ContextFJ

A Core Calculus for Context-Oriented Programming

Atsushi Igarashi
(Kyoto Univ.)

Joint work with
Robert Hirschfeld (HPI)
Hidehiko Masuhara (Univ. Tokyo)
Context-oriented Programming (COP) [Hirschfeld, Costanza, Nierstrasz JOT08]

- **Goal:** modularizing behavioral variations depending on the dynamic context of execution
  - e.g., editor key binding depending on buffer modes

- Several COP extensions of existing (OOP) languages has been proposed
  - Java, Smalltalk, Common Lisp, JavaScript
This Work

First step towards a formal account of COP langs

- **ContextFJ calculus**
  - In the style of Featherweight Java [Igarashi et al.'99]
  - Direct operational semantics
    - c.f., encoding COP programs into other formalisms [Molderez et al. '10; Schippers et al. '08]

- (Very Simple) Type System for ContextFJ
  - Proof of Type Soundness
Plan of the Talk

- COP Language Constructs
- ContextFJ
  - Syntax
  - Operational Semantics
  - Simple Type System
- Future Work
COP Language Constructs

- Partial methods
  - Smallest unit to describe behavioral variations
  - Comparable to advice in AOP
- Layers
  - A bunch of partial methods
  - Unit of modularity/cross-cutting concerns
- Dynamically scoped layer (de)activation
  - with/without statements
Example: Personal data class

- Fields: name, address, employer

- Behavioral variations for `toString()`
  - “Name: ” + name;
  - “Name: ” + name + “; Addr: ” + address
  - “Name: ” + name + “; Affil: ” + employer
  - ...

class Person {
    String name, addr, employer;
    Person(String name, String addr, String employer) {
    }
    String toString() {
        return "Name: " + name;
    }
}

layer Contact {
    String toString() {
        return proceed() + "; Addr: " + addr;
    }
}

layer Employment {
    String toString() {
        return proceed() + "; Affl: " + employer;
    }
}

• Partial method(s) in one layer will be simultaneously activated
• There may be other partial methods defined inside another class

```
layer Contact {
    String toString() {
        return proceed() + "; Addr: " + addr;
    }
}

layer Employment {
    String toString() {
        return proceed() + "; Affl: " + employer;
    }
}
```
class Person {
    String name, addr, employer;

    Person(String name, String addr, String employer) {
        // constructor code
    }

    String toString() {
        return "Name: " + name;
    }
}

layer Contact {
    String toString() {
        return proceed() + "; Addr: " + addr;
    }
}

layer Employment {
    String toString() {
        return proceed() + "; Affl: " + employer;
    }
}

- Partial method(s) in one layer will be simultaneously activated
- There may be other partial methods defined inside another class

Partial method defs. for toString()

Call to the "original" behavior
Person me = new Person("Igarashi", "Kyoto", "Kyoto U.");

println(me.toString()); // "Name: Igarashi"

with (Contact) {
  println(me.toString());
  // "Name: Igarashi; Addr: Kyoto"
  f(me); // x.toString() in f(x) will result in
  // the same string as above
}

with (Employment) {
  println(me.toString());
  // "Name: Igarashi; Affl: Kyoto U."
}
Person me = new Person("Igarashi", "Kyoto", "Kyoto U.");

println(me.toString()); // "Name: Igarashi"

with (Contact) {
  println(me.toString());
  // "Name: Igarashi; Addr: Kyoto"
  f(me); // x.toString() in f(x) will result in
  // the same string as above
}

with (Employment) {
  println(me.toString());
  // "Name: Igarashi; Affl: Kyoto U."
}
• Layer precedence depends on activation order

with (Contact) {
    with (Employment) {
        println(me.toString());
        // “Name: Igarashi; Addr: Kyoto; Affl: Kyoto U.”
    }
}

with (Employment) {
    with (Contact) {
        println(me.toString());
        // “Name: Igarashi; Affl: Kyoto U.; Addr: Kyoto”
    }
}
How a COP program is organized

Base classes

C1
m1()
m2()

C2
m3()
m4()

C3
m5()
m6()

C4
m7()
m8()

Layer L1

C1
m1()

C2
m4()

C4
m8()

Layer L2

C2
m4()

C3
m5()

C4
m7()
with (L1) { ... }

Base classes

C1
m1()
m2()

C2
m3()
m4()

C3
m5()
m6()

C4
m7()
m8()

Layer L1

C1
m1()
m4()

