GUI Reverse Engineering with Machine Learning

Inês Coimbra Morgado, inescm@fe.up.pt
Ana C. R. Paiva, apaiva@fe.up.pt
João Pascoal Faria, jpf@fe.up.pt
Rui Camacho, rcamacho@fe.up.pt
“the process of analysing a subject system to identify the system’s components and interrelationships and to create representations of the system in another form or at a higher level of abstraction”

Chikofsky and Cross, 1990
Software Reverse Engineering

Exploration of the system

“the process of analysing a subject system to identify the system’s components and interrelationships and to create representations of the system in another form or at a higher level of abstraction”

Chikofsky and Cross, 1990
Representation of the information

"the process of analysing a subject system to identify the system’s components and interrelationships and to create representations of the system in another form or at a higher level of abstraction"

Chikofsky and Cross, 1990
Motivation & Goal

- Hard to manually build a model
- Extract (part of) the model of a GUI automatically and dynamically (in run time)
- Reduce the effort of building a formal model
Application Under Analysis
Application Under Analysis
Application Under Analysis
Application Under Analysis
Application Under Analysis
Exploration Process

![Diagram](image-url)
# Execution Traces

<table>
<thead>
<tr>
<th>Trace ID</th>
<th>Event (User Action)</th>
<th>Text</th>
<th>Enter/Text(X)</th>
<th>Find</th>
<th>Exit</th>
<th>Find/Exit</th>
<th>Enter/Find/Exit(Y)</th>
<th>Find</th>
<th>Cancel</th>
<th>Ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Start</td>
<td>[]</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0.1</td>
<td>W1.Exit</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1.0</td>
<td>Start</td>
<td>[]</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1.1</td>
<td>W1.EnterText(&quot;a&quot;)</td>
<td>[a]</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1.2</td>
<td>W1.Find</td>
<td>[a]</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>[a]</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>-</td>
</tr>
<tr>
<td>1.3</td>
<td>W2.EnterFindWhat(&quot;a&quot;)</td>
<td>[a]</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>[a]</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>-</td>
</tr>
<tr>
<td>1.4</td>
<td>W2.Find</td>
<td>[a]</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1.5</td>
<td>W1.Exit</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2.0</td>
<td>Start</td>
<td>[]</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2.1</td>
<td>W1.EnterText(&quot;b&quot;)</td>
<td>[b]</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2.2</td>
<td>W1.Find</td>
<td>[b]</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>[b]</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>-</td>
</tr>
<tr>
<td>2.3</td>
<td>W2.EnterFindWhat(&quot;a&quot;)</td>
<td>[b]</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>[a]</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>-</td>
</tr>
<tr>
<td>2.4</td>
<td>W2.Find</td>
<td>[b]</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>[a]</td>
<td>D</td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>2.5</td>
<td>W3.Ok</td>
<td>[b]</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2.6</td>
<td>W1.Exit</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Model Generation Process

1. **Execution Traces**
2. **FSM Generation** *(FSM Generator)*
3. **Ambiguous FSM Model**
4. **FSM Disambiguation** *(ILP Engine)*
5. **Not Ambiguous FSM Model**
6. **Verification & Refinement** *(Verifier)*
# Execution Traces & FSM States

<table>
<thead>
<tr>
<th>TraceID</th>
<th>Event/Action</th>
<th>Next FSM State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Start</td>
<td>S0</td>
</tr>
<tr>
<td>0.1</td>
<td>W1.Exit</td>
<td>End</td>
</tr>
<tr>
<td>1.0</td>
<td>Start</td>
<td>S0</td>
</tr>
<tr>
<td>1.1</td>
<td>W1.EnterText(“a”)</td>
<td>S1</td>
</tr>
<tr>
<td>1.2</td>
<td>W1.Find</td>
<td>S2</td>
</tr>
<tr>
<td>1.3</td>
<td>W2.EnterFindWhat(“a”)</td>
<td>S3</td>
</tr>
<tr>
<td>1.4</td>
<td>W2.Find</td>
<td>S1</td>
</tr>
<tr>
<td>1.5</td>
<td>W1.Exit</td>
<td>End</td>
</tr>
<tr>
<td>2.0</td>
<td>Start</td>
<td>S0</td>
</tr>
<tr>
<td>2.1</td>
<td>W1.EnterText(“b”)</td>
<td>S1</td>
</tr>
<tr>
<td>2.2</td>
<td>W1.Find</td>
<td>S2</td>
</tr>
<tr>
<td>2.3</td>
<td>W2.EnterFindWhat(“a”)</td>
<td>S3</td>
</tr>
<tr>
<td>2.4</td>
<td>W2.Find</td>
<td>S4</td>
</tr>
<tr>
<td>2.5</td>
<td>W3.Ok</td>
<td>S1</td>
</tr>
<tr>
<td>2.6</td>
<td>W1.Exit</td>
<td>End</td>
</tr>
</tbody>
</table>
Finite State Machine

