in comparison with the overall outcome, which is that the mock-up layout algorithm displays the controls in the proper orientation and order, as found within the specification.

2. Control properties as defined in the specification are properly displayed within the mock-ups. For example, in the First Dialog, the radio button with text 'One' is initially selected, as per its specification.

3. The size of the container is defined by the number and orientation of the controls contained within it. Thus, the Main Window mock-up is smaller than what would typically be implemented, as noted by the title text being barely visible.

4. The multi-column list box control specified in the First Dialog is generated as a JTable. However, the JScrollPane the JTable is placed in is much larger than is required. We use generic sizing here to ensure that the control displays in a manner similar to an implementation.

The majority of the issues noted here are related to sizing, spacing, and alignment. In section 4.3.3 we discussed how we abstracted out as much of the visual aspects of the controls as possible, and instead use the layout mechanism put in place to handle actual layout. This was done to ensure that TUIL is a light-weight specification language. Thus, to include the exact values for spacing, alignment, sizing, etc, while improving the look of the mock-ups, would only increase the complexity of TUIL and be contrary to what we are attempting to achieve here.

Regardless of the issues found, the main goals we attempted to achieve with the mock-up generation were achieved. We can say that TUIL is expressive enough to specify the look and layout of a GUI, and that the mock-ups can give a designer some indication of whether their specification is specifying what they want it to. The issues found would be correctable during the implementation phase, without affecting our TUIL specification or the behaviour it defines.

Chapter 8

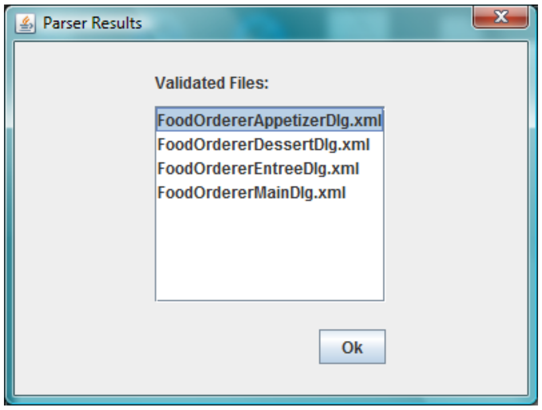
Application of Research

This chapter illustrates the approach developed in this work using a demonstration application, called Food Orderer. The purpose of Food Orderer is to allow a user to make menu selections for ordering food at a restaurant.

The design of the Food Orderer application is broken down into four windows:

- Main Dialog - Used to access the other three dialogs, and to enter any allergies that are applicable to the user.
- Appetizer Dialog - Allows the user to order from a selection of appetizers, with some variations allowed.
- Entrée Dialog - The user is given the choice of one main dish, from a selection of dishes, with a choice of sides.
- Dessert Dialog - The user is provided with a set of dessert options to choose from.

The specification for this application can be found in the supplementary files. The guideline of one container window to a specification file was followed. The first thing that is required is to load this specification into the TUIDE application, following the step-by-step instructions in Appendix D. When this is done the result dialog in Figure 8.1 is displayed.



FoodOrdererAppetizerDlg.xml
FoodOrdererDessertDlg.xml
FoodOrdererEntreeDlg.xml
FoodOrdererMainDlg.xml

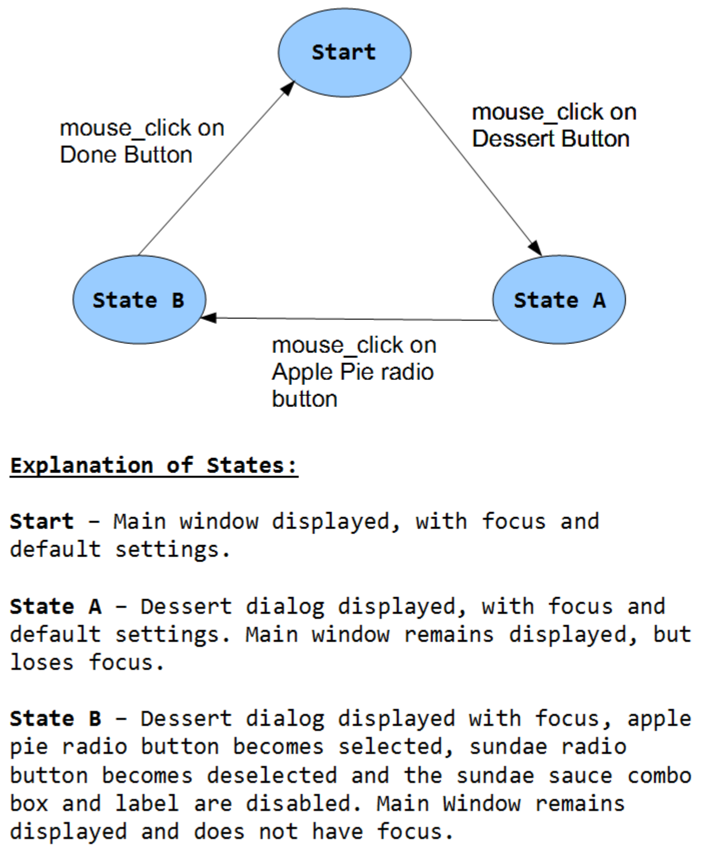
Figure 8.1: Specification Result Dialog

After the specification is successfully loaded into TUIDE, it can be used to generate test cases and mock-ups respectively.

8.1 Test Case Generation

Test case generation is described in chapter 5. Here we discuss this with respect to the Food Orderer application. The user sees very little when generating a graph or set of test cases, however they are provided with result messages denoting the outcome of each step. The graph generation step is successful here. Please see Figure 8.2 for an example partial graph for this specification.

When the *Generate Test Cases* menu is clicked, the user is prompted with the container selection dialog, which decides what containers will have test cases generated for them. As can be seen in Figure 8.3, all containers within Food Orderer are selected. Then the test cases for this specification are all generated successfully.


Start
mouse_click on Done Button
mouse_click on Dessert Button
State B
State A
mouse_click on Apple Pie radio button

Explanation of States:

Start – Main window displayed, with focus and default settings.

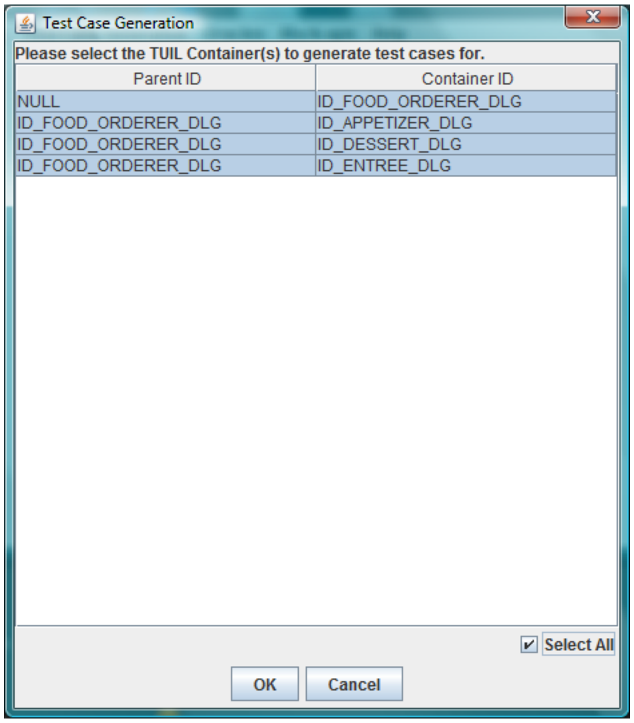
State A – Dessert dialog displayed, with focus and default settings. Main window remains displayed, but loses focus.

State B – Dessert dialog displayed with focus, apple pie radio button becomes selected, sundae radio button becomes deselected and the sundae sauce combo box and label are disabled. Main Window remains displayed and does not have focus.

Figure 8.2: Food Orderer Partial Graph Example

For the Food Orderer specification, the following is the breakdown of the number of test cases generated per container. Please see Appendix I for a few examples of these generated test case files.

- Main Dialog - 21
- Appetizer Dialog - 23
- Entrée Dialog - 39
- Dessert Dialog - 6



Please select the TUIL Container(s) to generate test cases for.

Parent ID	Container ID
NULL	ID_FOOD_ORDERER_DLG
ID_FOOD_ORDERER_DLG	ID_APPETIZER_DLG
ID_FOOD_ORDERER_DLG	ID_DESSERT_DLG
ID_FOOD_ORDERER_DLG	ID_ENTREE_DLG

Figure 8.3: Container Selection Dialog

The test cases generated were examined with respect to how well they covered the set of events provided by the specification. Table 8.1 gives the event coverage breakdown for the Main Dialog. Please see Appendix J for the other event coverage tables.

Table 8.1: Main Dialog Event Coverage

Control/Container	**Event**	**Usage Count**
`ID_APPETIZER_BUTTON`	`mouse_click`	23
`ID_ENTREE_BUTTON`	`mouse_click`	39
`ID_DESSERT_BUTTON`	`mouse_click`	6
`ID_ALLERGY_TEXTBOX`	`text_change`	20
`ID_PLACE_ORDER_BUTTON`	`mouse_click`	21
`ID_FOOD_ORDERER_DLG`	`initialize_container`	0

The following are the results of examining the event coverage:

1. 84% of events within the specification were used at least once, with some being used much more than this. The other results below explain why some events were not used.

2. No **initialize_container** actions were found in the generated test cases. This may be due to the fact that if this action will cause a cycle within the graph, as in starting and ending in the same state, it will be ignored.

3. In some cases, either the **return_key** action or the **mouse_click** action was never fired for a particular control. This may be due to the randomness of **TuilArc** selection, as described in section 5.4. Another reason could be that the action ignored was not on one of the shortest paths within the graph.

4. Within the *Entrée* dialog, the steak cook option slider's **mouse_click** action was never fired. The reason for this is currently unknown, but may result from it not lying on one of the shortest paths in the graph.

5. The **text_change** action was used more often than it needed to be for proper testing. This may result from the GUID generated each time a **text_change** action is encountered during the graph modeling process.

Each time test case generation occurred, it resulted in the same number of test cases per container. The contents of the test case files, however, were never guaranteed to stay the same, which is as designed. There are two reasons why this randomness occurs:

1. The random selection between **TuilArcs** could return a different index each time.

2. The paths through the graph may be returned in a different order from previous.

8.2 Implementation

This section introduces the implementation of the Food Orderer application, which will be used for the following sections in this chapter to:

- Compare the generated mock-ups against.
- Be used as the AUT during the application of oracles.
- Be used as the basis for seeding errors into the AUT for testing the oracles ability to find them.

The implementation of the application was done using the specification as a guide for layout, types of controls, events to hook-up, and default settings for each container. We followed the guideline that the **id** attribute of each control and container within the specification be mapped to the setName/getName methods of the equivalent Swing component.

The screen captures of the four dialogs within the application can be seen in Figures 8.4 to 8.7.

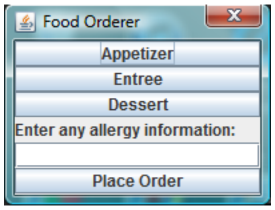
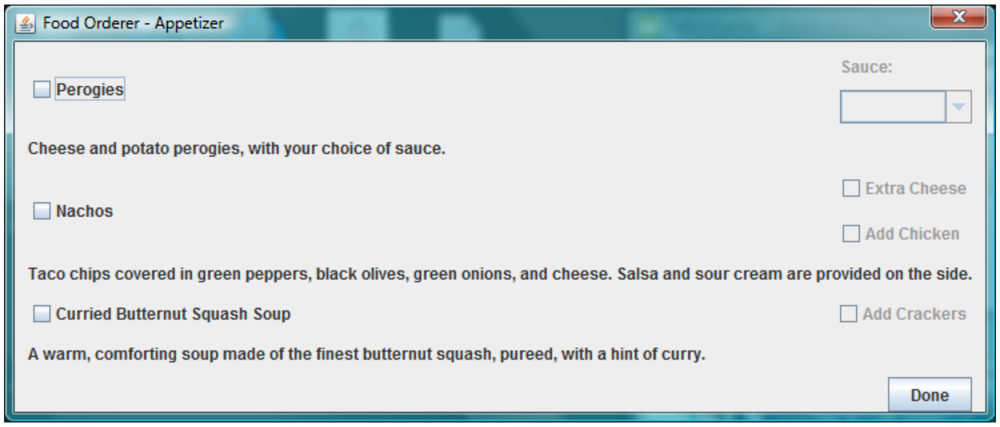


Figure 8.4: Main Dialog Implementation



Cheese and potato perogies, with your choice of sauce.

Taco chips covered in green peppers, black olives, green onions, and cheese. Salsa and sour cream are provided on the side.

A warm, comforting soup made of the finest butternut squash, pureed, with a hint of curry.

Figure 8.5: Appetizer Dialog Implementation

8.3 Mock-ups

The TUIDE mock-ups, as discussed in chapter 7, can be generated by selecting each container individually from a dialog similar to Figure 8.8. Figures 8.9 to 8.12 are shown in response to these selections.

Comparing these mock-ups to the implemented dialog screen captures from the previous section, we can see that overall the mock-ups generate a good representation of the containers. There are some issues to do with alignment and spacing during layout, as well as setting preferred sizes of containers and some controls, however, these are relatively minor. The general purpose of the mock-ups, which is to give the designer an idea of what the container could look like, has been met successfully.

8.4 Oracles

The oracles are discussed within Chapter 6. For the Food Orderer application, the amount of time required to apply all 89 test cases against the runnable JAR file for the AUT was approximately fifteen minutes.

