Data-Driven Synthesis for Object-Oriented Frameworks

Armando Solar-Lezama, Zhilei Xu, Kuat Yessenov

OOPSLA 2011
Programming with Frameworks

User Application

Glue Code
<!-- This is a simple configuration file -->

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

<body>
    <name>Configuration</name>
    <property p="234">Property</property>
    <outer>
        <inner>inner</inner>
    </outer>
</body>
What Novice Programmer Knows

FooEditor.setScanner(FooScanner);

FooEditor

RuleBasedScanner

TextEditor

Editor and Scanner Glue Code

class FooConfig extends SourceViewerConfiguration {
    @Override
    public PresentationReconciler(...) {
        DefaultDamagerRepairer dr = new DefaultDamagerRepairer(new FooScanner());
        PresentationReconciler rcr = new PresentationReconciler();
        rcr.setDamager(dr, ...);
        rcr.setRepairer(dr, ...);
        return rcr;
    }
}
class FooEditor extends TextEditor {
    @Override
    void initializeEditor() {
        setSourceViewerConfiguration(new FooConfig());
    }
}
class FooScanner extends RuleBasedScanner {...}
Welcome to MatchMaker!

Please specify source and target classes:

Source: `org.eclipse.ui.editors.text.TextEditor`

Target: `org.eclipse.jface.text.rules.RuleBasedScanner`

(type auto-completion is case-sensitive; click button below to generate code)

Match!

class MyConfig extends SourceViewerConfiguration {
    @Override
    public PresentationReconciler getPresentationReconciler(ISourceViewer arg0) {
        DefaultDamagerRepairer dr = new DefaultDamagerRepairer(new MyScanner());
        PresentationReconciler rcr = new PresentationReconciler();
        rcr.setRepairer(dr, ??);
        return rcr;
    }
}
class MyEditor extends TextEditor {
    @Override
    void initializeEditor() {
        setSourceViewerConfiguration(new MyConfig());
    }
}
class MyScanner extends RuleBasedScanner {}
Learn from Observation

User applications reuse the framework

Observe how the user application dynamically exercises the framework to infer the glue code.
Data-Driven Approach

Query: two types

Answer: (imperfect) Glue Code
MatchMaker Hypothesis

Objects interact via **pointer chains**

**Critical chain** is the first chain between two objects
## MatchMaker Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Collect execution data</td>
<td>Set-up</td>
</tr>
<tr>
<td>1</td>
<td>Build heap series and call tree</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Find critical chains between A and B</td>
<td>Query:</td>
</tr>
<tr>
<td>3</td>
<td>Thin-slice events creating the chain</td>
<td>“How do I connect A and B?”</td>
</tr>
<tr>
<td>4</td>
<td>Project the slice to the user code.</td>
<td></td>
</tr>
</tbody>
</table>

Output: synthesized code snippets modulo simple transformations
Data Collection

• Dynamic byte code instrumentation using ASM
• Records internal program events:
  – Field reads and writes
  – Method enters and exits, exceptions
• Filters out internal library events (e.g. JDK)
Data Analysis

Call Tree:

1. \( a.f \leftarrow b \)
2. `call m(a)`
3. \( c \leftarrow a.g \)
4. \( a.g \leftarrow \text{null} \)
5. `return c`
6. \( b.h \leftarrow c \)
7. `call n(c)`
8. `throw e`
9. `throw e`

Heap Series:

Statements update the heap

```
\begin{tikzpicture}
  \node (a) at (0,0) {a};
  \node (b) at (2cm,0) {b};
  \node (c) at (1cm,-1cm) {c};
  \draw[->] (a) -- (b) node [midway, above] {f};
  \draw[->] (a) -- (c) node [midway, left] {g};
  \draw[->] (b) -- (c) node [midway, right] {h};
\end{tikzpicture}
```

0 Collection 1 Heap series 2 Critical chain 3 Thin slicing 4 Projection
Data Analysis

Call Tree:
1. a.f ← b
2. call m(a)
3. c ← a.g
4. a.g ← null
5. return c
6. b.h ← c
7. call n(c)
8. throw e
9. throw e

Heap Series:
Combine heap configurations
0 Collection  | 1 Heap series  | 2 Critical chain  | 3 Thin slicing  | 4 Projection
Data Analysis

Call Tree:
1. a.f ← b
2. call m(a)
3. c ← a.g
4. a.g ← null
5. return c
6. b.h ← c
7. call n(c)
8. throw e
9. throw e

Heap Series:
Graph on objects labeled with fields and time intervals

0 Collection  |  1 Heap series  |  2 Critical chain  |  3 Thin slicing  |  4 Projection
Critical Chain in Heap Series

Naïve algorithm:

