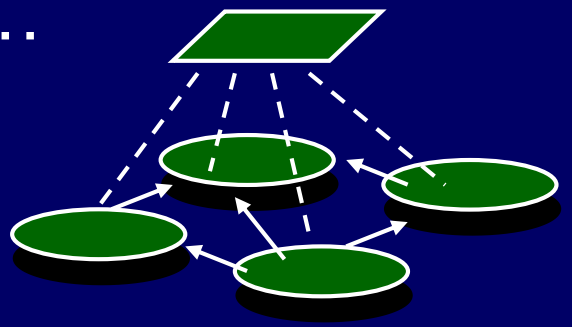
Reflection on (Reflection and) the Power of Pointcuts (or Aspects)

Hidehiko Masuhara
(The University of Tokyo)

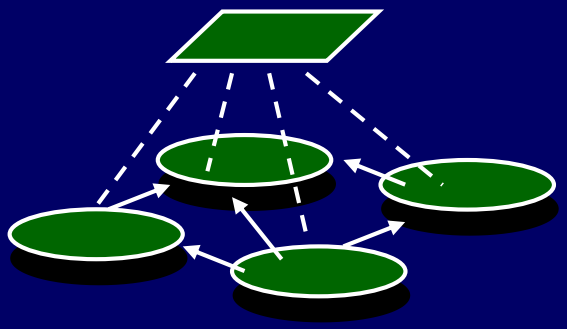
Computational reflection

- is computation about its own computational process [Smith'84, Maes'87]
- is useful to add controls into ***concurrent objects***
  - load-balancing, scheduling, distribution, time-warping, optimization…
  - ABCL/R[Watanabe88], ABCL/R2[Masuhara92] …



We didn’t conquer the world

- Difficult to program because of hard-to-predict effects
  - Changes at meta-level cannot be localized
- Difficult to develop tools because of flexibility in semantics

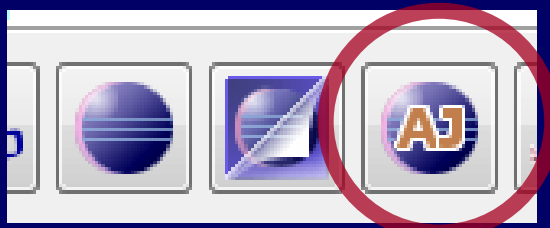


compilers, static analysis debuggers, IDEs, …

Aspect-oriented programming

[Kiczales97]

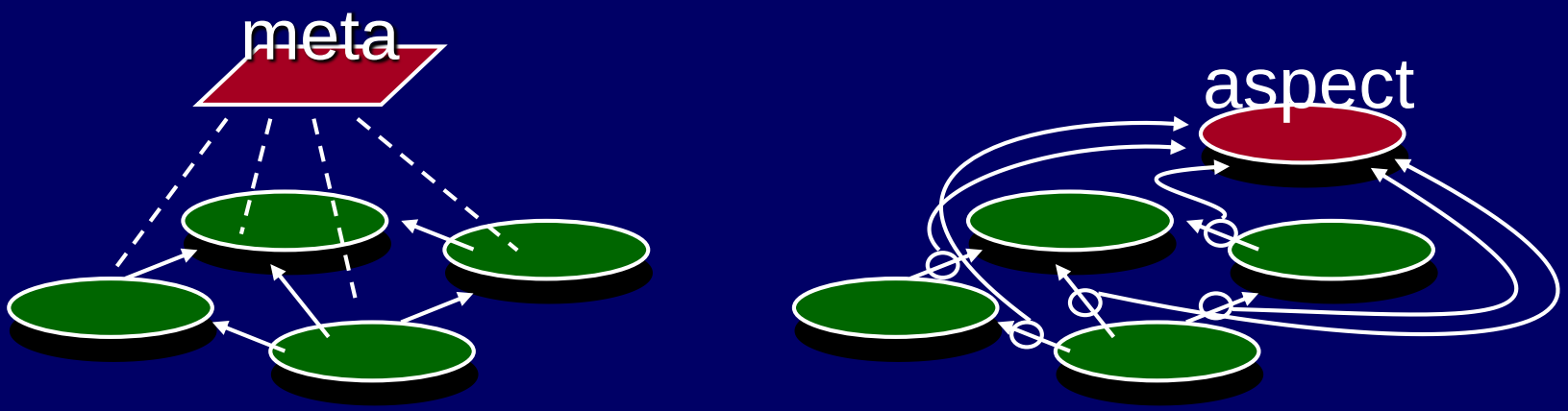
- offers limited ability
  - i.e., advice, or hooking on method calls
- but can realize many killer applications
- while enabling us to provide tools
  - e.g., AJDT, static analysis



Can AOP get closer to reflection without losing good properties?