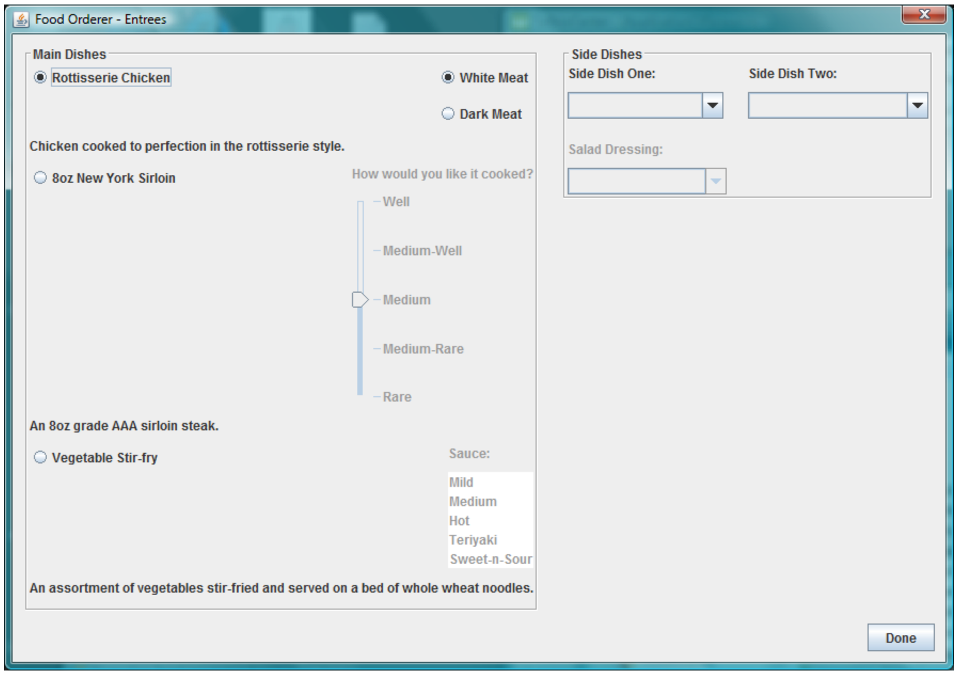


Figure 8.6: Entrée Dialog Implementation

Please see Appendix K for some generated oracle result files, using the proper implementation of the Food Orderer application. As a result, the test cases were all successful, except for the inability to catch the close of a dialog to get the proper end state for the test case. This is a known problem within the oracle side of the AUT implementation, where the closing event on a Window is not caught and thus, the Window is not removed from the IAUTEventListener list properly. Please refer to

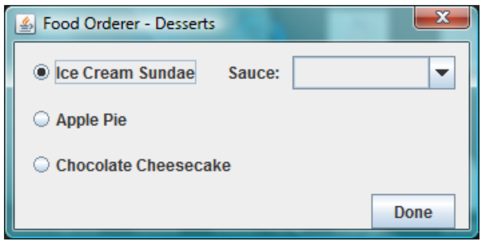
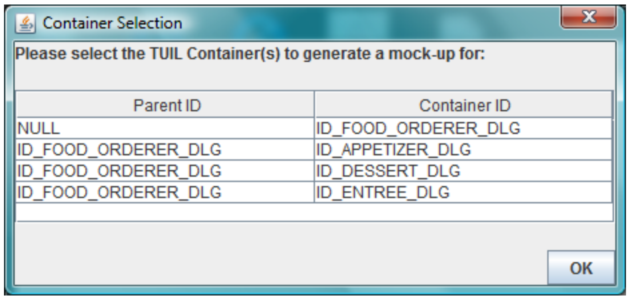


Figure 8.7: Dessert Dialog Implementation



Container Selection

Please select the TUIL Container(s) to generate a mock-up for:

Parent ID	Container ID
NULL	ID_FOOD_ORDERER_DLG
ID_FOOD_ORDERER_DLG	ID_APPETIZER_DLG
ID_FOOD_ORDERER_DLG	ID_DESSERT_DLG
ID_FOOD_ORDERER_DLG	ID_ENTREE_DLG

Figure 8.8: Container Selection Dialog for Mock-ups

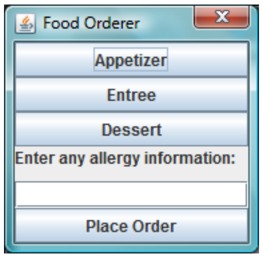
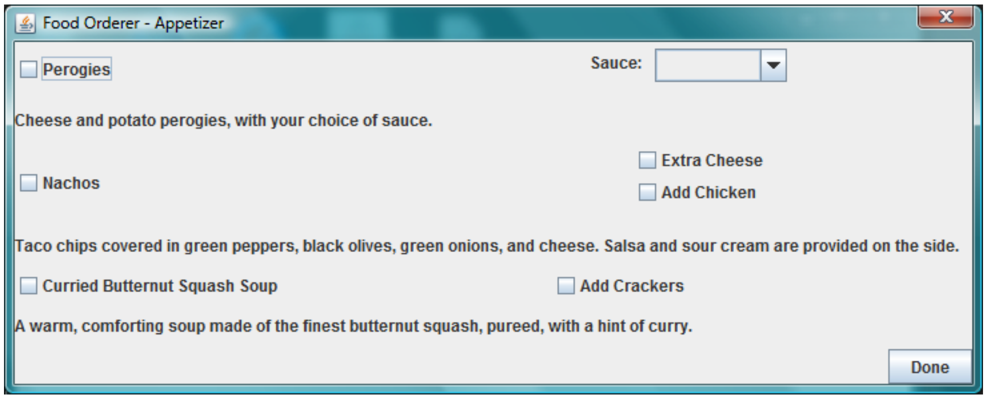


Figure 8.9: Main Dialog Mock-up

section 6.2.2 for details on this listener and the state request process.

8.5 Implementation Errors

Two copies were made of the Food Orderer implemented solution, in order to be able to seed each copy with certain faults to test for. One copy was seeded for faults within the Main dialog, while the other copy was seeded for faults within the Appetizer, Entrée, and Dessert dialogs. The reason for dividing them up is that only the first fault found is reported during the state comparison and logging phase. If we only used one version of the AUT for seeding faults, the faults in the Main dialog would be the only ones reported. Many of the faults seeded were based on mistakes that can easily happen during software implementation, such as copy-paste errors, etc. The following is a list of all faults seeded:


Food Orderer - Appetizer

Perogies

Sauce:

Cheese and potato perogies, with your choice of sauce.

Nachos

Extra Cheese

Add Chicken

Taco chips covered in green peppers, black olives, green onions, and cheese. Salsa and sour cream are provided on the side.

Curried Butternut Squash Soup

Add Crackers

A warm, comforting soup made of the finest butternut squash, pureed, with a hint of curry.

Done

Figure 8.10: Appetizer Dialog Mock-up

1. The Allergy text box within the Main dialog is disabled on start-up, even though it should be enabled.

2. The Allergy text box label in the Main dialog contains an extra space at the end.

3. The Dessert button will open the Appetizer dialog instead of opening the Dessert dialog.

4. The Sundae Sauce combo in the Dessert dialog is disabled on startup by checking for the wrong dessert being selected. Instead of checking for Sundae being the default dessert, which it is, it checks for the Cheesecake option.

5. The Soup Crackers check box in the Appetizer dialog uses the Nachos check box being selected to determine whether it should enable itself. This is wrong, as it should use the Soup check box. As well, the check to see if it should enable/disable itself is only done when the Soup check box is selected or deselected.

6. The Steak radio button in the Entrée dialog should disable the chicken sub-radio-buttons for white and dark meat, however here it enables them.

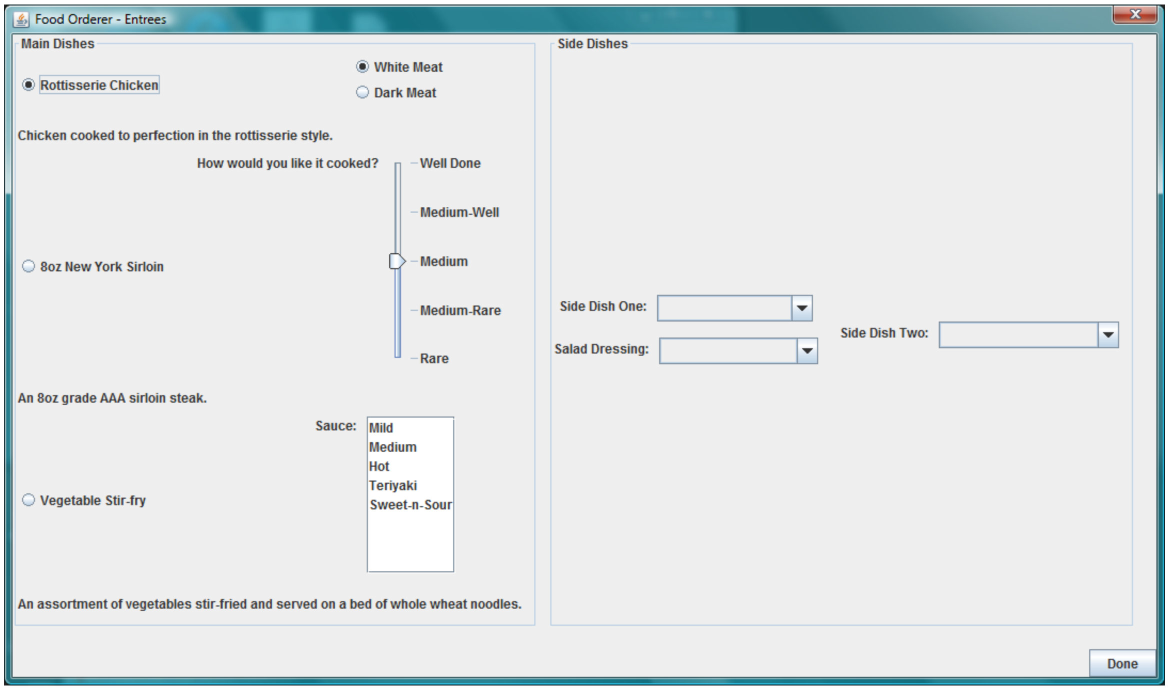


Figure 8.11: Entrée Dialog Mock-up

7. Selections in the Side Dish One combo in the Entrée dialog enable/disable the Dressing combo in the opposite manner. When Garden Salad item is selected, the Dressing combo is disabled, and when any other Side Dish One is selected, the Dressing combo is enabled.

8. Added an extra item, *Cherry*, to the Sundae Sauce combo in the Dessert dialog.

9. Removed the *Cajun Sauce* item from the Perogie Sauce combo in the Appetizer

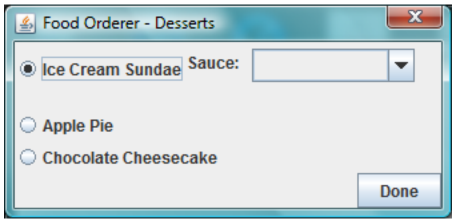


Figure 8.12: Dessert Dialog Mock-up

Table 8.2: Faults Seeded Into AUT

Fault	Java File	Line Number	Nature	Group	Compile Type
1	MainDlg	138	Incorrect boolean	N/A	Separate
2	MainDlg	129	Bad spelling	N/A	Separate
3	MainDlg	158-159	Copy-paste	N/A	Separate
4	DessertDlg	167	Incorrect enum	Enable/ Disable	Grouped
5	AppetizerDlg	342 and 452	Wrong control	Enable/ Disable	Grouped
6	EntreeDlg	657-658, 1283-1284	Incorrect boolean	Enable/ Disable	Grouped
7	EntreeDlg	1353-1361	Incorrect booleans	Enable/ Disable	Grouped
8	DessertDlg	136	Extra item	List Items	Grouped
9	AppetizerDlg	159	Missing item	List Items	Grouped
10	EntreeDlg	407	Bad spelling	List Items	Grouped

dialog.

10. Renamed the *Sweet-n-Sour* item, of the Stirfry Sauce list in the Entrée dialog, to be *Sweet and Sour*.

Table 8.2 lists the faults described above, in order and using more details.

As can be noted from the table, several compilations were required to prevent contamination between different faults. The generated JAR files were also named appropriately to prevent confusion. Faults within the three child dialogs were done in sets. One set represented enable/disable errors, while another represented list item errors.

Some of the Oracle results can be found within Appendix L. Table 8.3 provides a summary of these results.

As can be seen in Table 8.3, all seeded faults were successfully found. In some cases, where subsequent events were dependent upon the original outcome of the

Table 8.3: Oracle Results for Seeded Faults

Fault	**# of Test Cases That Found It**	**Test Case for Appendix L Result**
1,2	89	Main Dialog's 000.xml
3,4	6	Dessert Dialog's 004.xml
5	15	Appetizer Dialog's 011.xml, 013.xml, 016.xml
6	7	Entrée Dialog's 036.xml
7	22	Entrée Dialog's 036.xml
8	6	Dessert Dialog's 004.xml
9	23	Appetizer Dialog's 016.xml
10	39	Entrée Dialog's 036.xml

event with the fault, more errors were generated. We can compare each faulty result from Appendix L to the result from the proper implementation in Appendix K.

Thus, with all the aspects of this research applied to the Food Orderer demonstration application, we can see that the research goals have been met, and its usage is valid with respect to its current capabilities.

Chapter 9

Conclusion

The original goals of this research were the following:

1. Create a specification language that would describe both the GUI, as well as its behaviour, in a light-weight manner, and in a format that was easily configurable and transferable. The language would then be usable for multiple purposes within the realm of testing and GUI development.

2. Develop a method to automatically generate test cases for a particular specification.

3. Develop a method by which the test cases generated would be applied to an implementation of the GUI, called the Application Under Test or AUT, and results gathered.

4. Develop a method to take a particular specification and generate a mock-up of what different containers within the GUI could look like if implemented.

