Synthesizing Models of the Observer Pattern for Event-Driven Frameworks

Jinseong Jeon\textsuperscript{1)}, Xiaokang Qiu\textsuperscript{2)}, Jeffrey S. Foster\textsuperscript{1)}, and Armando Solar-Lezama\textsuperscript{2)}

1) University of Maryland, College Park
2) Massachusetts Institute of Technology
Model of Event-Driven Frameworks

• Many apps built on top of (event-driven) frameworks
  • Can’t analyze app in isolation

• Analyzing framework code together?
  • Might not be available; written in a different language;
  • Might be available but too complicated to analyze

• Solution: Use a model of framework
  • Simplify and abstract actual framework
  • Need not be precise
    • Over- or under-approximations can be acceptable, depending on the goal

• Currently, models are built manually
  • Difficult to write and maintain, error-prone
Synthesis of Framework Models

- Construct framework model automatically
- By Program Synthesis
  - Easy to revise
    - When analysis is changed/fixed
    - When bugs/mistakes in model spec are found
  - Easy to update
    - When frameworks are updated (e.g., Android)
  - Easy to improve/tune
    - When analysis is redesigned
  - Easy to apply to new (or less popular) frameworks
Our approach: Pasket

• Key observations
  • Many *examples* of how to use frameworks are already out there
    • Tutorials and documentation in SDK
  • Frameworks built using *design patterns*
    • Observer pattern for handling events
    • State machine, singleton, factory and builder, etc.

• Pasket (*Pattern Sketcher*)
  • *Samples* or *tutorials* that call key APIs
    • Run these to see framework behavior
  • *Annotated Template* of framework structure
    • Specify design patterns used
  • Based on the Sketch synthesizer
    • Represent role variables and constraints via holes (??) and assertions
  • Focus to date: Observer pattern in Java Swing
Logging Behavior

**Tutorials / Sample Programs**
- Programs that illustrate pieces of the API
- E.g., Swing comes with 79 tutorials
- Executable specification

```java
1 ButtonDemo.createAndShowGUI()
2 ButtonDemo.ButtonDemo()
3 JButton.setActionCommand(JButton@8, "disable")
4 JButton.setEnabled(JButton@4, false)
5 JButton.addActionListener(JButton@8, ButtonDemo@9)
6 JButton.addActionListener(JButton@4, ButtonDemo@9)
7 ...
8 ActionEvent.ActionEvent(JButton@8, 0, "disable")
9 ButtonDemo.actionPerformed(ButtonDemo@9, ActionEvent@7)
10 ActionEvent.getActionCommand(ActionEvent@7)
11 JButton.setEnabled(JButton@8, false)...
```

**Logs**

- **execute & record calls**
- **callback from framework**
Behavioral Constraints

Logs

```java
1 ButtonDemo.createAndShowGUI()
2 ButtonDemo.ButtonDemo()
3 JButton.addActionListener(JButton@8, ButtonDemo@9)
4 JButton.addActionListener(JButton@8, ButtonDemo@9)
5 JButton.addActionListener(JButton@4, ButtonDemo@9)
6 JButton.addActionListener(JButton@4, ButtonDemo@9)
7 ...
8 ActionEvent.ActionEvent(JButton@8, 0, "disable")
9 ActionEvent.ActionEvent(JButton@8, 0, "disable")
10 ActionEvent.ActionEvent(JButton@8, 0, "disable")
11 ... ...
12 ActionEvent.ActionEvent(JButton@8, 0, "disable")
13 ButtonDemo.actionPerformed(ButtonDemo@9, ActionEvent@7)
14 ActionEvent.getActionCommand(ActionEvent@7)
15 JButton.setEnable(JButton@8, false)
16 JButton.setEnable(JButton@8, false)
```

Sketch – behavioral constraints

```java
void addActionListener(V_Object self, V_Object l) {
  /* addActionListener has id 19 */
  int[] params = {19, self.obj_id, l.obj_id};
  check_log(params);
  observableCall(19, self, l, null);
  params = {-19};
  check_log(params);
}
```

behavior of synthesized model must match logged behavior
**Structural Constraints**

**Templates**
- Annotated framework APIs
  - names/types for *public* classes/methods
  - Hints about design patterns used

  ```java
  @ObserverPattern(ActionEvent)
  public class AbstractButton ... 
  ```

  “This class participates in observer pattern, triggered by `ActionEvent`.”

**Sketch – structural constraints**

```
int observable = ??; int attach = ??; ...
assert subcls [ observable ][ belongsTo[attach ]];
assert subcls [ observable ][ belongsTo[detach ]];
assert subcls [ observable ][ belongsTo[handle ]];
...
```

encode design pattern rules

```
int observable = ??; int attach = ??; ...
assert subcls [ observable ][ belongsTo[attach ]];
assert subcls [ observable ][ belongsTo[detach ]];
assert subcls [ observable ][ belongsTo[handle ]];
...
```

class/method ids, to be solved for

```
int observable = ??; int attach = ??; ...
assert subcls [ observable ][ belongsTo[attach ]];
assert subcls [ observable ][ belongsTo[detach ]];
assert subcls [ observable ][ belongsTo[handle ]];
...
```

these methods must be members of the class that is observed
Model Construction

Sketch – behavioral constraints

```java
void addActionListener(V_Object self, V_Object l) {
    /* addActionListener has id 19 */
    int [] params = { 19, self.obj_id, l.obj_id };
    check_log(params);
    observableCall(19, self, l, null);
    params = {-19};
    check_log(params);
}
```

Sketch – structural constraints

```java
int observable = ??; int attach = ??; ...
assert subcls [ observable ][ belongsTo[attach ]];
assert subcls [ observable ][ belongsTo[detach ]];
assert subcls [ observable ][ belongsTo[handle ]];
```

Sketch Solution

```java
observable = 11; class with id 11
attach = 19; method with id 19
```

Executable Model

```java
public class AbstractButton extends JComponent implements SwingConstants {
    LinkedList<ActionListener> _obs;
    public void addActionListener (ActionListener l) {
        _obs.add(l);
    }
}
```
Results

• Synthesized executable model
  • Sanity check: ran logged event trace to reproduce original trace
  • JPF’s own hand-written models are insufficient, whereas JPF along with our synthesized model can run tutorials.

• Experiments
  • annotated APIs: # classes = 36, # methods = 59, # fields = 33
  • # annotations = 4, # event kinds = 2
  • Terms: # Event, Lines of Code/Logs, Size of search space, and # Classes

| Demo name          | LoC | Sample #E | LoL | LoC          | |S| | Tm | Java LoC | #C |
|--------------------|-----|-----------|-----|--------------|---|-----|-----|----------|----|
| ButtonDemo         | 148 | 2         | 86  | 1,499        | 80,000 (= 2^2 \times 10^4 \times 2) | 354 | 437 | 25      |
| ButtonHtmlDemo     | 151 | 2         | 86  | 1,480        | 80,000 (= 2^2 \times 10^4 \times 2) | 356 | 437 | 25      |
| CheckBoxDemo       | 232 | 2         | 74  | 1,582        | 117,128 (= 2^2 \times 11^4 \times 2) | 274 | 470 | 26      |
| CustomIconDemo     | 233 | 2         | 88  | 1,559        | 80,000 (= 2^2 \times 10^4 \times 2) | 365 | 437 | 25      |
Future Plans

• More tutorials
  • with more different event sorts

• More generalized observer pattern
  • attach with property strings
  • multiple attach/update methods
  • multiple observable classes

• More design patterns
  • state machine, factory and builder, etc.

• Mechanized way to specify design patterns and encoding rules

• Finally, apply to modeling Android