Start

S0
{EnterText, Exit}

Exit

AppWnd.EnterText(x)

S1
{EnterText, Find, Exit}

AppWnd.Find

S2
{EnterFindWhat, Cancel}

FindDlg.EnterFindWhat(y)

S3
{EnterFindWhat, Find, Cancel}

FindDlg.Find

S4
{Ok}

NotFoundException.Ok
Ambiguity

(a)

Sx

Sy

Sz

(b)

Sx

Sy

E [cond1]

E [cond2]

Sz
Inductive Logic Programming Process

Execution Traces with FSM states

Automatic Encoding

Encoded Transitions and State Variables

Definition of Reusable Patterns

Encoded Patterns (actionInfo)

ILP Engine

Transition Rules

Automatic Guard Conditions Extraction

Not Ambiguous FSM

Activity

Artefact

Dataflow
Encoding of states and transitions

stateVariable(Control, TraceId, StepId, Value).
  stateVariable(text, trace1, step3, [b]).

transition (Source, Action, Target, TraceId, StepId).
  transition(s0, enterText, s1, trace1, step1).
Manual encoding of reusable patterns (once)
  - Login, RangeValidation, Mandatory Field, Find...

```prolog
actionInfo(Action, TraceId, StepId, Result).

actionInfo(find, TraceId, StepId, notFound):-
  stateVariable(text, TraceId, StepId, Text),
  stateVariable(findWhat, TraceId, StepId, FindWhat),
  not member(FindWhat, Text).
```
Inferring transition rules

- Returns the disambiguated transitions

\[ \text{transition}(Source, Action, Target, Traceld, Stepld):=} \]
\[ \text{stateName}(Source, s3), \]
\[ \text{stateName}(Target, s4), \]
\[ \text{actionInfo}(Action, Traceld, Stepld, notFound). \]
Guard Conditions

- Extract the guard conditions:

\[ \text{cond1} = \text{member}(\text{FindWhat}, \text{Text}). \]

\[ \text{cond2} = \text{not member}(\text{FindWhat}, \text{Text}). \]
Not Ambiguous FSM

Start

S0
{EnterText, Exit}

Exit

AppWnd.EnterText(x)

S1
{EnterText, Find, Exit}

AppWnd.Find

S2
{EnterFindWhat, Cancel}

FindDlg.EnterFindWhat(y)

S3
{EnterFindWhat, Find, Cancel}

FindDlg.Find[member(FindWhat, Text)]

FindDlg.Find[not member(FindWhat, Text)]

S4
{Ok}

NotFound.Ok
Conclusions

- Approach to extract model
- Approach to solve ambiguities
- Combines machine learning with software engineering
Future Work

- Explore the encoding of more powerful patterns
- Improve the automatic reuse of patterns
- Transform in iterative process
  - Complement the model at each iteration
  - Use the extracted information to guide the exploration
    - Extract more information for ILP
    - Provide a more complete and intelligent exploration
GUI Reverse Engineering with Machine Learning

Inês Coimbra Morgado, inescm@fe.up.pt
Ana C. R. Paiva, apaiva@fe.up.pt
João Pascoal Faria, jpf@fe.up.pt
Rui Camacho, rcamacho@fe.up.pt

Thank You!

RAISE’12, Zurich, 5th June 2012