ALIA4J’s [(Just-In-Time) Compile-Time] MOP for Advanced Dispatching

Christoph Bockisch\textsuperscript{1}, Andreas Sewe\textsuperscript{2}, Martin Zandberg\textsuperscript{1}

\textsuperscript{1}Universiteit Twente, The Netherlands
\textsuperscript{2}Technische Universität Darmstadt, Germany
Motivation and Goal

- Much research in programming languages
  - Increase modularity
  - Variations of late-binding ("advanced dispatching")
- Language creation involves two roles

  ![Language designer](image)

  ![Language implementer](image)

- High-level semantics
- Prototype based on interpretation
- Semantic interactions of language concepts

- Low-level implementation
- Product based on compilation
- Indentify optimizable special cases

- No unnecessary complexity
- Correctness
- Reusability

Oct. 23, 2011
http://www.alia4j.org – VMIL @ SPLASH 2011
Advanced-Dispatching Language-Implementation Architecture for Java (ALIA4J)

abstract classes

Concrete classes by language designer

META-OBJECTS

PROTOCOL: derives execution model from meta-objects

Language-independent meta-model

Language-specific program model

Dispatching model

Java bytecodes

co-implmented concrete classes by language implementer

Framework instantiation

Virtual Machine

Framework for execution environments
public aspect Aspect isSingleton() {
    before(Integer i) :
    call(void Main.m(Number)) && args(i) {
        // Advice body compiled to
        // method Aspect.before(Integer)
    }
}

- Attachment
  - MethodCallAction
    - method="Aspect.before(Integer)"
  - Specialization
  - ScheduleInfo
    - time="before"

- PerTupleContext
  - contexts=[]
  - type="Aspect"
- ArgumentContext
  - index=0
- BasicPredicate
- Instanceof-Predicate
  - type="Integer"
- MethodPattern
  - TypePattern
  - NamePattern
...
Deriving Execution Model by Example

Find join point shadows.

Attachment

MethodCallAction
method="Aspect. before(Integer)"

Specialization
time="before"

ScheduleInfo

PerTupleContext
contexts=[]
type="Aspect"

ArgumentContext
index=0

BasicPredicate

InstanceOf-Predicate
type="Integer"

MethodPattern

TypePattern

NamePattern...

Oct. 23, 2011
http://www.alia4j.org – VMIL @ SPLASH 2011
Deriving Execution Model by Example

Join point shadow

Create specification how to execute action.

Attachment

MethodCallAction
method="Aspect. before(Integer)"

Specialization

ScheduleInfo
time="before"

PerTupleContext
contexts=[]
type="Aspect"

ArgumentContext
index=0

BasicPredicate

InstanceOfPredicate
type="Integer"
Deriving Execution Model by Example

Join point shadow
- Aspect: before
- impl: action

Dynamically select actions to execute.
- Attachment
- Specialization
- ScheduleInfo time="before"
- BasicPredicate
- Instanceof-Predicate type="Integer"

Every join point has implicit attachment of called Java method.

Oct. 23, 2011
http://www.alia4j.org – VMIL @ SPLASH 2011
Deriving Execution Model by Example

Join point shadow

- Dispatch Function
  - instanceof
  - impl. action

- Aspect before impl action

Attachment

- ScheduleInfo
  - time=“before”

Order actions and consider other constraints.
Deriving Execution Model by Example

Join point shadow

Dispatch Function

instanceof

Aspect. before
impl. action

impl. action
Meta-Object Protocol by Example

```java
public class Main {
    public static void main(String[] args) {
        new Main().m(new Integer(1));
    }
    public void m(NumberOf n) {
        ...
    }
}
```

1. Find corresponding execution model
2. Perform semantics of **ArgumentContext**
3. Perform semantics of **InstanceofPredicate**, passing result of step 2
4. Execute actions
   in top box, if step 3 returns true in bottom box, otherwise
   a. Evaluate exposed context values and execute actions

Oct. 23, 2011
http://www.alia4j.org – VMIL @ SPLASH 2011
ALIA4J Execution Environments

- **Generic work flows** implemented in FIAL
  - Handle dynamic class loading and deployment
  - Derive execution model
  - Optimize execution model
  - ...

