Aspect-Oriented Programming with AspectJ™

AspectJ.org
PARC

Erik Hilsdale, Jim Hugunin, Wes Isberg,
Gregor Kiczales, Mik Kersten

partially funded by DARPA under contract F30602-97-C0246

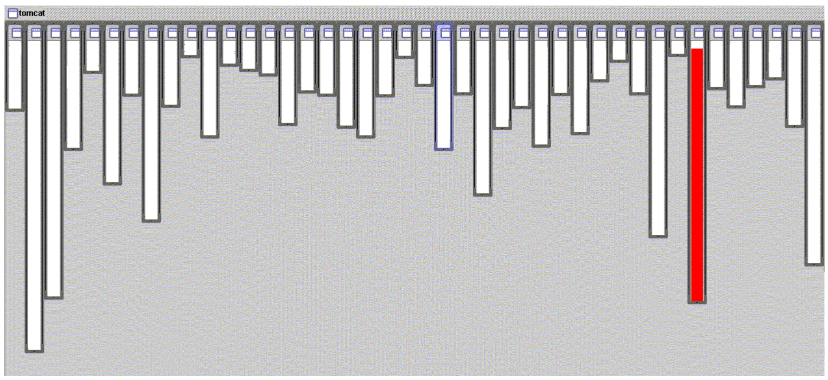
outline

- **I AOP and AspectJ overview**
  - problems, basic concepts, context, adoption
- **II AspectJ tutorial**
  - first example
  - language mechanisms
  - development environment
  - using aspects
- **III conclusion**
  - futures, references, summary

2

good modularity

XML parsing



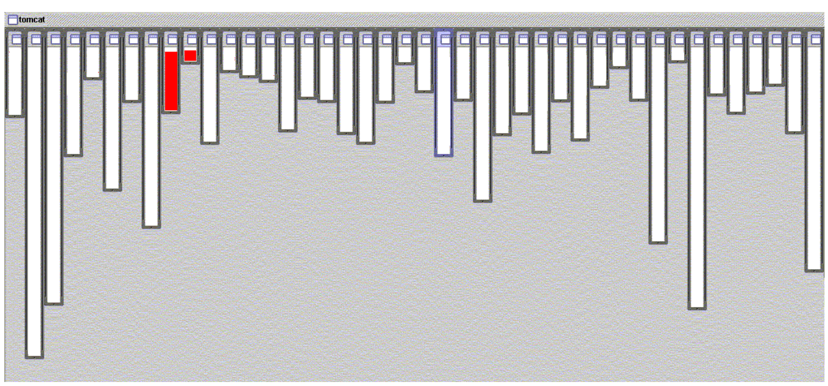
- **XML parsing in org.apache.tomcat**
  - red shows relevant lines of code
  - nicely fits in one box

aspectj.org

3

good modularity

URL pattern matching



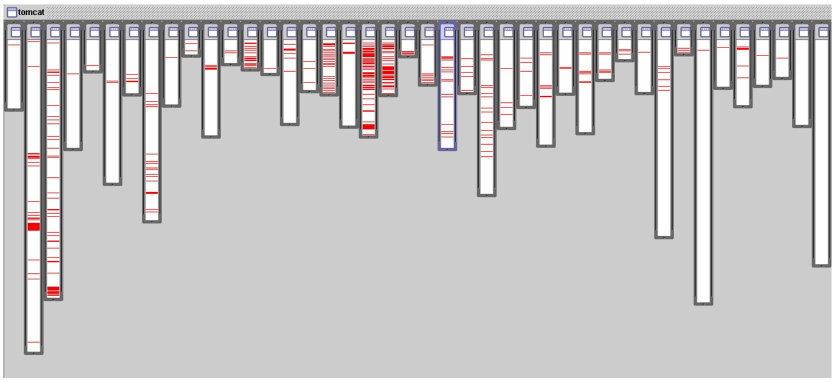
- **URL pattern matching in org.apache.tomcat**
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

aspectj.org

4

problems like…

logging is not modularized

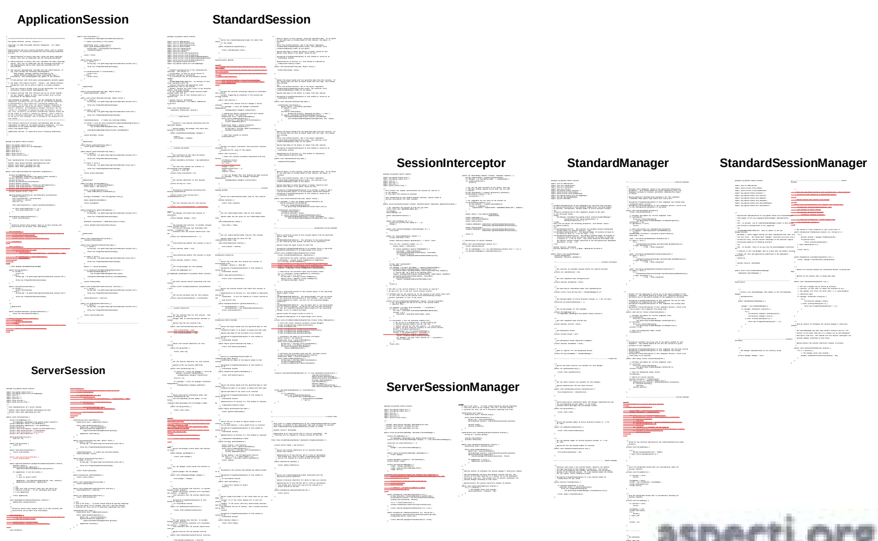


- **where is logging in org.apache.tomcat**
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

