Aspect Mining Using Event Traces

Silvia Breu

Lehrstuhl für Softwaresysteme

Universität Passau

03.05.2004

Workshop Software-Reengineering, Bad Honnef, May 2004

What is *Aspect Mining*?

- new research area
- identification of crosscutting concerns in legacy systems
- “isolation” of crosscutting concerns
- helpful for program understanding
- useful for refactoring

Existing Aspect Mining Approaches

Based on static program analysis techniques:

- Aspect Browser (Griswold et al.)
- AMT (Hannemann, Kiczales)
- AMTex (Zhang, Jacobsen)
- JQuery (Janzen, Volder)
- FEAT (Robillard, Murphy)
- Ophir (Shepherd, Pollock)

Basic Idea of Developed Aspect Mining Approach

- dynamic analysis technique
- based on investigation of program traces
- search for recurring execution relations (called ***aspect candidates***)
- aspect candidates indicate potential crosscutting concerns

Execution Relation

In a program trace we distinguish

- Outside-Execution Relations
  - Outside-Before-Execution Relations $\mathtt{u} \rightharpoonup \mathtt{v}$:

    method execution $\mathtt{u}$ before method execution $\mathtt{v}$
  - Outside-After-Execution Relations $\mathtt{u} \leftharpoonup \mathtt{v}$:

    method execution $\mathtt{u}$ after method execution $\mathtt{v}$
- Inside-Execution Relations
  - Inside-First-Execution Relations $\mathtt{u} \in_{\top} \mathtt{v}$:

    method execution $\mathtt{u}$ first inside method execution $\mathtt{v}$
  - Inside-Last-Execution Relations $\mathtt{u} \in_{\perp} \mathtt{v}$:

    method execution $\mathtt{u}$ last inside method execution $\mathtt{v}$

Execution Relation Constraints

Characterisation of recurring execution relations in program traces with three constraints:

Uniformity: always the same composition, e.g.

$a \rightharpoonup b, a \rightharpoonup b, a \rightharpoonup b$ ✔ $a \rightharpoonup b, c \rightharpoonup b, a \rightharpoonup b$ ✗

Non-Triviality: more than once

Crosscutting: more than one calling context, e.g.

$a \rightharpoonup b, a \rightharpoonup c, a \rightharpoonup b$ ✔ $a \rightharpoonup b, a \rightharpoonup b, a \rightharpoonup b$ ✗

DynAMiT - *Dyn*amic *A*spect *Mi*ning *T*ool

- aspect mining prototype
- application of constraints in two algorithms:
  - basic analysis (uniformity & non-triviality)
  - crosscutting analysis (uniformity & crosscutting)

  $\xrightarrow{\text{results in}}$ aspect candidates
- used to conduct several case studies

Case Study “AspectJ Example telecom”

- Java application (simulation of phone calls)
- extended with aspects (timing, billing) written in AspectJ
- results:
  - detected basic functionality
  - found ***all*** crosscutting functionality added by timing/billing aspect
  - identified no false positives
  - resulting aspect candidates like a manual of what happens

Result Part: Case Study “AspectJ Example telecom”

Basic algorithm, outside-/inside-aspect candidates:

`void Call.hangup(Customer)` $\rightharpoonup$ `void Customer.removeCall(Call)`

`void Customer.addCall(Call)` $\leftharpoonup$ `void Call.pickup()`

`long Timer.getTime()` $\in_{\perp}$ `void Call.hangup(Customer)`

Crosscutting algorithm, outside-aspect candidates:

`Timer Timing.getTimer(Connection)` $\rightharpoonup$
    `void Timer.start(),void Timer.stop(),long Timer.getTime()`

`Customer Billing.getPayer(Connection)` $\leftharpoonup$
    `long Local.callRate(),long LongDistance.callRate()`

Case Study “Graffiti”

- industrial-sized graph editor with toolkit for graph visualisation algorithms
- $\approx$ 450 classes/interfaces, 3.000 methods, 82 kLoC
- results:
  - numerous aspect candidates
  - information about architecture (e.g. extendability with algorithms) and controlflow (setting of plugin author, name, description, dependencies etc.)
  - real crosscutting concerns (e.g. plugin structure, logging)

$<rel>$	$\lvert R^{<rel>} \rvert$	Cand.
$\mathtt{u} \rightharpoonup \mathtt{v}$	40	10
$\mathtt{u} \leftharpoonup \mathtt{v}$	40	8
$\mathtt{u} \in_{\top} \mathtt{v}$	33	10
$\mathtt{u} \in_{\bot} \mathtt{v}$	25	7

Summary

- first dynamic aspect mining approach (light-weight)
- based on program traces and abstraction into execution relations
- ***automatic*** analysis
- finds seeded and existing crosscutting concerns
- high precision and recall
- generally applicable

Thanks for your attention!

Any questions?