C4
m8()
with (L1) { with (L2) { ... } }

Base classes

C1
m1()
m2()

C2
m3()
m4()

C3
m5()
m6()

C4
m7()
m8()

Layer L1

C1
m1()

C2
m4()

Layer L2

C2
m4()

C3
m5()

C4
m7()
with (L2) { with (L1) { ... } }

Base classes

C1
m1()
m2()

C2
m3()
m4()

C3
m5()
m6()

C4
m7()
m8()

Layer L1

C1
m1()

C2
m4()

Layer L2

C2
m4()

C3
m5()

C4
m7()
Plan of the Talk

• COP Language Constructs
• ContextFJ
  • Syntax
  • Operational Semantics
  • Simple Type System
• Future Work
ContextFJ

- ContextFJ =

Featherweight Java [Igarashi,Pierce,Wadler'99]
+ Partial methods
+ proceed(), super()
+ with/without expressions
Syntax (1/2)

\[
\begin{align*}
\text{CL} & ::= \text{class } C < D \{ \sim C \sim f; \sim M \} \\
\text{M} & ::= C \ m(\sim C \sim x)\{ \text{return } e; \} \\
\text{e} & ::= x \mid e.f \mid e.m(\sim e) \mid \text{new } C(\sim e) \\
& \quad | \ \text{with } L \ e \quad \text{layer activation} \\
& \quad | \ \text{without } L \ e \quad \text{layer deactivation} \\
& \quad | \ \text{proceed}(\sim e) \quad \text{proceed call} \\
& \quad | \ \text{super.m}(\sim e) \quad \text{super call}
\end{align*}
\]
Syntax (2/2)

ContextFJ program: \((CT, PT, e)\)

- **Class table:** \(CT(C) = CL\)
- **Partial method table:** \(PT(m, C, L) = M\)
- **Main expression:** \(e\)
Operational Semantics

FJ

- Lookup function: $mbody(m,C) = \sim x.e$
- Reduction relation: $e \rightarrow e'$

ContextFJ

- Lookup function: $mbody(m,C,\sim L1,\sim L2) = \sim x.e \text{ in } D,\sim L3$
- Reduction relation: $\sim L \vdash e \rightarrow e'$
Lookup function: \textit{mbody}

\textit{mbody}(m, C, \sim L_1, \sim L_2) = \sim x.e \text{ in D, } \sim L_3

- “Body of method m in C is e with params \sim x”
- \sim L_2 is the list of activated layers
- C, \sim L_1 denote the currently focused position
- D, \sim L_3 denote where \sim x.e is found
mbody(m, C3, (L1;L2), (L1;L2))
= mbody(m, C3, L1, (L1;L2))
= mbody(m, C3, ·, (L1;L2))
= mbody(m, C2, (L1;L2), (L1;L2))
= mbody(m, C2, L1, (L1;L2))
= x.e in C2, L1
\[ PT(m, C, L0) = C0 \ m(\sim C \ \sim x)\{ \text{return e; } \} \]

\[ mbbody(m, C, (\sim L1; L0), \sim L2) = \sim x.e \text{ in C, (\sim L1; L0)} \]

\[ PT(m, C, L0) \text{ undefined} \]
\[ mbbody(m, C, \sim L1, \sim L2) = \sim x.e \text{ in D, } \sim L3 \]

\[ mbbody(m, C, (\sim L1; L0), \sim L2) = \sim x.e \text{ in D, } \sim L3 \]
• \textit{mbody}(\texttt{toString}, \texttt{Person}, \cdot, \cdot) = \\
\hspace{1em}(\cdot.("\text{Name: }" + \text{this.name}) \text{ in Person}, \cdot)

• \textit{mbody}(\texttt{toString}, \texttt{Person}, \texttt{Contact}, \texttt{Contact}) = \\
\hspace{1em}(\cdot.\text{proceed}() + "\text{Addr: }" + \text{this.addr}) \\
\hspace{2em}\text{in Person, Contact}
Reduction: \( \sim L \vdash e \rightarrow e' \)

- “e reduces to e' in one step under activated layers \( \sim L \)”

- e.g.,
  - \( \vdash \) new Person(...).toString()
    \( \rightarrow \) “Name: “ + new Person(...).name
  - Contact \( \vdash \) new Person(...).toString()
    \( \rightarrow \) proceed() + “Affl: “ + new Person(...).addr
  - … actually, not quite correct! (Wait for a few slides!)
Reduction rule for layer activation

\[
\text{remove}(L, \sim L) = \sim L' \quad \sim L';L \vdash e \rightarrow e'
\]

\[
\sim L \vdash \text{with } L \ e \rightarrow \text{with } L \ e'
\]

