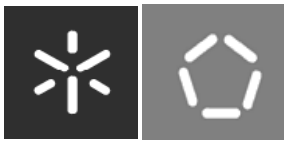


GUI Testing

Ana Paiva

apaiva@fe.up.pt

www.fe.up.pt/~apaiva

www.fe.up.pt/~softeng

Agenda

- Introduction
- GUI testing approaches
  - Manual testing
  - Static analysis
  - Automated GUI testing tools
    - Capture/Replay testing tools
    - Random testing
    - Unit testing
    - Model-Based testing
- Spec Explorer extensions for GUI testing

GUI testing raises specific challenges

- GUI test automation is harder than API test automation
  - Documentation; GUIs are slower that APIs
- Observing visible GUI state is difficult
- Observing invisible GUI state is tricky almost impossible
- Controlling GUI actions is difficult
  - Event based
  - Non-solicited events
  - How to simulate user actions?
- State space and test case explosion
  - Multiple ways (mouse, keyboard,…) to achieve the same goal
  - Almost all user actions are enabled most of the time

GUI errors – concrete examples

- Correct functioning
- Missing commands
- Correct window modality
- Mandatory fields
- Incorrect field defaults
- Data validation
- Error handling (messages to the user)
- Wrong values retrieved by queries
- Fill order
- (…)

Agenda

- Introduction
- GUI testing approaches
  - Manual testing
  - Static analysis
  - Automated GUI testing tools
    - Capture/Replay testing tools
    - Random testing
    - Unit testing
    - Model-Based testing
- Spec Explorer extensions for GUI testing

Manual testing (V&V) techniques

- **Heuristic Methods**
  - A group of *specialists* studies the SW in order to find problems that they can identify
- **Guidelines**
  - Recommendations about the SW and UI. E.g.,: how to organize the display and the menu structure
- **Cognitive walkthrough**
  - The *developers* walk through the SW in the context of core tasks a typical user will need to accomplish. The actions and the feedback of the interface are compared to the user's goals and knowledge, and discrepancies between user's expectations and the steps required by the interface are noted
- **Usability tests**
  - The SW is studied under real-world or controlled conditions (*real users*), with evaluators gathering data on problems that rise during its use

[JMWU91]

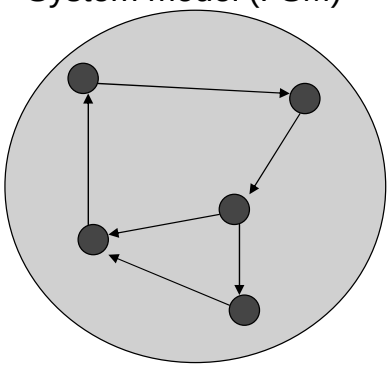
Manual testing

- Advantages
  - More bugs found per test cases executed (good specialist) (not necessarily by time or money invested)
  - Adaptability: bugs found provide hints to find other bugs
  - Can find bugs that are difficult to find with automated tests (the converse is also true)
    - Usability problems
  - Can be supported / made more systematic/repeatable by checklists of standard tests and application specific tests
  - Good for exploratory / initial testing

- Disadvantages
  - Regression testing
  - Effort required
  - Weak coverage
  - Repeatability / reproducibility
  - Good test specialists are difficult to find
  - Depends on the capabilities of the tester

Formal static analysis: Model-checking

System model (FSM)



x = T,T,
y = F,F,

Abstraction – System representation is replaced by a simpler one in which irrelevant low level details are removed.

Bounding the state space – the domains of the state variables are bound to a certain number of possible values.

Partial Order Reduction (POR) – POR is based on the fact that the order in which concurrent transitions are executed does not influence the result.

Symbolic model checking – an implicit representation of the states and transitions that model the system.

Binary Decision Diagrams (BDD) – A special case of Symbolic model checking techniques where the implicit representation of the states and transitions is based on Boolean formulas.

Formal static analysis: formal proofs

- Verify if the implementation (*I*) performs the specification (*S*). This can be expressed mathematically either by

  $I \rightarrow S$ or

  $I \equiv S$

- The theorem that has to be proved.
  - *S* and *I* expressed in the same formal language
  - The formal proof is rigorously constructed as a sequence of steps based on a set of axioms and inference rules, like simplification, rewriting, and induction

Static code analysis

- Lightweight "intellisense"-like checking
  - Check that guidelines are followed, e.g.,
    - Button placement
    - Use only colours which are distinguishable by colour-blind users
  - Check that UI components are used appropriately
    - E.g., button without a Click event handler

Agenda

- Introduction
- GUI testing approaches
  - Manual testing
  - Static analysis
  - Automated GUI testing tools
    - Capture/Replay testing tools
    - Random testing
    - Unit testing
    - Model-Based testing
- Spec Explorer extensions for GUI testing

