Dynamic Analysis and Design Pattern Detection in Java Programs

Lei Hu
Kamran Sartipi
{hu4, sartipi}@mcmaster.ca

Department of Computing and Software
McMaster University
Canada

SEKE’08
July 3, 2008

Outline

- Motivation
- Research Problem Definition and Solution
- Proposed Framework for Feature-Oriented Design Pattern Detection
  - Feature-oriented Dynamic Analysis
  - Two-phase Design Pattern Detection Process
- Case studies on Three Versions of JHotDraw Systems
- Contribution

Motivation

- Software product line is a group of software-intensive systems that share a common set of features to satisfy the specific needs from the market. Developed based on a reference architecture which consists of common parts and variable parts.
- An evolutionary development of a software product line starts from reverse engineering activities.
  - Understanding the existing systems
  - Locating common features to reuse
- Design pattern recovery can support the construction of software product line.
  - Understanding the existing system at design level
  - Reusing the existing system’s design artefacts

Research Definition

- Research Agenda:
  - To devise a methodology and supporting tools for recovering the instances of design patterns from the implementation of software system’s behavioural features, by the means of a high level pattern description method.
- Provided Solution:
  - We propose a reverse engineering framework which combines
    - feature-oriented dynamic analysis with
    - two-phase design pattern detection technique
  to identify the instances of design patterns for different software behavioural features.

Foundation of Reverse Engineering

- Reverse Engineering
  - A process of analyzing a software system to identify a system’s components and their interrelationships, and create representation of the system at a higher level of abstraction. [Chikofsky&Cross]
- Two major sub-areas in Reverse Engineering
  - Static Analysis
    - Clustering
    - Visualization
    - Pattern Matching (Design Pattern Recovery)
  - Dynamic Analysis
    - Feature identification
    - Behavioural design model extraction

Proposed Framework for Feature-oriented Design Pattern Detection
Feature-Oriented Dynamic Analysis

--- Execution Traces Generation

Using Eclipse Test and Performance Tools Platform (TPTP) to collect the execution traces generated by running the scenarios in the feature-specific scenario set.

- Reducing execution trace size using filter set mechanism

Feature-Specific Scenario Set
- Start, Draw a Ellipse
- Start, Draw a Rectangle
- Start, Draw an Ellipse
- Start, Draw a Polygon

TPTP On Eclipse

Execution traces

Apply Sequential Pattern Mining to generate Execution Patterns

Feature-Oriented Dynamic Analysis

--- Execution Pattern Extraction

Execution Traces for 3 Feature-specific Scenario Sets

<table>
<thead>
<tr>
<th>Feature 1</th>
<th>Feature 2</th>
<th>Feature 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1, C2, C3, C4, C5</td>
<td>C1, C2, C3, C4, C5</td>
<td>C1, C2, C3, C4, C5</td>
</tr>
<tr>
<td>C2, C5, C6, C8, C15</td>
<td>C2, C5, C6, C8, C15</td>
<td>C2, C5, C6, C8, C15</td>
</tr>
<tr>
<td>C1, C2, C3, C4, C5, C6, C8, C15</td>
<td>C1, C2, C3, C4, C5, C6, C8, C15</td>
<td>C1, C2, C3, C4, C5, C6, C8, C15</td>
</tr>
<tr>
<td>C4, C5, C6, C8, C15</td>
<td>C4, C5, C6, C8, C15</td>
<td>C4, C5, C6, C8, C15</td>
</tr>
<tr>
<td>C4, C5, C6, C8, C15, C18, C20</td>
<td>C4, C5, C6, C8, C15, C18, C20</td>
<td>C4, C5, C6, C8, C15, C18, C20</td>
</tr>
<tr>
<td>C1, C2, C3, C4, C5, C6, C8, C15</td>
<td>C1, C2, C3, C4, C5, C6, C8, C15</td>
<td>C1, C2, C3, C4, C5, C6, C8, C15</td>
</tr>
<tr>
<td>C3, C8, C15</td>
<td>C3, C8, C15</td>
<td>C3, C8, C15</td>
</tr>
<tr>
<td>C9, C10, C15</td>
<td>C9, C10, C15</td>
<td>C9, C10, C15</td>
</tr>
<tr>
<td>C17, C18, C20, C13, C14, C15</td>
<td>C17, C18, C20, C13, C14, C15</td>
<td>C17, C18, C20, C13, C14, C15</td>
</tr>
<tr>
<td>C20</td>
<td>C20</td>
<td>C20</td>
</tr>
</tbody>
</table>

