Structural Feature Interaction Patterns - Case Studies and Guidelines
VaMoS 2014, January 22–24, Nice, France
Sven Schuster, Sandro Schulze, Ina Schaefer, January 23, 2014
Feature Interactions

Behavioral Feature Interactions

- Fire & Flood Control
Feature Interactions

Structural Feature Interactions

- Database Locks & Statistics

- Locks (blue), Statistics (red)
- Overlapping (violet)
  - Locking the calculation of Statistics
  - Statistics of Locking
- Optional Feature Problem

```java
class Database {
    List locks;
    void lock() { /*...*/ }
    void unlock() { /*...*/ }

    void put(Object key, Object data) {
        lock();
        /*...*/
        unlock();
    }

    Object get(Object key) {
        lock();
        /*...*/
        unlock();
    }

    int getOpenLocks() {
        return locks.size();
    }

    int getDbSize() {
        return calculateDbSize();
    }

    static int calculateDbSize() {
        lock();
        /*...*/
        unlock();
    }
}
```
Problem Statement

Categorization by Apel et al.

- Which FIs exist?
- How do they manifest?
Problem Statement

Categorization by Apel et al.

- Which FIs exist?
- How do they manifest?

What about...

- Why do they exist?
- Are they intentional?

Focussing on...

- Internal Interactions
  - Structural Interactions
Problem Statement

Categorization by Apel et al.
- Which FIs exist?
- How do they manifest?

What about...
- Why do they exist?
- Are they intentional?

⇒ Structural Feature Interaction Patterns

Focussing on...
- Internal Interactions
  - Structural Interactions

⇒ Structural Feature Interaction Patterns
SPLs & FOP

Legend:
- Mandatory
- Optional
- Or
- Abstract
- Concrete

Superimposition

Feature Base

```java
class Stack {
    int backupPush;
    void undo() {
        original(v);
    }
    void push(int v) {
        backupPush = v;
    }
    int pop() {
        /* ... */
    }
}
```

Feature Undo

```java
class Stack {
    int backupPush;
    void undo() {
        /* ... */
    }
    void push(int v) {
        backupPush = v;
        original(v);
    }
    int pop() {
        /* ... */
    }
}
```

Motivation Background Reasoning Case Study Conclusion

January 23, 2014 | Sven Schuster, Sandro Schulze, Ina Schaefer | Page 5
Structural Feature Interaction Patterns - Case Studies and Guidelines
**SPLs & FOP**

Legend:
- Mandatory
- Optional
- Or
- Abstract
- Concrete

```
class Stack {
  int backupPush;
  int pop() { /* ... */ }
  void undo() { /* ... */ }
  void push(int v) { /* ... */ }
}
```

```
Superimposition

class Stack {
  int backupPush;
  int pop() { /* ... */ }
  void undo() { /* ... */ }
  void push(int v) { /* ... */ }
}
```

```
Feature Base

class Stack {
  *...*/
  int backupPush;
  void undo() { /* ... */ }
  void push(int v) {
    backupPush = v;
    original(v);
  }
}
```

```
Feature Undo

class Stack {
  /* ... */
  int backupPush;
  int pop() { /* ... */ }
  void undo() { /* ... */ }
  void push(int v) {
    backupPush = v;
    original(v);
  }
}
```

⇒ **FeatureHouse / Fuji**
Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Foreword by Grady Booch
Design Patterns

- Client
  - visitors
  - elements
- Visitor
  - + visit(ConcreteElement)
- Element
  - + accept(Visitor)
- ConcreteVisitor
  - + visit(ConcreteElement)
- ConcreteElement
  - + accept(Visitor)
Feature-Oriented Design Patterns

⇒ One-to-one mapping between Features & Strategies

1Sven Schuster, Sandro Schulze, Object-Oriented Design in Feature-Oriented Programming, FOSD’12
Feature Template Method

Feature DirectedOnlyVertices

```java
public class Graph {
    public void run(Vertex s) {
        /*empty */
    }
}
```

Feature TestProg

```java
class Main {
    public static void main(String[] args) {
        Graph g = new Graph();
        /* ... */
        g.run(g.getVertices().next());
    }
}
```

Feature Number

```java
public class Graph {
    public void run(Vertex s) {
        System.out.println("Number");
        NumberVertices();
        original(s);
    }
}
```

⇒ Implicit constraint to fill/refine template method in later features²

²Sven Schuster, Sandro Schulze, *Object-Oriented Design in Feature-Oriented Programming*, FOSD’12
Feature Template Method

Feature DirectedOnlyVertices

```java
public class Graph {
    public void run(Vertex s) {
        /*empty */
    }
}
```

Feature TestProg

```java
class Main {
    public static void main(String[] args) {
        Graph g = new Graph();
        g.run(g.getVertices().next());
    }
}
```