Based on the results noted in the previous chapters, all of these goals have been met. We have created a prototype of the specification language, which we have called

TUIL, or Testable User Interface Language. To go along with this, we have created a prototype application, called TUIDE, or Testable User Interface Development Environment, which combines the ability to generate test cases, run Oracles, and generate mock-ups.

The most telling results of this research include the following:

- The specification language does work to describe the functionality, look, and layout of a GUI. However, it is a cumbersome language to read and write manually, thus decreasing the usability of the language for the specifier.

- The mock-ups generated a reasonably accurate visual representation of the actual GUI, barring issues with alignment and sizing.

- The oracles were able to catch all faults seeded into the demonstration application. However, more testing would need to be performed in order to determine how well it can catch other types of faults.

- The test case generation process created test cases that not only were able to help accurately find the faults when applied by the oracles, they also covered the majority of events within the specification, except for some considerations, which are provided below. The Dijkstra's algorithm used in this research did not provide full graph coverage with respect to generating test cases for all events. Some of the missed events, however, were independent to the algorithm itself, and instead were affected by our own design considerations, also noted below. If these had been implemented differently, the event coverage would have been higher. As well, Dijkstra's shortest path algorithm is a good starting point for generating test cases, as it is simple to implement and use, and as can be seen in this research, even with its limitations, it was able to do a reasonable job.

- Some events are not applied if the start and end node for the action are the same. This is to prevent adding cycles to the graph. One example of this would be the **initialize_container** action.
- When multiple actions lead from the same start node in the graph to the same end node, we randomly select only one of these to apply. This can cause some actions to be *starved* of use.
- Due to Dijkstra's shortest path algorithm, some actions may lie on the non-shortest paths and thus, will not be included in the test cases generated.

There is great potential for this research. Currently, however, it is in the prototype stage and requires work in order to be more usable in practice. Improvements, as outlined in the next section, would increase this research from the prototype level to that of a proper toolkit.

9.1 Future Work

Since the TUIL language and TUIDE program are prototypes, there is still much room for improvements and features. These can be broken up into four separate groupings:

1. Implementation specific improvements - Requires modifications to different aspects of the tools and the language to increase functionality.
2. Expanded functionality - This represents larger pieces of work to increase usability and portability.
3. Tool support - Describes different plug-ins or add-ons that could be created

to be incorporated into applications such as Eclipse, Visual Studio, etc, or by further development of our own TUIDE tool.

4. Further testing

The following is the list of improvements that have been flagged as implementation specific, in order to increase the capabilities of this research:

1. Update the TUIL language to increase the number of container types, control types, and events/actions/reactions supported. Some of these would include adding in panel, canvas, menu, and menu item types, and scrolling events. This would then require updating both test case generation, as well as oracles, to handle the new additions.

2. Expand TUIL allow it to represent the model state of the GUI, to further improve the testing process. Subsequent changes would be required to test case generation and oracles.

3. The text bounds interaction property in the specification for TUIL labels is not directly assignable to a JLabel property. Consideration should be given to extending the JLabel class to add this in, or implement some sort of draw capability for this. This affects mock-ups, test case generation, and Oracles.

4. The knob orientation property in the specification for TUIL sliders is not directly assignable to a JSlider property. Again, we would either have to extend on the JSlider class to implement this property, or handle it through draw/paint capabilities. This also affects all three uses of the TUIL specification.

5. Fine tune the mock-up generation process to improve layout and alignment of controls.

6. Multiple default buttons may result in incorrect layout during mock-up generation. This would need to be tested more fully.

7. Currently, only one arc between two state nodes is selected randomly for the test case. If multiple arcs exist, and their shortest path weighting is equal, then we should generate a test case per arc. This could possibly be done using multigraph or pseudograph.

8. More work needs to be done to ensure the software will be able to generate test cases for specifications where non-modal containers, other than the main container, are allowed.

9. The code action initialize_container currently only works with default states during test case generation. Modifications should be made to ensure that saved state can also be worked with. This may require changes to the TUIL language as well, to denote whether a container allows for saved state initialization or not. This would work well with the model state future work item noted earlier.

10. Currently, combo boxes, list boxes, and slider controls all have statically defined sets of items. Many applications, however, use dynamically allocated lists to display information to the user. To this end, several additions to the functionality of these control types are required:

    - The control would need to specify whether its item set is defined as static or dynamic.
    - If the item set is to be static, then the burden of item selection should fall to the test case generation process, as opposed to the Oracles, which it is currently. This would simplify the oracle process here.

- Dynamic item sets require more consideration, as they complicate how we generate test cases. It is difficult to generate a test case containing a **select_item** action if we do not know that there is actually an item to select ahead of time. This may put the burden back on the oracle side, since at this point, we would know what the item set is.

11. When running Oracles, there is an existing issue whereby when a window is closed, it is still able to receive our state request event.

12. When a combo box, list box, or slider control item set is compared during the Oracles, it is assumed that the item sets are in the same order. We do not currently define whether a set of items is sortable within the specification language. This should be added, and the items organized appropriately before comparison.

13. The **MOUSE_DRAG** mouse action is not currently handled within the Oracles.

14. If an action is being applied to a *TuideTitledBorder*, then it is currently ignored within the Oracles, as it is not an underlying *Component* object. This needs to be resolved.

15. Expand the logging capabilities of Oracles to provide even more in depth results about failures.

16. Multiple column list states should contain the set of column names for completion. This would enable us to declare a problem if the implemented version has differing column names from the specification.

17. Mouse actions applied during the oracles cause the mouse clicks to occur on the center of the control in question. Perhaps this location should be more random, depending on the control type.

18. Investigate ways to improve the graph modeling and test case generation process. Different graph coverage schemes may find paths that Dijkstra's shortest path algorithm did not, thus increasing the capabilities of our test cases and oracles.

19. Reduce the number of states generated for a **text_change** action by investigating the type of text to use to test this control properly.

20. Expand the action and reaction capabilities by allowing the specifier to add in constraints on whether the action or reaction is feasible, given the states of other controls/containers. This would require changes to the test case generation process as well.

21. Investigate different oracle application methods, including not using the AUT to contain the bulk of the oracle functionality.

22. Update the Food Orderer application mentioned in Chapter 8 for the new functionality and retest. As well, create two more demonstration applications to compare results with.

The following describes the expanded functionality:

1. Put in the ability to allow multiple specifications to concurrently exist within TUIDE. This, in turn, would require that multiple test graphs, sets of test cases, and JAR files be concurrently allowed. Changes would need to be made

to ensure that when oracles are run, we choose what test case set and JAR file are to be used beforehand. As well, when generating test cases, we would need to add the ability to select what graph to generate from and what containers to generate for.

2. Currently TUIDE only has the ability to mock-up a single TUIL container at a time. Consider allowing multiple to be selected and mocked-up. This would need to take into account screen real-estate.

3. The only way to close a mock-up dialog is to click the X button in the title bar, or to hit ESC or ALT+F4. It would be nice to have another more built-in way to do this.

4. Add in the ability to mock-up event interactions and behaviour, so that the specifier can also test this as the specification is being developed.

5. Consider porting TUIL to C++ or C# to make it even more powerful.

6. Potentially put in the ability to automatically generate the implementation class code for the specified GUI in different implementation languages.

The following deals with tool support for the different aspects of this research. These would include:

1. A plug-in or add-on to handle creation of TUIL specifications in a visual manner, and generation of mock-ups. These two parts compliment each other, and would work well as a single plug-in/add-on.

2. A plug-in or add-on to encompass test case generation and oracle application, as again, these two would work well together.

3. Further development to the TUIDE application itself, to generate the specification in a visual manner rather than manually. This is important to increase usability of the language as a whole.

References

[1] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Using a goal-driven approach to generate test cases for GUIs," in *Proceedings of the 21st International Conference on Software Engineering*, pp. 257–266, May 1999.

[2] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Automated test oracles for GUIs," in *SIGSOFT '00/FSE-8: Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, (New York, NY, USA), pp. 30–39, ACM Press, 2000.

[3] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage critiria for GUI testing," in *Proceedings of the 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)*, pp. 256–267, Sept 2001.

[4] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Hierarchical GUI test case generation using automated planning," *IEEE Transactions on Software Engineering*, vol. 27, no. 2, pp. 144–155, 2001.

[5] A. M. Memon, "GUI testing: Pitfalls and process," *IEEE Computer*, vol. 35, pp. 90–91, Aug 2002.

[6] A. M. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should i use for effective GUI testing?," in *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)*, 2003.

[7] A. M. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing," in *Proceedings of the 10th Working Conference on Reverse Engineering*, Nov 2003.

[8] A. M. Memon, "An event-flow model of GUI-based applications for testing," *Software Testing, Verification and Reliability*, 2007.

[9] C. Campbell and M. Veanes, "State exploration with multiple state groupings," in *Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05)*, Abstract State Machines, (Paris, France), pp. 119–130, March 8-11 2005.

[10] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. SE-4, pp. 178–187, Sept 1978.

[11] R. K. Shehady and D. P. Siewiorek, "A method to automate user interface testing using variable finite state machines," in *FTCS'97: Proceedings of the 27th Annual International Symposium on Fault-Tolerant Computing*, pp. 80–88, June 1997.

[12] R. E. K. Stirewalt, S. Rugaber, and G. D. Abowd, "Automating the design of specification interpreters," GVU Technical Report GIT-GVU-96-14, Georgia Institute of Technology, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, 30332-0280, 1996.

[13] F. Belli, "Finite-state testing and analysis of graphical user interfaces," in *Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 34–43, Nov 2001.

[14] A. C. R. Paiva, N. Tillmann, J. C. P. Faria, and R. F. A. M. Vidal, "Modeling and testing hierarchical GUIs," in *Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05)*, Abstract State Machines, (Paris, France), pp. 329–344, March 8-11 2005.

[15] L. White, "Controlling the effects of complexity in software testing (testing of GUI systems)," in *Workshop on Software Assessment (WOSA), Part of the 2001 International Symposium on Software Reliability Engineering (ISSRE'01)*, (Hong Kong), Nov 2001.

[16] L. Apfelbaum and J. Schroeder, "Reducing the time to thoroughly test a GUI," in *Proceedings of the 11th International Software Quality Week Conference (QW'98)*, (Sheraton Palace Hotel, San Francisco, CA, USA), May 1998.

[17] C. Erickson, R. Palmer, D. Crosby, M. Marsiglia, and M. Alles, "Make haste, not waste: Automated system testing," in *Extreme Programming and Agile Methods - XP/Agile Universe 2003*, vol. 2753/2003 of *Lecture Notes in Computer Science*, pp. 120–128, Springer Berlin/Heidelberg, September 2003.

[18] M. Caswell, V. Aravamudhan, and K. Wilson, "Introduction to jfcUnit." `http://jfcunit.sourceforge.net/`, Last visited on September 24, 2014.

[19] "Jemmy framework." `https://jemmy.java.net/`, Last visited on September 24, 2014.

[20] Oracle Corporation and/or its affiliates, "Class Robot." http://docs.oracle.com/javase/7/docs/api/java/awt/Robot.html, Last visited on September 24, 2014.

[21] T. Wall, "Getting started with the Abbot java GUI test framework." http://abbot.sourceforge.net/doc/overview.shtml, Last visited on September 24, 2014.

[22] Jalian Systems Pvt. Ltd., "Marathon: Opensource java GUI testing tool." http://marathontesting.com/marathon/, Last visited on September 24, 2014.

[23] D. K. Peters and D. L. Parnas, "Using test oracles generated from program documentation," *IEEE Transactions on Software Engineering*, pp. 161–173, March 1998.

[24] D. K. Peters, M. Lawford, and B. T. con y Widemann, "An IDE for software development using tabular expressions," in *Newfoundland Electrical and Computer Engineering Conference*, (St. John's, NL, Canada), Nov 2007.

[25] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[26] "XML." http://en.wikipedia.org/wiki/XML, Last visited on February 17, 2015.

[27] "Extensible markup language (XML)." http://www.w3.org/XML/, Last visited on February 17, 2015.

[28] J. Bishop and N. Horspool, "Developing principles of GUI programming using views," *SIGCSE Bulletin*, vol. 36, pp. 373–377, March 2004.

[29] J. Bishop, "Multi-platform user interface construction a challenge for software engineering-in-the-small," in *Proceedings of the 28th International Conference on Software Engineering*, pp. 751–760, May 20-28 2006.

[30] "Reflection." `https://msdn.microsoft.com/en-us/library/cxz4wk15(v=vs.80).aspx`, Last visited on February 17, 2015.

[31] W. Xin, "XML specification of GUI," masters of science: computer systems engineering programme, Technical University of Denmark, Technical University of Denmark, Informatics and Mathematical Modelling Building 321, DK-2800 Kongens Lyngby, Denmark, March 2006.

[32] M. A. A. Salam, A. E. Keshk, N. A. W. Ismail, and H. M. Nassar, "Specification-driven automated testing of java swing GUIs using XML," in *5th International Conference on Information and Communications Technology (ICICT)*, pp. 84–88, 2007.

[33] M. A. A. Salam, A. E. Keshk, N. A. W. Ismail, and H. M. Nassar, "Automated testing of java menu-based GUIs using XML visual editor," in *International Conference on Computer Engineering and Systems (ICCES)*, pp. 313–318, 2007.

[34] M. Tubishat, I. Alsmadi, and M. Al-Kabi, "Using XML files to document the user interfaces of applications," in *5th IEEE GCC Conference and Exhibition*, pp. 1–4, 2009.

[35] Microsoft, "XAML overview (WPF)." `http://http://msdn.microsoft.com/en-us/library/ms752059(v=vs.110).aspx`, Last visited on September 24, 2014.

[36] "Extensible application markup language." `http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language`, Last visited on September 24, 2014.

[37] Soyatec, "eFace." `http://www.soyatec.com/eface/`, Last visited on September 24, 2014.

[38] Mozilla, "XUL." `http://developer.mozilla.org/En/XUL`, Last visited on September 24, 2014.