aspectj.org

problems like…

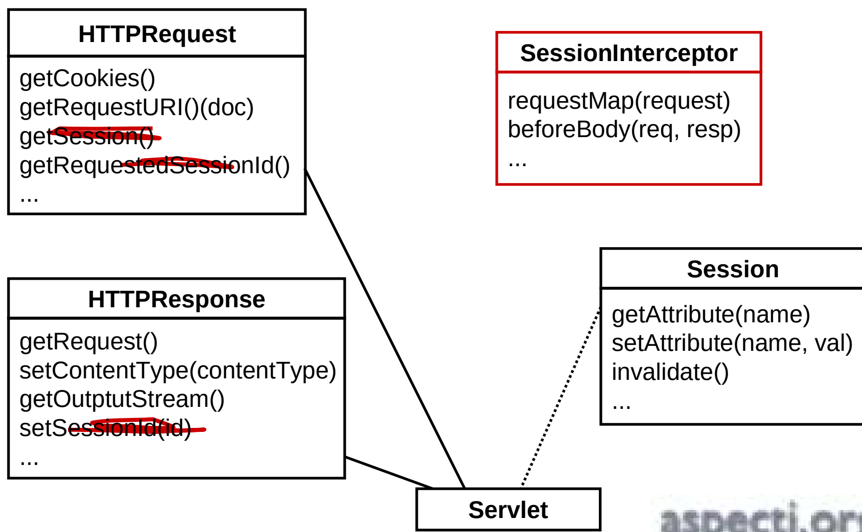
session expiration is not modularized



aspectj.org

problems like…

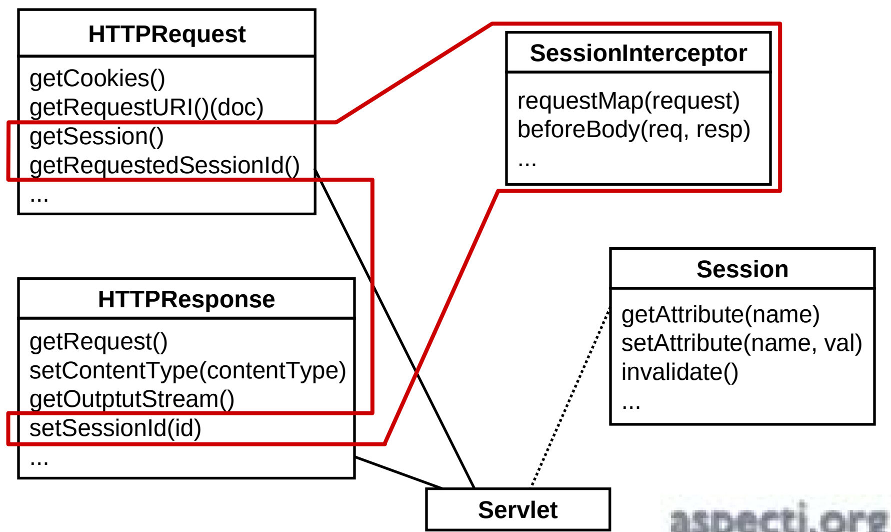
session tracking is not modularized



the cost of tangled code

- **redundant code**
  - same fragment of code in many places
- **difficult to reason about**
  - non-explicit structure
  - the big picture of the tangling isn't clear
- **difficult to change**
  - have to find all the code involved
  - and be sure to change it consistently
  - and be sure not to break it by accident

crosscutting concerns



9

the AOP idea

aspect-oriented programming

- **crosscutting is inherent in complex systems**
- **crosscutting concerns**
  - have a clear purpose
  - have a natural structure
    - defined set of methods, module boundary crossings, points of resource utilization, lines of dataflow…
- **so, let’s capture the structure of crosscutting concerns explicitly...**
  - in a modular way
  - with linguistic and tool support
- **aspects are**
  - well-modularized crosscutting concerns
- **Aspect-Oriented Software Development: AO support throughout lifecycle**

10

this tutorial is about...

- **using AOP and AspectJ to:**
  - improve the modularity of crosscutting concerns
    - design modularity
    - source code modularity
    - development process
- **aspects are two things:**
  - concerns that crosscut [design level]
  - a programming construct [implementation level]
    - enables crosscutting concerns to be captured in modular units
- **AspectJ is:**
  - an aspect-oriented extension to Java™ that supports general-purpose aspect-oriented programming

aspectj.org

11

language support to…

aspectj.org

12

the context: IT trends

Complex New Technologies

Mobile Computing

J2EE

Components

Web Services

XML

P2P

.NET

Applications are becoming more complex and more modular
– DRW

Increasing Pressures

Variability

Historically, enterprises have had to update these complex relationships by hard-coding single-use applications and relationships - Gartner

Quality

Requirements for quality have never been higher - Giga

Integration

This is a crucial need for almost all corporations – Giga

Agility

Agile application architecture is a critical element of an effective strategy to deal with continuing innovation - Giga

aspectj.org

13

the unsolved problem

- **Each new technology solves specific problems**
- **But how do you glue it together with flexibility while supporting variability?**
  - 10% of code causes 90% of problems
- **Structured → Objects → Components → Aspects**
  - each offers an additional kind of "modularity technology"
  - each enables significantly more complex software
- **Alternatives (many are ad hoc AOP, lacking flexibility, leverage, generality, explicit structure)**
  - EJB, servlet deployment descriptors, other XML languages
  - interceptors, proxies, specialized design patterns (command...)
  - code generation/wizards
  - instrumentation (profilers, coverage tools, …)
  - preprocessors (jContract, etc.)
  - meta-object programming: too complex