Automated testing approaches

- Capture-Replay tools
  - E.g., WinRunner, Rational Robot, Android
- Random input testing tools
  - E.g., Rational's TestFactory uses dumb monkey method
- Unit testing frameworks
  - E.g., JUnit, NUnit
- Model-based testing tools
  - E.g., Spec Explorer (extended)
  - E.g., Guitar (for GUI testing)

GUI testing tools

- In the Marketplace:
  - CompuWare TestPartner - www.compuware.com
  - Rational Robot
  - Rational Visual Test - www.rational.com
  - WinRunner
  - LoadRunner - www.mercuryinteractive.com
  - Segue's SilkTest - www.segue.com
- Open Source
  - Abbot
  - GUITAR
  - Pounder
  - qftestJUI
  - Android - (http://www.wildopensource.com/activities/larry-projects/android.php)
  - GUI test drivers – (http://www.testingfaqs.org/t-gui.html)

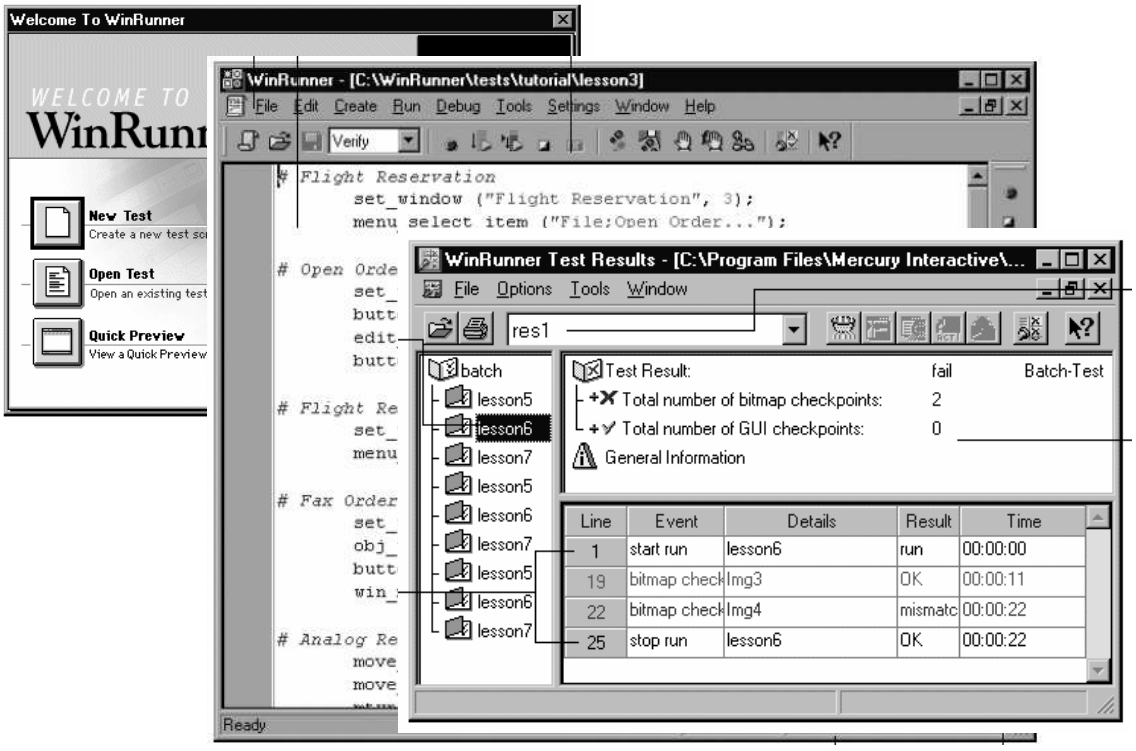
Capture replay tools: how it works?

- Two different interacting modes
  - **Capture**: The user/tester interacts with the tool
    The tools saves user actions and output
  - **Replay**: Saved user actions are reproduced
    The output obtained is compared with the one expected
- Three different execution modes:
  - Position based; Object based; Mixed
- Output verification
  - Property/content comparison; Bitmap comparison; Optical character recognition (OCR)
- Scripts
  - Saved from user actions; From scratch; Mixed
- A higher level of abstraction is needed to reach independence of GUI updates

Capture replay tools

- Advantages
  - Test case recording (capture) (similar to macro recording in Excel)
  - Recorded test scripts can be made more generic by programming
    - E.g., use logical names in test script and map logical names to physical objects in a separate file
  - Automatic test case execution (replay)
  - Output checking
- Disadvantages
  - Can be used only when the SW application is working correctly
  - (good for regression testing but) Sensitivity to physical details
  - **Do not support automatic generation of test cases.**

Capture/Replay tool: WinRunner



Random testing tools

- Do you know that a monkey using a piano keyboard could play a Vivaldi opera? Could the same monkey, using your application, discovery defects?
- Actions performed randomly without knowledge of how humans use the application
- Microsoft says that 10 to 20% of the bugs in Microsoft projects are found by these tools
- Two kinds of tools
  - **Dumb monkeys** – low IQ; they can't recognize an error when they see one
  - **Smart monkeys** – generate inputs with some knowledge to reflect expected usage; get knowledge from state table or model of the AUT.

