On bytecode slicing and AspectJ interferences

Antonio Castaldo D’Ursi    Luca Cavallaro    Mattia Monga

March 13th 2007
Outline

1. Motivations
2. XCutter
3. Interference analysis results
4. Conclusions and future developments
Outline

1. Motivations
2. XCutter
3. Interference analysis results
4. Conclusions and future developments
Aspects and modularization

- Aspects describe crosscutting computations, referring to an abstract view of the system.

- Composition is performed by the weaving process.

- Code affected by an aspect is oblivious about that (it does not contain any clue about if an aspect might be advised on it).

- As a result while aspect code units are physically separated they might unwittingly be not logically distinct.
Aspect interference

- We defined a notion of aspect interference.
- To quantify aspect interference we focused on the portion of a program affected by an aspect.
- Our definition is based on the observation of the system state.

**Definition**

An aspect $A$ does not interfere with a code unit $C$ if and only if every interesting predicate on the state manipulated by $C$ is not changed by the application of $A$. 

---

**Interference analysis results**

---

**Conclusions and future developments**

---
Interference for AspectJ programs

- We derived an operative test for AspectJ programs
- Our test is based on backward static slicing
- Given a criterion the backward static slice is the set of instructions in the source code that influence the criterion

**Definition**

- $A_1$ and $A_2$ are two aspects
- $S_1$ and $S_2$ the corresponding backward slices obtained by using all the statements defined in $A_1$ and $A_2$ as slicing criteria
- $A_1$ does not interfere with $A_2$ if $A_1 \cap S_2 = \emptyset$. 
Outline

1. Motivations
2. XCutter
3. Interference analysis results
4. Conclusions and future developments
A bytecode level slicer featuring AspectJ constructs

- To verify our operative condition we built XCutter

- It is a bytecode level slicer that can analyze both Java and AspectJ programs

Main features

- Based on Soot\(^a\)
- Uses an IR to do analysis
- Features our space efficient slicing algorithm
- More details in our master thesis \(^b\)

\(^a\)http://www.sable.mcgill.ca/soot/
\(^b\)http://www.elet.polimi.it/upload/cavallaro/thesis/thesis.pdf
Outline

1. Motivations
2. XCutter
3. Interference analysis results
4. Conclusions and future developments
What we expected vs what we found

**Expected result**
- Given two pieces of advice $A_1$ and $A_2$
- If $A_1$ reads a variable $x$ and $A_2$ writes the same $x$
- $A_1$ does not interfere with $A_2$

**Obtained result**
We have interference also in this case
The after finally advice

- The tool detects spurious interferences in presence of after finally pieces of advice.
- The translation introduces control dependencies between aspect bytecode instructions.
- These dependencies may not be interesting for the programmer.
The after finally advice

Write Advice

Read Advice

Control dependence

catch Throwable t

throw Throwable r1

throw Throwable r2

throw Throwable r3

throw Throwable r4
Outline

1. Motivations
2. XCutter
3. Interference analysis results
4. Conclusions and future developments
What we can do

- Using annotations, the tool could ignore these dependencies

- Deciding if ignoring weaving introduced dependencies is safe is not trivial

- In our examples they were not significative

- A formal analysis of the problem is needed
Future work

- A deeper study on correctness of ignoring dependencies is needed
- The proposed system is easy to implement
- It can be unsafe
- Even if unsafe this analysis could discover other useful information
The End

Thanks for your attention