17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

FEATURE-ORIENTED PROGRAMMING

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH
PROJECT UPDATE
READING
ASSIGNMENT NOV 4


remainder of Chapter 6

LEARNING GOALS

Understand key concepts of feature-oriented programming
Implement variations with feature-oriented programming
Understand modularity implications of feature-oriented programming and mechanisms to mitigate issues
TOOL-BASED TRACEABILITY

Feature Model

Tool manages mapping

Implementation
LANGUAGE-BASED TRACEABILITY

Feature Model

1:1 / 1:n mapping

Implementation
FEATURE COMPOSITION
FEATURE COMPOSITION
FEATURE COMPOSITION
# PARTIAL CLASSES

<table>
<thead>
<tr>
<th>Features</th>
<th>Graph</th>
<th>Edge</th>
<th>Node</th>
<th>Weight</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Directed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weighted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colored</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
COLLABORATIONS AND ROLES

Search
Directed
Weighted
Colored

Graph
Edge
Node
Weight
Color

Role
class Graph {
    Vector nv = new Vector();
    Vector ev = new Vector();
    Edge add(Node n, Node m) {
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m);
        ev.add(e); return e;
    }
    void print() {
        for(int i = 0; i < ev.size(); i++)
            ((Edge)ev.get(i)).print();
    }
}

class Node {
    int id = 0;
    void print() {
        System.out.print(id);
    }
}

class Edge {
    Node a, b;
    Edge(Node _a, Node _b) {
        a = _a; b = _b;
    }
    void print() {
        a.print(); b.print();
    }
}

refines class Graph {
    Edge add(Node n, Node m) {
        Edge e = Super.add(n, m);
        e.weight = new Weight();
    }
    Edge add(Node n, Node m, Weight w) {
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        e.weight = w; return e;
    }
}

refines class Edge {
    Weight weight = new Weight();
    void print() {
        Super.print(); weight.print();
    }
}

class Weight {
    void print() {
        ...
    }
}
DIRECTORY STRUCTURE
CLASS REFINEMENT

Stepwise Refinement

“Imprecise” Definition of Super Class
METHOD REFINEMENT (AHEAD)

class Edge {
    void print() {
        System.out.print(" Edge between " + node1 + " and " + node2);
    }
}

refines class Edge {
    private Node start;
    void print() {
        Super().print();
        System.out.print(" directed from " + start);
    }
}

refines class Edge {
    private int weight;
    void print() {
        Super().print();
        System.out.print(" weighted with " + weight);
    }
}
class Edge {
    void print() {
        System.out.print("Edge between " + node1 + " and " + node2);
    }
}

class Edge {
    private Node start;
    void print() {
        original();
        System.out.print("directed from " + start);
    }
}

class Edge {
    private int weight;
    void print() {
        original();
        System.out.print("weighted with " + weigth);
    }
}
AHEAD

Feature modules (directories)

Composer

jampack

mixin

jak2java

Composed .jak files

.java files
C = B

A

Code

R.drc

Htm

X.jak

Y.jak

W.htm

Z.htm

X.jak

= X.jak • X.jak

Code

R.drc

Htm

X.jak

Y.jak

Z.htm

Code

R.drc

Htm

X.jak

Y.jak

W.htm
<table>
<thead>
<tr>
<th>Base</th>
<th>Graph</th>
<th>Edge</th>
<th>Node</th>
<th>Weight</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Directed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weighted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colored</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TOOLS

AHEAD Tool Suite + Documentation
- Command line tools for Jak (Java 1.4 extension)
- http://www.cs.utexas.edu/users/schwartz/ATS.html

FeatureHouse
- Command-line tools for Java, C#, C, Haskell, UML, …
- http://www.fosd.de/fh

FeatureC++
- Command-line tool for C++
- http://www.fosd.de/fcpp

FeatureIDE
- Eclipse plugin for AHEAD, FeatureHouse and FeatureC++
- Automates compilation; syntax highlighting; etc
- http://www.fosd.de/featureide
public refines class Edge {

    private int weight;

    public int getWeight() {
        return this.weight;
    }

    public void display() {
        System.out.print( "Weight=" + weight );
        super.display();
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

}
UNIFORMITY

Features are implemented by a diverse selection of software artifacts and any kind of software artifact can be subject of subsequent refinement.

– Don Batory
<table>
<thead>
<tr>
<th>Base</th>
<th>Edge</th>
<th>Node</th>
<th>Buildfile</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>Edge</td>
<td>Node</td>
<td>Buildfile</td>
<td>Doc</td>
</tr>
<tr>
<td>Base</td>
<td>Base</td>
<td>Base</td>
<td>Base</td>
<td>Base</td>
</tr>
<tr>
<td>Search</td>
<td>Search</td>
<td>Search</td>
<td>Search</td>
<td>Search</td>
</tr>
<tr>
<td>Directed</td>
<td>Directed</td>
<td>Directed</td>
<td>Directed</td>
<td>Directed</td>
</tr>
<tr>
<td>Weighted</td>
<td>Weighted</td>
<td>Weighted</td>
<td>Weighted</td>
<td>Weighted</td>
</tr>
<tr>
<td>Colored</td>
<td>Colored</td>
<td>Colored</td>
<td>Colored</td>
<td>Colored</td>
</tr>
</tbody>
</table>

Graph: Grap h.jak
Edge: Edge.jak
Node: Node.jak
Buildfile: build.xml
Doc: index.htm
\[ \bullet : F \times F \rightarrow F \]

\[ p = f_n \bullet f_{n-1} \bullet \ldots \bullet f_2 \bullet f_1 \]

(associative, not commutative)
package util;
class Calc {
    int e0 = 0, e1 = 0, e2 = 0;
    void enter(int val) {
        e2 = e1; e1 = e0; e0 = val;
    }
    void clear() {
        e0 = e1 = e2 = 0;
    }
    String top() {
        return String.valueOf(e0);
    }
}
package util;
class Calc {
    void add() {
        e0 = e1 + e0;
        e1 = e2;
    }
}

package util;
class Calc {
    int e0 = 0, e1 = 0,
    e2 = 0;
    void enter(int val) {
        e2 = e1; e1 = e0;
        e0 = val;
    }
    void clear() {
        e0 = e1 = e2 = 0;
    }
    String top() {
        return String.valueOf(e0);
    }
}

package util;
class Calc {
    int e0 = 0, e1 = 0,
    e2 = 0;
    void enter(int val) {
        e2 = e1; e1 = e0;
        e0 = val;
    }
    void clear() {
        e0 = e1 = e2 = 0;
    }
    String top() {
        //...
    }
    void add() {
        e0 = e1 + e0;
        e1 = e2;
    }
}
COMPOSITION OF TERMINAL NODES

class Calc {
    int count = 0;
    void enter(int val) {
        original(val);
        count++;
    }
}

class Calc {
    int count;
    void enter(int val) {  
        e2 = e1;
        e1 = e0;
        e0 = val;
    }
}

class Calc {
    int count = 0;
    void enter(int val) {
        e2 = e1;
        e1 = e0;
        e0 = val;
        count++;
    }
}
RESTRICTIONS

Hierarchical structure
Nodes need name and type
Child elements require unique name/type combinations
Composition rules required for terminal nodes

„feature-ready languages“

?
FEATUREHOUSE

Java

C#

C

Haskell

FSTs

Pretty Printer

Parser

Pretty Printer

Pretty Printer

Parser

Parser

Parser

Parser
DISCUSSION

Modularity, Cohesion
Traceability
FURTHER READINGS