[39] F. Sauer, "XMLTalk." `http://sourceforge.net/projects/xmltalk/`, Last visited on September 24, 2014.

[40] F. Sauer, "XMLTalk: A framework for automatic GUI rendering from XML specs," *Java Report*, pp. 16–23, October 2001.

[41] The Forms Working Group, "XForms." `http://www.w3.org/Markup/Forms/`, Last visited on September 24, 2014.

[42] XIML Forum, "XIML." `http://www.ximl.org/`, Last visited on September 24, 2014.

[43] H. Tran, "Test generation using model checking." Report from CSC2108 Automated Verification, Fall'00, Instructor: Marsha Chechik, Department of Computer Science, University of Toronto, 2000.

[44] M. B. Dwyer, V. Carr, and L. Hines, "Model checking graphical user interfaces using abstractions," in *Proceedings of the European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pp. 244–261, 1997.

[45] S. R. Ganov, C. Killmar, S. Khurshid, and D. E. Perry, "Test generation for graphical user interfaces based on symbolic execution," in *In Proceedings of the 3rd International Workshop on Automation of Software Test (AST'08)*, (New York, NY, USA), pp. 33–40, 2008.

[46] X. Lee, "JavaScript + DOM: Intro to event-based programing." `http://xahlee.info/js/events.html`, Last visited on September 24, 2014.

[47] "Mealy machine." `http://en.wikipedia.org/wiki/Mealy_machine`, Last visited on February 10, 2015.

[48] Oracle Corporation and/or its affiliates, "JAXB Reference Implementation." `http://jaxb.java.net/`, Last visited on September 24, 2014.

[49] Oracle Corporation and/or its affiliates, "JAXB Tutorial." `http://jaxb.java.net/tutorial/`, Last visited on September 24, 2014.

[50] Oracle Corporation and/or its affiliates, "JAXB RI 2.2.4 - binding compiler (xjc)." `http://jaxb.java.net/2.2.4/docs/xjc.html`, Last visited on September 24, 2014.

[51] IBM, "xjc command for JAXB applications." `http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.websphere.express.doc%2Finfo%2Fexp%2Fae%2Frwbs_xjc.html`, Last visited on September 24, 2014.

[52] Oracle Corporation and/or its affiliates, "JAXBContext Javadoc." `http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/JAXBContext.html`, Last visited on September 24, 2014.

[53] Merriam-Webster, Inc., “Graph.” `http://www.merriam-webster.com/dictionary/graph`, Last visited on September 24, 2014.

[54] S. Wilson, “annas.” `https://sites.google.com/site/annasproject/`, Last visited on September 24, 2014.

[55] Oracle Corporation and/or its affiliates, “The event dispatch thread.” `http://docs.oracle.com/javase/tutorial/uiswing/concurrency/dispatch.html`, Last visited on September 24, 2014.

[56] Oracle Corporation and/or its affiliates, “Worker threads and SwingWorker.” `http://docs.oracle.com/javase/tutorial/uiswing/concurrency/worker.html`, Last visited on September 24, 2014.

[57] Oracle Corporation and/or its affiliates, “SwingUtilities Javadoc.” `http://docs.oracle.com/javase/7/docs/api/javax/swing/SwingUtilities.html`, Last visited on September 24, 2014.

[58] Houghton Mifflin Company, “The american heritage dictionary of the english language.” `http://www.thefreedictionary.com/mock-up`, Last visited on September 24, 2014.

[59] Oracle Corporation and/or its affiliates, “Class GridBagLayout.” `http://download.java.net/jdk7/archive/b123/docs/api/java/awt/GridBagLayout.html`, Last visited on September 24, 2014.

Appendix A

TUIL Action/Reaction Definitions

Table A.1: Breakdown of Actions

Name	Type	Description
mouse_click	mouse	A single mouse click
l_mouse_click	mouse	A single left button click
r_mouse_click	mouse	A single right button click
mouse_dbl_click	mouse	A double mouse click
l_mouse_dbl_click	mouse	A double left button click
r_mouse_dbl_click	mouse	A double right button click
mouse_button_down	mouse	A mouse button press
mouse_button_up	mouse	A mouse button release
mouse_drag	mouse	Dragging the mouse cursor across the screen
ctrl_plus_mouse_click	key+mouse	Holding the Ctrl key and clicking the mouse
shift_plus_mouse_click	key+mouse	Holding the Shift key and clicking the mouse
return_key	key	The Enter or Return key being pressed and released
tab_key	key	The Tab key being pressed and released
select_item	item	Select an item within a combo/list
deselect_item	item	Deselects a previously selected item in a combo/list
toggle_check_on	button	Used in conjunction with another action that toggle's the selected state of the radio/check button. Represents when it is toggled on or selected, and is paired with the reactions that should result when this happens.
toggle_check_off	button	Used in conjunction with another action that toggle's the selected state of the radio/check button. Represents when it is toggled off or deselected, and is paired with the reactions that should result when this happens.
initialize_container	code	Called from the code behind the GUI to initialize the container when it is first created and displayed.

. . .

Name	Type	Description
update_container	code	Similar to initialize_container, but it can occur while the container is already displayed.
text_action	text	Combines the text to be applied along with the type of text action to perform. Currently only text_change is allowed.
select_specific_action	item	Represents an item_action that is performed on either a specific item within a combo/list, or on any item but that specific one.

Table A.2: Breakdown of Reactions

Name	Type	Description
enable	base	Enables a control/container for user interaction.
disable	base	Disables a control/container to prevent user interaction.
focus_on	base	Gives a control/container focus. An example is that a text control can only have text entered into it when it has focus.
hide	base	Hides a control/container from view.
show	base	Shows a control/container so that it is visible to the user.
toggle_check	button	Represents when an action is performed on a radio button or check-box that would cause it to be selected/deselected, meaning the selected state is toggled. When a toggle_check reaction is included for one of these controls, it is advised to have both the toggle_check_on and toggle_check_off actions also included, as they go together. These two actions are explained in the previous table.

...

Name	Type	Description
radio_group_select	button	Implies that a button within a button group was selected, and all other buttons within that group should be deselected. These reactions are handled differently, in that their associated event_object id parameter is set to the value of the button_group attribute for said group.
radio_group_enable	button	Enables all buttons within a button group. Again, the event_object id parameter is set to the button_group attribute value for said group.
radio_group_disable	button	Disables all buttons within a button group. The event_object id parameter here is also set to the button_group attribute value for said group.
show_list	combo	The reaction when the drop down list of a combo-box is displayed to the user.
remove_selection	item	Decreases the number of selected items in a list by one. This is not allowed for combo-box or slider controls, as these only allow single selection.
single_selection	item	Selects one item within a list-type control. This is allowed for all list, combo-box, and slider controls.
multi_selection	item	Increases the number of selected items in a list by one. This is only allowed for list controls that are defined as multiple selection, and that have at least one more item available for selection.
open_container	container	Displays a container to the user. At the moment, this means with the default settings, as defined in the specification, loaded.
close_container	container	Closes a container and hides it from the user. A typical example of this would be when the user clicks on an Ok button in a dialog.

. . .

Name	**Type**	**Description**
container_updated	container	The reaction that occurs when a code action is performed. Denotes that the container's settings have been initialized or updated in some way.
text_update	text	Denotes that the text within a text control has been modified.

Appendix B

Generate Java Package tuil

The Java application XJC is used to compile the schema files mentioned in section 4.3 into a package of Java stub-classes. It will also validate the syntax of the schema files to ensure they are correct with respect to standard XML syntax, as well as any new constructs created as part of the TUIL language. The command line used to perform the compilation for the TUIL schema files is:

```
xjc.exe "TUIL/Specification/TUILContainer.xsd" "TUIL/TestCase/TU-
ILTestCase.xsd" -d "TUIL/GeneratedFiles" -p tuil
```

To breakdown the command:

- The reference paths to the TUILContainer.xsd file and TUILTestCase.xsd file denote the main elements of the specification and test case sides of the TUIL language respectively.
- The -d command and path value after it denote where the generated Java files should be placed.

- The -p and its value denote that name of the Java package to generate, which will contain all the Java files generated.

The following section contains one of the generated files.

B.1 Generated Java Files

Listing B.1: TuilContainer.java

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference
     Implementation, vJAXB 2.1.10 in JDK 6
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source
    schema.
// Generated on: 2014.01.24 at 04:02:16 PM NST
//


package tuil;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;


/**
 *
 *         The main element for the TUIL language, and which represents a container
     GUI component, such
 *         as a window/frame/dialog/etc.
 *
 *
 * <p>Java class for anonymous complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
     this class.
 *
 * <pre>
 * &lt;complexType>
 *   &lt;complexContent>
 *     &lt;extension base="{}tuil_elem">
 *       &lt;sequence>
 *         &lt;element name="children" type="{}child_group_type" minOccurs="0"/>
 *       &lt;/sequence>
 *       &lt;attGroup ref="{}container_base_attribGroup"/>
 *       &lt;attribute name="use_default_button_placement" type="{http://www.w3.org
     /2001/XMLSchema}boolean" default="true" />
 *     &lt;/extension>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "children"
})
@XmlRootElement(name = "tuil_container")
public class TuilContainer
    extends TuilElem
{

    protected ChildGroupType children;
    @XmlAttribute(name = "use_default_button_placement")
    protected Boolean useDefaultButtonPlacement;
    @XmlAttribute(required = true)
    protected ContainerType type;
    @XmlAttribute(name = "is_modal")
    protected Boolean isModal;

    /**
     * Gets the value of the children property.
     *
     * @return
     *     possible object is
     *     {@link ChildGroupType }
     *
     */
    public ChildGroupType getChildren() {
        return children;
    }

    /**
     * Sets the value of the children property.
     *
     * @param value
     *     allowed object is
     *     {@link ChildGroupType }
     *
     */
    public void setChildren(ChildGroupType value) {
        this.children = value;
    }

    /**
     * Gets the value of the useDefaultButtonPlacement property.
     *
     * @return
     *     possible object is
     *     {@link Boolean }
     *
     */
    public boolean isUseDefaultButtonPlacement() {
        if (useDefaultButtonPlacement == null) {
            return true;
        } else {
            return useDefaultButtonPlacement;
        }
    }
```

```
/**
 * Sets the value of the useDefaultButtonPlacement property.
 *
 * @param value
 *     allowed object is
 *     {@link Boolean }
 *
 */
public void setUseDefaultButtonPlacement(Boolean value) {
    this.useDefaultButtonPlacement = value;
}

/**
 * Gets the value of the type property.
 *
 * @return
 *     possible object is
 *     {@link ContainerType }
 *
 */
public ContainerType getType() {
    return type;
}

/**
 * Sets the value of the type property.
 *
 * @param value
 *     allowed object is
 *     {@link ContainerType }
 *
 */
public void setType(ContainerType value) {
    this.type = value;
}

/**
 * Gets the value of the isModal property.
 *
 * @return
 *     possible object is
 *     {@link Boolean }
 *
 */
public boolean isIsModal() {
    if (isModal == null) {
        return false;
    } else {
        return isModal;
    }
}

/**
 * Sets the value of the isModal property.
 *
 * @param value
 *     allowed object is
 *     {@link Boolean }
 *
```

```
     */
    public void setIsModal(Boolean value) {
        this.isModal = value;
    }

}
```

B.2 Unmarshalling a TUIL XML File

Listings B.2 and B.3 are provided as examples of how JAXB can be used to convert a single TUIL XML file into a TuilContainer Java class instance for processing/-manipulation. TuilTestCase objects may also be loaded in this manner. They are examples only, as the TUIDE code declares the variables differently and performs error checking. However, how TUIDE instantiates these variables is similar.

Listing B.2: Example JAXB Variable Instantiation

```
  JAXBContext tuilContext = JAXBContext.newInstance("tuil");
  Unmarshaller unmarshaller = tuilContext.createUnmarshaller();
```

Listing B.3: Example JAXB Unmarshal a Single XML File

```
public TuilContainer UnmarshalFile(String filename)
{
  JAXBContext tuilContext = JAXBContext.newInstance("tuil");
  Unmarshaller unmarshaller = tuilContext.createUnmarshaller();
  TuilContainer c = (TuilContainer)unmarshaller.unmarshal(new File(filename));
}
```

Appendix C

Example of State Object Classes

Listing C.1: BaseState.java

```
/**
 * Represents all the base data state attributes that all other state objects extend
     on.
 */
package tuide.test_case.state_objects;

import tuil.TcBase;

/**
 * @author Krista A King
 * @version 1.0.10
 */
public class BaseState
{
  /**
   * Makes it possible to tell the difference between state objects, and to know what
        to cast the BaseState object to, in order to access
   * the correct information.
   */
  public enum Types { BASE, BUTTON, COMBO, GROUP_BOX, LABEL, TEXT_BOX, SLIDER,
      LISTBOX, MULTI_SELECT_LISTBOX, MULTI_COLUMN_LISTBOX,
      MULTI_SELECT_MULTI_COLUMN_LISTBOX, CONTAINER };

  protected Types m_Type = Types.BASE;
  protected boolean m_HasFocus = false;
  protected boolean m_Enabled = false;
  protected boolean m_Visible = false;
  protected String m_Text = new String("");

  public BaseState(boolean focused, boolean enabled, boolean visible, String text)
  {
    m_HasFocus = focused;
    m_Enabled = enabled;
    m_Visible = visible;

    // Leave as empty string if this value is null
    if(text != null)
    {
      m_Text = text;
    }
  }

  /**
   * Makes a deep copy of this instance and returns it.
   */
  public BaseState Copy()
  {
    BaseState ret = new BaseState(m_HasFocus, m_Enabled, m_Visible, m_Text);
    return ret;
  }

  // @return The Types value representing what kind of control this state object is
      associated with.
  public Types GetType() { return m_Type; }

