Nu: a Dynamic Aspect-Oriented Intermediate Language Model and Virtual Machine for Flexible Runtime Adaptation

Robert Dyer and Hridesh Rajan

Department of Computer Science
Iowa State University
{rdyer,hridesh}@cs.iastate.edu

7th International Conference on Aspect-Oriented Software Development
Motivation: Supporting dynamic aspect-oriented constructs

Approach: Nu

Evaluation: Expressiveness and performance

Technical Contributions:
- Flexible, dynamic AO intermediate language model
- Implementation in industrial strength VM (Sun Hotspot)
- Dedicated AO caching mechanism
Need For a Dynamic IL Model

- Currently: Dynamic, high-level AO constructs $\rightarrow$ low-level OO representation
- AO compilers need “building blocks”!
- Perhaps an example...
History-based Pointcuts

- History-based pointcuts [Douence, Fradet, and Südholt]
- Temporal constructs in AspectJ [Stolz and Bodden]
History-based Pointcuts

- History-based pointcuts [Douence, Fradet, and Südholt]
- Temporal constructs in AspectJ [Stolz and Bodden]
History-based Pointcuts - Static Translation

- entry(setWord)
- exit(setWord)
- afterP1
- afterP1P2
- entry(setBit)
- exit(setBit)
- afterP2
- afterP1P2
History-based Pointcuts - Static Translation

Overview
Motivation
Our Approach
Evaluation
Summary

Robert Dyer and Hridesh Rajan

Nu: Dynamic AO IL Model and VM
History-based Pointcuts - Dynamic Support

- entry(setWord) → exit(setWord)
- afterP1
- afterP1P2

- entry(setBit) → exit(setBit)
History-based Pointcuts - Dynamic Support

Overview
Motivation
Our Approach
Evaluation
Summary

Robert Dyer and Hridesh Rajan
9
Nu: Dynamic AO IL Model and VM
History-based Pointcuts - Dynamic Support

- **Motivation**
- **Our Approach**
- **Evaluation**
- **Summary**

Nu: Dynamic AO IL Model and VM
History-based Pointcuts - Dynamic Support

Overview
Motivation
Our Approach
Evaluation
Summary

Robert Dyer and Hridesh Rajan

Nu: Dynamic AO IL Model and VM
Need For a Dynamic IL Model

- Inadequate support for dynamic use cases in current ILs
  - Dynamic deployment,
  - Dynamic adaptation,
  - Policy changes, etc
- Dynamically adapting set of advised join points
  - Morphing aspects [Hanenberg et al.]
  - Open Aspects [Hirschfeld and Hanenberg]
Overview of the Nu Model

- JPM: Point-in-time model [Masuhara et al.]
- New primitives: *bind* and *remove*
- Advice as delegate methods
- Library of patterns
  - First-class, immutable objects
### Two New Primitives: Bind and Remove

<table>
<thead>
<tr>
<th>Stack Transition</th>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bind</code></td>
<td><code>nullpointerexception</code></td>
</tr>
<tr>
<td><code>..., Pattern, Delegate</code></td>
<td><code>nullpointerexception</code></td>
</tr>
<tr>
<td><code>..., BindHandle</code></td>
<td><code>nullpointerexception</code></td>
</tr>
<tr>
<td><code>remove</code></td>
<td><code>illegalargumentexception</code></td>
</tr>
<tr>
<td><code>..., BindHandle</code></td>
<td><code>illegalargumentexception</code></td>
</tr>
</tbody>
</table>

This talk uses source representation for ease.
public class AuthLogger {

}
public class AuthLogger {
    protected static Pattern p;
    protected static Delegate d;

    static {
        p = new Execution(new Method("*.login"));
        d = new Delegate(AuthLogger.class, "log");
    }

    public static void log() { // record the time of login }
}
public class AuthLogger {
    protected static Pattern p;
    protected static Delegate d;

    static {
        p = new Execution(new Method("*.login"));
        d = new Delegate(AuthLogger.class, "log");
    }

    public static void log() { // record the time of login }
}
public class AuthLogger {
    protected static Pattern p;
    protected static Delegate d;

    static {
        p = new Execution(new Method("*.login"));
        d = new Delegate(AuthLogger.class, "log");
    }

    public static void log() { // record the time of login }
}
public class AuthLogger {
    protected static Pattern p;
    protected static Delegate d;
    protected static BindHandle id = null;

    static {
        p = new Execution(new Method("*.login"));
        d = new Delegate(AuthLogger.class, "log");
    }

    public static void enable() { id = bind(p, d); }

    public static void log() { // record the time of login }
}
public class AuthLogger {
    protected static Pattern p;
    protected static Delegate d;
    protected static BindHandle id = null;

    static {
        p = new Execution(new Method("*.login"));
        d = new Delegate(AuthLogger.class, "log");
    }

    public static void enable() { id = bind(p, d); }
    public static void disable() { remove(id); }

    public static void log() { // record the time of login }
}
Implementation Overview

