Dependent Advice
A General Approach to Optimizing History-based Aspects
AspectJ as an intermediate language

Various specification languages

JavaMOP

LTL spec.
J-LO

S2A

LSC

History-based AJ aspect

abc

tracematches

relational aspects

MOFScript

M2Aspects

Java-STAIIIRS Aspects
Example concern

Do not write to a disconnected connection.
aspect ConnectionClosed {
  Set closed = new WeakIdentityHashSet();

  after /*disconn*/ (Connection c) returning:
    call(*/ Connection.disconnect()) && target(c) {
      closed.add(c);
    }

  after /*reconn*/ (Connection c) returning:
    call(*/ Connection.reconnect()) && target(c) {
      closed.remove(c);
    }

  before /*write*/ (Connection c) :
    call(*/ Connection.write (...) ) && target(c) {
      if (closed.contains(c))
        error(c+" is closed !");
    }
}
boolean foo(Iterator i, Iterator j) {
    while (i.hasNext() && j.hasNext()) {
        if (i.next() != j.next())
            return false;
    }
    return true;
}
Problem:
Potentially large runtime overhead
aspect ConnectionClosed {
  Set closed = new WeakIdentityHashSet();

  after /*disconn*/ (Connection c) returning:
  call(* Connection.disconnect()) && target(c) {
    closed.add(c);
  }

  after /*reconn*/ (Connection c) returning:
  call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
  }

  before /*write*/ (Connection c):
  call(* Connection.write (...) ) && target(c) {
    if (closed.contains(c))
      error(c+" is closed !");
  }
}
Now: general case

Static program analysis

History-based AJ aspects

Optimized Runtime
/* Monitor aspect for HasNext+*/
import java.util.*/;
import org.apache.commons.collections.map.*/;
class HasNextMonitorPM {
    Vector monitorList = new Vector();
synchronized public void hasNext(Iterator i) {
        HashSet monitorSet = new HashSet();
        Iterator it = monitorList.iterator();
        while (it.hasNext()){
            HasNextMonitor monitor = (HasNextMonitor)it.next();
            monitor.hasNext(i);
            if (monitorSet.contains(monitor) || monitor.failed())
                it.remove();
            else {
                monitorSet.add(monitor);
                if (monitor.succeeded()){
Static program analysis
Contributions and Outline

Syntax of dep. advice

```java
aspect ConnectionClosed {
    Set closed = new WeakIdentityHashSet();

    after disconnect(connection c) returning:
        call("Connection.disconnect()") & target(c) {
            closed.add(c);
        }

    ... before write(Connection c)
        call("Connection.write (...)") & target(c) {
            if (closed.contains(c))
                error("c is closed!");
        }

    dependency( strong disconnect,write; )
}
```

Semantics of dep. advice

![Diagram showing strong and weak semantics]

Generating dep. advice

```plaintext
dependency( strong a,d; weak c; )
dependency( strong a,c,d; weak b; )
```

Experimental results

![Graph showing experimental results with different scenarios and outcomes]

```plaintext
- Quick-Check, 24
- Orphan-sessions analysis, 12
- Shadows remaining, 36
- Reduced to 24%, 6
- Some reduction, 18
- No reduction, 10
```
aspect ConnectionClosed {
    Set closed = new WeakIdentityHashSet();

    after disconnect(connection c) returning:
        call(Connection.disconnect()) && target(c) {
            closed.add(c);
        }

    ...

    before write(Connection c):
        call(Connection.write(..)) && target(c) {
            if (closed.contains(c))
                error(c+" is closed ");
        }

    dependency{ strong disconnect,write; }
aspect ConnectionClosed {
    Set closed = new WeakIdentityHashSet();

    after disconn(Connection c) returning:
        call(\ Connection.disconnect() && target(c) {
            closed.add(c);
        }

    after reconn(Connection c) returning:
        call(\ Connection.reconnect() && target(c) {
            closed.remove(c);
        }

    before write(Connection c):
        ...

        dependency{ strong disconn,write; weak reconn; }
dependency{ strong disconn, write; weak reconn; }
Verbose syntax

dependency{
    strong disconn, write;
    weak reconn;
}

... is a shorthand

dependency{
    strong disconn(c), write(c);
    weak reconn(c);
}
When is a dependency fulfilled?

dependency{
    strong disconn(c), write(c);
    weak reconn(c);
}

Dependency is fulfilled for Connection c if both disconn(c) and write(c) do execute on c at some point in time.
Example

```java
try{
    disconnect(c), write(c);
    conn(c);
}

Connection c1 = new Connection();
Connection c2 = new Connection();
c1.disconnect();
c2.write("foo");
c1.reconnect();
c1.write("bar");

→ dependency fulfilled for c1, but not fulfilled for c2
```
When does a Dep. Adv. execute?

Dependent advice a **must** execute at a joinpoint $j$ on objects $o$ if there exists a dependency $d$ that references $a$ and is fulfilled for objects $o$. 
Example

```java
try{
    disconnect(c), write(c);
    reconnect(c);
}
```

Connection `c1` = new Connection();
Connection `c2` = new Connection();

`c1`.disconnect();
`c2`.write("foo");
`c1`.reconnect();
`c1`.write("bar");

→ disconnect/write/reconnect *will* execute on `c1`, do not have to execute (but *may*) on `c2`
Optimizing Dependent Advice

Motivated by tracematch-based analysis, Bodden, Hendren & Lhotak (ECOOP 2007)

Two analysis stages:
- Quick check
  - syntactic
- Flow-insensitive Orphan-shadows analysis
  - uses context-sensitive points-to information
Auto-generating dependent advice

Various specification languages

JavaMOP

History-based AJ aspect

abc

tracematches

relational aspects

Java-STAIRS Aspects

MOFScript

M2Aspects

LTL spec.