  // @return A boolean denoting whether the associated control/container has focus or
       not.
  public boolean HasFocus() { return m_HasFocus; }
```

```
/**
 * Sets whether the associated control/container has focus or not.
 * @param focused A boolean denoting focused (true) or unfocused (false).
 */
public void SetFocus(boolean focused) { m_HasFocus = focused; }

// @return A boolean denoting whether the associated control/container is enabled/
    disabled.
public boolean IsEnabled() { return m_Enabled; }

/**
 * Sets whether the associated control/container is enabled or not.
 * @param enable A boolean denoting enabled (true) or disabled (false).
 */
public void SetEnabled(boolean enable) { m_Enabled = enable; }

// @return A boolean denoting whether the associated control/container is visible/
    hidden.
public boolean IsVisible() { return m_Visible; }

/**
 * Sets whether the associated control/container is visible or not.
 * @param visible A boolean denoting visible (true) or hidden (false).
 */
public void SetVisible(boolean visible) { m_Visible = visible; }

// @return The String representing the text part of  the associated control/
    container.
public String GetText() { return m_Text; }

/**
 * Sets the text component of this control
 * @param text The String representing the new text component.
 */
public void SetText(String text) { m_Text = text; }

/**
 * Determines the equality between this BaseState and the o parameter.
 * @param o An Object to compare against.
 * @return A boolean denoting equality, true mean equal, false means not equal.
 */
public boolean equals(Object o)
{
  if(o == null || o.getClass() != BaseState.class)
  {
    return false;
  }
  return IsEquals((BaseState)o);
}

/**
 * Generates the hash code for this BaseState.
 */
public int hashCode()
{
  return GetHashCode();
}

/**
 * Determines equality between the parameter b and this BaseState.
```

```
   * @param b A BaseState object to compare against.
   * @return A boolean denoting equality. True means equal, false means not equal.
   */
  public boolean IsEquals(BaseState b)
  {
    if(m_Type == b.GetType())
    {
      if(m_HasFocus == b.HasFocus())
      {
        if(m_Enabled == b.IsEnabled())
        {
          if(m_Visible == b.IsVisible())
          {
            if(m_Text.compareTo(b.GetText()) == 0)
            {
              return true;
            }
          }
        }
      }
    }
    return false;
  }

  /**
   * Generates the hash code for this BaseState. Used by the extension classes so
   * that they can determine the base hash code.
   * @return An int value which represents the hash code for this instance.
   */
  public int GetHashCode()
  {
    int ret = 31;
    ret += m_Type.hashCode();
    ret += (m_HasFocus) ? 1 : 0;
    ret += (m_Enabled) ? 1 : 0;
    ret += (m_Visible) ? 1 : 0;
    ret += m_Text.hashCode();
    return ret;
  }

  /**
   * @return A TcBase instance which represents the current
   * state of this object.
   */
  public TcBase ToTcObject()
  {
    TcBase ret = new TcBase();
    FillTcBase(ret);
    return ret;
  }

  /**
   * Fills the passed in TcBase instance with this object's
   * current settings.
   * @param base The TcBase instance to set values on.
   */
  public void FillTcBase(TcBase base)
  {
    if(base == null)
    {
      return;
    }
    base.setHasFocus(m_HasFocus);
    base.setIsEnabled(m_Enabled);
    base.setIsVisible(m_Visible);
    base.setText(GetText());
  }
}
```

Appendix D

How to Load a TUIL Specification into TUIDE

To load a TUIL specification into TUIDE, the first step is to open the *TUIL* menu as shown in Figure D.1.

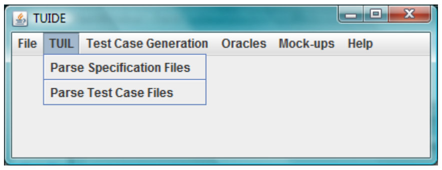


Figure D.1: The TUIL Menu

By selecting the *Parse Specification Files* menu item, the user is prompted with a directory open dialog. This allows the user to select a directory, within which it will search for .xml files. All sub-directories will also be searched. Once it has found a set of XML files to parse, it will use JAXB to load the files into corresponding sets of **TuilContainer** instances.

When the parsing has completed, TUIDE will display a summary to the user denoting the validation state of all files. Figure D.2 shows the result when an example specification has been loaded into TUIDE, however, one of the specification files contained an error, and thus failed validation.

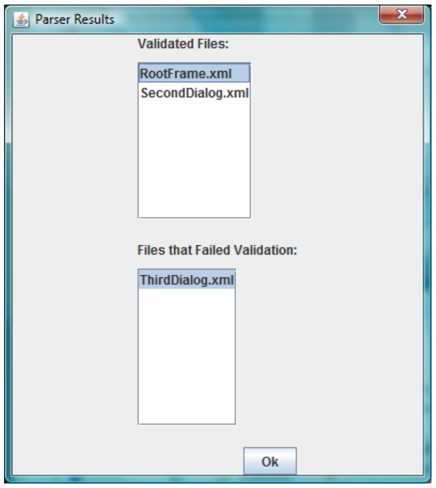


Figure D.2: The TUIL Result Dialog

Appendix E

How to Generate Test Cases From TUIDE

In order to generate test cases from TUIDE, a TUIL specification must have been loaded into the program (see Appendix D), and a graph must have been generated (see section 5.3).

The Test Case Generation menu, as seen in Figure E.1, contains two items of interest:

1. *Generate Graph*
2. *Generate Test Cases*

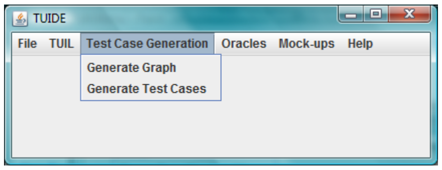


Figure E.1: The Test Case Generation Menu

E.1 Generate Graph Menu Item

To begin this process, the *Generate Graph* menu item must be selected from the *Test Case Generation* menu.

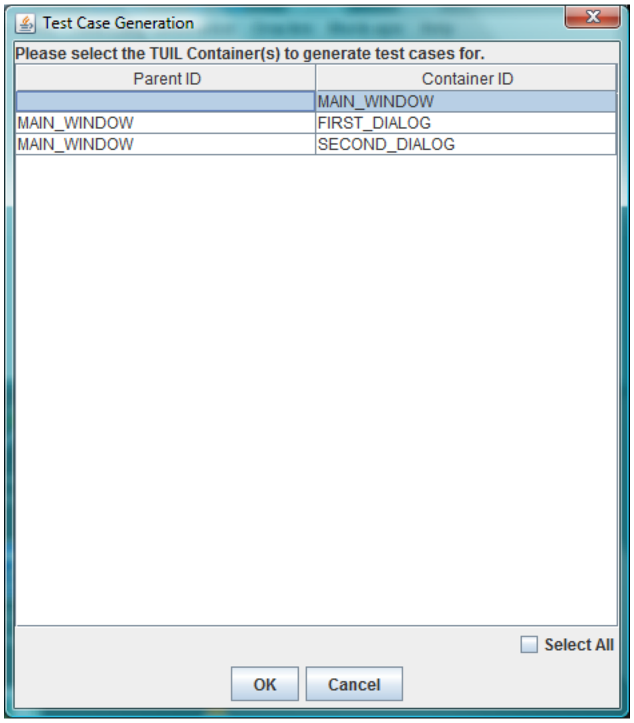
The first thing that is checked is that there is a proper TUIL specification loaded into the system, otherwise TUIDE will prevent graph generation from occurring. As well, if an error condition or exception occurs during the generation process, execution of the process will end.

Regardless of whether the process ends with an error or success, the user is provided with a message denoting the outcome.

Please see section 5.3 for more details into the algorithm used to handle generating the graph.

E.2 Generating Test Cases Menu Item

The process begins with selecting the *Generate Test Cases* menu item. A dialog is then displayed which allows one or more of the containers within the specified GUI to be selected for test case generation (Please see Figure E.2). There is also a check-box which, if checked, will select all of the containers. The selected containers are then saved into a set and passed on for further processing.



Please select the TUIL Container(s) to generate test cases for.

Parent ID	Container ID
	MAIN_WINDOW
MAIN_WINDOW	FIRST_DIALOG
MAIN_WINDOW	SECOND_DIALOG

Figure E.2: Test Case Generation Container Selection

If the graph could not be generated successfully, TUIDE will prevent the test case generation from continuing. If it was, then a worker thread is started, which is responsible for generating the test cases and displaying a message box of the outcome.

If the outcome is successful, then the directory into which the test cases were saved is provided to the user.

Please see section 5.4 for the complete explanation of the test case generation process.

Appendix F

How to Generate a Mock-up in TUIDE

To generate a mock-up within TUIDE, the designer must use the Mock-up menu as shown in Figure F.1. There is only one option available here, which is *Generate Mock-ups*.

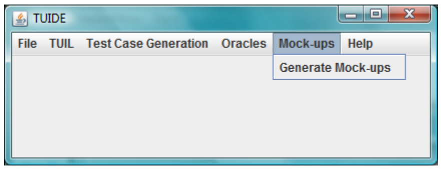
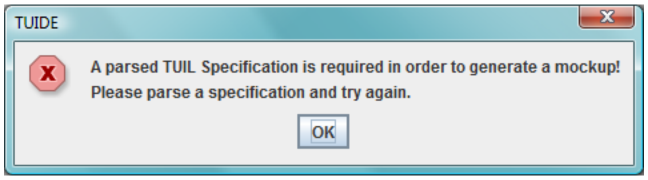


Figure F.1: The TUIDE Mock-up Menu

A TUIL specification must already have been loaded into the application, using the process outlined in Appendix D. If a TUIL specification has not been loaded into the system, an error message is displayed as seen in Figure F.2.

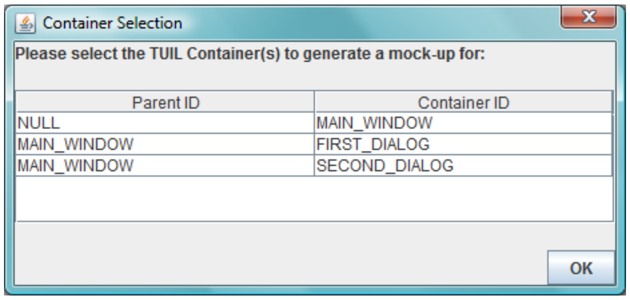

TUIDE

A parsed TUIL Specification is required in order to generate a mockup!
Please parse a specification and try again.

Figure F.2: The No TUIL Specification Error

When the *Generate Mock-ups* menu item is selected, a dialog, as shown in Figure F.3, is displayed. It will have the list of containers within a table, where both the **parent_id** and **id** attributes are shown.

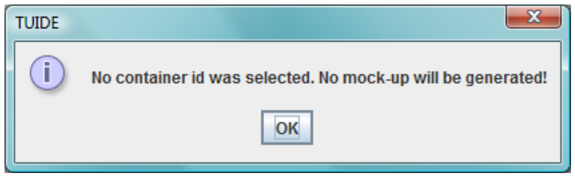
If the designer closes the dialog without selecting a container, an error message is displayed, as seen in Figure F.4.



Please select the TUIL Container(s) to generate a mock-up for:

Parent ID	Container ID
NULL	MAIN_WINDOW
MAIN_WINDOW	FIRST_DIALOG
MAIN_WINDOW	SECOND_DIALOG

Figure F.3: Select a Container To Mock-up Dialog



No container id was selected. No mock-up will be generated!

Figure F.4: No Container Selected Error

Appendix G

How to Run an Oracle from TUIDE

G.1 How to Load a Set of TUIL Test Cases

The following describes the steps required to load a set of TUIL test cases into the TUIDE program for use with different sections of it.

First, select the option *Parse Test Case Files* from the TUIL menu as shown in Figure G.1.

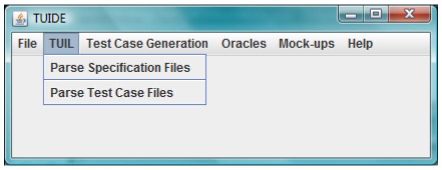
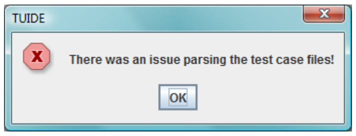


Figure G.1: The TUIL Menu

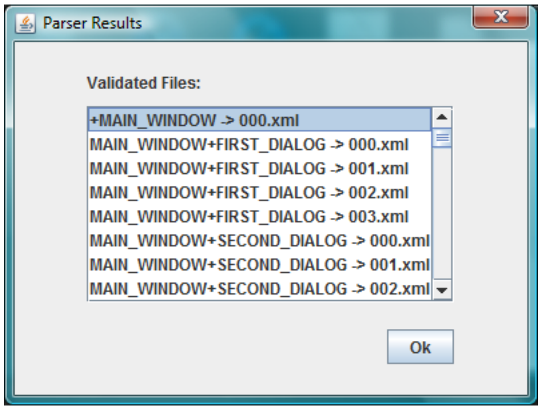
This will display a directory find dialog. Navigate to the directory containing the test cases to parse, and select ok/open. This will start the process of iterating through the directory and sub-directories to find all XML files. Then each of these XML files is parsed against the TUIL test case schema files to ensure they are valid. If no test cases could be found, then an error message is displayed as seen in Figure G.2.


TUIDE

There was an issue parsing the test case files!

Figure G.2: The No Test Cases Parsed Error Message

If test cases could be found, then the validation summary is displayed in another dialog. Figure G.3 shows how the dialog will look when all files were successfully validated. If some files failed, then a second list is shown containing information for those files.