[N00]

Random input testing tools

- Advantages
  - Good for finding system crashes
  - No effort in generating test cases
  - Independent of GUI updates
  - Increase confidence on the SW when running several hours without finding errors
  - "Easy" to implement
- Disadvantages
  - Not good for finding other kinds of errors
  - Difficult to reproduce the errors
  - Unpredictable

Unit testing frameworks

- E.g., JUnit, NUnit
- Advantages
  - Flexibility
  - Automatic test execution
  - Support TDD (Test Driven Development)
  - Used in combination with appropriate GUI test libraries (e.g., Jemmy and Abbot) can be used for GUI testing (essentially reduce GUI to API testing
- Disadvantages
  - Test cases are programmed manually
  - Usually more lines of testing code than application code

[UTF-site]

Model-based GUI testing

- Advantages
  - Higher degree of automation (test case generation)
  - Allows more exhaustive testing
  - Good for correctness/functional testing
  - Model can be easily adapted to changes
- Disadvantages
  - Requires a formal specification/model
  - Test case explosion
  - Test case generation has to be controlled appropriately to generate a test case of manageable size
  - Small changes to the model can result in a totally different test suite

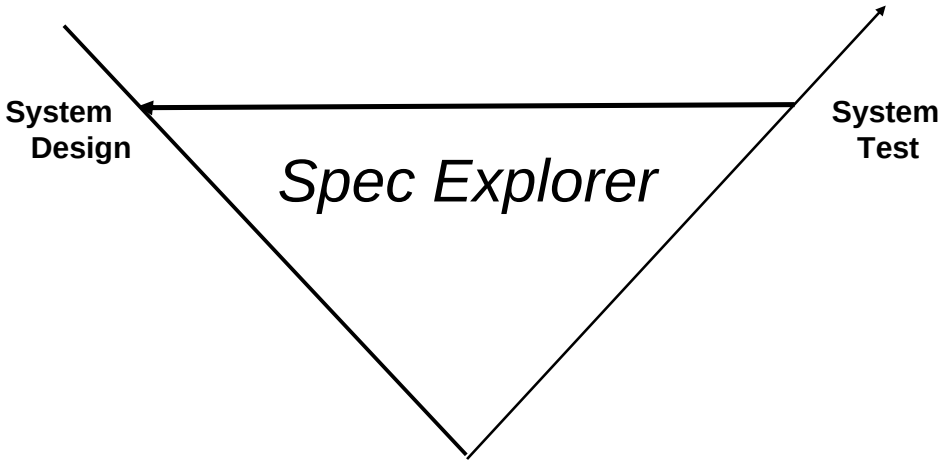
GUI model

- Abstract models. Ex.: PIE and RED-PIE.
- Grammars. Ex.: BNF.
- State-based. Ex.: FSM, Petri nets.
- Model-based. Ex.: VDM, Z.
- Property-based
  - Algebraic. Ex.: OBJ.
- Behavior-based:
  - Process Algebras. Ex.: CSP
- Hybrid approaches. Ex.: VDM and CSP.

Agenda

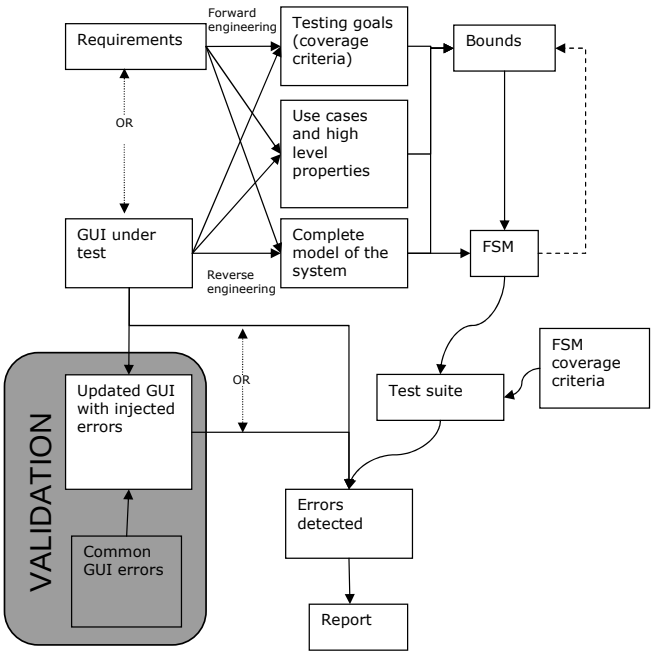
- Introduction
- GUI testing approaches
  - Manual testing
  - Static analysis
  - Automated GUI testing tools
    - Capture/Replay testing tools
    - Random testing
    - Unit testing
    - Model-Based testing
- Spec Explorer extensions for GUI testing

