Mostly Modular Compilation of Crosscutting Concerns by Contextual Predicate Dispatch

Shigeru Chiba
Salikh Zakirov

Atsushi Igarashi

Tokyo Institute of Technology

Kyoto University

TOKYO INSTITUTE OF TECHNOLOGY

TOKYO TECH

Aspect Oriented Programming

- … modularizes the un-modularizable.
- … breaks modularity.

10th AOSD

2011 March in Brazil

Object Oriented Programming

- Modular type checking
  and Separate compilation
  - Yes if super classes are given

- Modular reasoning
  - Fairly? but dynamic dispatch

~~Object Oriented~~ Programming

Aspect Oriented

- Modular type checking
  and Separate compilation
  - Yes if super classes are given

- Modular reasoning
  - Fairly? but dynamic dispatch

TOKYO INSTITUTE OF TECHNOLOGY

GluonJ

- A simple AOP extension to Java
  - A subset of AOP functionality of AspectJ
  - An OOP-like mechanism
    - Enhanced method overriding and dispatching
    - For ease of comparison between AOP and OOP
    - Are type checking and compilation modular?

TOKYO INSTITUTE OF TECHNOLOGY

TOKYO TECH

AOP functionality (1)

- Destructive extension
  - An advice can obliviously modify a method.
    - Unlike subclass, mixin, traits, …

```
class AddExpr {
  Value eval() { … }
}
```

```
Value around(AddExpr ae):
    execution(Value AddExpr.eval())
    && this(ae) {
    if (…) proceed();
    else … ;
}
```

AOP functionality (2)

- Limited scope
  - An advice can modify a method call in a body
  - Breaking modularity?

```
class VarDecl {
  Value init() {
    v = right.eval();
} }

    class AddExpr {
      Value eval() { … }
    }
```

```
aspect Logging {
  before():
    call(void Expr.eval())
    && withincode(* VarDecl.init()) {
    ...
} }
```

GluonJ: A reviser

- Destructive extension
  - A reviser can add and override a method, and add a field to an existing class.
    - It cannot have an explicit constructor.

```
class AddExpr {
  Value eval() { … }
}
```

```
class FloatEx revises AddExpr {
  Value eval() {
    if (…) super.eval();
    else … ;
  }
}
```


TOKYO INSTITUTE OF TECHNOLOGY
TOKYO TECH

TOKYO INSTITUTE OF TECHNOLOGY
TOKYO TECH

GluonJ: A within method

- Limited scope
  - A method may have a predicate.
  - Its method overriding is effective only when it is called from …

```
class VarDecl {
  Value init() {
    v = right.eval();
}}

    class AddExpr {
      Value eval() { … }
    }
```

```
class Log revises AddExpr {
  Value eval()
            within VarDecl.init() {
      …
}}
```

Contextual predicate dispatch

- GluonJ
  - Predicates refers to non-local contexts i.e. **within** who is a caller.
    - Currently only **within** is available.
  - to deal with crosscutting concerns

- Original predicate dispatch
  - Predicates refers to only local contexts such as arguments and receiver's fields
    - for unambiguity and exhaustiveness

GluonJ: requires

- Some revisers may modify the same.
  - Like mixin, they are linearized and applied one by one.
  - **requires** must be a total order

```
class FloatEx revises AddExpr {
  Value eval() { … }
}
```

```
class AddExpr {
  Value eval() { … }
}
```

```
class StringEx requires FloatEx
                          revises AddExpr {
  Value eval() {
    if (…) super.eval();
    else … ; }
}
```

GluonJ: using

- How to access revised methods/fields
  - Effects by a required reviser is visible.
  - A **using** declaration make them visible from classes.

```
class Printing revises AddExpr {
  void print() { … } }
```

```
class AddExpr {
  Value eval() { … }
}
```

```
using Printing;
class Printer {
  static void print(AddExpr e) {
      e.print();
  } }
```

Core Calculus of GluonJ

- An extension of Featherweight Java
  - to rigorously discuss modular compilation

- Syntax

```
CL ::= class C extends C using R { C f; M }
       | class R revises C using R { M }
L    ::= C | R
M    ::= C m(C ;x) { return e; } [ within L ]
e    ::= x | e.f | e.m(e) | new C(e) | e in L
v,w ::= new C(v)
```

S-REFL

$$\texttt{C} <: \texttt{C}$$

S-TRANS

$$\frac{\texttt{C} <: \texttt{D} \qquad \texttt{D} <: \texttt{E}}{\texttt{C} <: \texttt{E}}$$