- Concrete execution environments realize execution model by
  - **Interpretation** (Steamloom^{ALIA}, SiRIn, NOIRIn)
  - **Bytecode generation** (Steamloom^{ALIA}, SiRIn)
  - **Machine code generation** (Steamloom^{ALIA})
Strengths of MOPs

- Plain MOP
  - No knowledge of execution environment
  - Implemented in plain Java
- Compile-time MOP
  - Avoid indirection
  - Access to compilation context
  - Deferred code generation: can consider runtime state
- Just-in-time (JIT) compile-time MOP
  - As compile-time MOP
  - More precise compilation context
  - Access to low-level operations
    Direct memory access, runtime services, etc.
Implementing Semantics/Optimizations

public class PerTupleContext extends Context implements ... {
  public Object getObjectValue (Object[] tuple) {
    ...
  }
  public void build(BytecodeBuilder builder, JoinPointSite call) {
    ...
  }
  public void generateASM(Assembler asm, BaselineCompilerState compilerState) {
    ...
  }
  public void generate("IR/CGIR/bc2ir, GenerationContext gc) {
    ...
  }
}

- Modular implementation in meta-model entity
- Choose best strategy supported by execution environment
- **Transparent to the user**
Levels of Meta-Object Protocol

1. Find corresponding execution mode
2. Perform semantics of **ArgumentContext**
3. Perform semantics of **InstanceOfPredicate**
   passing result of step 2
4. Execute actions
   in top box, if step 3
   in bottom box, otherwise
   a. Evaluate exposed objects and execute actions

Oct. 23, 2011  http://wwwalia4jorg ~ VMIL @ SPLASH 2011
JIT Compile-Time MOP: Case Study

AspectJ and other aspect-oriented languages:

- Aspects are types and can be instantiated to objects
- Advice execute in context of aspect instance
- Advice are invoked implicitly

aspect instance creation/selection specified by rules

- PerTupleContext implements generalized aspect-instantiation model
  - Creation: on-demand (implicit) or in-advance (explicit)
  - Selection: based on context-value tuple
JIT Compile-Time MOP: Case Study

- Generic default implementation using map
- Optimizations depend on usage of PerTupleContext

- Example AspectJ issingleton():
  *implicit, context-insensitive*
  - Implicit
    - Implementer has control over instantiation
  - Context-insensitive
    - Result may be known at JIT compile-time
public class PerTupleContext extends Context implements ...

    public void generateASM(Assembler asm, BaselineCompilerState compilerState) {
        if (this.isInsensitive() && this.getContexts().isEmpty()) {
            if (!this.hasInstance()) {
                // instance still uninitialized?
                // allocate instance in non-moving heap
                // store instance
                // load instance
            } else {
                asm.emitPUSH_Imm(
                    Magic.objectAsAddress(this.getInstance()).toInt()
                );
            }
        } else { ... }
    }

Preliminary performance evaluation promising
Lessens Learnt

- Separation of roles works

Andre Loker  Martin Zandberg

- ALIA4J’s system of MOPs helps

Correct transition
- Regression test suite defined by Andre, applied by Martin

Re-use previous implementation
- Martin re-uses Andre’s implementation for non-special cases

Avoiding unnecessary complexity
- See second case study in paper
http://www.alia4j.org

Christoph Bockisch, c.m.bockisch@cs.utwente.nl
Andreas Sewe, sewe@st.informatik.tu-darmstadt.de
Martin Zandberg
Re-use of LIAM entities

- ALIA4J includes implementations of many concepts
- **Most of them** implement at least **BytecodeSupport**
- Explored mapping many different languages