```
+MAIN_WINDOW -> 000.xml
MAIN_WINDOW+FIRST_DIALOG -> 000.xml
MAIN_WINDOW+FIRST_DIALOG -> 001.xml
MAIN_WINDOW+FIRST_DIALOG -> 002.xml
MAIN_WINDOW+FIRST_DIALOG -> 003.xml
MAIN_WINDOW+SECOND_DIALOG -> 000.xml
MAIN_WINDOW+SECOND_DIALOG -> 001.xml
MAIN_WINDOW+SECOND_DIALOG -> 002.xml
```

Figure G.3: The Test Case Parsing Summary Dialog

Once this dialog is closed, the system is now setup with a parsed set of test cases to work with.

G.2 How to Load a Runnable JAR File

This section will describe how to load a Runnable JAR file into TUIDE so that corresponding test cases can be run against it using Oracles.

The first thing to do is to go to the *Oracles* menu and choose the *Select Runnable JAR File* option, as seen in Figure G.4.

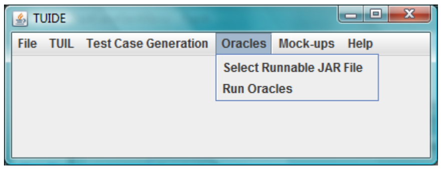


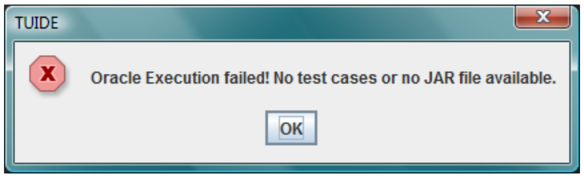
Figure G.4: The Oracle Menu

This will display a file selection dialog. Navigate to where the JAR file is located, select it in the file list and press Open. The TUIDE application will give no output to tell the user if the operation was successful or not. All it simply does is save the filename for future use.

G.3 How to Run an Oracle

This section will describe how to state the Oracles running from TUIDE.

The first thing to do is to go to the *Oracles* menu and choose the *Run Oracles* option, as seen in Figure G.4. The user must have first loaded a set of test cases into the system already, as describe in section G.1, and selected a runnable JAR file, as described in section G.2. If one or both of these were not completed previously, then the application will display a warning, as seen in Figure G.5, and the Oracles will not be performed.


TUIDE

Oracle Execution failed! No test cases or no JAR file available.

Figure G.5: The Oracle Warning Message If Missing Information

Otherwise, the system will start running the Oracles. Once the Oracles have completed, TUIDE will display a message to the user denoting the folder containing the results of the Oracles.

Appendix H

Guidelines for Application Under Test GUI Design

The following are guidelines on how the AUT's GUI should be implemented for our oracle application process to be successful. If these are not followed, then this process may fail. Some of the guidelines make use of classes and/or interfaces from the TUI_Utils library, that was designed as part of this research.

- The GUI should be written using Java Swing objects.
- The **id** attribute within each TUIL container and control specification object should be assigned to the associated Swing object using the *setName* property.
- Any Window type object, which is part of the specification, needs to implement the *IAUTEventListener*, and register itself with the *AUTHandler* class. This is to be able to receive certain event calls from the AUT oracle code. As well, the Window objects should also call *addWindowListener* and pass an instance of the *WindowClosingAdapter* class provided.

- As the **initialize_container** action only handles the default state of a container for now, the Java implementation code for the CodeEvent methods should also only load the default settings. This is to ensure consistency for now.

- The specification of all controls contains a **text** element. For some controls, this can be directly associated to one of the control's properties. An example of this is a container object, whose **text** is associated with the title or caption for the actual Swing container. Others do not have a property that is directly associated. An example of this would be the combo box. It is a list type control, and thus, the **text** value represents the type of information the combo box contains. This should be added to a *JLabel*. Similar controls to the combo box are:

  – Lists

  – Sliders

- Since *ButtonGroup* objects and *TitledBorders* are not extended from Component or JComponent, TUI_Utils provides extension classes for these which have name property available through the *getName* and *setName* functions. It is encouraged that these are used in the implementation of the AUT instead.

Appendix I

Food Orderer: Subset of Generated Test Cases

Listing I.1: Main Dialog Test Case 1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tuil_test_case>
  <path>
    <edge>
      <start root_node="true">
        <entry>
          <id>+ID_FOOD_ORDERER_DLG</id>
          <container is_modal="false" type="dialog">
            <has_focus>true</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Food Orderer</text>
            <entry>
              <id>ID_ALLERGY_ENTRY_LABEL</id>
              <tc_label>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Enter any allergy information:</text>
              </tc_label>
            </entry>
            <entry>
              <id>ID_ALLERGY_TEXTBOX</id>
              <tc_textbox>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text></text>
                <is_readOnly>false</is_readOnly>
                <is_multiline>false</is_multiline>
              </tc_textbox>
            </entry>
            <entry>
              <id>ID_APPETIZER_BUTTON</id>
              <tc_button>
                <has_focus>true</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Appetizer</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_DESSERT_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Dessert</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_ENTREE_BUTTON</id>
              <tc_button>
```

```
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Entree</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_PLACE_ORDER_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Place Order</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
          </container>
        </entry>
      </start>
      <end>
        <entry>
          <id>+ID_FOOD_ORDERER_DLG</id>
          <container is_modal="false" type="dialog">
            <has_focus>true</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Food Orderer</text>
            <entry>
              <id>ID_ALLERGY_ENTRY_LABEL</id>
              <tc_label>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Enter any allergy information:</text>
              </tc_label>
            </entry>
            <entry>
              <id>ID_ALLERGY_TEXTBOX</id>
              <tc_textbox>
                <has_focus>true</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>06c1f6d6-7af5-4205-b623-4bd6d2289b11</text>
                <is_readOnly>false</is_readOnly>
                <is_multiline>false</is_multiline>
              </tc_textbox>
            </entry>
            <entry>
              <id>ID_APPETIZER_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Appetizer</text>
                <is_checked>false</is_checked>
```

```xml
            <type>push</type>
            <button_group></button_group>
          </tc_button>
        </entry>
        <entry>
          <id>ID_DESSERT_BUTTON</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Dessert</text>
            <is_checked>false</is_checked>
            <type>push</type>
            <button_group></button_group>
          </tc_button>
        </entry>
        <entry>
          <id>ID_ENTREE_BUTTON</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Entree</text>
            <is_checked>false</is_checked>
            <type>push</type>
            <button_group></button_group>
          </tc_button>
        </entry>
        <entry>
          <id>ID_PLACE_ORDER_BUTTON</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Place Order</text>
            <is_checked>false</is_checked>
            <type>push</type>
            <button_group></button_group>
          </tc_button>
        </entry>
      </container>
    </entry>
  </end>
  <action_pair>
    <action>
      <text_action>
        <type>text_change</type>
        <text>06c1f6d6-7af5-4205-b623-4bd6d2289b11</text>
      </text_action>
    </action>
    <event_object parent_id="ID_FOOD_ORDERER_DLG" id="ID_ALLERGY_TEXTBOX" />
  </action_pair>
</edge>
<edge>
  <start>
    <entry>
      <id>+ID_FOOD_ORDERER_DLG</id>
      <container is_modal="false" type="dialog">
        <has_focus>true</has_focus>
        <is_enabled>true</is_enabled>
```

```
<is_visible>true</is_visible>
<text>Food Orderer</text>
<entry>
  <id>ID_ALLERGY_ENTRY_LABEL</id>
  <tc_label>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Enter any allergy information:</text>
  </tc_label>
</entry>
<entry>
  <id>ID_ALLERGY_TEXTBOX</id>
  <tc_textbox>
    <has_focus>true</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>06c1f6d6-7af5-4205-b623-4bd6d2289b11</text>
    <is_readOnly>false</is_readOnly>
    <is_multiline>false</is_multiline>
  </tc_textbox>
</entry>
<entry>
  <id>ID_APPETIZER_BUTTON</id>
  <tc_button>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Appetizer</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
<entry>
  <id>ID_DESSERT_BUTTON</id>
  <tc_button>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Dessert</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
<entry>
  <id>ID_ENTREE_BUTTON</id>
  <tc_button>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Entree</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
<entry>
  <id>ID_PLACE_ORDER_BUTTON</id>
```

```
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Place Order</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
          </container>
        </entry>
      </start>
      <end />
      <action_pair>
        <action>
          <mouse>mouse_click</mouse>
        </action>
        <event_object parent_id="ID_FOOD_ORDERER_DLG" id="ID_PLACE_ORDER_BUTTON" />
      </action_pair>
    </edge>
  </path>
</tuil_test_case>
```

Listing I.2: Main Dialog Test Case 9

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tuil_test_case>
  <path>
    <edge>
      <start root_node="true">
        <entry>
          <id>+ID_FOOD_ORDERER_DLG</id>
          <container is_modal="false" type="dialog">
            <has_focus>true</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Food Orderer</text>
            <entry>
              <id>ID_ALLERGY_ENTRY_LABEL</id>
              <tc_label>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Enter any allergy information:</text>
              </tc_label>
            </entry>
            <entry>
              <id>ID_ALLERGY_TEXTBOX</id>
              <tc_textbox>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text></text>
                <is_readOnly>false</is_readOnly>
                <is_multiline>false</is_multiline>
              </tc_textbox>
            </entry>
            <entry>
              <id>ID_APPETIZER_BUTTON</id>
              <tc_button>
                <has_focus>true</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Appetizer</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_DESSERT_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Dessert</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_ENTREE_BUTTON</id>
              <tc_button>
```