S-EXTENDS

$$\frac{\texttt{C ext D}}{\texttt{C} <: \texttt{D}}$$

T-VAR

$$\texttt{L}; \Gamma \vdash \texttt{x} : \Gamma(\texttt{x})$$

T-FIELD

$$\frac{\texttt{L}; \Gamma \vdash \texttt{e}_0 : \texttt{C}_0 \qquad \mathit{fields}(\texttt{C}_0) = \bar{\texttt{C}}\ \bar{\texttt{f}}}{\texttt{L}; \Gamma \vdash \texttt{e}_0.\texttt{f}_i : \texttt{C}_i}$$

T-NEW

$$\frac{\mathit{fields}(\texttt{C}) = \bar{\texttt{D}}\ \bar{\texttt{f}} \quad \texttt{L}; \Gamma \vdash \bar{\texttt{e}} : \bar{\texttt{C}} \qquad \bar{\texttt{C}} <: \bar{\texttt{D}}}{\texttt{L}; \Gamma \vdash \texttt{new C}(\bar{\texttt{e}}) : \texttt{C}}$$

T-IN

$$\frac{\texttt{L}'; \Gamma \vdash \texttt{e} : \texttt{C}}{\texttt{L}; \Gamma \vdash \texttt{e in L}' : \texttt{C}}$$

T-INVK

$$\frac{\texttt{L using } \bar{\texttt{R}} \qquad \begin{array}{c}\texttt{L}; \Gamma \vdash \texttt{e}_0 : \texttt{C}_0 \qquad \texttt{L}; \Gamma \vdash \bar{\texttt{e}} : \bar{\texttt{C}} \\ \mathit{mtype}(\texttt{m}, \mathit{lowest}(\texttt{C}_0, \bar{\texttt{R}}), \bar{\texttt{R}}, \texttt{L}) = \bar{\texttt{D}} \rightarrow \texttt{C}\end{array} \qquad \bar{\texttt{C}} <: \bar{\texttt{D}}}{\texttt{L}; \Gamma \vdash \texttt{e}_0.\texttt{m}(\bar{\texttt{e}}) : \texttt{C}}$$

T-METHOD

$$\frac{\begin{array}{c}\texttt{C}; \bar{\texttt{x}} : \bar{\texttt{C}}, \texttt{this} : \texttt{C} \vdash \texttt{e}_0 : \texttt{E}_0 \qquad \texttt{E}_0 <: \texttt{C}_0 \\ \text{for any } \texttt{L}, \text{ if } \mathit{mtype}(\texttt{m}, \mathit{super}(\texttt{C}), \mathit{dom}(RT), \texttt{L}) = \bar{\texttt{D}} \rightarrow \texttt{D}_0, \text{ then } \bar{\texttt{C}} = \bar{\texttt{D}} \text{ and } \texttt{C}_0 = \texttt{D}_0\end{array}}{\texttt{C}_0\ \texttt{m}(\bar{\texttt{C}}\ \bar{\texttt{x}})\{\ \texttt{return e}_0;\ \}\ \texttt{OK IN C}}$$

T-METHODR

$$\frac{\begin{array}{c}\texttt{R rev D} \qquad \texttt{R}; \bar{\texttt{x}} : \bar{\texttt{C}}, \texttt{this} : \texttt{D} \vdash \texttt{e}_0 : \texttt{E}_0 \qquad \texttt{E}_0 <: \texttt{C}_0 \\ \text{for any } \texttt{L}, \text{ if } \mathit{mtype}(\texttt{m}, \mathit{super}(\texttt{R}), \mathit{dom}(RT), \texttt{L}) = \bar{\texttt{D}} \rightarrow \texttt{D}_0, \text{ then } \bar{\texttt{C}} = \bar{\texttt{D}} \text{ and } \texttt{C}_0 = \texttt{D}_0\end{array}}{\texttt{C}_0\ \texttt{m}(\bar{\texttt{C}}\ \bar{\texttt{x}})\{\ \texttt{return e}_0;\ \}\ [\texttt{within L}']\ \texttt{OK IN R}}$$

T-CLASS

$$\frac{\bar{\texttt{M}}\ \texttt{OK IN C}}{\texttt{class C extends D using } \bar{\texttt{R}}\ \{\ \bar{\texttt{C}}\ \bar{\texttt{f}};\ \bar{\texttt{M}}\ \}\ \texttt{OK}}$$

T-RCLASS

$$\frac{\bar{\texttt{M}}\ \texttt{OK IN R}}{\texttt{class R revises D using } \bar{\texttt{R}}\ \{\ \bar{\texttt{M}}\ \}\ \texttt{OK}}$$