- Activated layer \( L \) always comes at the top
  - Even when it's already been activated

- \textit{e.g.,}
  \[
  \vdash \text{with Contact (new Person(...)\text{.toString()})}
  \rightarrow \text{with Contact (}
  \text{proceed()} + \text{"Affil: " + new Person(...)\text{.addr}}
  \)\]
Run-time expression to deal with proceed and super

\[ e ::= \ldots | \text{new } C(\sim e)<D,\sim L_1,\sim L_2> \]

- Essentially new \( C(\sim e) \)
- Annotation \( <D,\sim L_1,\sim L_2> \) remembers
  - where method lookup starts next time \( (D, \sim L_1) \)
  - what layers have been activated \( (\sim L_2) \)
- Contact \( \vdash \text{new } \text{Person}(\ldots).\text{toString}() \)
  \[ \rightarrow \text{new } \text{Person}<\text{Person}, \cdot, \text{Contact}>(\).\text{toString}() \]
  \[ + \text{“Affl: “} + \text{new } \text{Person}(\ldots).\text{addr} \]
Reduction Rules for Method Invocation

\[ \sim L \vdash \text{new } C(\sim v) < C, \sim L, \sim L > . m(\sim w) \rightarrow e' \]

\[ \sim L \vdash \text{new } C(\sim v) . m(\sim w) \rightarrow e' \]

\[ mbody(m, D, \sim L1, \sim L2) = \sim x. e \text{ in } E, (\sim L3, L) \]

\[ \text{class } E < F \]

\[ \sim L4 \vdash \text{new } C(\sim v) < D, \sim L1, \sim L2 > . m(\sim w) \rightarrow \]

\[ [\text{new } C(\sim v) / \text{this,} \]

\[ \sim w / \sim x, \]

\[ \text{new } C(\sim v) < E, \sim L3, \sim L2 > . m / \text{proceed,} \]

\[ \text{new } C(\sim v) < F, \sim L2, \sim L2 > / \text{super } ] \ e \]
Reduction Rules for Method Invocation

~L ├ new C(~v)<C,~L,~L>.m(~w) → e'

~L ├ new C(~v).m(~w) → e'

Invocation on an “unannotated” object is affected by currently activated layers ~L

[new C(~v)/ this,
  ~w / ~x,
  new C(~v)<E,~L3,~L2>.m / proceed,
  new C(~v)<F,~L2,~L2> / super ] e
Reduction Rules

Self calls will be affected by with/without in e but super/proceed calls won't

\[ \sim L \vdash \text{new } C(\sim v).m(\sim w) \rightarrow e' \]

\[ mbody(m, D, \sim L1, \sim L2) = \sim x.e \text{ in } E, (\sim L3, L) \]
class E < F

\[ \sim L4 \vdash \text{new } C(\overline{\sim v})<D,\sim L1,\sim L2>.m(\sim w) \rightarrow \]
\[ [\text{new } C(\sim v)/ \text{this}, \]
\[ \sim w \quad / \sim x, \]
\[ \text{new } C(\sim v)<E, \sim L3, \sim L2>.m \quad / \text{proceed}, \]
\[ \text{new } C(\sim v)<F, \sim L2, \sim L2> \quad / \text{super} \quad ] \quad e \]
\[ m(x) \] 

\[
\begin{align*}
\text{return } e;
\end{align*}
\]
Type System for ContextFJ

- Main problem: ensure proceed() to succeed
  - Non-trivial as layer configuration changes dynamically!
- A simple (but restrictive) answer: every partial method has to override one in a base class
  - rather than to introduce new behavior

⇒ Mostly the same type system as FJ!
  - Covariant return type overriding only for base methods
  - Type Soundness by Preservation + Progress
Summary

• Language Constructs for COP
  • Partial methods in layers
  • Layer (de)activation
• ContextFJ for direct account of COP programs
  • Operational semantics
  • Simple and sound type system
Summary

“We can talk about COP languages at Starbucks, even without Mac!” – Hirschfeld

- Layer (de)activation
- ContextFJ for direct account of COP programs
  - Operational semantics
  - Simple and sound type system
Future work

• Sophisticated type system to allow partial methods to *introduce* new behavior
  • c.f., FOP type systems

• Formal accounts of advanced COP features
  • Stateful layers
  • First-class layers