Common pattern
Noise pattern
Feature-specific pattern

Describe Design Pattern using PDL

- Different types of the classes in PDL
  - Main-seed class, Depth1 class, Depth2 class and Seed-depth1 class

Example:

Static Analysis

Two-Phase Design Pattern Detection

Dynamic Analysis

Feature-Oriented Dynamic Analysis
Two-phase Design Pattern Detection Process

Subject System

Obtain inter-class relations by parsing the Subject System to get a list of source-class clusters.

Depth1Matching

Set of all combinations of matched source-classes of all depth1-classes

Depth2Matching

Set of all combinations of matched source-classes of all depth2-classes

Merge Depth1 combination and Depth2 combination

Identified design pattern instances

Design Pattern Repository

Approximate Matching

Merge Depth1 combination and Depth2 combination

Identified design pattern instances

Relation matrices

Depth Matching

Structural Matching

Two-phase Design Pattern Detection

---- Approximate Matching

Attribute Vector

- The attribute vector includes the following items:
  - Number of Inherit_From / Inherited_By relation
  - Number of in_Association / out_Association relation

- Similarity Function

Given the attribute vectors \( \text{Attr}_1 \) and \( \text{Attr}_2 \) of two classes \( c_1 \) and \( c_2 \), the approximate similarity function is defined as:

\[
\text{sim}_A(\text{Attr}_1, \text{Attr}_2) = \begin{cases} 
\Delta(\text{Attr}_1, \text{Attr}_2) & \text{Attr}_1 \geq \text{Attr}_2 \\
0 & \text{otherwise}
\end{cases}
\]

where \( \Delta(c_1, c_2) = \sum_{i=1}^{n} \frac{|c_1(i) - c_2(i)|}{|c_1(i)| + |c_2(i)|} \)

Result: a group of source-class clusters

Two-phase Design Pattern Detection

---- Structural Matching

Identifying all the instances of the target design pattern within a source-class cluster.

Depth1Matching

- Input: a source-class cluster, a candidate main-seed class, and a target design pattern.
- Output: set of all combinations of matched source-classes of all the depth1-classes.

Depth2Matching

- Input: a source-class cluster, a combination of matched source-classes of all the depth1-classes, and a target design pattern.
- Output: set of instances of the target design pattern.

An Example

Class diagram of Bridge Design Pattern

PDL representation of Bridge Design Pattern

Attribute Vector

\( \text{Attr}_c(\text{Implementor}) = [0, 2, 1, 0, 1] \)

An Example…

Search Space

Through applying approximate matching on the search space, we obtain two candidates of main-seed class \( c_2 \) and \( c_3 \).

\( \text{Attr}_c(c_2) = [1, 2, 1, 1, 1] \)

\( \text{Attr}_c(c_3) = [1, 2, 1, 1, 1] \)

\( \text{Attr}_c(\text{Implementor}) = [0, 2, 1, 0, 1] \)
Experiments with JHotDraw System

Statistics of three versions of JHotDraw systems

<table>
<thead>
<tr>
<th>System</th>
<th>Version</th>
<th># Classes</th>
<th># Files</th>
<th># LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw</td>
<td>5.1</td>
<td>112</td>
<td>445</td>
<td>1041</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>6.0b1</td>
<td>416</td>
<td>289</td>
<td>1791</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>7.0.7</td>
<td>351</td>
<td>309</td>
<td>1212</td>
</tr>
</tbody>
</table>