TOKYO INSTITUTE OF TECHNOLOGY
TOKYO TECH

Type judgment for methods

- T-MethodR

$$\frac{\begin{array}{c}\texttt{R rev D} \qquad \texttt{R}; \overline{\texttt{x}} : \overline{\texttt{C}}, \texttt{this} : \texttt{D} \vdash \texttt{e}_0 : \texttt{E}_0 \qquad \texttt{E}_0 \;\texttt{<:}\; \texttt{C}_0 \\ \text{for any } \texttt{L}, \text{if } \mathit{mtype}(\texttt{m}, \mathit{next}(\texttt{R}), \mathit{dom(RT)}, \texttt{L}) = \overline{\texttt{D}} \rightarrow \texttt{D}_0, \\ \text{then } \overline{\texttt{C}} = \overline{\texttt{D}} \text{ and } \texttt{C}_0 = \texttt{D}_0\end{array}}{\texttt{C}_0\ \texttt{m}(\overline{\texttt{C}}\ \overline{\texttt{x}})\{\ \texttt{return e}_0;\ \}\ [\texttt{within L}']\ \texttt{OK IN R}} \quad \text{(T-MethodR)}$$

```
class BinExpr {}
```

```
class StringEx revises BinExpr {
  String eval() { … } }
```

```
class AddExpr ex. BinExpr {}
```

```
class FloatEx revises AddExpr {
  float eval() { … } }
```

Discussion

inherent in crosscutting concerns?

- Mostly modular type checking because of this global analysis:
  - TMETHODR requires all the revisers are known at compile time.
  - **requires** must be a total order on every target class.

TOKYO INSTITUTE OF TECHNOLOGY
TOKYO TECH

Implementation (1)

- 1st stage: translation
  - source code to Java bytecode
  - a reviser becomes a Java class.
  - Inserting type casts

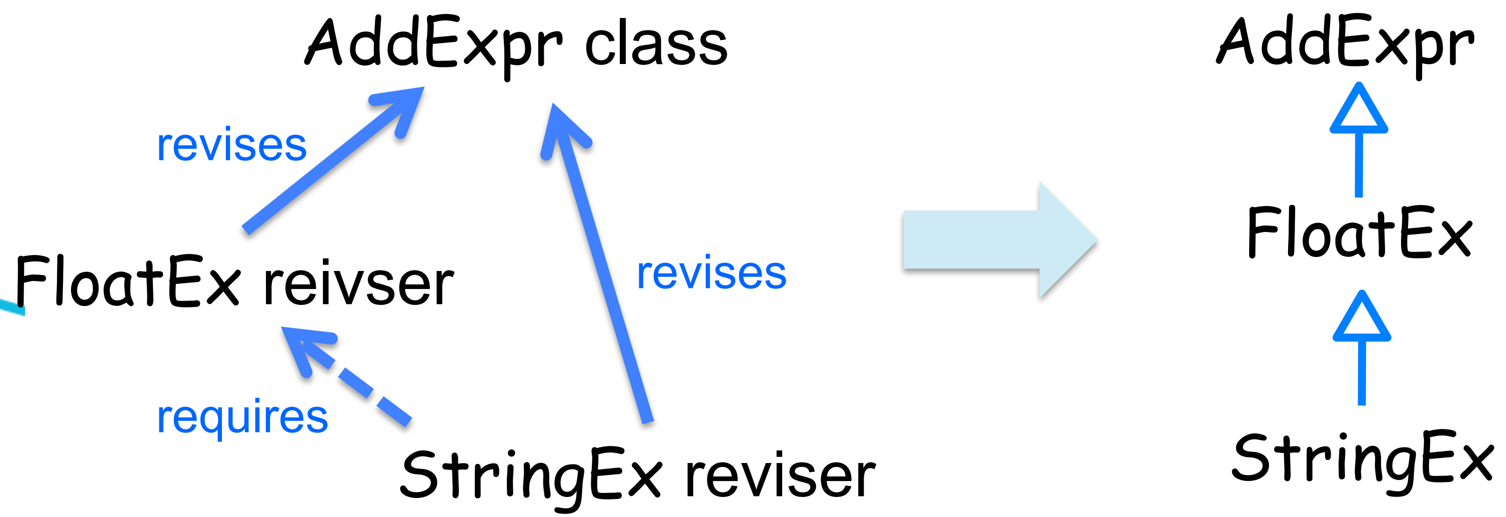
```
using Printing;
class Printer {
  static void print(AddExpr e) {
      e.print();
  } }
```

```
((Printing)e).print();
```

Implementation (2)

- 2nd stage: linking
  - Final type checking
  - A reviser inherits from its target class
    - Revisers targeting the same class are linearized.
    - Object creation is modified.

