Semantic/Foundational Issues of AOP:

Challenges for FOAL

Shmuel Katz

The Technion

What have we accomplished?

- Several semantic definitions for aspect languages similar to AspectJ (Denotational, Operational, …)
  - Simpler object structure (Featherweight Java)
  - Define pointcuts and weaving strategies
- Specifications for aspects
  - Assume-Guarantee based
  - Correctness criteria, including for interference
- Verification for aspects
  - Based on model checking
  - Based on theorem proving (extended Hoare logic)
  - Both for woven system, and libraries of aspect models

More accomplishments

- Language extensions
  - Richer pointcut languages (with history, context)
  - More dynamic
  - For functional languages
- Static analysis tools
  - Dataflow and slicing for defining kinds of aspects
  - Detecting potential interference among aspects (A changes the value of a variable used in B)

A (Big) Problem with Aspects

- Interest may be subsiding!!
- Is it just research, or is the practice not catching on?
- One claim:
  - Non-standard, might be dangerous to use
- Another:
  - Too conservative, doesn't provide what I want

- Can Foundational Studies make any difference?

What exactly is the problem?

- Maybe AspectJ is the wrong aspect language
- Maybe the perceived benefit is too small
- Maybe aspects are too nonstandard and seen as dangerous
- Maybe verification is too expensive and hard to do

Trends from the mainstream

- Fundamental semantics are understood, new variants are suggested, but challenged with
  - " How does this help me?"
- Settle for bug detection, rather than full formal verification (e.g., bounded model checking with SAT)
- Runtime verification
- Make formal verification practical
  - "Under the hood" philosophy
  - Hoare's verifying compiler—directly from code
  - No user involvement, or interactive queries
  - Microsoft's SLAM verifier for software drivers

Implications for Aspect Language Constructs

- New ideas for modularity and cleaner constructs

1. Symmetric models?
   - HyperJ. Classpects, …
   - Need better ways of combining and merging
2. Better interfaces, treating fragile pointcuts
3. More abstract aspects
   - Combining aspects into more complex aspects
   - Use terminology natural to the aspect, not to the underlying system and its method calls
4. Move upstream: language independent constructs, mapped to various languages
5. Evaluate constructs with user experiments

Implications for Aspect Verification

1. Combining and chaining aspect analysis tools
   - Very little done so far
   - Dataflow for potential influence, model checking to detect real interference using specifications
   - Common Aspect Proof Environment (CAPE)
2. Automatic checks for aspects, interferences, and weaving, for fixed domain properties (“no harm”)
3. Combining static analysis and runtime checks
4. Verify code: Extend Java tools to treat aspects independently
5. Evaluate and compare tools using experiments

More Implications

1. Aspects still need clear programming styles that increase reliability
   - Verified design patterns for aspects
2. Can use aspects to modularize
   - Specifications and annotations
   - Abstractions (reducing state space)
   - Runtime checks for hard questions

Can this help?

- Would like to revitalize aspect research agenda (with more new ideas in the following discussion)

- For the wider picture---only time will tell

My Suggestions

L1. Symmetric models?
L2. Improved interfaces
L3. Abstract aspects (combining aspects, natural terms)
L4. Move upstream
L5. Evaluate with user experiments
V1. Combine and chain tools
V2. Automatic built-in checks
V3. Combine static analysis and runtime verification
V4. Verify directly from code
V5. Evaluate and compare tools
M1. Verified styles and design patterns for aspects
M2. Use aspects for modular specification and verification