Feature Number

```java
public class Graph {
    public void run(Vertex s) {
        System.out.println("Number");
        NumberVertices();
        original(s);
    }

    public void NumberVertices() {
        /* ... */
    }
}
```

⇒ Implicit constraint to fill/refine template method in later features

---

2 Sven Schuster, Sandro Schulze, *Object-Oriented Design in Feature-Oriented Programming*, FOSD’12
Feature Template Method

Feature \textit{DirectedOnlyVertices}

```java
public class Graph {
    public void run(Vertex s) {
        /*empty */
    }
}
```

Feature \textit{TestProg}

```java
class Main {
    public static void main( String[] args ) {
        Graph g = new Graph();
        /* ... */
        g.run(g.getVertices().next());
    }
}
```

Feature \textit{Number}

```java
public class Graph {
    public void run(Vertex s) {
        System.out.println("Number");
        NumberVertices();
        original(s);
    }
}
```

⇒ Implicit constraint to fill/refine template method in later features\textsuperscript{2}

\textsuperscript{2}Sven Schuster, Sandro Schulze, \textit{Object-Oriented Design in Feature-Oriented Programming}, FOSD’12
Research Questions

⇒ **RQ1:** To what extent do design patterns exist in feature-oriented SPLs?

⇒ **RQ2:** Do feature interactions occur in the context of design patterns and if so, how do the features interact?
Setup & Methodology

Study

- Automated pattern detection technique
Setup & Methodology

Study
- Automated pattern detection technique
- SPLs of different sizes and domains

SPLs
- Ahead
- BerkeleyDB
- GameOfLife
- GPL
- GUIDSL
- TankWar
- Violet
Setup & Methodology

Study
- Automated pattern detection technique
- SPLs of different sizes and domains
- Common design patterns

SPLs
- Ahead
- BerkeleyDB
- GameOfLife
- GPL
- GUIDSL
- TankWar
- Violet

Patterns
- Visitor
- Observer
- Strategy
- FTM
Design Pattern Detection for FOP

Automated detection approach

- Adapted from Heuzeroth et al.
- Family-based $\Rightarrow$ static analysis
Design Pattern Detection for FOP

Automated detection approach
- Adapted from Heuzeroth et al.
- Family-based $\Rightarrow$ static analysis
- Searching for tuples of elements using Fuji AST

Visitor Pattern

VisitorCandidate = {
    Visitor.visit(ConcreteElement),
    ConcreteVisitor.visit(ConcreteElement),
    Element.accept(Visitor),
    ConcreteElement.accept(Visitor)
}

Design Pattern Detection for FOP

Automated detection approach
- Adapted from Heuzeroth et al.
- Family-based \(\Rightarrow\) static analysis
- Searching for tuples of elements using Fuji AST
- Manual elimination of false positives

Visitor Pattern

```
VisitorCandidate = {
    Visitor.visit(ConcreteElement),
    ConcreteVisitor.visit(ConcreteElement),
    Element.accept(Visitor),
    ConcreteElement.accept(Visitor)
}
```
Design Pattern Detection - Visitor in GPL

Feature **DFS**

```java
public class WorkSpace {
    public void preVisitAction(Vertex v) {} 
}

public class Vertex {
    public void nodeSearch(WorkSpace w) {
        /*...*/
        w.preVisitAction(this);
        /*...*/
    }
}
```

Feature **Number**

```java
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;

    public NumberWorkSpace() {
        vertexCounter = 0;
    }

    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;
        }
    }
}
```
**Design Pattern Detection - Visitor in GPL**

Visitor.visit(ConcreteElement)

**Feature DFS**

```java
1 public class WorkSpace {
2   public void preVisitAction(Vertex v) {}   
3 }
4
5 public class Vertex {
6   public void nodeSearch(WorkSpace w) {
7       /*...*/
8       w.preVisitAction(this);
9       /*...*/
10   }                                               
11 }
```

**Feature Number**

```java
1 public class NumberWorkSpace extends WorkSpace {
2   int vertexCounter;
3
4   public NumberWorkSpace() {
5       vertexCounter = 0;
6   }
7
8   public void preVisitAction(Vertex v) {
9       if (v.visited != true) {
10          v.VertexNumber = vertexCounter++;
11       }
12   }
13 }
```
Design Pattern Detection - Visitor in GPL

Element.accept(Visitor)

Feature **DFS**

```
public class WorkSpace {
    public void preVisitAction(Vertex v) {}  
}

public class Vertex {
    public void nodeSearch(WorkSpace w) {
       /*...*/
        w.preVisitAction(this);
       /*...*/
    }
}
```

Feature **Number**

```
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;
    public NumberWorkSpace() {
        vertexCounter = 0;
    }
    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;  
        }
    }
}
```
ConcreteVisitor.visit(ConcreteElement)