Commonality and difference between Reflection & AOP

Common: can “hook” on everything



Different when we ***selectively*** hook

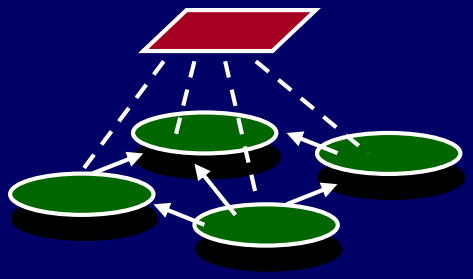
Key difference: namespaces

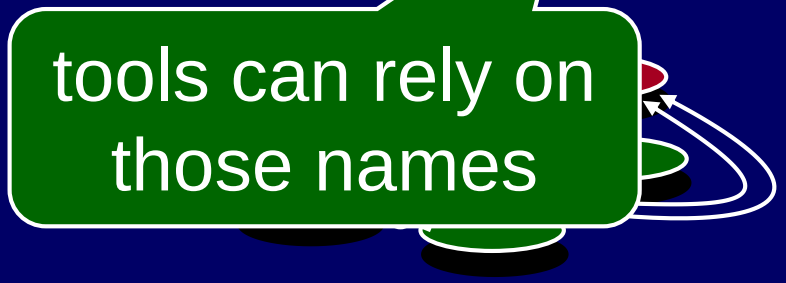
Reflection reifies base-level names to ***values*** at meta-level

```
s = "ma" + "in";
m = o.getClass()
      .getMethod(s);
m.appendInstructions(…);
```

Aspects live in the base-level namespace

```
import com.acme.Main;
after():
  call(int Main.main(..)) {
   … }
```





Enrich ***pointcuts*** in AOP with meta-information

- Allow aspects to “hook” by using richer information
  - like forward control flow / ***dataflow*** / results of ***static analysis*** and test executions
- Then they can modularize more things
  - like security / optimization
- Challenge:
  without contaminating namespaces

Dataflow pointcut for AspectJ

joint work with Kazunori Kawauchi

- can hook based on “where the data comes from”
- useful for security aspects, e.g., selective data sanitization

those only originating from user inputs

all outputs shall be quoted to avoid XSS

```
aspect XssSanitizingAspect {
  around (String s) :
        call(void print(String)) && args(s)  {
    proceed(quote(s));  } }
```

Sanitization with dataflow pointcut

```
aspect XssSanitizingAspect {
  around (String s) :
        call(void print(String)) && args(s) &&
        dflow[ s, userinput ]
          ( call(String get())
            && returns(userinput) )
    proceed(quote(s)); } }
```

when there is a dataflow from get() to print()

- More declarative; more robust against changes

SCoPE AspectJ compiler

joint work with Tomoyuki Aotani

- brings the power of reflection into AOP
  - can selectively hook based on user-defined static analysis
  - like forward control flow, dataflow, safety checks
- has conservative effects on semantics
  - does not contaminate namespaces

An example: making safety aspect more efficient

- Safety aspect replaces null argument with a default value

```
void around(URL a) : call(* request(URL))  && args(a) {
  if (a == null) a = new DefaultURL();
  proceed(a);
}
```

how to exclude obviously non-null cases?

any call to request method

```
T v;
if (...) v = new URL(...);
request(v);
```

```
request(new URL( ... ) );
```

with SCoPE: define and use "maybeNull" pointcut

1. Get an existing static analysis package (e.g., FindBugs)
2. Write a method that runs the analysis on a given method name

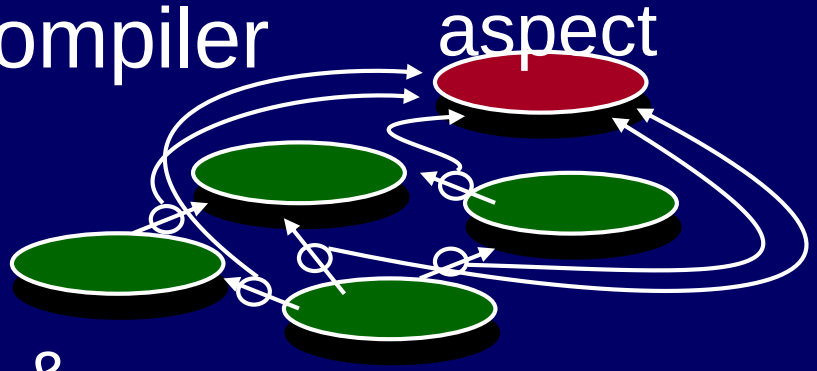
```
static boolean maybeNull(tjp){
  return FindBugs.nullCheck(tjp.getMethod()…);  }
```

3. Add "if" pointcut into the safety aspect

```
void around(a) : call(* request(URL)) && args(a)
        && if(maybeNull(tjp)) {
```