aspectj.org

14

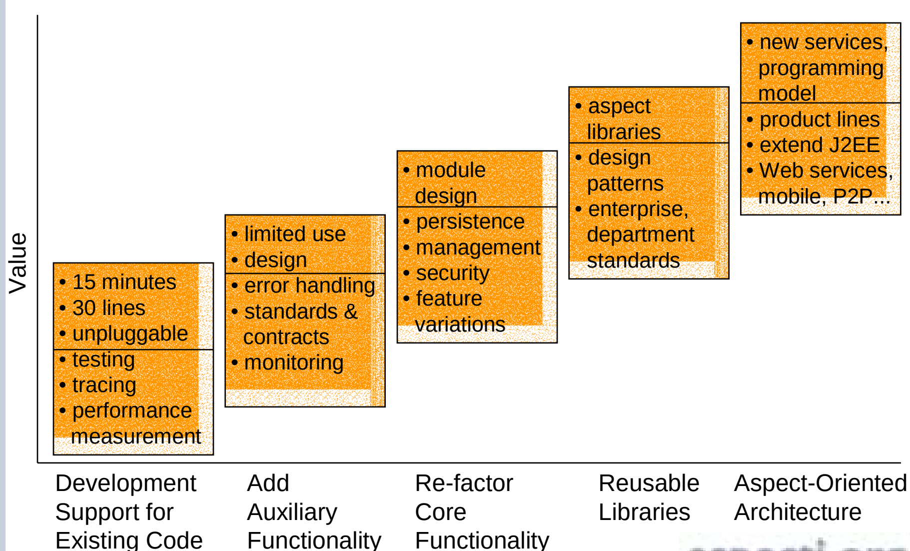
Aspect-Oriented Programming

- **an idea whose time has come**
  - enables significantly more complex and flexible software
  - improve quality with consistent policies
  - manage complexity and variability
  - leverage design skills
- **research momentum**
- **luminary endorsements**
  - There's something deeper, something that's truly beyond objects… I note subtle signs that point to a marked transformation, a disruptive technology, on the horizon. – Grady Booch, Chief Scientist, Rational
  - Charles Simonyi, Creator of Microsoft Word and Excel
  - Michael Jackson, leader in software engineering for 30 years
  - Linda Northrup, Director SEI Product Line Systems Program

aspectj.org

15

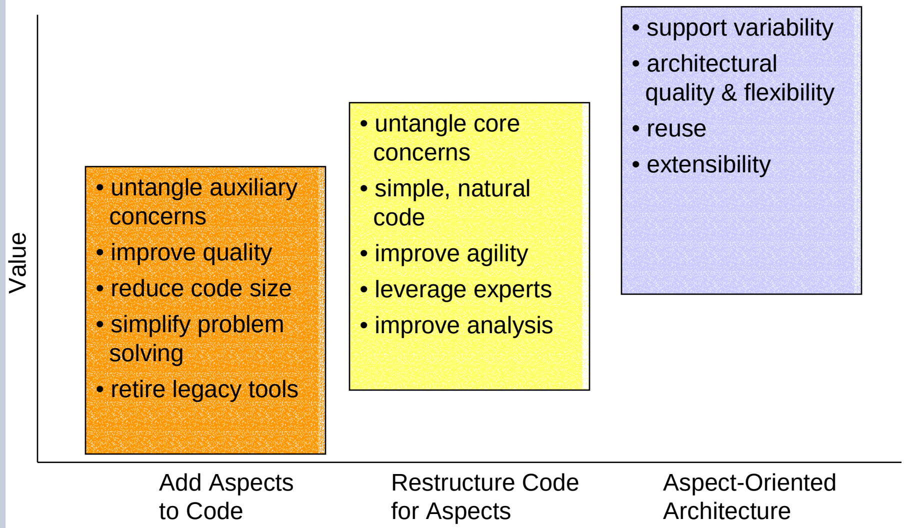
AspectJ smooth adoption curve



aspectj.org

16

expected benefits

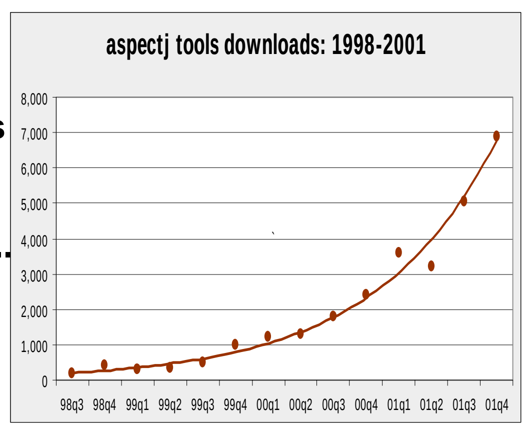


aspectj.org

17

AspectJ adoption

- **rapid growth**
- **user mailing list: 700 active members**
- **MIT Tech Review, Java Report, IBM, Java World, CACM...**
- **production use**



CheckFree The Way Money Moves on the Web

Sirius

- **presentations**
  – JavaOne, OOPSLA, Software Development, SIGS Java, ICSE, TOOLS, ECOOP, JAOO, FSE, …

aspectj.org

18

commercialization

- **1.0 release**
- **focus on successful customer deployments**
- **offerings**
  - tutorials
  - customized trainings
  - architecture workshops
  - consulting support
  - project reviews
- **product development**
  - support, joint development, advanced edition
- **business planning**

aspectj.org

19

AspectJ™ is…

- **a small and well-integrated extension to Java**
  - outputs .class files compatible with any JVM
  - all Java programs are AspectJ programs
- **a general-purpose AO language**
  - just as Java is a general-purpose OO language
- **includes IDE support**
  - emacs, JBuilder, Forte 4J, Eclipse
- **freely available implementation**
  - compiler is Open Source
- **user feedback is driving language design**
  - users@aspectj.org, support@aspectj.org

aspectj.org

20

looking ahead

problem structure

examples:

crosscutting in the design, and
how to use AspectJ to capture that

AspectJ language

language mechanisms:

crosscutting in the code
mechanisms AspectJ provides

aspectj.org

Part II

Tutorial

aspectj.org

language mechanisms

- **Goal: present basic mechanisms**
  - using one simple example
    - emphasis on what the mechanisms do
    - small scale motivation
- **later**
  - environment, tools
  - larger examples, design and SE issues