Chain lifetime: [8, 14]
Challenge: Heaps Are Large

Solution: connectivity-preserving abstractions of the heap series
Challenge: Heaps Are Large

- Exhaustively enumerate chains in the heap abstraction

  - chain found
  - fails: learn from failing prefix

- Concretize chain in the heap series

  - succeeds

- Output the chain

0 Collection 1 Heap series 2 Critical chain 3 Thin slicing 4 Projection
Type-Based Heap Abstraction

Type A

- a → b
- c → d

Type B

A → B

α
Time-Based Heap Abstraction

\[(f, [0,5] \cup [8,15] \cup [19,33])\]
Projection of Thin Slices

Thin slicing computes a data-dependency relation effective for program understanding.
Projection of Thin Slices

Thin slicing computes a data-dependency relation effective for program understanding.
Projection generalizes to the essential code creating the chain.

- calls from the user to the framework
- returns from the user code
- user producers of framework consum
Evaluation

• User Study
  – MatchMaker generates imperfect glue code.
  – Does it still help improve productivity?

• Case Study
  – Can MatchMaker generate glue code for many class pairs other than editor-scanner?
User Study - Task

- Implement Syntax Highlighting in an editor, highlighting keywords "implements" and "??"
User Study - Task

```
TextEditor

MyEditor
{
}

RuleBasedScanner

MyScanner
{
  code for scanning "implements"
  code for scanning "??"
}
```
User Study - Task

```
MyEditor
{
}
```

```
RuleBasedScanner
{
  MyScanner
  {
    code for scanning "implements"
    code for scanning "??" (task code)
  }
}
```
User Study - Task

```java
g^A TextEditor
{
    ...
}

g^A RuleBasedScanner
{
    code for scanning "implements"
    code for scanning "??"
    (task code)
}
```

```plaintext
g^A MyEditor
{
    ...
}

g^A MyScanner
{
    code for scanning "implements"
    code for scanning "??"
    (task code)
}
User Study - Participants

• 9 participants
  – One Eclipse framework expert
    • Years of experience writing Eclipse plug-ins
    • No experience implementing syntax highlighting

  – 8 novice users
    • 4 control subjects
    • 4 MatchMaker users
    • All of them are experienced Java programmers
User Study - Result

• Average time to complete the whole task:

  - Expert: 31 mins
  - MatchMaker Users: 50 mins
  - Control Subjects: 98 mins
User Study - Result

Expert

- Web browsing for glue code
- Writing & debugging glue code
- Time spent on task code

MatchMaker Users

Control Subjects

10 6 15
24 11 15
65
18 15
User Study - Observation

• Documentation on the web is not enough
  – JavaDoc: class-by-class, not helpful for interactions
  – Tutorials: lengthy, hard to get the essential piece

• MatchMaker makes browsing more efficient
  – Generated glue code depicts relations among classes
  – Only use web to find holes / missing statements
  – Users have correct class/method names as keyword to quickly skim the documentation
MatchMaker improves productivity even when generated glue code is imperfect
Case Study

Can MatchMaker do more than connect editors and scanners?

Can MatchMaker generate glue code for other interesting class pairs?
Case Study
eclipse.jdt.internal.ui
Case Study

eclipse.jdt.internal.ui
grep

200+ classes extend the framework
Case Study

eclipse.jdt.internal.ui

IQuickAssistProcessor

QuickAssistAssistant

16 interesting class pairs
# Case Study - Result

<table>
<thead>
<tr>
<th>No.</th>
<th>Class pair</th>
<th># holes</th>
<th># missing statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>QuickAssistAssistant-IQuickAssistProcessor</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>ITextEditor-ITextHover</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>ITextEditor-TextFileDocumentProvider</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>ITextEditor-QuickAssistAssistant</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>ITextEditor-SelectionHistory</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>ITextEditor-IAutoEditStrategy</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>ISpellCheckEngine-ISpellChecker</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>ITextEditor-IContentOutlinePage</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>MonoReconciler-IReconcilingStrategy</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>IContentAssistProcessor-ContentAssistInvocationContext</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>ITextEditor-ContextBasedFormattingStrategy</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>ITextEditor-SemanticHighlighting</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Case Study - Result

MatchMaker generates reasonable glue code for 12 / 16 class pairs we have tested.

even the program execution database is quite limited.
Conclusion

Software today becomes so large and dynamic
Seeing the code is not enough to learn about it

Data-driven approach: the power of reasoning
about dynamic behaviors of programs in real time

Opens a new gate for programming tools
Thank you!

Try MatchMaker at sketch1.csail.mit.edu/matchmaker

still a prototype, not mature
the program execution database is very limited