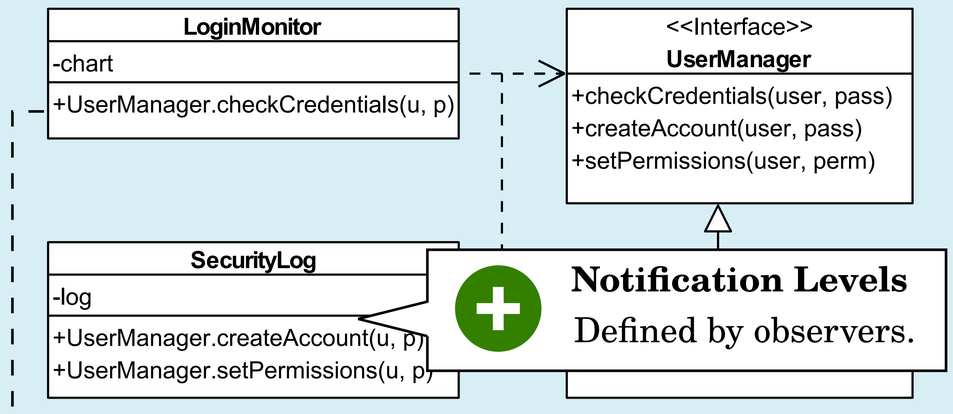
Classes as Layers: Rewriting Design Patterns with COP

Matthias Springer[†], Hidehiko Masuhara[†], Robert Hirschfeld[‡]
([†]Tokyo Institute of Technology; [‡]Hasso Plattner Institute, University of Potsdam)

Context-oriented Programming

- COP is a technique for modularizing cross-cutting concerns that allows for dynamic adaptation at runtime
- *Partial Methods* defined in *Layers* can add/modify behavior in other classes and be combined (layering)
- **Our approach:** Classes act as Layers
- **This poster:** Rewrite design patterns with COP to overcome shortcomings compared to trad. implementations

Observer Design Pattern




```
class LoginMonitor {
  def UserManager.checkCredentials(u, p) {
    def result = super(u, p);
    if (result) { /* ... */ } else { /* ... */ }
    return result;
  }
}
```

Notification Trigger Granularity
Observers can listen to only method invocations (not inside a method).

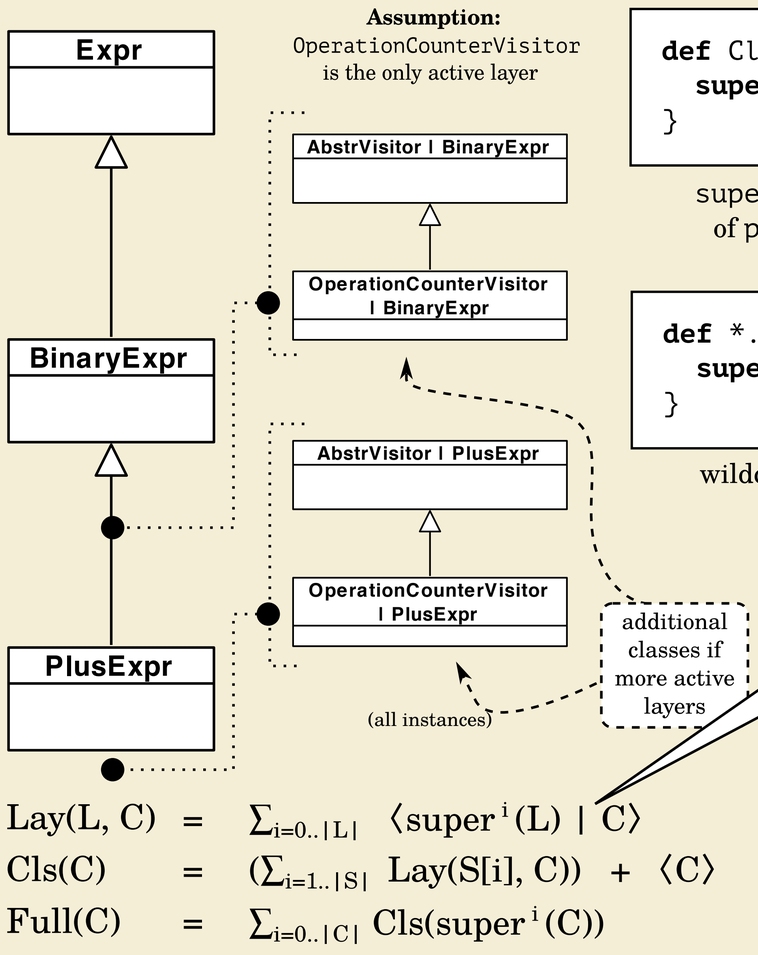
```
def loginMonitor = new LoginMonitor();

// per-object activation
manager.activate(loginMonitor);

// global activation
loginMonitor.activate(
```

Group Observation
Observe all instances of a class/interface.

Method Lookup



L | C: projection of L by C
Contains only methods in L targeting C.

$$Lay(L, C) = \sum_{i=0..|L|} \langle super^i(L) \mid C \rangle$$
$$Cls(C) = (\sum_{i=1..|S|} Lay(S[i], C)) + \langle C \rangle$$
$$Full(C) = \sum_{i=0..|C|} Cls(super^i(C))$$

|C| number of superclasses of C
S layer composition stack (S[i] is i-th element)
<C> list with only C