System Testing with model programs



Microsoft Research (FSE) : Colin Campbell, Wolfgang Grieskamp, Yuri Gurevich, Lev Nachmanson, Wolfram Schulte, Nikolai Tillmann, Margus Veanes

GUI testing process



GUI modelling with Spec#

- One module (sub-model) for each window/dialog
- State variables inside modules to model the state of the window
- Methods annotated as actions to model the possible user actions on that window
- Probe methods (with Get as name's prefix) to observe the state of the GUI
- Actions inside each module has at least one pre-condition: the corresponding window is enabled (one window is enabled when it is open and doesn't have a child modal window on top)

GUI modelling with Spec# - Example

```
namespace MyNotepad;

//State variables
string text="",selText="";
bool dirty=false;
int posCursor=0;

// Start and close the Notepad application
[Action] void LaunchNotepad()
requires !IsOpen("Notepad"); {
  AddWindow("Notepad","",false);
  //... state variables initialization ...
}
```

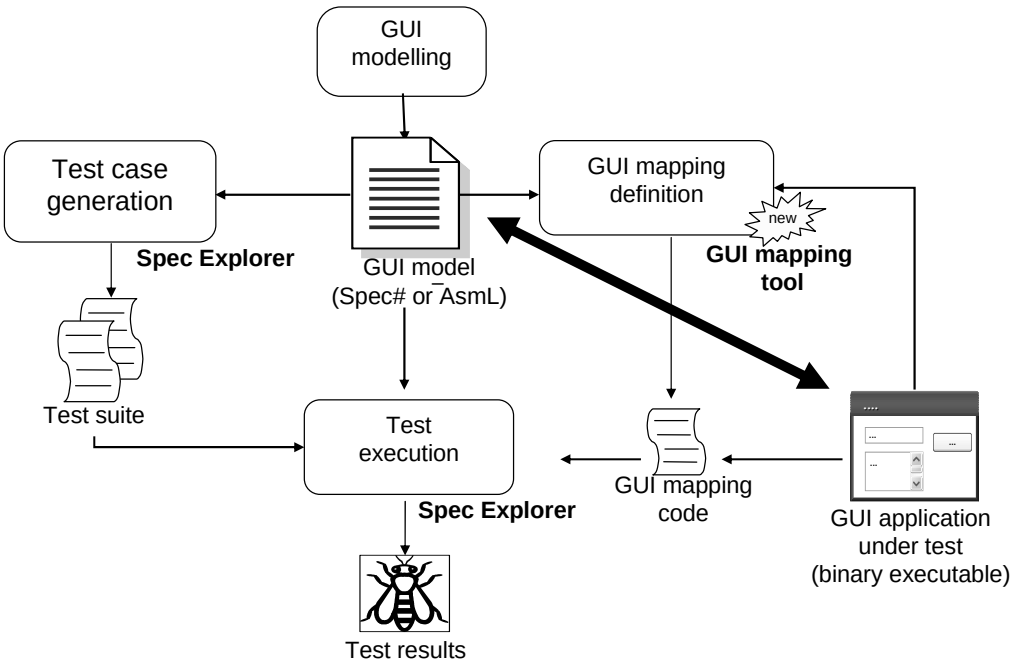
```
[Action]
void MsgSaveChangesBeforeClose(string option)
requires IsEnabled("MsgClose"); {
  RemoveWindow("MsgClose");
  switch (option){
    case "No": RemoveWindow("Notepad"); return;
    case "Yes": AddWindow("Save"); return;
    case "Cancel": return;
    default: return;
  }
}

// change and query the content of the main window
[Action(Kind=Probe)] string GetText()
requires IsEnabled("Notepad"); {
  return text;
}
```

Window manager

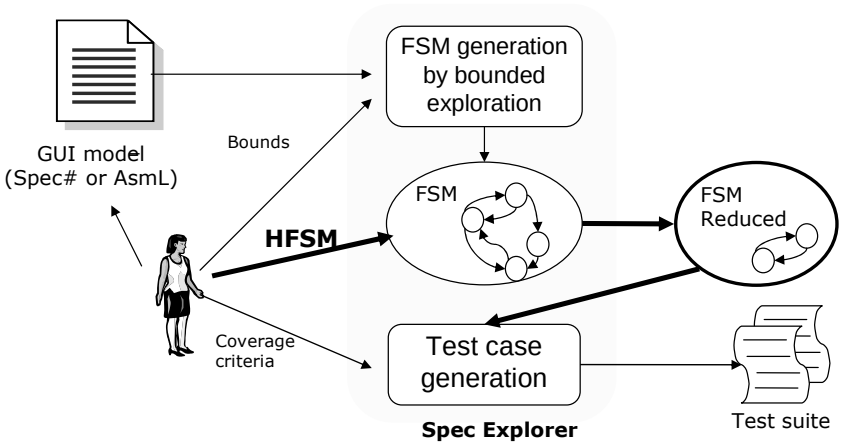
- **AddWindow**(windowName, parentWindowName, isModal)
- **RemoveWindow**(windowName)
- **IsEnabled**(windowName)
- **IsOpen**(windowName)

  - The state keeps information about the hierarchical active windows' structure.
  - When a method open/closes a window, such window must be added/removed to/from the window manager.
  - When a window is closed, all its child windows are closed.
  - One window is enabled when it is open and there isn't a modal window on top.