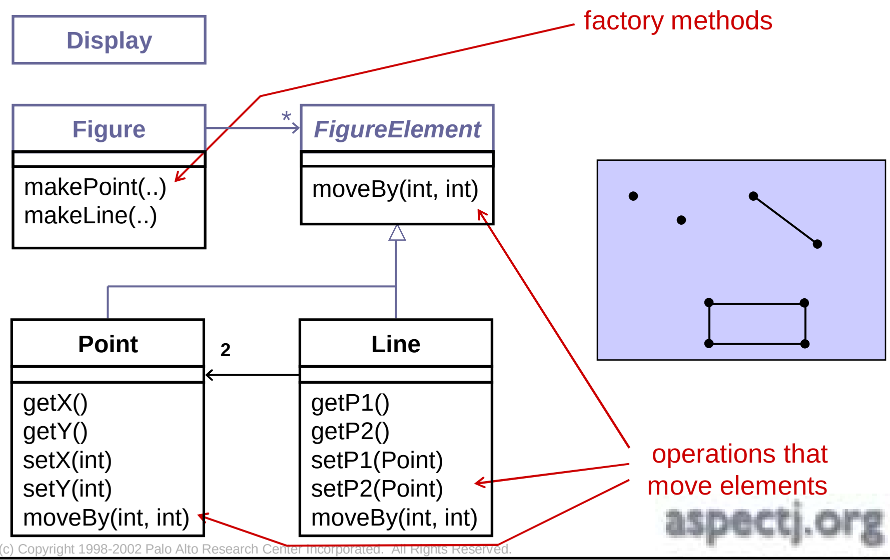
aspectj.org

basic mechanisms

- **1 overlay onto Java**
  - dynamic join points
    - "points in the execution" of Java programs
- **4 small additions to Java**
  - pointcuts
    - pick out join points and values at those points
      - primitive, user-defined pointcuts
  - advice
    - additional action to take at join points in a pointcut
  - inter-class declarations (aka "open classes")
  - aspect
    - a modular unit of crosscutting behavior
      - comprised of advice, inter-class, pointcut, field,constructor and method declarations

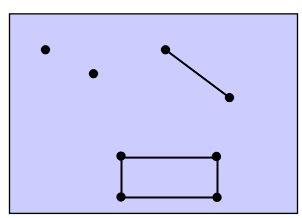
aspectj.org

a simple figure editor



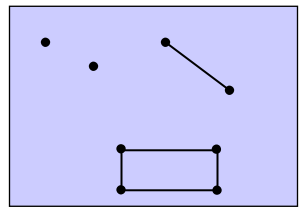
a simple figure editor

```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }
}

class Point implements FigureElement {
  private int x = 0, y = 0;
  int getX() { return x; }
  int getY() { return y; }
  void setX(int x) { this.x = x; }
  void setY(int y) { this.y = y; }
  void moveBy(int dx, int dy) { ... }
}
```



display updating

- **collection of figure elements**
  - that move periodically
  - must refresh the display as needed
  - complex collection
  - asynchronous events
- **other examples**
  - session liveness
  - value caching



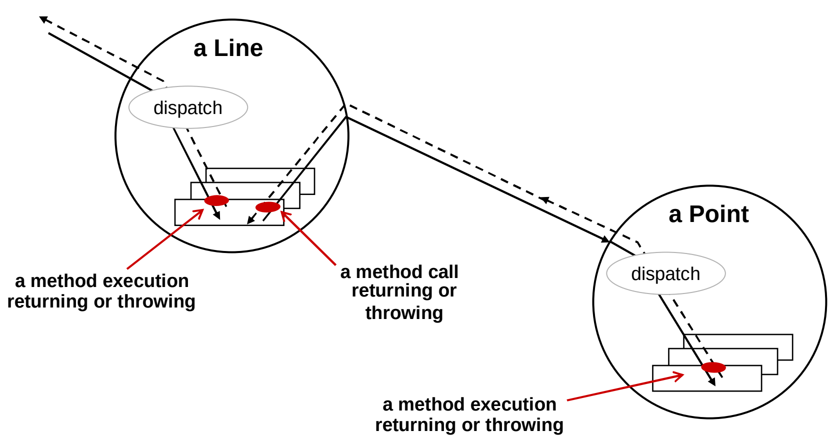
we will initially assume just a single display

aspectj.org

join points

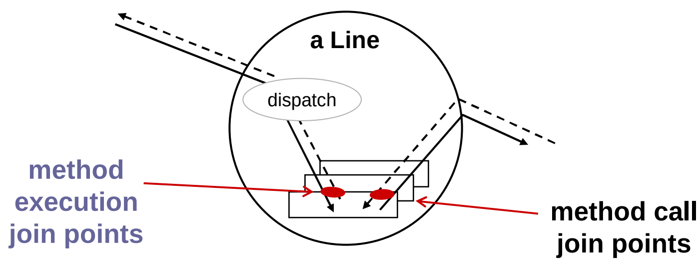
key points in dynamic call graph

imagine `l.moveBy(2, 2)`



aspectj.org

join point terminology



- **several kinds of join points**
  - method & constructor call
  - method & constructor execution
  - field get & set
  - exception handler execution
  - static & dynamic initialization

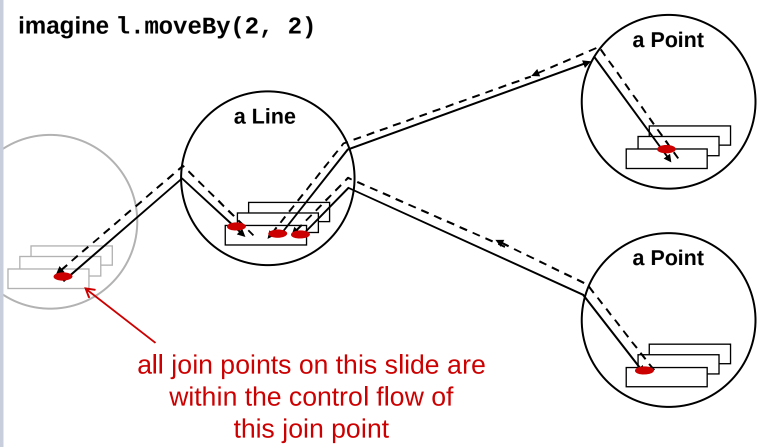
aspectj.org

29

join point terminology

key points in dynamic call graph

imagine `l.moveBy(2, 2)`



aspectj.org

30

primitive pointcuts

"a means of identifying join points"

a pointcut is a kind of predicate on join points that:

