Rejuvenate Pointcut[1]

A Tool for Pointcut Expression Recovery in Evolving Aspect-Oriented Software

Raffi Khatchadourian, Awais Rashid
khatchad@cse.ohio-state.edu,awais@comp.lancs.ac.uk

Ohio State University, Lancaster University

September 29, 2008

[1]This material is based upon work supported in part by European Commission grants IST-33710 (AMPLE) and IST-2-004349 (AOSD-Europe).

Outline

Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example PCEs

- `execution(* Foo.*(..))`
- `execution(* Foo.methodA())`

Example PCEs

- `execution(* Foo.*(..))`
- `execution(* Foo.methodA())`

Fragile Pointcut Problem

- Constructing optimal pointcut expressions that capture true intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions (PCEs) should remain valid as the software evolves.
- However, situations arise where base-code alterations induce the invalidity of existing PCEs.
- PCEs must be *amended* as a result.
- But which join points should be included?
- Can we automatically *infer* join points that should be included in the new version of the PCE upon evolution?

Fragile Pointcut Problem

- Constructing optimal pointcut expressions that capture true intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions (PCEs) should remain valid as the software evolves.
- However, situations arise where base-code alterations induce the invalidity of existing PCEs.
- PCEs must be *amended* as a result.
- But which join points should be included?
- Can we automatically *infer* join points that should be included in the new version of the PCE upon evolution?

Fragile Pointcut Problem

- Constructing optimal pointcut expressions that capture true intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions (PCEs) should remain valid as the software evolves.
- However, situations arise where base-code alterations induce the invalidity of existing PCEs.
- PCEs must be *amended* as a result.
- But which join points should be included?
- Can we automatically *infer* join points that should be included in the new version of the PCE upon evolution?

Fragile Pointcut Problem

- Constructing optimal pointcut expressions that capture true intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions (PCEs) should remain valid as the software evolves.
- However, situations arise where base-code alterations induce the invalidity of existing PCEs.
- PCEs must be *amended* as a result.
- But which join points should be included?
- Can we automatically *infer* join points that should be included in the new version of the PCE upon evolution?

Fragile Pointcut Problem

- Constructing optimal pointcut expressions that capture true intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions (PCEs) should remain valid as the software evolves.
- However, situations arise where base-code alterations induce the invalidity of existing PCEs.
- PCEs must be *amended* as a result.
- But which join points should be included?
- Can we automatically *infer* join points that should be included in the new version of the PCE upon evolution?

Fragile Pointcut Problem

- Constructing optimal pointcut expressions that capture true intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions (PCEs) should remain valid as the software evolves.
- However, situations arise where base-code alterations induce the invalidity of existing PCEs.
- PCEs must be *amended* as a result.
- But which join points should be included?
- Can we automatically *infer* join points that should be included in the new version of the PCE upon evolution?

The Idea

- Capture rich program element relationships in a *concern graph* adapted for AOP.
- Associate graph elements with join point shadows contained in the PCE.
- Extract *patterns* from finite, acyclic paths in the graph.
- Place *wildcards* in place of the associated elements.
- Apply the patterns to the evolved version of the base-code.

The Idea

- Capture rich program element relationships in a *concern graph* adapted for AOP.
- Associate graph elements with join point shadows contained in the PCE.
- Extract *patterns* from finite, acyclic paths in the graph.
- Place *wildcards* in place of the associated elements.
- Apply the patterns to the evolved version of the base-code.

The Idea

- Capture rich program element relationships in a *concern graph* adapted for AOP.
- Associate graph elements with join point shadows contained in the PCE.
- Extract *patterns* from finite, acyclic paths in the graph.
- Place *wildcards* in place of the associated elements.
- Apply the patterns to the evolved version of the base-code.

The Idea

- Capture rich program element relationships in a *concern graph* adapted for AOP.
- Associate graph elements with join point shadows contained in the PCE.
- Extract *patterns* from finite, acyclic paths in the graph.
- Place *wildcards* in place of the associated elements.
- Apply the patterns to the evolved version of the base-code.

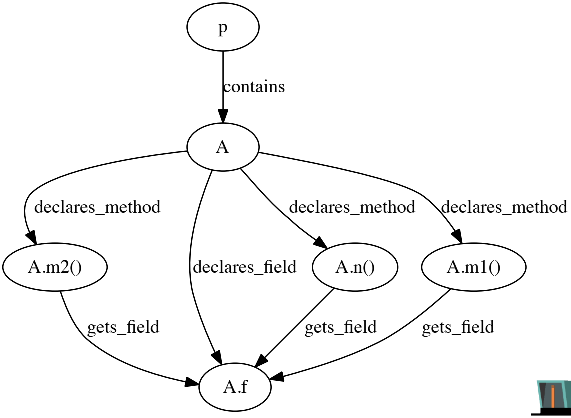
The Idea

- Capture rich program element relationships in a *concern graph* adapted for AOP.
- Associate graph elements with join point shadows contained in the PCE.
- Extract *patterns* from finite, acyclic paths in the graph.
- Place *wildcards* in place of the associated elements.
- Apply the patterns to the evolved version of the base-code.