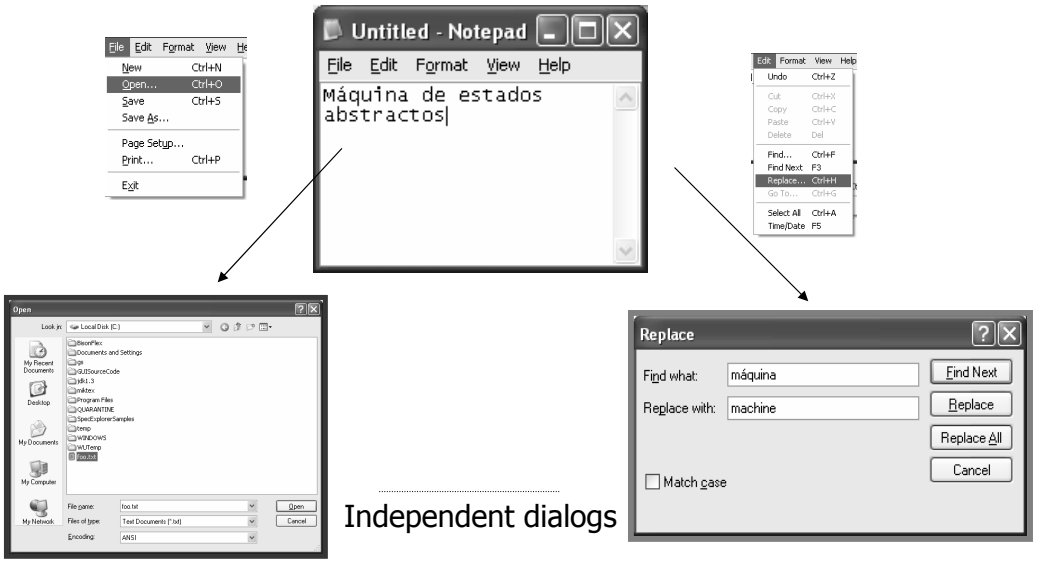
Extensions to Spec Explorer



Test case generation (extensions)



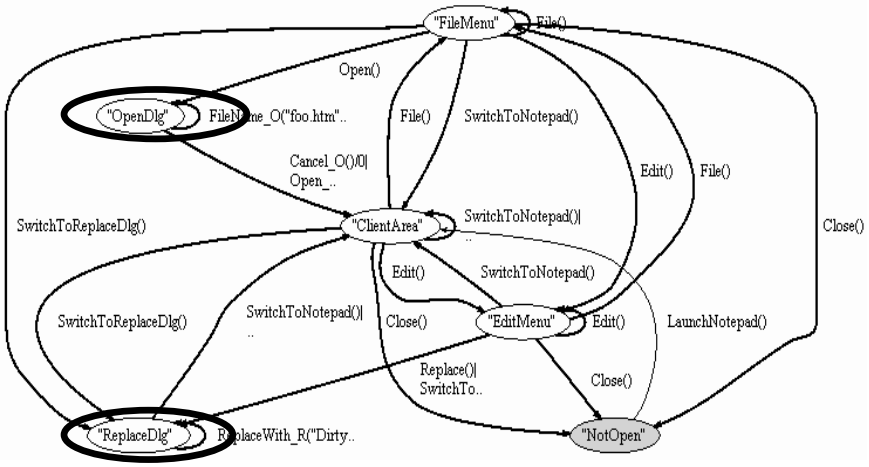
FSM reduction: example



Independent dialogs

HFSM: top level

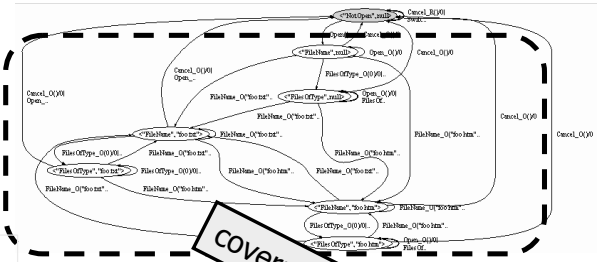
$$Navigation\ map(FSM) = \{(\pi_{GetWindowWithFocus()}s, a, \pi_{GetWindowWithFocus()}s') \mid (s,a,s') \in T\}.$$