J-LO

S2A

LSC

MSC
abc compiler

tracematch

Finite-state machine

Generic Algorithm

Dependency Declarations

JavaMOP

ERE spec.
FTLTL spec.
PTLTL spec.

Finite-state machine

Generic Algorithm
FSM $\rightarrow$ dependency declarations

dependency{ strong a; }
dependency{ strong a d; weak b; }
Proven: Algorithm is "stable"

Equivalent automata yield equivalent dependency declarations
## Benchmarks - Properties

<table>
<thead>
<tr>
<th>ASyncIter</th>
<th>FailSafeIterM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASyncIterM</td>
<td>HasNext</td>
</tr>
<tr>
<td>FailSafeEnum</td>
<td>LeakingSync</td>
</tr>
<tr>
<td>FailSafeEnumHT</td>
<td>Reader</td>
</tr>
<tr>
<td>FailSafeIter</td>
<td>Writer</td>
</tr>
</tbody>
</table>
Benchmarks - Properties

For each of the ten properties:
- Hand-coded AspectJ aspect & annotations
- Tracematch

Where possible:
- JavaMOP specification in ERE

For three specifications also:
- JavaMOP specification in FTLTL
- JavaMOP specification in PTLTL
Benchmark programs

DaCapo benchmark suite:

<table>
<thead>
<tr>
<th>antlr</th>
<th>hsqldb</th>
</tr>
</thead>
<tbody>
<tr>
<td>bloat</td>
<td>jython</td>
</tr>
<tr>
<td>chart</td>
<td>lucene</td>
</tr>
<tr>
<td>eclipse</td>
<td>pmd</td>
</tr>
<tr>
<td>fop</td>
<td>xalan</td>
</tr>
</tbody>
</table>
Runtime overhead

380 woven programs

<=10%, 308

>10%, 72
Elimination of all shadows

Quick-Check, 24
Orphan-shadows analysis
Shadows remaining, 36

zero overhead after optimization

72 programs with overhead >10%
Reduction of runtime overhead

36 programs with overhead >10% and shadows remaining
Limitations

Law-of-Demeter Checker (Lieberherr et al.)

```java
after() returning(Object o):IgnoreTargets() {
    ignoredTargets.put(o, o);
}

after(Object thiz, Object targt):
    Any.MethodCall(thiz, targt) && !IgnoreCalls() {
        if (!ignoredTargets.containsKey(tarzt) &&
            !Pertarget.aspectOf(thiz).contains(tarzt)) {
            objectViolations.put(tjSP, tjSP);
        }
    }
```
Related work

Various specification languages
JavaMOP

History-based AJ aspect
abc
tracematches
relational aspects

LTL spec.
J-LO

S2A
LSC

M2Aspects
MSC

MOFScript
Java-STAIRS Aspects
Related work

Already mentioned:
- **S2A** (Maoz & Harel, FSE 2006)
- **M2Aspects** (Krüger et al., SCESM 2006)
- **Java-STAIRS Aspects** (Oldevik & Haugen, 5pm)
- **J-LO** (Bodden, Diploma Thesis)

Other possible clients of dependent advice:
- **Association aspects** (Sakurai et al., AOSD 2004)
- **LogicAJ** (Kniesel et al., RAM-SE 2004)
- **Dataflow pointcuts** (Masuhara & Kawauchi, APLAS 2003; and tomorrow, 14:30)
- **Conditional compilation** (Adams et al., Friday)
Related work

Optimizations for tracematches:
- Bodden, Hendren & Lhotak, ECOOP 2007
- Bodden, Lam & Hendren, FSE 2008
- Naeem & Lhotak, OOPSLA 2008

Optimizations of the Runtime Monitor:
- Avgustinov et al., OOPSLA 2007
- Chen & Rosu, TACAS 2009
Important conclusion

Approach hard to formalize without AOP

- History-based aspect modularizes instrumentation
- Hence can use modular dependency annotation
Acknowledgements

col-workers:
- Laurie Hendren
- Patrick Lam

developer of S2A:
- Shahar Maoz

abc/tracematch maintainers:
- Pavel Avgustinov
- Julian Tibble
aspect ConnectionClosed {
    Set closed = new HashSet();

    after disconn(Connection c) returning:
        call[• Connection.disconnect()] as target(c) {
            closed.add(c);
        }
    
    after write(Connection c) after-trying:
        call[• Connection.write (..)] as target(c) {
            if (closed.contains(c))
                error(c" in close()");
        }

    dependency{ strong disconn, write; }
}
Static analysis

dependency{
    strong disconnect(c), write(c);
    weak reconnect(c);
}

Connection c1 = new Connection(); // (1)

Connection c2 = new Connection(), c3; // (2)
c1.disconnect();
c2.write("foo");
c1.reconnect();
c3 = c1;
c3.write("bar");
Results – Static-analysis time

Average total: 12 minutes
Max total: 58 minutes