- Built on top of Sun Hotspot VM
- Adds code to the interpreter for join point dispatch
- bind and remove implemented as native methods
Implementation Overview

- Interpreter (asm)
- JRE (Java)
- VM (c++)
Implementation Overview

- Interpreter (asm)
- JP Dispatcher
- JRE (Java)
- Pattern Library
- VM (c++)
- Matcher
- bind/remove
Novel AO Caching Mechanism

- Interpreter (asm)
  - JP Dispatcher

- JRE (Java)
  - Pattern Library

- The Nu Intermediate Language Model
  - The Nu Virtual Machine

- MethodOop
  - Local Counter
  - Delegate Cache

- Classloader
  - Cache Initialization

- VM (c++)
  - Matcher
  - Global Counter

  bind/remove
Caching Mechanism

- **Local counter == Global counter?**
  - **Yes**
    - **Cached Delegates?**
      - **Yes**
        - Invoke Cached Delegates
      - **No**
        - Continue Normal JP Execution
  - **No**
    - Perform Incremental Match
Caching Mechanism

Local counter == Global counter?
  Yes
  Cached Delegates?
    Yes
    Invoke Cached Delegates
    No
    Continue Normal JP Execution
  No
    Perform Incremental Match

Robert Dyer and Hridesh Rajan
Expressing Constructs

- Supports multiple language constructs:
  - AspectJ aspects, pointcuts and advice
  - cflow, cflowbelow
  - History-based pointcuts
  - CaesarJ’s deploy
Expressing Constructs

- Supports multiple language constructs:
  - AspectJ aspects, pointcuts and advice
  - cflow, cflowbelow
  - History-based pointcuts
  - CaesarJ’s deploy
cflow(execution(setWord)) && execution(setBit)

if (counter == 0) {
    id = bind(call(setBit), advice);
    counter++;
}
counter--;
if (counter == 0) {
    remove(id);
}
cflow(execution(setWord)) && execution(setBit)

if (counter == 0)
    id = bind(call(setBit), advice);
    counter++;

counter--;
if (counter == 0)
    remove(id);

entry(setWord)

advice

entry(setBit) exit(setBit)

exit(setWord)
cflow(execution(setWord)) && execution(setBit)

if (counter == 0)
    id = bind(call(setBit), advice);
counter++;

counter--;
if (counter == 0)
    remove(id);

entry(setWord) --

entry(setBit) --

exit(setBit) --

exit(setWord) --
Performance Evaluation Overview

- Java Grande, SPEC JVM98, and custom micro-benchmarks
- Unmodified VM, Nu VM (caching), Nu VM (no caching)
- Only about 1.5% overhead
Java Grande Method Benchmark
SPEC JVM98 Benchmark

- **Nu**
- **JVM**
- **Nu (with caching)**

<table>
<thead>
<tr>
<th>Category</th>
<th>Nu</th>
<th>JVM</th>
<th>Nu (with caching)</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>200</td>
<td>120</td>
<td>140</td>
</tr>
<tr>
<td>jess</td>
<td>40</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>db</td>
<td>60</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>javac</td>
<td>80</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>mpegaudio</td>
<td>100</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>mtrt</td>
<td>120</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>jack</td>
<td>140</td>
<td>120</td>
<td>130</td>
</tr>
</tbody>
</table>
Related Work

- Steamloom - Bockisch, Haupt, Mezini, and Ostermann
- Prose - Popovici, Alonso, and Gross
- Delegation-based Machine Model - Haupt and Schippers
- Morphing Aspects and Continuous Weaving - Hanenberg, Hirschfeld, and Unland
Future Work

- Around Advice
- Dynamic Introductions
- Additional Optimizations
Motivation: Supporting dynamic aspect-oriented constructs
- cflow, deploy, history-based, etc
- Compiled into static constructs
- Lower-level support may yield run-time benefits

Approach: Nu
- IL-level primitives for dynamic deployment
- Dedicated caching mechanism (low overhead)

Evaluation: Expressiveness and performance
- Supports large subset of dynamic AO constructs
- Only about 1.5% overhead

Technical Contributions:
- Flexible, dynamic AO intermediate language model
- Implementation in industrial strength VM (Sun Hotspot)
- Dedicated AO caching mechanism
Questions?

http://www.cs.iastate.edu/~nu/
Caching Mechanism Assembly Code

```
movl   eax, methodOop.counter
movl   ecx, globalCounter

// methodOop.counter == globalCounter?
    cmpl   eax, ecx
    jcc    equals, InvokeDelegates

    call_VM incrementalMatcher

InvokeDelegates:

    movl   eax, methodOop.cache.head

    // cache.head == NULL?
    testl  eax, eax
    jcc    zero, ContinueJP

    // invoke the cached delegates

ContinueJP:
```
Pattern Library Example

All Join Points → m = new Method("*.login")

- new Execution(m)
- new Return(m)
- new Call(m)
- new Reception(m)
- ...

