

An Overview of AspectJ

Gregor Kiczales, **UBC**

Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, **Xerox PARC**

William G. Griswold, **UCSD**

*talk will focus on key semantic elements
of the language*

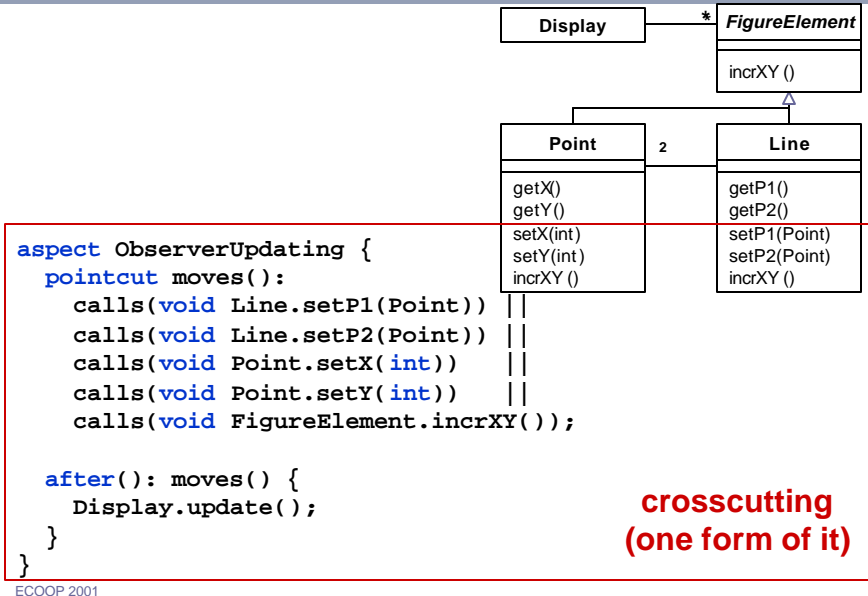
ECOOP 2001

AspectJ is...

- **a general-purpose AO language**
 - just as Java is a general-purpose OO language
 - unlike examples in ECOOP'97 talk
- **an integrated extension to Java**
 - outputs .class files compatible with any JVM
 - accepts all java programs as input
 - integrated with tools
 - programming feels like Java
- **enables two kinds of crosscutting**
 - static (introduction) – similar to, but weaker than HyperJ
 - dynamic – focus of today's talk

ECOOP 2001

crosscutting in AspectJ



three key language elements

- join point (JP) model
- means of identifying JPs
- means of specifying behavior at JPs

three key language elements

- **join point (JP) model**
 - the possible “wheres”
- **means of identifying JPs**
 - picking out join points of interest
- **means of specifying behavior at JPs**
 - what happens

ECOOP 2001

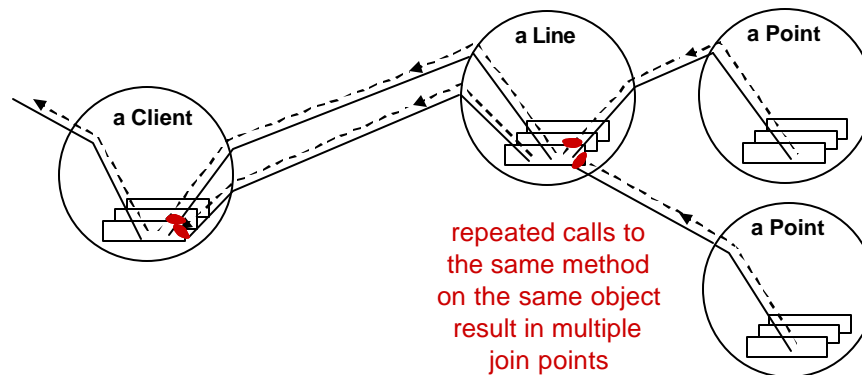
three key language elements

- **join point (JP) model**
 - the possible “wheres”
 - certain principled points in program execution
- **means of identifying JPs**
 - picking out join points of interest
 - pointcut designators
- **means of specifying behavior at JPs**
 - what happens
 - advice declarations

ECOOP 2001

dynamic join points

key points in dynamic control graph



ECOOP 2001

pointcuts

- filters picking out join points
- primitive and user-definable
- composable with `&&`, `||`, `!`
- calls pointcut matches call join points by signature

```
pointcut moves():
  calls(void Line.setP1(Point)) ||
  calls(void Line.setP2(Point)) ||
  calls(void Point.setX(int)) ||
  calls(void Point.setY(int)) ||
  calls(void FigureElement.incrXY());
```