Implementation issues & approach

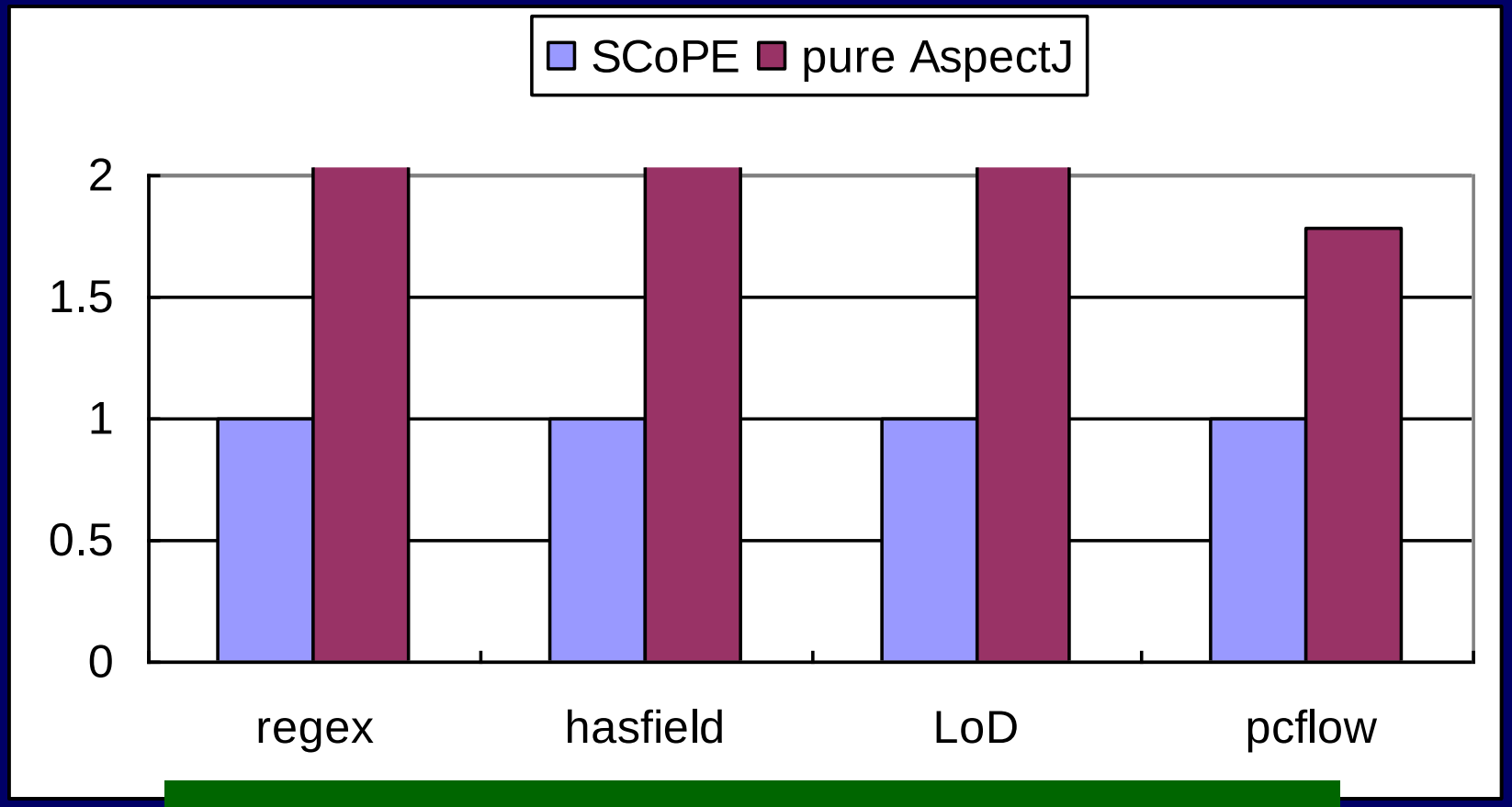
- Observing other aspects' effects
- Exploiting existing compiler implementations
- Our approach
  - analyze woven code & backpatch to eliminate runtime checks
  - conservative effect in semantics: merely eliminating conditional branches



Static analyses realized with SCoPE

- Null pointer check (via FindBugs)
- “Predictive” control flow [Kiczales03]
- Side-effect freeness
- The Law of Demeter [Lieberherr03]
- Checking class structures, like existence of fields, methods, constructors
- Regular expression-based matching
- …

Execution times relative to manual selection (fastest)



no runtime overheads in SCoPE!

Summary

- Living in the same namespace is crucial to providing tools
- We can bring the power of reflection into aspects
  - by selectively hooking based on user-defined static analysis
  - can be useful for security and optimization
  - without contaminating namespaces