The experimental results of execution pattern extraction

<table>
<thead>
<tr>
<th>Specific Feature</th>
<th>Number of Executions</th>
<th>Average Trace Size</th>
<th>Average Precedence Size</th>
<th>Number of Extracted Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangle</td>
<td>1.1.1</td>
<td>390</td>
<td>678</td>
<td>111</td>
</tr>
<tr>
<td>Ellipse</td>
<td>2.1.1</td>
<td>394</td>
<td>988</td>
<td>131</td>
</tr>
<tr>
<td>Polygon</td>
<td>3.1.1</td>
<td>564</td>
<td>1098</td>
<td>172</td>
</tr>
<tr>
<td>Line</td>
<td>4.1.1</td>
<td>1091</td>
<td>2084</td>
<td>192</td>
</tr>
<tr>
<td>Area</td>
<td>5.1.1</td>
<td>2006</td>
<td>3988</td>
<td>195</td>
</tr>
<tr>
<td>Group</td>
<td>6.1.1</td>
<td>1003</td>
<td>1892</td>
<td>162</td>
</tr>
<tr>
<td>Join</td>
<td>7.1.1</td>
<td>605</td>
<td>1198</td>
<td>102</td>
</tr>
</tbody>
</table>

Legend: A / B / C A: data for JHotDraw 5.1 B: data for JHotDraw 6.0b1 C: data for JHotDraw 7.0.7

Legend:
- A: data for JHotDraw 5.1
- B: data for JHotDraw 6.0b1
- C: data for JHotDraw 7.0.7

Experiments with JHotDraw System

Results of feature-specific classes assignment for 10 features of JHotDraw 5.1

<table>
<thead>
<tr>
<th>Specific Feature</th>
<th>Feature-specific Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw a Rectangle</td>
<td>Rectangle, Ellipse, Ellipse</td>
</tr>
<tr>
<td>Draw an Ellipse</td>
<td>Ellipse, Ellipse</td>
</tr>
<tr>
<td>Draw a Polygon</td>
<td>Polygon, Polygon, Polygon, Polygon</td>
</tr>
<tr>
<td>Draw a Tool</td>
<td>Tool, Tool, Tool, Tool</td>
</tr>
<tr>
<td>Group Figures</td>
<td>Group, Group, Group, Group</td>
</tr>
<tr>
<td>Move a Figure</td>
<td>Figure, Figure, Figure, Figure</td>
</tr>
<tr>
<td>Delete a Figure</td>
<td>Figure, Figure, Figure, Figure</td>
</tr>
<tr>
<td>Draw a Line</td>
<td>Line, Line, Line, Line</td>
</tr>
<tr>
<td>Draw a LineConnection</td>
<td>LineConnection, LineConnection, LineConnection, LineConnection</td>
</tr>
</tbody>
</table>

Experiments with JHotDraw System

Concept lattice representation of features and classes in JHotDraw 5.1

Experiments with JHotDraw System

Results of identified Adapter design pattern instances and related features in JHotDraw 5.1 system

<table>
<thead>
<tr>
<th>Design Pattern Instance</th>
<th>Related Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE/DA/Draw/Line</td>
<td>Draw a Line</td>
</tr>
<tr>
<td>CE/DA/Draw/Transfer</td>
<td>Draw a Transfer</td>
</tr>
<tr>
<td>CE/DA/Draw/Command</td>
<td>Draw a Command</td>
</tr>
<tr>
<td>CE/DA/Draw/Connection</td>
<td>Draw a Connection</td>
</tr>
</tbody>
</table>

Summary

We presented:
- A methodology to identify individual design pattern instances from the implementation of system behavioural features.
- A new design pattern representation, PDL (Pattern Description Language), which enables users to describe the structural information of design patterns efficiently and conveniently.
- A two-phase design pattern matching process (approximate matching & structure matching) to reduce the complexity of the matching process.
- A prototype toolkit for the proposed approach on the Eclipse open platform.

Future Work

Our future work will mainly concentrate on the following directions:
- Extending the pattern repository to support more design patterns identification.
- Extracting more inter-class relations, such as delegation and method invocations, to improve the accuracy of the technique.
- Validating our approach on large-scale software systems.
- Tracking the evolution of software systems at design level by analyzing the evolution of design patterns.
Dynamic Analysis and Design Pattern Detection in Java Programs

Lei Hu
Kamran Sartipi
{hu4, sartipi}@mcmaster.ca

Department of Computing and Software
McMaster University
Canada

SEKE'08
July 3, 2008