```
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Entree</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_PLACE_ORDER_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Place Order</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
          </container>
        </entry>
      </start>
      <end />
      <action_pair>
        <action>
          <mouse>mouse_click</mouse>
        </action>
        <event_object parent_id="ID_FOOD_ORDERER_DLG" id="ID_PLACE_ORDER_BUTTON" />
      </action_pair>
    </edge>
  </path>
</tuil_test_case>
```

Listing I.3: Dessert Dialog Test Case 1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tuil_test_case>
  <path>
    <edge>
      <start root_node="true">
        <entry>
          <id>+ID_FOOD_ORDERER_DLG</id>
          <container is_modal="false" type="dialog">
            <has_focus>true</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Food Orderer</text>
            <entry>
              <id>ID_ALLERGY_ENTRY_LABEL</id>
              <tc_label>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Enter any allergy information:</text>
              </tc_label>
            </entry>
            <entry>
              <id>ID_ALLERGY_TEXTBOX</id>
              <tc_textbox>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text></text>
                <is_readOnly>false</is_readOnly>
                <is_multiline>false</is_multiline>
              </tc_textbox>
            </entry>
            <entry>
              <id>ID_APPETIZER_BUTTON</id>
              <tc_button>
                <has_focus>true</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Appetizer</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_DESSERT_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Dessert</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_ENTREE_BUTTON</id>
              <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Entree</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_PLACE_ORDER_BUTTON</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Place Order</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
    </container>
  </entry>
</start>
<end>
  <entry>
    <id>+ID_FOOD_ORDERER_DLG</id>
    <container is_modal="false" type="dialog">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Food Orderer</text>
      <entry>
        <id>ID_ALLERGY_ENTRY_LABEL</id>
        <tc_label>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Enter any allergy information:</text>
        </tc_label>
      </entry>
      <entry>
        <id>ID_ALLERGY_TEXTBOX</id>
        <tc_textbox>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text></text>
          <is_readOnly>false</is_readOnly>
          <is_multiline>false</is_multiline>
        </tc_textbox>
      </entry>
      <entry>
        <id>ID_APPETIZER_BUTTON</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Appetizer</text>
          <is_checked>false</is_checked>
        <type>push</type>
        <button_group></button_group>
      </tc_button>
    </entry>
    <entry>
      <id>ID_DESSERT_BUTTON</id>
      <tc_button>
        <has_focus>true</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Dessert</text>
        <is_checked>false</is_checked>
        <type>push</type>
        <button_group></button_group>
      </tc_button>
    </entry>
    <entry>
      <id>ID_ENTREE_BUTTON</id>
      <tc_button>
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Entree</text>
        <is_checked>false</is_checked>
        <type>push</type>
        <button_group></button_group>
      </tc_button>
    </entry>
    <entry>
      <id>ID_PLACE_ORDER_BUTTON</id>
      <tc_button>
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Place Order</text>
        <is_checked>false</is_checked>
        <type>push</type>
        <button_group></button_group>
      </tc_button>
    </entry>
  </container>
</entry>
<entry>
  <id>ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG</id>
  <container is_modal="true" type="dialog">
    <has_focus>true</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Food Orderer - Desserts</text>
    <entry>
      <id>ID_APPLE_PIE</id>
      <tc_button>
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Apple Pie</text>
        <is_checked>false</is_checked>
        <type>radio</type>
        <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
      </tc_button>
      </entry>
      <entry>
        <id>ID_CHEESECAKE</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Chocolate Cheesecake</text>
          <is_checked>false</is_checked>
          <type>radio</type>
          <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_OK</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Done</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_SUNDAE</id>
        <tc_button>
          <has_focus>true</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Ice Cream Sundae</text>
          <is_checked>true</is_checked>
          <type>radio</type>
          <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_SUNDAE_SAUCE</id>
        <tc_combo>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Sauce:</text>
          <select_state>no_selection</select_state>
          <items>
            <item>Chocolate Fudge</item>
            <item>Strawberry</item>
            <item>Pineapple</item>
            <item>Caramel</item>
            <item>Butterscotch</item>
          </items>
          <item_change_state>no_change</item_change_state>
          <selected_item>-1</selected_item>
          <list_shown>false</list_shown>
        </tc_combo>
      </entry>
    </container>
  </entry>
</end>
```

```
    <action_pair>
      <action>
        <mouse>mouse_click</mouse>
      </action>
      <event_object parent_id="ID_FOOD_ORDERER_DLG" id="ID_DESSERT_BUTTON" />
    </action_pair>
  </edge>
  <edge>
    <start>
      <entry>
        <id>+ID_FOOD_ORDERER_DLG</id>
        <container is_modal="false" type="dialog">
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Food Orderer</text>
          <entry>
            <id>ID_ALLERGY_ENTRY_LABEL</id>
            <tc_label>
              <has_focus>false</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Enter any allergy information:</text>
            </tc_label>
          </entry>
          <entry>
            <id>ID_ALLERGY_TEXTBOX</id>
            <tc_textbox>
              <has_focus>false</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text></text>
              <is_readOnly>false</is_readOnly>
              <is_multiline>false</is_multiline>
            </tc_textbox>
          </entry>
          <entry>
            <id>ID_APPETIZER_BUTTON</id>
            <tc_button>
              <has_focus>false</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Appetizer</text>
              <is_checked>false</is_checked>
              <type>push</type>
              <button_group></button_group>
            </tc_button>
          </entry>
          <entry>
            <id>ID_DESSERT_BUTTON</id>
            <tc_button>
              <has_focus>true</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Dessert</text>
              <is_checked>false</is_checked>
              <type>push</type>
              <button_group></button_group>
            </tc_button>
          </entry>
        <entry>
          <id>ID_ENTREE_BUTTON</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Entree</text>
            <is_checked>false</is_checked>
            <type>push</type>
            <button_group></button_group>
          </tc_button>
        </entry>
        <entry>
          <id>ID_PLACE_ORDER_BUTTON</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Place Order</text>
            <is_checked>false</is_checked>
            <type>push</type>
            <button_group></button_group>
          </tc_button>
        </entry>
      </container>
    </entry>
    <entry>
      <id>ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG</id>
      <container is_modal="true" type="dialog">
        <has_focus>true</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Food Orderer - Desserts</text>
        <entry>
          <id>ID_APPLE_PIE</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Apple Pie</text>
            <is_checked>false</is_checked>
            <type>radio</type>
            <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
          </tc_button>
        </entry>
        <entry>
          <id>ID_CHEESECAKE</id>
          <tc_button>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Chocolate Cheesecake</text>
            <is_checked>false</is_checked>
            <type>radio</type>
            <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
          </tc_button>
        </entry>
        <entry>
          <id>ID_OK</id>
          <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Done</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_SUNDAE</id>
        <tc_button>
          <has_focus>true</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Ice Cream Sundae</text>
          <is_checked>true</is_checked>
          <type>radio</type>
          <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_SUNDAE_SAUCE</id>
        <tc_combo>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Sauce:</text>
          <select_state>no_selection</select_state>
          <items>
            <item>Chocolate Fudge</item>
            <item>Strawberry</item>
            <item>Pineapple</item>
            <item>Caramel</item>
            <item>Butterscotch</item>
          </items>
          <item_change_state>no_change</item_change_state>
          <selected_item>-1</selected_item>
          <list_shown>false</list_shown>
        </tc_combo>
      </entry>
    </container>
  </entry>
</start>
<end>
  <entry>
    <id>+ID_FOOD_ORDERER_DLG</id>
    <container is_modal="false" type="dialog">
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Food Orderer</text>
      <entry>
        <id>ID_ALLERGY_ENTRY_LABEL</id>
        <tc_label>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Enter any allergy information:</text>
        </tc_label>
```

```
</entry>
<entry>
  <id>ID_ALLERGY_TEXTBOX</id>
  <tc_textbox>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text></text>
    <is_readOnly>false</is_readOnly>
    <is_multiline>false</is_multiline>
  </tc_textbox>
</entry>
<entry>
  <id>ID_APPETIZER_BUTTON</id>
  <tc_button>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Appetizer</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
<entry>
  <id>ID_DESSERT_BUTTON</id>
  <tc_button>
    <has_focus>true</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Dessert</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
<entry>
  <id>ID_ENTREE_BUTTON</id>
  <tc_button>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Entree</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
<entry>
  <id>ID_PLACE_ORDER_BUTTON</id>
  <tc_button>
    <has_focus>false</has_focus>
    <is_enabled>true</is_enabled>
    <is_visible>true</is_visible>
    <text>Place Order</text>
    <is_checked>false</is_checked>
    <type>push</type>
    <button_group></button_group>
  </tc_button>
</entry>
```

```
        </container>
      </entry>
      <entry>
        <id>ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG</id>
        <container is_modal="true" type="dialog">
          <has_focus>true</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Food Orderer - Desserts</text>
          <entry>
            <id>ID_APPLE_PIE</id>
            <tc_button>
              <has_focus>false</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Apple Pie</text>
              <is_checked>false</is_checked>
              <type>radio</type>
              <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
            </tc_button>
          </entry>
          <entry>
            <id>ID_CHEESECAKE</id>
            <tc_button>
              <has_focus>true</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Chocolate Cheesecake</text>
              <is_checked>true</is_checked>
              <type>radio</type>
              <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
            </tc_button>
          </entry>
          <entry>
            <id>ID_OK</id>
            <tc_button>
              <has_focus>false</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Done</text>
              <is_checked>false</is_checked>
              <type>push</type>
              <button_group></button_group>
            </tc_button>
          </entry>
          <entry>
            <id>ID_SUNDAE</id>
            <tc_button>
              <has_focus>false</has_focus>
              <is_enabled>true</is_enabled>
              <is_visible>true</is_visible>
              <text>Ice Cream Sundae</text>
              <is_checked>false</is_checked>
              <type>radio</type>
              <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
            </tc_button>
          </entry>
          <entry>
            <id>ID_SUNDAE_SAUCE</id>
            <tc_combo>
            <has_focus>false</has_focus>
            <is_enabled>false</is_enabled>
            <is_visible>true</is_visible>
            <text>Sauce:</text>
            <select_state>no_selection</select_state>
            <items>
              <item>Chocolate Fudge</item>
              <item>Strawberry</item>
              <item>Pineapple</item>
              <item>Caramel</item>
              <item>Butterscotch</item>
            </items>
            <item_change_state>no_change</item_change_state>
            <selected_item>-1</selected_item>
            <list_shown>false</list_shown>
          </tc_combo>
        </entry>
      </container>
    </entry>
  </end>
  <action_pair>
    <action>
      <mouse>mouse_click</mouse>
    </action>
    <event_object parent_id="ID_DESSERT_DLG" id="ID_CHEESECAKE" />
  </action_pair>
</edge>
<edge>
  <start>
    <entry>
      <id>+ID_FOOD_ORDERER_DLG</id>
      <container is_modal="false" type="dialog">
        <has_focus>false</has_focus>
        <is_enabled>true</is_enabled>
        <is_visible>true</is_visible>
        <text>Food Orderer</text>
        <entry>
          <id>ID_ALLERGY_ENTRY_LABEL</id>
          <tc_label>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Enter any allergy information:</text>
          </tc_label>
        </entry>
        <entry>
          <id>ID_ALLERGY_TEXTBOX</id>
          <tc_textbox>
            <has_focus>false</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text></text>
            <is_readOnly>false</is_readOnly>
            <is_multiline>false</is_multiline>
          </tc_textbox>
        </entry>
        <entry>
          <id>ID_APPETIZER_BUTTON</id>
          <tc_button>
            <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Appetizer</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_DESSERT_BUTTON</id>
        <tc_button>
          <has_focus>true</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Dessert</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_ENTREE_BUTTON</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Entree</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
      <entry>
        <id>ID_PLACE_ORDER_BUTTON</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Place Order</text>
          <is_checked>false</is_checked>
          <type>push</type>
          <button_group></button_group>
        </tc_button>
      </entry>
    </container>
  </entry>
  <entry>
    <id>ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG</id>
    <container is_modal="true" type="dialog">
      <has_focus>true</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Food Orderer - Desserts</text>
      <entry>
        <id>ID_APPLE_PIE</id>
        <tc_button>
          <has_focus>false</has_focus>
          <is_enabled>true</is_enabled>
          <is_visible>true</is_visible>
          <text>Apple Pie</text>
      <is_checked>false</is_checked>
      <type>radio</type>
      <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
    </tc_button>
  </entry>
  <entry>
    <id>ID_CHEESECAKE</id>
    <tc_button>
      <has_focus>true</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Chocolate Cheesecake</text>
      <is_checked>true</is_checked>
      <type>radio</type>
      <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
    </tc_button>
  </entry>
  <entry>
    <id>ID_OK</id>
    <tc_button>
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Done</text>
      <is_checked>false</is_checked>
      <type>push</type>
      <button_group></button_group>
    </tc_button>
  </entry>
  <entry>
    <id>ID_SUNDAE</id>
    <tc_button>
      <has_focus>false</has_focus>
      <is_enabled>true</is_enabled>
      <is_visible>true</is_visible>
      <text>Ice Cream Sundae</text>
      <is_checked>false</is_checked>
      <type>radio</type>
      <button_group>ID_DESSERT_CHOICE_GROUP</button_group>
    </tc_button>
  </entry>
  <entry>
    <id>ID_SUNDAE_SAUCE</id>
    <tc_combo>
      <has_focus>false</has_focus>
      <is_enabled>false</is_enabled>
      <is_visible>true</is_visible>
      <text>Sauce:</text>
      <select_state>no_selection</select_state>
      <items>
        <item>Chocolate Fudge</item>
        <item>Strawberry</item>
        <item>Pineapple</item>
        <item>Caramel</item>
        <item>Butterscotch</item>
      </items>
      <item_change_state>no_change</item_change_state>
      <selected_item>-1</selected_item>
      <list_shown>false</list_shown>
    </tc_combo>
```

```
            </entry>
          </container>
        </entry>
      </start>
      <end>
        <entry>
          <id>+ID_FOOD_ORDERER_DLG</id>
          <container is_modal="false" type="dialog">
            <has_focus>true</has_focus>
            <is_enabled>true</is_enabled>
            <is_visible>true</is_visible>
            <text>Food Orderer</text>
            <entry>
              <id>ID_ALLERGY_ENTRY_LABEL</id>
              <tc_label>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Enter any allergy information:</text>
              </tc_label>
            </entry>
            <entry>
              <id>ID_ALLERGY_TEXTBOX</id>
              <tc_textbox>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text></text>
                <is_readOnly>false</is_readOnly>
                <is_multiline>false</is_multiline>
              </tc_textbox>
            </entry>
            <entry>
              <id>ID_APPETIZER_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Appetizer</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_DESSERT_BUTTON</id>
              <tc_button>
                <has_focus>true</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Dessert</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_ENTREE_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Entree</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
            <entry>
              <id>ID_PLACE_ORDER_BUTTON</id>
              <tc_button>
                <has_focus>false</has_focus>
                <is_enabled>true</is_enabled>
                <is_visible>true</is_visible>
                <text>Place Order</text>
                <is_checked>false</is_checked>
                <type>push</type>
                <button_group></button_group>
              </tc_button>
            </entry>
          </container>
        </entry>
      </end>
      <action_pair>
        <action>
          <mouse>mouse_click</mouse>
        </action>
        <event_object parent_id="ID_DESSERT_DLG" id="ID_OK" />
      </action_pair>
    </edge>
  </path>
</tuil_test_case>
```

Appendix J

Food Orderer: Event Coverage

The term event coverage, with respect to this research, and the Food Orderer application, means how often each event occurred within the entire set of test cases generated. The usage counts provided were tabulated manually.

The Event column shows only the action type, as in this scenario each action is unique within the set of events for each control. For example, the control ID_PEROGIES_CHECK has two events listed, one initiated by a mouse_click action, and one initiated by a return_key action.

Table J.1: Appetizer Dialog Events

Control/Container	**Event**	**Usage Count**
ID_PEROGIES_CHECK	mouse_click	12
ID_PEROGIES_CHECK	return_key	7
ID_PEROGIE_SAUCE_COMBO	mouse_click	1
ID_PEROGIE_SAUCE_COMBO	select_item	1
ID_NACHOS_CHECK	mouse_click	8
ID_NACHOS_CHECK	return_key	9
ID_NACHOS_EXTRA_CHEESE_CHECK	mouse_click	3
ID_NACHOS_EXTRA_CHEESE_CHECK	return_key	3
ID_NACHOS_ADD_CHICKEN_CHECK	mouse_click	3
ID_NACHOS_ADD_CHICKEN_CHECK	return_key	3
ID_SQUASH_SOUP_CHECK	mouse_click	14
ID_SQUASH_SOUP_CHECK	return_key	10
ID_SOUP_CRACKERS_CHECK	mouse_click	4
ID_SOUP_CRACKERS_CHECK	return_key	3
ID_OK	mouse_click	13
ID_OK	return_key	10
ID_APPETIZER_DLG	initialize_container	0

Table J.2: Entrée Dialog Events

Control/Container	**Event**	**Usage Count**
ID_CHICKEN	mouse_click	0
ID_CHICKEN	return_key	5
ID_CHICKEN_MEAT_WHITE	mouse_click	0
ID_CHICKEN_MEAT_WHITE	return_key	1
ID_CHICKEN_MEAT_DARK	mouse_click	4
ID_CHICKEN_MEAT_DARK	return_key	8
ID_NEWYORK_STEAK	mouse_click	2
ID_NEWYORK_STEAK	return_key	5
ID_STEAK_COOK_OPTION	mouse_click	0
ID_VEGGIE_STIRFRY	mouse_click	8
ID_VEGGIE_STIRFRY	return_key	15
ID_STIRFRY_SAUCE	select_item	15
ID_STIRFRY_SAUCE	deselect_item	12
ID_SIDEDISH_ONE	mouse_click	25
ID_SIDEDISH_ONE	select_item on Garden Salad Only	19
ID_SIDEDISH_ONE	select_item on all but Garden Salad	4
ID_SIDEDISH_ONE_DRESSING	mouse_click	15
ID_SIDEDISH_ONE_DRESSING	select_item	15
ID_SIDEDISH_TWO	mouse_click	26
ID_SIDEDISH_TWO	select_item	22
ID_OK	mouse_click	31
ID_OK	return_key	8
ID_ENTREE_DLG	initialize_container	0

Table J.3: Dessert Dialog Events

Control/Container	**Event**	**Usage Count**
ID_SUNDAE	mouse_click	0
ID_SUNDAE	return_key	1

. . .