- can match or not match any given join point and
- optionally, can pull out some of the values at that join point

```
call(void Line.setP1(Point))
```

matches if the join point is a method call with this signature

aspectj.org

pointcut composition

pointcuts compose like predicates, using &&, || and !

a "void Line.setP1(Point)" call

or

```
call(void Line.setP1(Point)) ||
call(void Line.setP2(Point));
```

a "void Line.setP2(Point)" call

whenever a Line receives a
"void setP1(Point)" or "void setP2(Point)" method call

aspectj.org

user-defined pointcuts

defined using the pointcut construct

user-defined (aka named) pointcuts

– can be used in the same way as primitive pointcuts

name parameters

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
```

more on parameters and how pointcut can expose values at join points in a few slides

aspectj.org

pointcuts

user-defined pointcut

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
```

primitive pointcut, can also be:

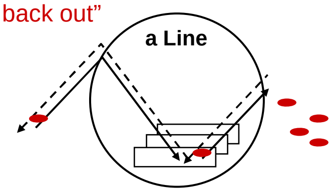
- call, execution **- this, target**
- get, set **- within, withincode**
- handler **- cflow, cflowbelow**
- initialization, staticinitialization

aspectj.org

after advice

action to take after computation under join points

after advice runs
“on the way back out”




```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));

after() returning: move() {
  <code here runs after each move>
}
```


a simple aspect

DisplayUpdating v1

an aspect defines a special class
that can crosscut other classes

```
aspect DisplayUpdating {

  pointcut move():
    call(void Line.setP1(Point)) ||
    call(void Line.setP2(Point));

  after() returning: move() {
    Display.update();
  }
}
```

box means complete running code

without AspectJ

DisplayUpdating v1

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update();
  }
}
```

- **what you would expect**
  - update calls are tangled through the code
  - “what is going on” is less explicit

37

pointcuts

can cut across multiple classes

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point)) ||
  call(void Point.setX(int))   ||
  call(void Point.setY(int));
```

38

pointcuts

can use interface signatures

```
pointcut move():
  call(void FigureElement.moveBy(int, int)) ||
  call(void Line.setP1(Point))              ||
  call(void Line.setP2(Point))              ||
  call(void Point.setX(int))                ||
  call(void Point.setY(int));
```

aspectj.org

39

a multi-class aspect

DisplayUpdating v2

```
aspect DisplayUpdating {

  pointcut move():
    call(void FigureElement.moveBy(int, int)) ||
    call(void Line.setP1(Point))              ||
    call(void Line.setP2(Point))              ||
    call(void Point.setX(int))                ||
    call(void Point.setY(int));

  after() returning: move() {
    Display.update();
  }
}
```

aspectj.org

40

using values at join points

demonstrate first, explain in detail afterwards

- pointcut can explicitly expose certain values
- advice can use value

parameter mechanism being used

```
pointcut move(FigureElement figElt):
  target(figElt) &&
  (call(void FigureElement.moveBy(int, int)) ||
   call(void Line.setP1(Point))              ||
   call(void Line.setP2(Point))              ||
   call(void Point.setX(int))                ||
   call(void Point.setY(int)));

after(FigureElement fe) returning: move(fe) {
  <fe is bound to the figure element>
}
```

aspectj.org

41

explaining parameters…

of user-defined pointcut designator

- **variable is bound by user-defined pointcut declaration**
  - pointcut supplies value for variable
  - value is available to all users of user-defined pointcut

pointcut parameters

```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

typed variable in place of type name

```
after(Line line): move(line) {
  <line is bound to the line>
}
```

aspectj.org

42

explaining parameters…

of advice

- **variable is bound by advice declaration**
  - pointcut supplies value for variable
  - value is available in advice body

```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

advice parameters

typed variable in place of type name

```
after(Line line): move(line) {
  <line is bound to the line>
}
```

aspectj.org

43

explaining parameters…

- **value is 'pulled'**
  - right to left across ':' left side : right side
  - from pointcuts to user-defined pointcuts
  - from pointcuts to advice, and then advice body

```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

```
after(Line line): move(line) {
  <line is bound to the line>
}
```

aspectj.org

44

target

primitive pointcut designator

```
target(<type name> | <formal reference>)
```

does two things:

- **exposes target**
- **predicate on join points - any join point at which target object is an instance of type name (a dynamic test)**

```
target(Point)
target(Line)
target(FigureElement)
```

"any join point" means it matches join points of all kinds

- method & constructor call join points
- method & constructor execution join points
- field get & set join points
- exception handler execution join points
- static & dynamic initialization join points

45

idiom for...

getting target object in a polymorphic pointcut

```
target(<supertype name>) &&
```

- **does not further restrict the join points**
- **does pick up the target object**

```
pointcut move(FigureElement figElt):
  target(figElt) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)) ||
   call(void Point.setX(int))   ||
   call(void Point.setY(int)));

after(FigureElement fe): move(fe) {
  <fe is bound to the figure element>
}
```

46

pointcuts

can expose values at join points

```
pointcut move(FigureElement fe):
  target(fe) &&
  (call(void FigureElement.moveBy(int, int)) ||
   call(void Line.setP1(Point))              ||
   call(void Line.setP2(Point))              ||
   call(void Point.setX(int))                ||
   call(void Point.setY(int)));
```


aspectj.org

context & multiple classes

DisplayUpdating v3

