Test Oracles

• Discussion
  - Automation of oracle necessary
  - Expected behavior given
  - Necessary parts of an oracle
  - Name spaces
Test Oracle

• A test oracle determines whether a system behaves correctly for test execution

• Webster Dictionary – Oracle
  - a person giving wise or authoritative decisions or opinions
  - an authoritative or wise expression or answer
Purpose of Test Oracle

- Sequential Systems
  - Check functionality
- Reactive (event-driven) Systems
  - Check functionality
  - Timing
  - Safety
Reactive Systems

• Complete specification requires use of multiple computational paradigms
• Oracles must judge all behavioral aspects in comparison with all system specifications and requirements
• Hence oracles may be developed directly from formal specifications
Parts of an Oracle

• **Oracle information**
  - Specifies what constitutes correct behavior
    • Examples: input/output pairs, embedded assertions

• **Oracle procedure**
  - Verifies the test execution results with respect to the oracle information
    • Examples: equality

• **Test monitor**
  - Captures the execution information from the run-time environment
    • Examples
      - Simple systems: directly from output
      - Reactive systems: events, timing information, stimuli, and responses
Approach

• Test class
  - Set of test data described by a condition that constrains input data and the initial system state

• Every test class will have an explicitly represented test oracle

• Results are monitored and verified against the oracle corresponding to all test classes satisfied for the test data
Phases of the Approach

- Oracle derivation
  - From specifications for each test class
- Monitoring test execution
- Mapping and applying the oracle procedure to the execution results
Automated Test Oracles for GUIs

Foundations of Software Engineering, 2000
GUI Test Cases

- GUI Test Case
  - Sequence of events
A Test Case for WordPad

This is the text.

SelectText ("This") → Format → Font → 18 → OK → SelectText ("text")

Format → Font → Underline → OK
What is Correct Behavior

SelectText ("This") → Format → Font → 18 → OK

SelectText ("text") → Format → Font → Underline → OK

Check State, not only Output !!
Research Focus

• **Goal**
  - To check the GUI's state after each event

• **Approaches**
  - Manual
  - Automated

• **Challenges**
  - Generating expected state
  - Extracting actual state
  - Comparing expected & actual states
Outline

• Overview of GUI Oracle

• Generating Expected State
  – Modeling the GUI’s State
    • Objects
    • Properties
  – Modeling the Events

• Obtaining Actual GUI’s State

• Comparing Actual & Expected States
Overview of GUI Oracle

- Test Case
- Expected-state Generator
- Verifier
- Verdict
- Oracle
- Actual State
- Expected State
- Formal GUI Model
- Run-time information from executing GUI

Diagram:
- Test Case flows to Expected-state Generator.
- Expected-state Generator produces Expected State.
- Expected State flows to Verifier.
- Verifier receives Actual State.
- Verdict is generated.
- Oracle is involved in the process.
Modeling the GUI

A GUI consists of Objects

---

**Form**

- Window State: wsNormal
- Width: 1088
- AutoScroll: TRUE

**Label**

- Align: alNone
- Caption: Files of type:
- Color: clBtnFace
- Font: (tFont)

**Button**

- Caption: Cancel
- Enabled: TRUE
- Visible: TRUE
- Height: 65
All Properties of Cancel
Determining Properties

- Manual Examination of GUI
- Specifications (Reduced Set)
  - GUI being tested
- Toolkit/Language (Complete Set)
  - All available properties

Now we know how to represent the GUI's state
Modeling Events

- Events are State Transducers

SelectText ("This")

Event: \( e \)

Notation: \( S_j = [S_i, e] \)

State: \( S_i \)

State: \( S_j \)
Representing Events

• We define an event as:
  \[ \text{State}_j = [\text{State}_i, \text{event}] \]

• For example:
  \[ \text{State}_j = [\text{State}_i, \text{cut}] \]

• Need a compact representation
**Operators**

Operator :: **CUT**

**Preconditions:**

\[ \text{isCurrent(Menu2).} \]

**Effects:**

\[
\text{FORALL Obj in Objects}
\]

\[
\text{Selected(Obj) } \Rightarrow \\
\text{ADD inClipboard(Obj) }
\]

\[
\text{DEL onScreen(Obj)}
\]

\[
\text{DEL Selected(Obj)}
\]

\[
\text{ADD isCurrent(Menu1)}
\]

\[
\text{DEL isCurrent(Menu2).}
\]

*Obtaining next state*
Deriving Expected State

- Given $S_0$, the initial state,
- A sequence of events
- Obtain $S_1 = [S_0, e_1]$
- And $S_i = [S_{i-1}, e_i]$
Obtaining Actual GUI's State

- **Execution Monitor**
  - Screen Scraping
  - Queries
  - Compatible with Expected State
  - Returns `<Object, Property, Value>`
    - `<Button1, "Caption", "Cancel">`
Automated Execution

- Test Executor
- GUI Under Test
- Execution Monitor
- Verifier
- Test Cases
- Expected State

ACTUAL STATE:
(isCurrent ROOT)
(Contains ROOT D)
(Contains ROOT D)
Comparing Actual and Expected States

• Verifier

• Three Levels of Testing
  - Changed Property Set *(Operators)*
  - GUI Relevant Property Set *(Specifications)*
  - Complete Property Set *(Toolkit/Language)*

• Hybrid Approach
  - Use all 3