COP With Only Layers

Robert Hirschfeld, Hidehiko Masuhara, Atsushi Igarashi

Hasso-Plattner-Institut Potsdam
Tokyo Institute of Technology
Kyoto University

2013-05-15
Outline

• COP
  – Basics
  – Layer activation
  – Symmetric composition

• L
  – Symmetric composition
  – Examples
  – Bootstrapping runtime

• Related work

• Ongoing and future work
  – …
COP Basics

- **Behavioral variations**
  - Partial class and method definitions
- **Layers**
  - Groups of related context-dependent behavioral variations
- **Activation**
  - Activation and deactivation of layers at runtime
- **Context**
  - Anything computationally accessible
- **Scoping**
  - Well-defined explicitly-controlled scopes
Dynamically-scoped Layer Activation

- Constructs
  
  \[
  \text{with \ (\ldots) \ \{\ldots\} / without \ (\ldots) \ \{\ldots\}}
  \]

- Activate (deactivate) layers for the \textit{current thread}
  
  - Does not interfere with other layer activations/deactivations in other threads

- Layers are activated/deactivated only for the \textit{dynamic extent} of the associated code block

- Activation order determines method precedence
COP Extensions

- ContextS → Squeak/Smalltalk
- ContextJS → Lively/JavaScript
- ContextFJ → Featherweight Java
- … (many more)

- Incremental
  - Mainly partial method definitions
    → composed into classes/objects

- Practical
  - Existing code artifacts such as frameworks and libraries
    written in original base language
    → usable in COP systems
    → can be adapted by COP systems
Asymmetric Layer Composition

- Base system implemented by classes/objects
- A set of layers to provide behavioral variations
- Class composition
  → static
  → at compile-time
  * dispatching
- Layer composition
  → dynamic
  → at run-time
  * dispatching
L: Symmetric Layer Composition

- Experimental COP language
- Based on ContextFJ
- To explore the design space for COP languages

- Only layers, no classes → unification of both
- Remove asymmetry
L₁: Example 1 – Objects and Layers

```java
object Person {
    String name, residence, employer;
    Person(String _name,
            String _residence,
            String _employer) {
        name = _name;
        residence = _residence;
        employer = _employer;
    }
}

object Student extends Person {
    String major;
    Person(String _name,
            String _residence,
            String _employer,
            String _major) {
        super(_name, _residence, _employer);
        major = _major;
    }
}

layer LPerson {
    object Person {
        String toString() {
            return "Name: " + name;
        }
    }
}

layer LContact {
    object Person {
        String toString() {
            return proceed() + "; Residence: " + residence;
        }
    }
}

layer LEmployment {
    object Person {
        String toString() {
            return proceed() + "; Affil.: " + employer;
        }
    }
}
```
Person atsushi =
   new Person("Atsushi", "Kyoto", "Kyodai");
new Printer().println(atsushi);
→ "an Instance"

with LPerson {
   new Printer().println(atsushi);
}
→ "Name: Atsushi"

with LPerson {
   with LContact {
      new Printer().println(atsushi);
   }
}
→ "Name: Atsushi; Residence: Kyoto"

with LPerson {
   with LEmployment {
      with LContact {
         new Printer().println(atsushi);
      }
   }
}
→ "Name: Atsushi; Affil.: Kyodai; Residence: Kyodai"
L2: Example 2 – Objects and Layers

layer LCustomer {
  object Customer {
    String name;
    void setName(String _name) {
      name = _name;
    }
  }
}

layer LConnection {
  object Connection {
    Customer from, to; // <1>
    void setFrom(Customer _from) {
      from = _from;
    }
    void setTo(Customer _to) {
      to = _to;
    }
    void complete() { ... }
    void drop() { ... }
  }
}

layer LTiming {
  object Timer {
    void start() { ... }
    void stop() { ... }
    int getTime() { ... }
  }
  object Connection {
    Timer timer; // <2>
    void setTimer(Timer _t) {
      timer = _t;
    }
    void complete() {
      proceed();
      timer.start();
    }
    void drop() {
      timer.stop();
      proceed();
    }
    int getTime() {
      return timer.getTime();
    }
  }
}

layer LBilling {
  // requires LTiming
  object Connection {
    Timer timer; // <3>
    void setTimer(Timer _t) {
      timer = _t;
    }
    void charge() {
      // cost = ...getTime()...;
      // ...charge cost on caller...
    }
    void drop() {
      proceed();
      charge();
    }
  }
}
L₂: Example 2: Transcript

// for convenience, available from somewhere
Connection simulate() {
    Customer a = new Customer();
a.setName("Atsushi");
Customer h = new Customer();
h.setName("Robert");
Connection c = new Connection();
c.setFrom(a);
c.setTo(h);
c.complete();
c.drop();
return c;
}

with LTiming {
    with LBilling {
        Connection c = simulate();
        new Printer().printLn(c.getTime());
    }
} → "712" // seconds
layer LCommon {
  object Base {
    String toString () {
      return "an Instance";
    }
  }
}

layer LCommon {
  object Printer {
    void println(Base o) {
      // PRIMITIVE_println(o.toString());
    }
  }
}

with LCommon {
  // [[read-eval-print|main]]
}
Related Work

• Hyper/J
• Early AspectJ
• Smalltalk
• Self
• Newspeak
• …
### AOP, FOP, and COP

<table>
<thead>
<tr>
<th></th>
<th>AOP</th>
<th>FOP</th>
<th>COP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverse dependencies</td>
<td>😄</td>
<td>📍</td>
<td>📍</td>
</tr>
<tr>
<td>1:n relationships</td>
<td>😄</td>
<td>😄</td>
<td>😄</td>
</tr>
<tr>
<td>Layers</td>
<td>📍</td>
<td>📍</td>
<td>📍</td>
</tr>
<tr>
<td>Dynamic activation</td>
<td>😄</td>
<td>😄</td>
<td>😄</td>
</tr>
<tr>
<td>Scoping</td>
<td>😄</td>
<td>😄</td>
<td>😄</td>
</tr>
</tbody>
</table>
L: COP With Only Layers

- Unification of classes and layers
- Simplification of core model
- Practical…