```
aspect DisplayUpdating {

  pointcut move(FigureElement figElt):
    target(figElt) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  after(FigureElement fe): move(fe) {
    Display.update(fe);
  }
}
```


aspectj.org

without AspectJ

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;

  }
  void setP2(Point p2) {
    this.p2 = p2;

  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
}
```

aspectj.org

without AspectJ

DisplayUpdating v1

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update();
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
}
```

aspectj.org

without AspectJ

DisplayUpdating v2

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update();
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
    Display.update();
  }
  void setY(int y) {
    this.y = y;
    Display.update();
  }
}
```

aspectj.org

without AspectJ

DisplayUpdating v3

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update(this);
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update(this);
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
    Display.update(this);
  }
  void setY(int y) {
    this.y = y;
    Display.update(this);
  }
}
```

- **no locus of “display updating”**
  - evolution is cumbersome
  - changes in all classes
  - have to track & change all callers

aspectj.org

with AspectJ

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
  }
  void setP2(Point p2) {
    this.p2 = p2;
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
  }
  void setY(int y) {
    this.y = y;
  }
}
```

aspectj.org

53

with AspectJ

DisplayUpdating v1

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
  }
  void setP2(Point p2) {
    this.p2 = p2;
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
  }
  void setY(int y) {
    this.y = y;
  }
}
```

```
aspect DisplayUpdating {

  pointcut move():
    call(void Line.setP1(Point)) ||
    call(void Line.setP2(Point));

  after() returning: move() {
    Display.update();
  }
}
```

aspectj.org

54

with AspectJ

DisplayUpdating v2

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
  }
  void setP2(Point p2) {
    this.p2 = p2;
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
  }
  void setY(int y) {
    this.y = y;
  }
}
```

```
aspect DisplayUpdating {

  pointcut move():
    call(void FigureElement.moveBy(int, int) ||
    call(void Line.setP1(Point))             ||
    call(void Line.setP2(Point))             ||
    call(void Point.setX(int))               ||
    call(void Point.setY(int));

  after() returning: move() {
    Display.update();
  }
}
```

55

aspectj.org

with AspectJ

DisplayUpdating v3

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
  }
  void setP2(Point p2) {
    this.p2 = p2;
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
  }
  void setY(int y) {
    this.y = y;
  }
}
```

```
aspect DisplayUpdating {

  pointcut move(FigureElement figElt):
    target(figElt) &&
    (call(void FigureElement.moveBy(int, int) ||
     call(void Line.setP1(Point))             ||
     call(void Line.setP2(Point))             ||
     call(void Point.setX(int))               ||
     call(void Point.setY(int)));


  after(FigureElement fe) returning: move(fe) {
    Display.update(fe);
  }
}
```

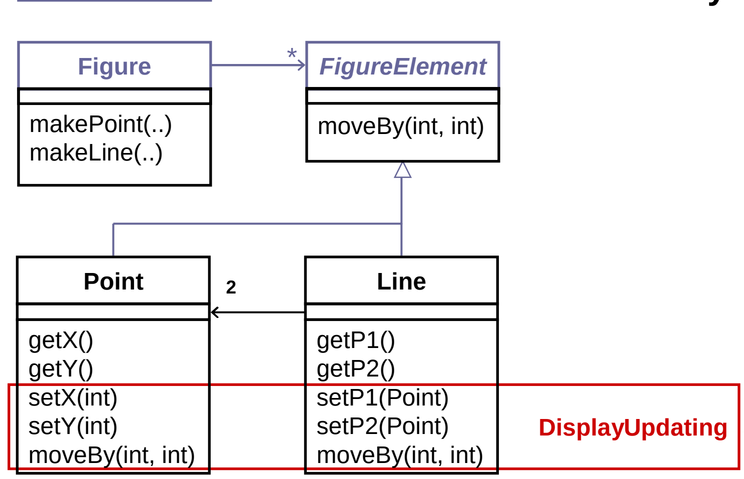
- **clear display updating module**
  - all changes in single aspect
  - evolution is modular

56

aspectj.org

aspects crosscut classes

aspect modularity cuts across class modularity



57

advice is

additional action to take at join points

- **before** — before proceeding at join point
- **after returning** — a value to join point
- **after throwing** — a throwable to join point
- **after** — returning to join point either way
- **around** — on arrival at join point gets explicit control over when&if program proceeds

aspectj.org

58

contract checking

simple example of before/after/around

- **pre-conditions**
  - check whether parameter is valid
- **post-conditions**
  - check whether values were set
- **condition enforcement**
  - force parameters to be valid

59

pre-condition

using before advice

```
aspect PointBoundsPreCondition {

  before(int newX):
      call(void Point.setX(int)) && args(newX) {
    assert(newX >= MIN_X);
    assert(newX <= MAX_X);
  }
  before(int newY):
      call(void Point.setY(int)) && args(newY) {
    assert(newY >= MIN_Y);
    assert(newY <= MAX_Y);
  }

  private void assert(boolean v) {
    if ( !v )
      throw new RuntimeException();
  }
}
```

what follows the ':' is always a pointcut – primitive or user-defined

60

post-condition

using after advice

```
aspect PointBoundsPostCondition {

  after(Point p, int newX) returning:
      call(void Point.setX(int)) && target(p) && args(newX) {
    assert(p.getX() == newX);
  }

  after(Point p, int newY) returning:
      call(void Point.setY(int)) && target(p) && args(newY) {
    assert(p.getY() == newY);
  }

  private void assert(boolean v) {
    if ( !v )
      throw new RuntimeException();
  }
}
```


condition enforcement

using around advice

```
aspect PointBoundsEnforcement {

  void around(int newX):
      call(void Point.setX(int)) && args(newX) {
    proceed(clip(newX, MIN_X, MAX_X));
  }

  void around(int newY):
      call(void Point.setY(int)) && args(newY) {
    proceed(clip(newY, MIN_Y, MAX_Y));
  }

  private int clip(int val, int min, int max) {
    return Math.max(min, Math.min(max, val));
  }
}
```

special method

for each around advice with the signature

```
<Tr> around(T1 arg1, T2 arg2, …)
```

there is a special method with the signature

```
<Tr> proceed(T1, T2, …)
```

available only in around advice

means “run what would have run if this around advice had not been defined”

property-based crosscutting