Base-code

```
package p;
public class A {
    int f;
    void m1() {
        int a = f + 1;
    }
    void m2() {
        int b = f + 2;
    }
    void n() {
        int c = f + 3;
    }
}
```

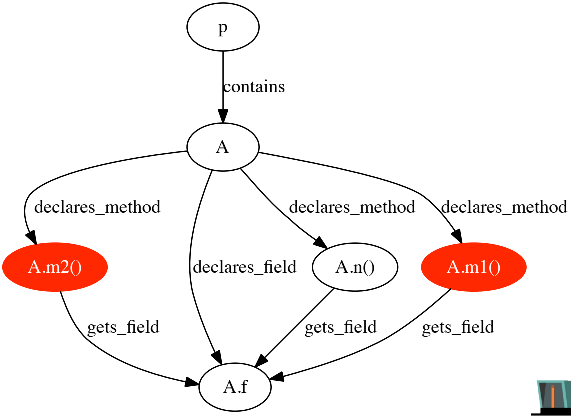
Concern Graph



An Aspect

```
package p;
public aspect B {
    before() : execution(* m*(..)) {
    }
}
```

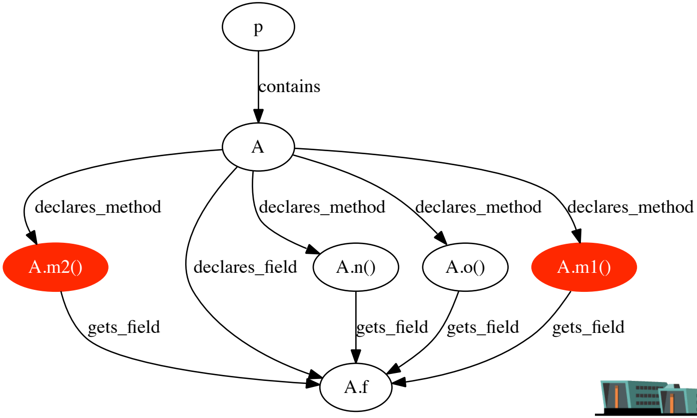
Concern Graph



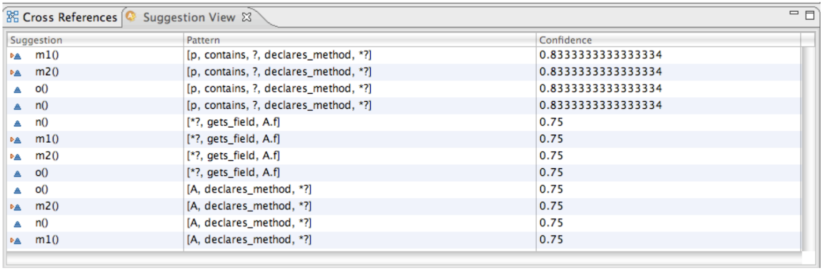
Base-code

```
//...
void o() {
    int d = f + 4;
}
//...
```

Concern Graph



Patterns and Suggestions



Suggestion	Pattern	Confidence
m1()	[p, contains, ?, declares_method, *?]	0.8333333333333334
m2()	[p, contains, ?, declares_method, *?]	0.8333333333333334
o()	[p, contains, ?, declares_method, *?]	0.8333333333333334
n()	[p, contains, ?, declares_method, *?]	0.8333333333333334
n()	[*?, gets_field, A.f]	0.75
m1()	[*?, gets_field, A.f]	0.75
m2()	[*?, gets_field, A.f]	0.75
o()	[*?, gets_field, A.f]	0.75
o()	[A, declares_method, *?]	0.75
m2()	[A, declares_method, *?]	0.75
n()	[A, declares_method, *?]	0.75
m1()	[A, declares_method, *?]	0.75

Tool and Material Downloads

- Tool research prototype publicly available at `http://code.google.com/p/rejuvenate-pc`.
- Research related material publicly available at `http://sites.google.com/site/pointcutrejuvenation`.

InfoLab21

Question for SCAM

Do more expressive PCE languages truly solve the fragile pointcut problem?