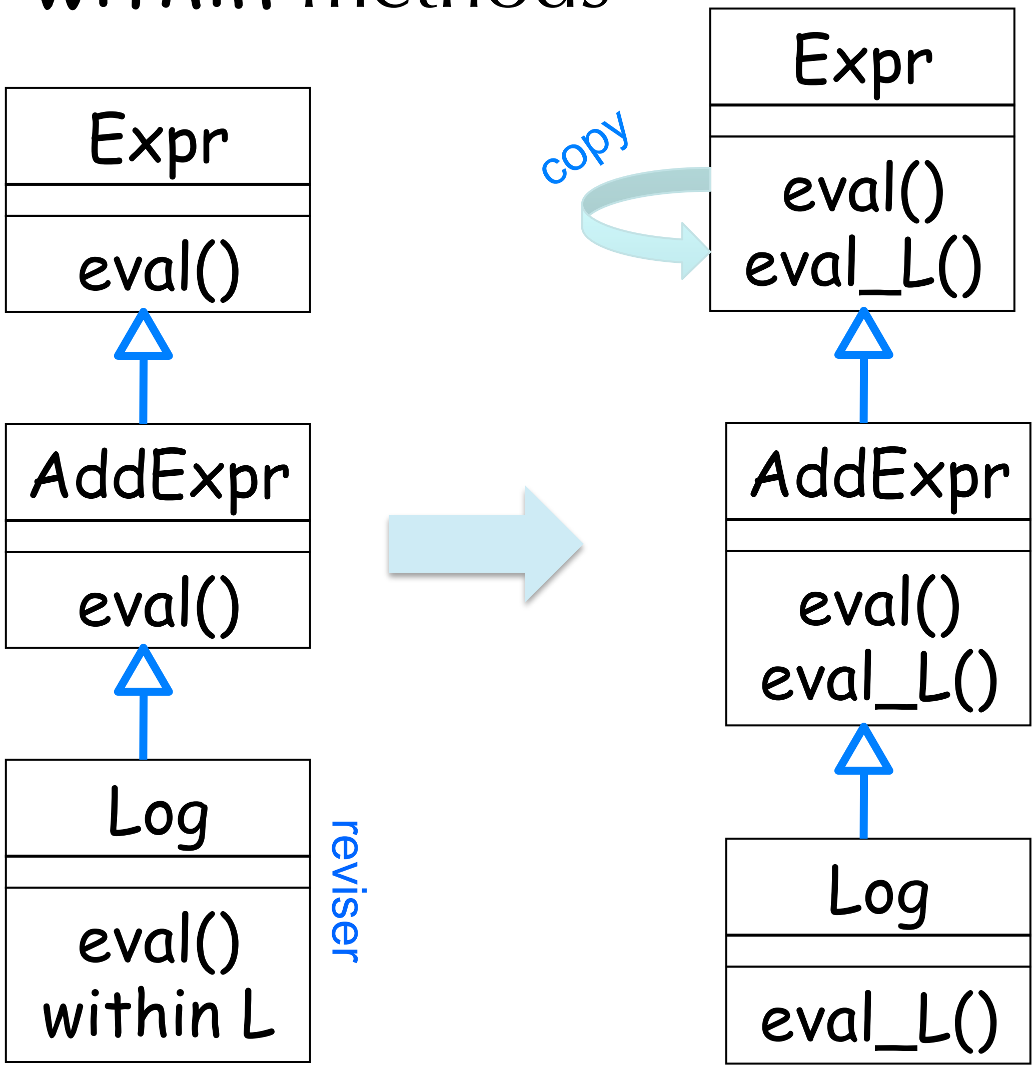
new AddExpr() → new StringEx()



TOKYO TECH

Implementation (3)

- within methods




```
class L { …
      v = right.eval();
 … }
```

```
v = right.eval_L();
```

Related work

- Predicate dispatch and AOP
  - D. Orleans pointed out in 2001
  - C. Bockisch [SPLAT'06], M. Haupt [ECOOP'07]
- Destructive extension
  - MixJuice, FOP, JavaGI, …
  - Open classes, expanders, …
- Limited scope
  - Classbox/J, COP, …

Not discussed AOP functionalities

- Pattern matching
  - A single advice may modify multiple methods matching a given pattern.
- dynamic pointcuts like `cflow`

Summary

- GluonJ
  - A simple AOP language with OOP-like constructs
    - Revisers and within methods
    - mostly modular   inherent in AOP?
- More in the paper
  - The type system is sound.
  - A formal model for compilation
    - GluonFJ to FJ translation
    - It preserves typing.
  - Benchmarking
    - Execution overheads are negligible.
  - http://www.javassist.org

TOKYO INSTITUTE OF TECHNOLOGY
TOKYO TECH