```
package com.xerox.pri
public class C1 {
  …
  public void foo() {
    A.doSomething(…);
    …
  }
  …
}
```

```
package com.xerox.scan;
public class C2 {
  …
  public int frotz()
    A.doSomething(…);
    …
  }
  public int bar() {
    A.doSomething(…);
    …
  }
  …
}
```

```
package com.xerox.copy;
public class C3 {
  …
  public String s1() {
    A.doSomething(…);
    …
  }
  …
}
```

- **crosscuts of methods with a common property**
  - public/private, return a certain value, in a particular package
- **logging, debugging, profiling**
  - log on entry to every public method

property-based crosscutting

```
aspect PublicErrorLogging {

  Log log = new Log();

  pointcut publicInterface():
    call(public * com.xerox..*.*(..));

  after() throwing (Error e): publicInterface() {
    log.write(e);
  }
}
```

neatly captures public interface of mypackage

consider code maintenance

- **another programmer adds a public method**
  - i.e. extends public interface – this code will still work
- **another programmer reads this code**
  - “what’s really going on” is explicit

aspectj.org

65

wildcarding in pointcuts

“*” is wild card
“..” is multi-part wild card

`target(Point)`	
`target(graphics.geom.Point)`	
`target(graphics.geom.*)`	**any type in graphics.geom**
`target(graphics..*)`	**any type in any sub-package of graphics**
`call(void Point.setX(int))`	
`call(public * Point.*(..))`	**any public method on Point**
`call(public * *(..))`	**any public method on any type**
`call(void Point.getX())`	
`call(void Point.getY())`	
`call(void Point.get*())`	
`call(void get*())`	**any getter**
`call(Point.new(int, int))`	
`call(new(..))`	**any constructor**

aspectj.org

66

other primitive pointcuts

```
this(<type name>)
within(<type name>)
withincode(<method/constructor signature>)
```

any join point at which
- currently executing object is an instance of type name
- currently executing code is contained within type name
- currently executing code is specified method or constructor

```
get(int Point.x)
set(int Point.x)
```

field reference or assignment join points

aspectj.org

fine-grained protection

a runtime error

```
class Figure {
  public Line  makeLine(Line p1, Line p2) { new Line...  }
  public Point makePoint(int x, int y)    { new Point... }
  ...
}
```

want to ensure that any creation of figure elements goes through the factory methods

```
aspect FactoryEnforcement {
  pointcut illegalNewFigElt():
    (call(Point.new(..)) || call(Line.new(..)))
    && !withincode(* Figure.make*(..));

  before(): illegalNewFigElt() {
    throw new Error("Use factory method instead.");
  }
}
```


aspectj.org

fine-grained protection

a compile-time error

```
class Figure {
  public Line  makeLine(Line p1, Line p2) { new Line...  }
  public Point makePoint(int x, int y)    { new Point... }
  ...
}
```

want to ensure that any creation of figure elements goes through the factory methods

```
aspect FactoryEnforcement {
  pointcut illegalNewFigElt():
    (call(Point.new(..)) || call(Line.new(..)))
    && !withincode(* Figure.make*(..));

  declare error: illegalNewFigElt():
    "Use factory method instead.";
  }
}
```

must be a "static pointcut" (more on this later)

aspectj.org

69

fine-grained protection

a compile-time error

```
class Figure {
  public Line  makeLine(Line p1, Line p2) { new Line...  }
  public Point makePoint(int x, int y)    { new Point... }
  ...
}
```

want to ensure that any creation of figure elements goes through the factory methods

```
aspect FactoryEnforcement {
  pointcut illegalNewFigElt():
    call(FigureElement+.new(..))
    && !withincode(* Figure.make*(..));

  declare error: illegalNewFigElt():
    "Use factory method instead.";
  }
}
```

must be a "static pointcut" (more on this later)

aspectj.org

70

fine-grained protection

as a static inner aspect

```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }

  static aspect SetterEnforcement {
    declare error: set(Point Line.*) &&
                   !withincode(void Line.setP*(Point))
      "Use setter method.";
  }
}
```

aspectj.org

71

fine-grained protection

as a static inner aspect

```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }

  static aspect SetterEnforcement {
    declare error: set(Point Line.*) &&
                   !withincode(void Line.setP*(Point))
      "Use setter method, even inside Line class.";
  }
}
```

aspectj.org

72

special value

reflective* access to the join point

```
thisJoinPoint.
     Signature  getSignature()
     Object[]   getArgs()
     ...
```

available in any advice

* introspective subset of reflection consistent with Java

73

using thisJoinPoint

in highly polymorphic advice

```
aspect PointCoordinateTracing {

  before(int newVal): set(int Point.*) && args(newVal) {
    System.out.println("At " +
                       thisJoinPoint.getSignature() +
                       " field is set to " +
                       newVal +
                       ".");
  }
}
```

using thisJoinPoint makes it possible for the advice to recover information about where it is running

74

other primitive pointcuts

`execution(void Point.setX(int))`
method/constructor execution join points (actual running method)

`initialization(Point)`
object initialization join points

`staticinitialization(Point)`
class initialization join points (as the class is loaded)

aspectj.org

75

other primitive pointcuts

cflow(*pointcut designator*)
all join points within the dynamic control flow of any join point in *pointcut designator*

cflowbelow(*pointcut designator*)
all join points within the dynamic control flow below any join point in *pointcut designator*

aspectj.org

76

only top-level moves

DisplayUpdating v4

```
aspect DisplayUpdating {

  pointcut move(FigureElement fe):
    target(fe) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  pointcut topLevelMove(FigureElement fe):
    move(fe) && !cflowbelow(move(FigureElement));

  after(FigureElement fe) returning: topLevelMove(fe) {
    Display.update(fe);
  }
}
```

aspectj.org

77

one display per figure element

DisplayUpdating v5