Language Features

```
def Cls.foo() {
  super();
}
```
super instead of proceed

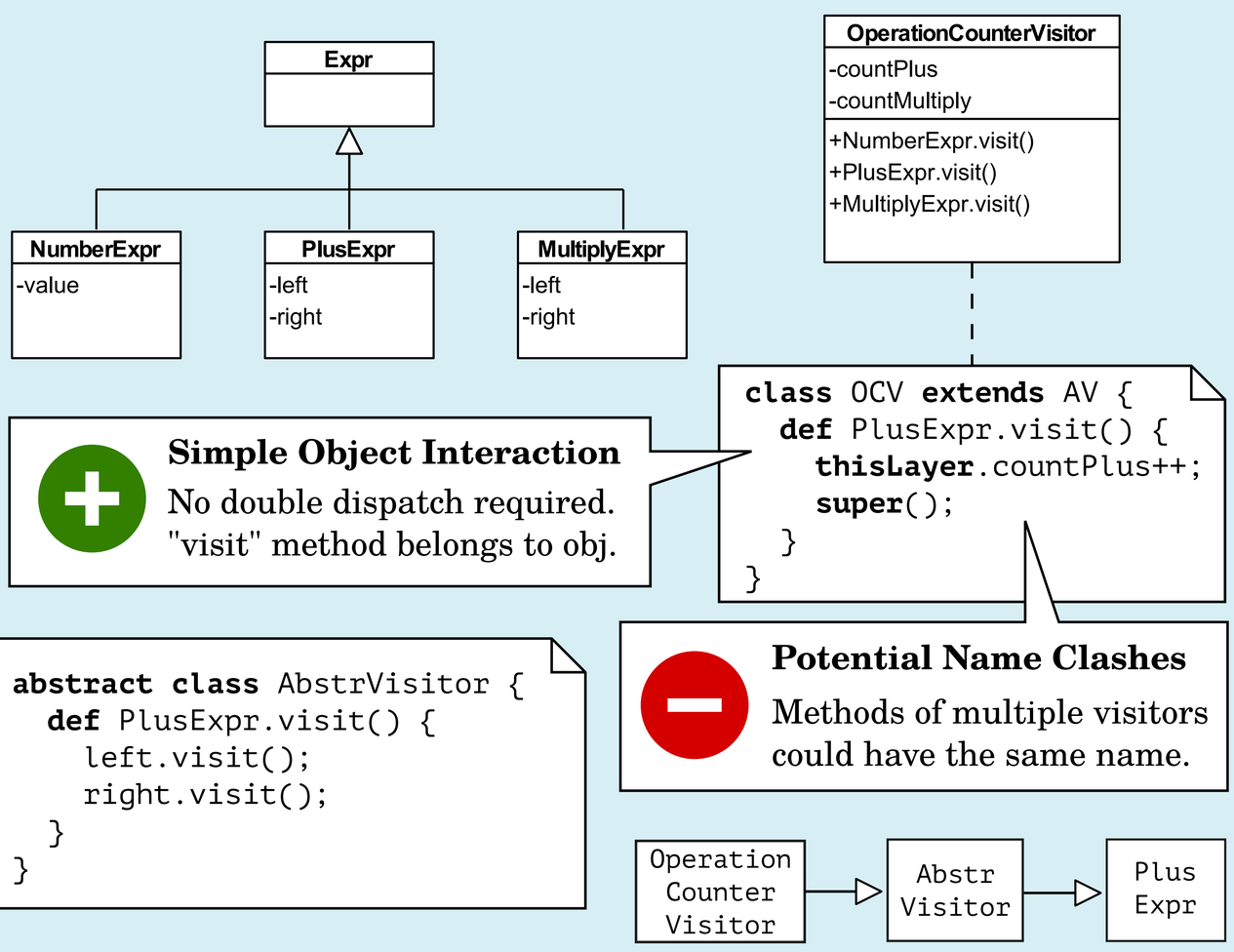
```
def Cls.foo() {
  this.bar;
}
```
affected object identifier

```
def *.foo() {
  super();
}
```
wildcard class

```
def Cls.foo() {
  thisLayer.bar;
}
```
layer object identifier

Dynamic Adaptation
Design patterns can be added at any time.

Visitor Design Pattern



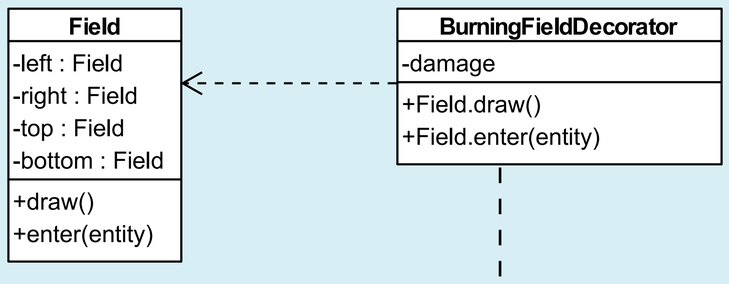
Simple Object Interaction
No double dispatch required.
"visit" method belongs to obj.

```
class OCV extends AV {
  def PlusExpr.visit() {
    thisLayer.countPlus++;
    super();
  }
}
```

Potential Name Clashes
Methods of multiple visitors could have the same name.

```
abstract class AbstrVisitor {
  def PlusExpr.visit() {
    left.visit();
    right.visit();
  }
}
```


Decorator Design Pattern



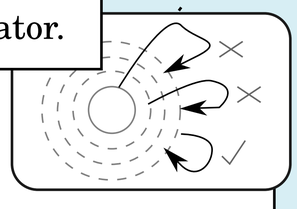

```
class BurningFieldDecorator {
  def Field.enter(entity) {
    entity.health -= thisLayer.damage;
    super(entity);
  }
}
```

Global Visibility
Internal method calls are also affected.

```
def field = new Field();
field.activate(new BurningFieldDecorator());

// global activation
new BurningFieldDecorator().activate();
```

No Object Schizophrenia
Object identity is preserved when applying a decorator.



```
class BurningSomethingDecorator {
  def *.enter(entity) {
    /* ... */
  }
}
```