HFSM: middle level

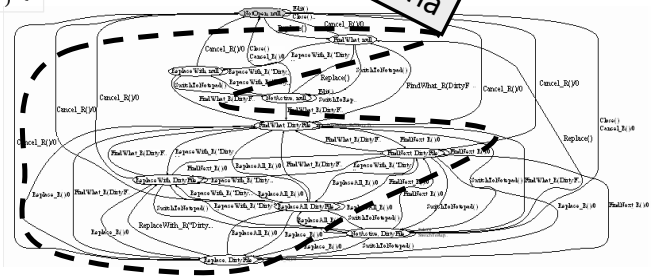
FSM of Open dialog

$$PFSM_{Dia\log} = \{(\pi_{ManipulatedVariables("Dia\log")}(s), a, \pi_{ManipulatedVariables("Dia\log")}(s')) \mid (s,a,s') \in T \wedge s,s' \in \sigma_{IsEnabled("Dia\log")}S\}.$$



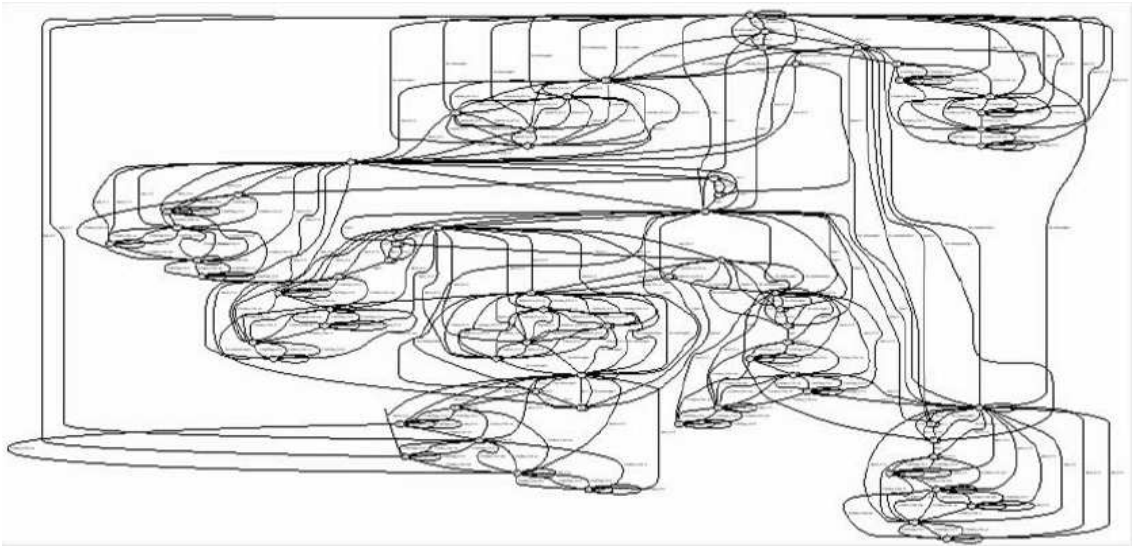
coverage criteria

...



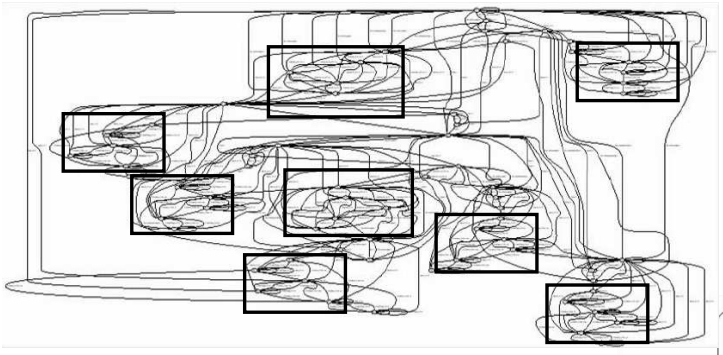
FSM of Replace dialof

HFSM: bottom level

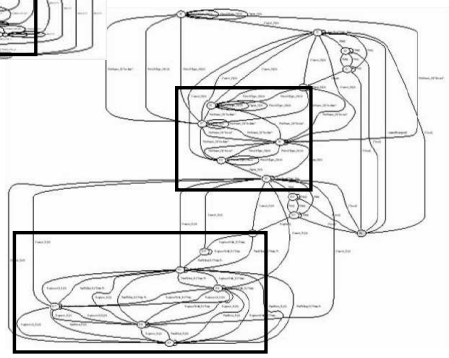


FSM original:
70 states and 399 transitions
Test suite with 673 steps

FSM reduction (preserving top level)