**Feature DFS**

```java
public class WorkSpace {
    public void preVisitAction(Vertex v) {} 
}

public class Vertex {
    public void nodeSearch(WorkSpace w) {
        /*...*/
        w.preVisitAction(this);
        /*...*/
    }
}
```

**Feature Number**

```java
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;
    public NumberWorkSpace() {
        vertexCounter = 0;
    }
    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;
        }
    }
}
```
ConcreteElement.accept(Visitor)

Feature DFS

```java
public class WorkSpace {
    public void preVisitAction(Vertex v) {} 
}

public class Vertex {
    public void nodeSearch(WorkSpace w) {
        w.preVisitAction(this);
    }
}
```

Feature Number

```java
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;
    public NumberWorkSpace() {
        vertexCounter = 0;
    }
    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;
        }
    }
}
```
Results - Complete Number of Patterns

- Ahead
- BerkeleyDB
- GameOfLife
- GPL
- GUIDSL
- TankWar
- Violet

Number of patterns
Results - Number of decomposed Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Visitor</th>
<th>Strategy</th>
<th>Observer</th>
<th>FTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahead</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BerkeleyDB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GameOfLife</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUIDSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TankWar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Violet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of patterns

Visitor, Strategy, Observer, FTM
Results - Decomposition of Visitor

Feature Number

```java
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;
    public NumberWorkSpace() {
        vertexCounter = 0;
    }
    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;
        }
    }
}
```

Feature Cycle

```java
public class CycleWorkSpace extends WorkSpace {
    public void preVisitAction(Vertex v) {
        /* ... */
    }
}
```

Feature Connected

```java
public class ConnectedWorkSpace extends WorkSpace {
    public void preVisitAction(Vertex v) {
        /* ... */
    }
}
```
Results - Feature Template Method

Feature DFS

```java
public class WorkSpace {
    public void preVisitAction(Vertex v) {}
}

public class Vertex {
    public void nodeSearch(WorkSpace w) {
        /*...*/
    }
}

public class Graph {
    public void GraphSearch(WorkSpace w) {
        /*...*/
        v.nodeSearch(w);
        /*...*/
    }
}
```

Feature Number

```java
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;

    public NumberWorkSpace() {
        vertexCounter = 0;
    }

    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;
        }
    }
}

public class Graph {
    public void run(Vertex s) {
        /*empty */
    }

    public void NumberVertices() {
        GraphSearch(new NumberWorkSpace());
    }
}
```

Feature DirectedOnlyVertices

```java
public class Graph {
    public void run(Vertex s) {
        /*empty */
    }
}
```
Results - Feature Template Method

Feature DFS

```java
public class WorkSpace {
    public void preVisitAction(Vertex v) {} }

public class Vertex {
    public void nodeSearch(WorkSpace w) {
        /*...*/
    }
}

public class Graph {
    public void GraphSearch(WorkSpace w) {
        /*...*/
        v.nodeSearch(w);
        /*...*/
    }
}
```

Feature Number

```java
public class NumberWorkSpace extends WorkSpace {
    int vertexCounter;

    public NumberWorkSpace() {
        vertexCounter = 0;
    }

    public void preVisitAction(Vertex v) {
        if (v.visited != true) {
            v.VertexNumber = vertexCounter++;
        }
    }
}

public class Graph {
    public void run(Vertex s) {
        System.out.println("Number");
        NumberVertices();
        original(s);
    }

    public void NumberVertices() {
        GraphSearch(new NumberWorkSpace());
    }
}
```

Feature DirectedOnlyVertices

```java
public class Graph {
    public void run(Vertex s) {
        /*empty */
    }
}
```
Design Patterns in SPLs

- Design Patterns do occur in feature-oriented SPLs...
  - frequently.
  - across several features.

Feature Interactions within Design Patterns

- Addition of concrete pattern classes in sibling features
- No refinements of the actual pattern implementation
- Feature Template Method to register/call new objects
Towards Guidelines

Application Scenarios

- Similar to design patterns in OOP
  - Visitor to traverse through grammar
  - Observer to realize MVC
  - Strategy to interchange similar algorithms

Feature Structure

- Concrete classes of decomposed patterns introduced by sibling features
- Design patterns are encapsulated by feature group
  ⇒ Reflection of design pattern in feature model
  ⇒ Synergy of modularity of variability model and Implementation
Conclusion & Future Work

We have...

... introduced the idea of structural feature interaction patterns
... proposed a detection technique for design patterns in FOP
... conducted a case study detecting structural feature interaction within design patterns
... introduced ideas for guidelines
Conclusion & Future Work

We have...

... introduced the idea of structural feature interaction patterns
... proposed a detection technique for design patterns in FOP
... conducted a case study detecting structural feature interaction within design patterns
... introduced ideas for guidelines

We should...

... improve and extend the detection technique
... conduct more studies to derive more detailed guidelines
... reason about how to apply such guidelines (automatically)?