Control/Container	**Event**	**Usage Count**
ID_SUNDAE_SAUCE	mouse_click	3
ID_SUNDAE_SAUCE	select_item	2
ID_APPLE_PIE	mouse_click	1
ID_APPLE_PIE	return_key	1
ID_CHEESECAKE	mouse_click	1
ID_CHEESECAKE	return_key	0
ID_OK	mouse_click	3
ID_OK	return_key	3
ID_DESSERT_DLG	initialize_container	0

Appendix K

Food Orderer: Generated Oracle Results

K.1 Main Dialog Oracle Results

Listing K.1: Main Dialog Oracle Result 1

```
Action: Start-up
Initial State:
  Pass: States identical

Action: text_change with text to apply equal to "06c1f6d6-7af5-4205-b623-4bd6d2289b11
    " on ID_ALLERGY_TEXTBOX within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: mouse_click on ID_PLACE_ORDER_BUTTON within container ID_FOOD_ORDERER_DLG
State 2:
  Pass: States identical
```

K.2 Appetizer Dialog Oracle Results

Listing K.2: Appetizer Dialog Oracle Result 12

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 2:
  Pass: States identical

Action: mouse_click on ID_SOUP_CRACKERS_CHECK within container ID_APPETIZER_DLG
State 3:
  Pass: States identical

Action: mouse_click on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 4:
  Pass: States identical

Action: return_key pressed on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 5:
  Pass: States identical

Action: mouse_click on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 6:
  Pass: States identical

Action: return_key pressed on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 7:
  Pass: States identical

Action: mouse_click on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 8:
  Pass: States identical

Action: return_key pressed on ID_OK within container ID_APPETIZER_DLG
State 9:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing K.3: Appetizer Dialog Oracle Result 14

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 2:
  Pass: States identical
```

```
Action: return_key pressed on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 3:
  Pass: States identical

Action: return_key pressed on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 4:
  Pass: States identical

Action: mouse_click on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 5:
  Pass: States identical

Action: return_key pressed on ID_OK within container ID_APPETIZER_DLG
State 6:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing K.4: Appetizer Dialog Oracle Result 17

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: mouse_click on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 2:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 3:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 4:
  Pass: States identical

Action: mouse_click on ID_OK within container ID_APPETIZER_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

K.3 Entrée Dialog Oracle Results

Listing K.5: Entrée Dialog Oracle Result 37

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_ENTREE_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: mouse_click on ID_SIDEDISH_ONE within container ID_ENTREE_DLG
State 2:
  Pass: States identical

Action: select_item with text equal to "Garden Salad" on ID_SIDEDISH_ONE within
    container ID_ENTREE_DLG
State 3:
  Pass: States identical

Action: return_key pressed on ID_VEGGIE_STIRFRY within container ID_ENTREE_DLG
State 4:
  Pass: States identical

Action: return_key pressed on ID_NEWYORK_STEAK within container ID_ENTREE_DLG
State 5:
  Pass: States identical

Action: return_key pressed on ID_CHICKEN within container ID_ENTREE_DLG
State 6:
  Pass: States identical

Action: return_key pressed on ID_OK within container ID_ENTREE_DLG
State 7:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

K.4 Dessert Dialog Oracle Results

Listing K.6: Dessert Dialog Oracle Result 5

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_DESSERT_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: mouse_click on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 2:
  Pass: States identical

Action: select_item on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 3:
  Pass: States identical

Action: return_key pressed on ID_SUNDAE within container ID_DESSERT_DLG
State 4:
  Pass: States identical

Action: return_key pressed on ID_OK within container ID_DESSERT_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Appendix L

Food Orderer: Generated Oracle Results with Faults

L.1 Main Dialog Oracle Results

Listing L.1: Result 1 with Label Fault

```
Action: Start-up
Initial State:
  Failure: Comparison between +ID_FOOD_ORDERER_DLG expected and generated Containers
      yielded the following error: Comparison between ID_ALLERGY_ENTRY_LABEL expected
       and generated Controls yielded the following error: Text value not same,
      expected (Enter any allergy information:) vs generated (Enter any allergy
      information: )

Action: text_change with text to apply equal to "06c1f6d6-7af5-4205-b623-4bd6d2289b11
    " on ID_ALLERGY_TEXTBOX within container ID_FOOD_ORDERER_DLG
State 1:
  Failure: Comparison between +ID_FOOD_ORDERER_DLG expected and generated Containers
      yielded the following error: Comparison between ID_ALLERGY_ENTRY_LABEL expected
       and generated Controls yielded the following error: Text value not same,
      expected (Enter any allergy information:) vs generated (Enter any allergy
      information: )

Action: mouse_click on ID_PLACE_ORDER_BUTTON within container ID_FOOD_ORDERER_DLG
State 2:
  Pass: States identical
```

[1]

Listing L.2: Result 1 with Textbox Fault

```
Action: Start-up
Initial State:
  Failure: Comparison between +ID_FOOD_ORDERER_DLG expected and generated Containers
      yielded the following error: Comparison between ID_ALLERGY_TEXTBOX expected and
       generated Controls yielded the following error: Enabled state not same,
      expected (true) vs generated (false)

Action: text_change with text to apply equal to "06c1f6d6-7af5-4205-b623-4bd6d2289b11
    " on ID_ALLERGY_TEXTBOX within container ID_FOOD_ORDERER_DLG
State 1:
  Failure: Comparison between +ID_FOOD_ORDERER_DLG expected and generated Containers
      yielded the following error: Comparison between ID_ALLERGY_TEXTBOX expected and
       generated Controls yielded the following error: Text value not same, expected
      (06c1f6d6-7af5-4205-b623-4bd6d2289b11) vs generated ()

Action: mouse_click on ID_PLACE_ORDER_BUTTON within container ID_FOOD_ORDERER_DLG
State 2:
  Pass: States identical
```

Listing L.3: Result 5 with the Dessert Button Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_DESSERT_BUTTON within container ID_FOOD_ORDERER_DLG
```

[1]The expected and generated label contents in the first Listing differ by only a space. This space is located at the end of the generated version.

```
State 1:
  Failure: Generated TcNode did not contain all of the Containers expected

Action: mouse_click on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 2:
  Failure: Generated TcNode did not contain all of the Containers expected

Action: select_item on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 3:
  Failure: Generated TcNode did not contain all of the Containers expected

Action: return_key pressed on ID_SUNDAE within container ID_DESSERT_DLG
State 4:
  Failure: Generated TcNode did not contain all of the Containers expected

Action: return_key pressed on ID_OK within container ID_DESSERT_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

L.2 Appetizer Dialog Oracle Results

Listing L.4: Result 12 with Soup Crackers Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 2:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: mouse_click on ID_SOUP_CRACKERS_CHECK within container ID_APPETIZER_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Focused state not same, expected (true) vs generated (false)

Action: mouse_click on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: IsChecked state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 5:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: IsChecked state not same, expected (true) vs generated (false)

Action: mouse_click on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 6:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: IsChecked state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 7:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: IsChecked state not same, expected (true) vs generated (false)

Action: mouse_click on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 8:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: IsChecked state not same, expected (true) vs generated (false)
```

```
Action: return_key pressed on ID_OK within container ID_APPETIZER_DLG
State 9:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing L.5: Result 14 with Soup Crackers Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 2:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: mouse_click on ID_PEROGIES_CHECK within container ID_APPETIZER_DLG
State 5:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_OK within container ID_APPETIZER_DLG
State 6:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing L.6: Result 17 with Soup Crackers Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical
```

```
Action: mouse_click on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 2:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 3:
  Pass: States identical

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SOUP_CRACKERS_CHECK expected and generated Controls yielded the following
      error: Enabled state not same, expected (false) vs generated (true)

Action: mouse_click on ID_OK within container ID_APPETIZER_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing L.7: Result 17 with List Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_APPETIZER_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_PEROGIE_SAUCE_COMBO expected and generated Controls yielded the following
      error: List sizes are not same, expected (4) vs generated (3)

Action: mouse_click on ID_NACHOS_CHECK within container ID_APPETIZER_DLG
State 2:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_PEROGIE_SAUCE_COMBO expected and generated Controls yielded the following
      error: List sizes are not same, expected (4) vs generated (3)

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_PEROGIE_SAUCE_COMBO expected and generated Controls yielded the following
      error: List sizes are not same, expected (4) vs generated (3)

Action: return_key pressed on ID_SQUASH_SOUP_CHECK within container ID_APPETIZER_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_APPETIZER_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_PEROGIE_SAUCE_COMBO expected and generated Controls yielded the following
      error: List sizes are not same, expected (4) vs generated (3)

Action: mouse_click on ID_OK within container ID_APPETIZER_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

L.3 Entrée Dialog Oracle Results

Listing L.8: Result 37 with Enable Faults

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_ENTREE_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Pass: States identical

Action: mouse_click on ID_SIDEDISH_ONE within container ID_ENTREE_DLG
State 2:
  Pass: States identical

Action: select_item with text equal to "Garden Salad" on ID_SIDEDISH_ONE within
    container ID_ENTREE_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SIDEDISH_ONE_DRESSING expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_VEGGIE_STIRFRY within container ID_ENTREE_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SIDEDISH_ONE_DRESSING expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_NEWYORK_STEAK within container ID_ENTREE_DLG
State 5:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_CHICKEN_MEAT_DARK expected and generated Controls yielded the following
      error: Enabled state not same, expected (false) vs generated (true)

Action: return_key pressed on ID_CHICKEN within container ID_ENTREE_DLG
State 6:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SIDEDISH_ONE_DRESSING expected and generated Controls yielded the following
      error: Enabled state not same, expected (true) vs generated (false)

Action: return_key pressed on ID_OK within container ID_ENTREE_DLG
State 7:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing L.9: Result 37 with List Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_ENTREE_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_STIRFRY_SAUCE expected and generated Controls yielded the following error:
      The generated set of list items did not have all required items. Inequality on
      expected (Sweet-n-Sour) vs generated (Sweet and Sour)

Action: mouse_click on ID_SIDEDISH_ONE within container ID_ENTREE_DLG
State 2:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_STIRFRY_SAUCE expected and generated Controls yielded the following error:
      The generated set of list items did not have all required items. Inequality on
      expected (Sweet-n-Sour) vs generated (Sweet and Sour)

Action: select_item with text equal to "Garden Salad" on ID_SIDEDISH_ONE within
    container ID_ENTREE_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_STIRFRY_SAUCE expected and generated Controls yielded the following error:
      The generated set of list items did not have all required items. Inequality on
      expected (Sweet-n-Sour) vs generated (Sweet and Sour)

Action: return_key pressed on ID_VEGGIE_STIRFRY within container ID_ENTREE_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_STIRFRY_SAUCE expected and generated Controls yielded the following error:
      The generated set of list items did not have all required items. Inequality on
      expected (Sweet-n-Sour) vs generated (Sweet and Sour)

Action: return_key pressed on ID_NEWYORK_STEAK within container ID_ENTREE_DLG
State 5:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_STIRFRY_SAUCE expected and generated Controls yielded the following error:
      The generated set of list items did not have all required items. Inequality on
      expected (Sweet-n-Sour) vs generated (Sweet and Sour)

Action: return_key pressed on ID_CHICKEN within container ID_ENTREE_DLG
State 6:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_ENTREE_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_STIRFRY_SAUCE expected and generated Controls yielded the following error:
      The generated set of list items did not have all required items. Inequality on
      expected (Sweet-n-Sour) vs generated (Sweet and Sour)

Action: return_key pressed on ID_OK within container ID_ENTREE_DLG
State 7:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

L.4 Dessert Dialog Oracle Results

Listing L.10: Result 5 with Sauce Combo Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_DESSERT_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SUNDAE_SAUCE expected and generated Controls yielded the following error:
      Enabled state not same, expected (true) vs generated (false)

Action: mouse_click on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 2:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between ID_SUNDAE
      expected and generated Controls yielded the following error: Focused state not
      same, expected (false) vs generated (true)

Action: select_item on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between ID_SUNDAE
      expected and generated Controls yielded the following error: Focused state not
      same, expected (false) vs generated (true)

Action: return_key pressed on ID_SUNDAE within container ID_DESSERT_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SUNDAE_SAUCE expected and generated Controls yielded the following error:
      Selection state not same, expected (SINGLE_SELECTION) vs generated (
      NO_SELECTION)

Action: return_key pressed on ID_OK within container ID_DESSERT_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```

Listing L.11: Result 5 with List Fault

```
Action: Start-up
Initial State:
  Pass: States identical

Action: mouse_click on ID_DESSERT_BUTTON within container ID_FOOD_ORDERER_DLG
State 1:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SUNDAE_SAUCE expected and generated Controls yielded the following error:
      List sizes are not same, expected (5) vs generated (6)

Action: mouse_click on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 2:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SUNDAE_SAUCE expected and generated Controls yielded the following error:
      List sizes are not same, expected (5) vs generated (6)

Action: select_item on ID_SUNDAE_SAUCE within container ID_DESSERT_DLG
State 3:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SUNDAE_SAUCE expected and generated Controls yielded the following error:
      List sizes are not same, expected (5) vs generated (6)

Action: return_key pressed on ID_SUNDAE within container ID_DESSERT_DLG
State 4:
  Failure: Comparison between ID_FOOD_ORDERER_DLG+ID_DESSERT_DLG expected and
      generated Containers yielded the following error: Comparison between
      ID_SUNDAE_SAUCE expected and generated Controls yielded the following error:
      List sizes are not same, expected (5) vs generated (6)

Action: return_key pressed on ID_OK within container ID_DESSERT_DLG
State 5:
  Failure: The Node was expected to contain 1 Containers, instead contained 2
      Containers
```