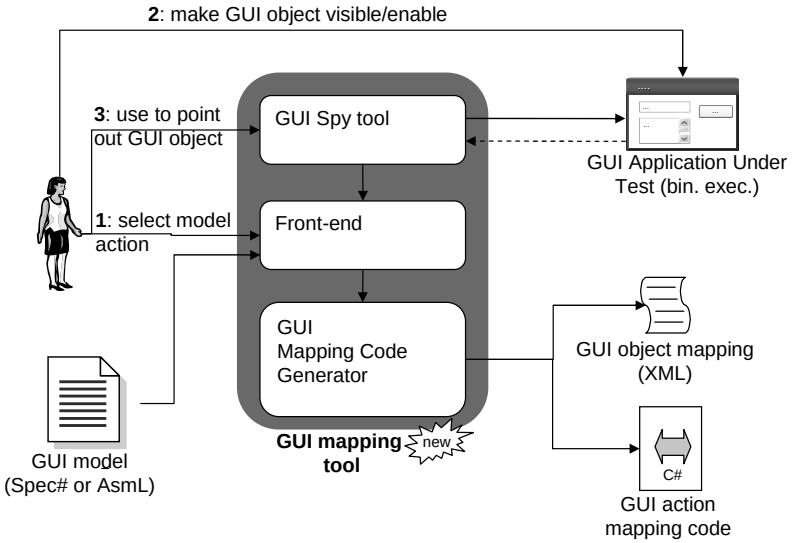
FSM original:
70 states and 399 transitions
Test suite with 673 steps