ECOOP 2001

advice

- code body attached to a pointcut
- code runs at every join point picked out by a pointcut
- after advice runs after the computation of a join point
 - both returning normally and throwing an exception

```
after(): moves() {  
    Display.update();  
}
```

ECOOP 2001

three key language elements

Join Points

- method call
- object creation (call to new)
- field get/set

- exception handler execution
- method execution
- class-specific initialization
- static initializer execution

Pointcuts

- calls
- gets
- sets
- instanceof
- cflow, cflowsub
- user-defined

- executions
- handlers
- initializations
- staticinitializations
- within
- withincode
- callsto

Advice

- before
- after
- after throwing
- after finally
- around

ECOOP 2001

field assignment join point

- captures assignments to fields
- picked out by the “sets” pointcut by signature
- example: postconditions

```
after(): sets(int Point.x) {  
    if (Point.x > 100)  
        throw new PointOutOfBoundsException();  
}
```

ECOOP 2001

instanceof pointcut

- picks out all kinds of join points based on a dynamic property
 - if the currently executing object is an instance of a particular type
- example: noting potentially unsafe calls

```
after(): instanceof(FigureElement) &&  
    calls(void Display.update()) {  
    Display.checkInvariants();  
}
```

ECOOP 2001

around advice

- allows fine-grained control of execution
- runs in place of join point
- “proceed” call in the body proceeds with original join point computation
- example: selective execution

```
around(): calls(Display.update()): {  
    if (! Display.disabled())  
        proceed();  
}
```

ECOOP 2001

orthogonality

- any join point
- works with any pointcut
- works with any advice
- special-purpose constructs aren't special

ECOOP 2001

language elements

Join Points

- method call
- object creation (call to new)
- field get/set

- exception handler execution
- method execution
- class-specific initialization
- static initializer execution

Pointcuts

- calls
- gets
- sets
- instanceof
- cflow, cflowsub
- user-defined

- executions
- handlers
- initializations
- staticinitializations
- within
- withincode
- callsto

Advice

- before
- after
- after throwing
- after finally
- around

ECOOP 2001

constructor bodies

- orthogonality informs design

```
Point(int x, int y) {  
    // super();  
    this.x = x; this.y = y;  
}
```

- capturing entry and exit not possible around constructor bodies.
- no before/around advice possible...
- so it should not be a join point

ECOOP 2001

fine-grained access control

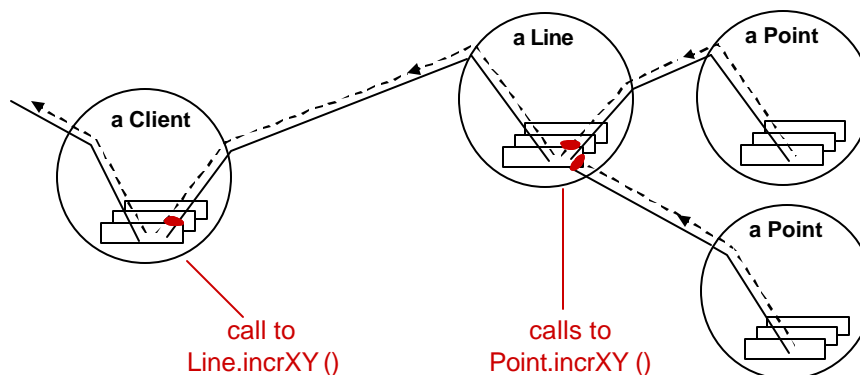
- only setter methods can modify fields

```
before(): (sets(int Point.x)) &&  
          !withincode(void Point.setX()) ||  
          (sets(int Point.y) &&  
           !withincode(void Point.setY())) {  
    throw new AccessDeniedException();  
}
```

- special-purpose withincode fits the model

ECOOP 2001

control flow



ECOOP 2001

control flow

```
aspect MoveCounting {
  int count = 0;
  pointcut moves():
    ...as before ...

  after(): moves() && !cflowSub(moves()) {
    count++;
  }
}
```

- **more interesting flow relationships possible**

ECOOP 2001

conclusions

- **dynamic join point model**
- **orthogonal join points, pointcuts, advice**
- **open source compiler**
- ...

ECOOP 2001

AspectJ is...

- **a general-purpose AO language**
- **an integrated extension to Java**
- **has static and dynamic crosscutting**
 - looked at kernel of dynamic crosscutting
 - skipped: values at join points, aspects, reuse, reflection
static crosscutting, compiler implementation...
- **freely available implementation**
 - compiler & tools are Open Source, MPL
- **includes IDE support**
 - emacs, JBuilder, Forte 4J
- **currently at 0.8 release**
 - 1.0 planned for September 2001