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 $u \rightarrow v$: method execution $u$ before method execution $v$
  - Outside-After-Execution Relations $u \leftarrow v$: method execution $u$ after method execution $v$

- **Inside-Execution Relations**
  - Inside-First-Execution Relations $u \in_T v$: method execution $u$ first inside method execution $v$
  - Inside-Last-Execution Relations $u \in_\bot v$: method execution $u$ last inside method execution $v$
Execution Relation Constraints

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

**Uniformity**: always the same composition, e.g.

\[
\begin{align*}
a \rightarrow b, a \rightarrow b, a \rightarrow b & \checkmark \\
\end{align*}
\]

**Non-Triviality**: more than once

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

\[
\begin{align*}
a \rightarrow b, a \rightarrow c, a \rightarrow b & \checkmark \\
\end{align*}
\]
DynAMiT - *Dynamic Aspect Mining Tool*

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

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) ↭ void Customer.removeCall(Call)

void Customer.addCall(Call) ↭ void Call.pickup()

long Timer.getTime() ∈⊥ void Call.hangup(Customer)

Crosscutting algorithm, outside-aspect candidates:

Timer Timing.getTimer(Connection) ↭
   void Timer.start(), void Timer.stop(), long Timer.getTime()

Customer Billing.getPayer(Connection) ↭
   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)

\begin{tabular}{l|l|l}
\text{\textless rel\textgreater} & \mid \text{R\textless rel\textgreater} \mid & \text{Cand.} \\
\hline
\text{u $\rightarrow$ v} & 40 & 10 \\
\text{u $\leftarrow$ v} & 40 & 8 \\
\text{u $\in$ $\top$ v} & 33 & 10 \\
\text{u $\in$ $\bot$ v} & 25 & 7 \\
\end{tabular}
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?