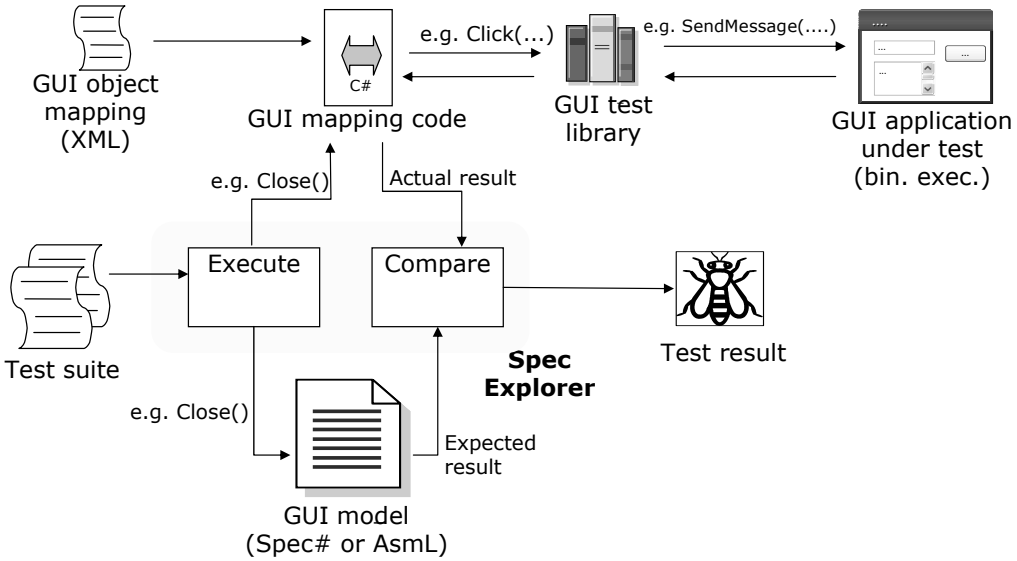
Garbage

FSM reduced:
20 states and 95 transitions
Test suite with 147 steps

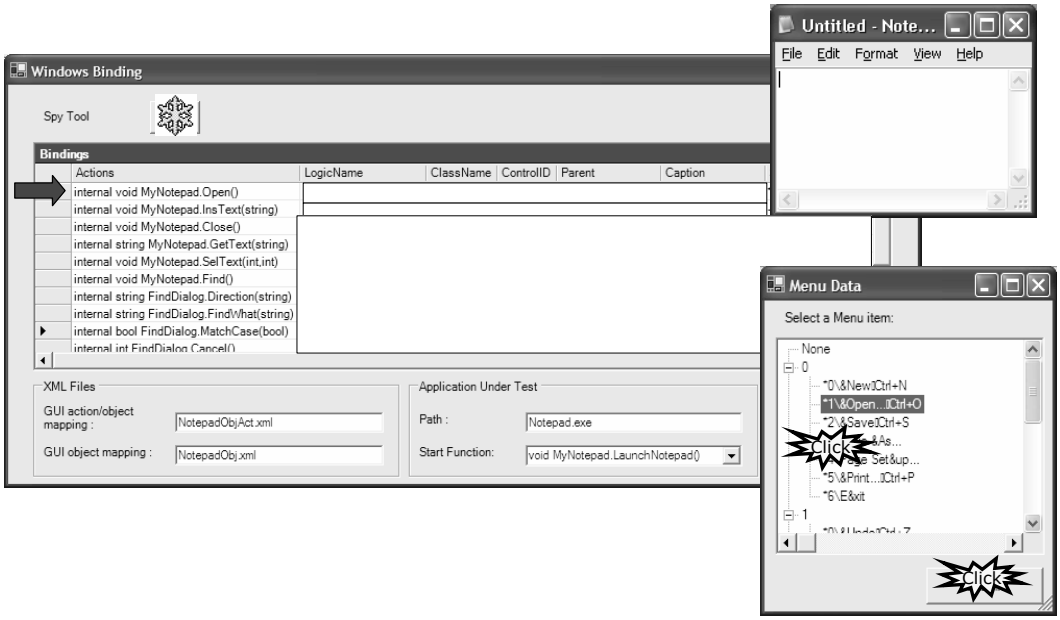
GUI mapping tool architecture



Test execution and the GUI test library



Front-end



Generated code

```
#region automatically generated code
{
    //…
    public static string MyNotepad_GetText(){
            return UserEvents.Event_GetText("MyNotepad.Text");
    }
    public static void MyNotepad_InsText(string p0){
            UserEvents.Event_SendText("MyNotepad.Text", p0);
    }
    public static void MyNotepad_Close(){
            UserEvents.Event_SelectOption("MyNotepad.Close");
    }
    public static void MyNotepad_AcknMsgCannotFind(){
            UserEvents.Event_MessageBox("MyNotepad.AcknMsgCannotFind","");
    }
    public static void ReplaceDialog_Cancel(){
            UserEvents.Event_Click("ReplaceDialog.RCancel");
    }
    // …
}
#endregion
```

XML files

Logical to physical object mapping

```
<obj diffgr:id="obj2" msdata:rowOrder="1"
   diffgr:hasChanges="inserted">
      <LogicalName>MyNotepad.Text</LogicalName>
      <ClassName>Edit</ClassName>
      <ControlID>15</ControlID>
      <Caption>-</Caption>
      <SubOption>-</SubOption>
      <ParentClassName>Notepad</ParentClassName>
      <ParentCaption>Untitled - Notepad</ParentCaption>
      <SubOpClassName>-</SubOpClassName>
      <FullPath>-</FullPath>
</obj>
```


GUI test library

// To act upon GUI objects

```
void Click(string GUIObjName);
void SendText(string GUIObjName, string txt);
void SelectText(string GUIObjName, int start, int end);
void SelectSubOption(string GUIObjName, string option);
void SelectCheckBox(string GUIObjName, bool check);
void SelectMsgBoxOp(string GUIObjName, string option);
```

// To observe properties of GUI objects

```
string GetText(string GUIObjName);
string GetSelectedText(string GUIObjName);
int GetInsertionPoint(string GUIObjName);
bool GetCheckBox(string GUIObjName);
```

// To map logical object names to physical objects

```
void LoadXMLObjMapping(string XMLFileName);
```

GUI Errors

Pre-condition mismatch:

- a test case is trying to act on a control that is not enabled or cannot be found; or
- a test case is trying to act on a window that is not reachable or is not opened (e.g., a modal dialog is open and the window we want to reach is behind that dialog);

Post-condition mismatch:

- the expected result was not displayed (e.g., a text box does not display the expected content);

Demo

References

Main references

- [P07] Ana C. R. Paiva, PhD Thesis untitled “Automated Model-Based Testing of Graphical User Interfaces”: www.fe.up.pt/~apaiva/PhD/PhDGUITesting.pdf
- [UL07] Mark Utting and Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan Kaufmann, Elsevier, 2007.

Other references

- [N00] N. Nyman, "Using Monkey Test Tools," in *STQE - Software Testing and Quality Engineering Magazine*, 2000.
- [UTF-site] For Unit testing frameworks visit sites – www.nunit.org; www.junit.org
- [JMWU91] R. Jeffries, J. R. Miller, C. Wharton, and K. M. Uyeda, "User Interface Evaluation in the Real World: A Comparison of Four Techniques," 1991
- [GC96] C. Gram and G. Cockton, *Design Principles for Interactive Software*, Chapman & Hall, isbn:0412724707, 1996
- [HVCKR01] J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, "A Practical Tutorial on Modified Condition / Decision Coverage," NASA/TM-2001-210876, 2001.
- [BS02] J. Bredereke and B.-H. Schlingloff, "An Automated, Flexible Testing Environment for UMTS", in Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV, 2002.
- [ABM98] P. E. Ammann, P. E. Black, and W. Majurski, "Using Model Checking to Generate Tests from Specifications", in Proceedings of the 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98), M. G. H. John Staples, and Shaoying Liu(Eds.), Brisbane, Australia, 1998.

SpecExplorer and Spec# references

- [BLS04] M. Barnett, K. R. M. Leino, and W. Schulte, "The Spec# Programming System: An Overview," presented at CASSIS'04 - International workshop on Construction and Analysis of Safe, Secure and Interoperable Smart devices, Marseille, 2004.
- [CGNSTV05] C. Compbell, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veans, "Testing Concurrent Object-Oriented Systems with Spec Explorer," presented at FM, 2005.
- [FSE-site] Visit FSE web site for more information: research.microsoft.com/foundations
- [SE-site] Visit Spec Explorer site for download: research.microsoft.com/SpecExplorer/

Additional reading

- [W03] James A. Whittaker, “How to Break Software: A Practical Guide to Testing”, ISBN: 0201796198
- [A-site] Alan Hartman – http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf
- Model-Based Testing papers – www.geocities.com/model_based_testing/online_papers.htm

END