Pattern Library Example

All Join Points \[ \rightarrow \] m = new Method(*.login) \[ \rightarrow \]

- new Execution(m)
- new Return(m)
- new Call(m)
- new Reception(m)

...
Pattern Library Example

```
All Join Points  m = new Method('*.login')

new Execution(m)
new Return(m)
new Call(m)
new Reception(m)
```

...
aspect Tcheck {
    pointcut p1(): call(* Word.set(..));
    int p1 = 1;

    pointcut p2(): call(* Bit.set(..));
    int p2 = 2;

    Formula state = Globally(p1, Finally(p2));
    Set<int> propSet = new Set<int>();

    ...
}
aspect Tcheck {
    pointcut p1(): call(* Word.set(..));
    int p1 = 1;

    pointcut p2(): call(* Bit.set(..));
    int p2 = 2;

    Formula state = Globally(p1, Finally(p2));
    Set<int> propSet = new Set<int>();

    ...
}
aspect Tcheck {
    pointcut p1(): call(* Word.set(..));
    int p1 = 1;

    pointcut p2(): call(* Bit.set(..));
    int p2 = 2;

    Formula state = Globally(p1, Finally(p2));
    Set<int> propSet = new Set<int>();

    ...
}
... 

after(): p1() { propSet.add(p1); } 
after(): p2() { propSet.add(p2); } 

after(): p1() || p2() { 
    state = state.transition(propSet);
    if (state.equals(Formula.TT))
        // report formula as satisfied
    else if(state.equals(Formula.FF))
        // report formula as falsified
    state.clear(); //reset proposition vector 
}
... 

after(): p1() { propSet.add(p1); } 

after(): p2() { propSet.add(p2); } 

after(): p1() || p2() { 
    state = state.transition(propSet); 
    if (state.equals(Formula.TT)) 
        // report formula as satisfied 
    else if(state.equals(Formula.FF)) 
        // report formula as falsified 
    state.clear(); // reset proposition vector 
}
class Tcheck {
    int p1 = 1;
    int p2 = 2;

    Formula state = Globally(p1, Finally(p2));
    Set<int> propSet = new Set<int>();

    ...
}
class Tcheck {
    int p1 = 1;
    int p2 = 2;

    Formula state = Globally(p1, Finally(p2));
    Set<int> propSet = new Set<int>();

    ...

class Tcheck {
    int p1 = 1;
    int p2 = 2;

    Formula state = Globally(p1, Finally(p2));
    Set<int> propSet = new Set<int>();

    ...

protected static Pattern prop2;
protected static Delegate d2;

static {
    Pattern prop1 = new Call("* Word.set(..)"iciencies");
    d1 = new Delegate(Tcheck.class, "afterP1");
    bind(prop1, d1);

    prop2 = new Call("* Bit.set(..)"iciencies");
    d2 = new Delegate(Tcheck.class, "afterP2");

    Pattern afterP1P2 = new Or(prop1, prop2);
    d3 = new Delegate(Tcheck.class, "afterP1P2");
    bind(afterP1P2, d3);
}
protected static Pattern prop2;
protected static Delegate d2;

static {
    Pattern prop1 = new Call("* Word.set(..)");
    d1 = new Delegate(Tcheck.class, "afterP1");
    bind(prop1, d1);

    prop2 = new Call("* Bit.set(..)");
    d2 = new Delegate(Tcheck.class, "afterP2");

    Pattern afterP1P2 = new Or(prop1, prop2);
    d3 = new Delegate(Tcheck.class, "afterP1P2");
    bind(afterP1P2, d3);
}
protected static Pattern prop2;
protected static Delegate d2;

static {
    Pattern prop1 = new Call("\* Word.set(\..\)");
    d1 = new Delegate(Tcheck.class, "afterP1");
    bind(prop1, d1);

    prop2 = new Call("\* Bit.set(\..\)");
    d2 = new Delegate(Tcheck.class, "afterP2");

    Pattern afterP1P2 = new Or(prop1, prop2);
    d3 = new Delegate(Tcheck.class, "afterP1P2");
    bind(afterP1P2, d3);
}
protected static BindHandle id;
public void afterP1() {
    propSet.add(p1);
    id = bind(prop2, d2);
}
public void afterP2() {
    propSet.add(p2);
    remove(id);
}
public void afterP1P2() {
    ... // same as before
}
... protected static BindHandle id;
public void afterP1() {
    propSet.add(p1);
    id = bind(prop2, d2);
}
public void afterP2() {
    propSet.add(p2);
    remove(id);
}
public void afterP1P2() {
    ... // same as before
}
protected static BindHandle id;
public void afterP1() {
    propSet.add(p1);
    id = bind(prop2, d2);
}
public void afterP2() {
    propSet.add(p2);
    remove(id);
}
public void afterP1P2() {
    ... // same as before
}