```
aspect DisplayUpdating {

  private Display FigureElement.display;

  static void setDisplay(FigureElement fe, Display d) {
    fe.display = d;
  }

  pointcut move(FigureElement figElt):
    <as before>;

  after(FigureElement fe): move(fe) {
    fe.display.update(fe);
  }
}
```

aspectj.org

78

field/getter/setter idiom

```
aspect DisplayUpdating {

  private Display FigureElement.display;

  public static void setDisplay(FigureElement fe, Display d) {
    fe.display = d;
  }

  pointcut
    <as bef

  after(Fig
    fe.disp
  }
}
```

private with respect to enclosing aspect declaration

the display field

- is a field in objects of type `FigureElement`, but
- belongs to `DisplayUpdating` aspect
- `DisplayUpdating` should provide getter/setter

79

one-to-many

DisplayUpdating v6

```
aspect DisplayUpdating {

  private List FigureElement.displays = new LinkedList();

  public static void addDisplay(FigureElement fe, Display d) {
    fe.displays.add(d);
  }
  public static void removeDisplay(FigureElement fe, Display d) {
    fe.displays.remove(d);
  }

  pointcut move(FigureElement figElt):
    <as before>;

  after(FigureElement fe): move(fe) {
    Iterator iter = fe.displays.iterator();
    ...
  }
}
```

80

inheritance & specialization

- **pointcuts can have additional advice**
  - aspect with
    - concrete pointcut
    - perhaps no advice on the pointcut
  - in figure editor
    - **`move()`** can have advice from multiple aspects
  - module can expose certain well-defined pointcuts
- **abstract pointcuts can be specialized**
  - aspect with
    - abstract pointcut
    - concrete advice on the abstract pointcut

role types and reusable

```
abstract aspect Observing {

  protected interface Subject  { }
  protected interface Observer { }

  public void    addObserver(Subject s, Observer o) { ... }
  public void removeObserver(Subject s, Observer o) { ... }
  public static List getObservers(Subject s) { ... }

  abstract pointcut changes(Subject s);

  after(Subject s): changes(s) {
    Iterator iter = getObservers(s).iterator();
    while ( iter.hasNext() ) {
      notifyObserver(s, ((Observer)iter.next()));
    }
  }
  abstract void notifyObserver(Subject s, Observer o);
}
```


this is the concrete reuse

DisplayUpdating v7

```
aspect DisplayUpdating extends Observing {

  declare parents: FigureElement implements Subject;
  declare parents: Display       implements Observer;

  pointcut changes(Subject s):
    target(s) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  void notifyObserver(Subject s, Observer o) {
    ((Display)o).update(s);
  }
}
```

83

design invariants

```
aspect FactoryEnforcement {

  pointcut newFigElt():
    call(FigureElement.new(..));

  pointcut inFactory():
    within(Point Figure.make*(..));

  pointcut illegalNewFigElt():
    newFigElt() && !inFactory();


  declare error: illegalNewFigElt():
    "Must call factory method to create figure elements.";

}
```

84

summary

join points
- method & constructor
  - call
  - execution
- field
  - get
  - set
- exception handler
  - execution
- initialization

aspects
- crosscutting type

pointcuts
- **-primitive-**
  - call
  - execution
  - handler
  - get set
  - initialization
  - this target
  - within withincode
  - cflow cflowbelow
- **-user-defined-**
  - pointcut
- declaration
  - abstract
  - overriding

advice
- before
- after
- around

inter-type decls
- Type.field
- Type.method()

declare
- warning
- error
- parents

reflection
- thisJoinPoint
- thisJoinPointStaticPart

aspectj.org

where we have been…

… and where we are going

problem structure

↓

examples:

crosscutting in the design, and
how to use AspectJ to capture that

↑

AspectJ language

language mechanisms:

crosscutting in the code
mechanisms AspectJ provides

aspectj.org

using aspects

- **present examples of aspects in design**
  – intuitions for identifying aspects
- **present implementations in AspectJ**
  – how the language support can help
  – putting AspectJ into practice
- **discuss style issues**
  – objects vs. aspects
- **when are aspects appropriate?**

aspectj.org

87

example

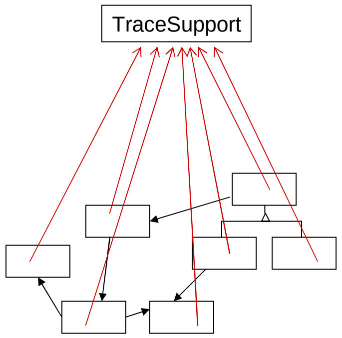
plug & play tracing

- **simple tracing**
  – exposes join points and uses very simple advice
- **an unpluggable aspect**
  – core program functionality is unaffected by the aspect

aspectj.org

88

tracing without AspectJ



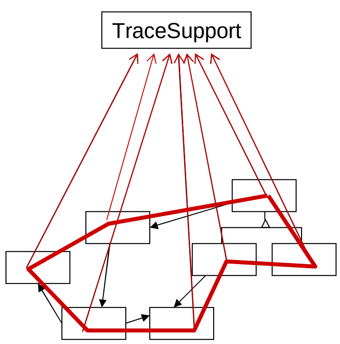

```
class TraceSupport {
  static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  static void init(PrintStream _s) {stream=_s;}

  static void traceEntry(String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  static void traceExit(String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
}
```

```
class Point {
  void set(int x, int y) {
    TraceSupport.traceEntry(“Point.set”);
    this.x = x; this.y = y;
    TraceSupport.traceExit(“Point.set”);
  }
}
```

aspectj.org

89

a clear crosscutting structure



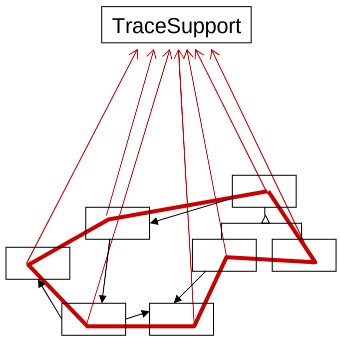
this line is about <u>interacting</u> with the trace facility

all modules of the system use the trace facility in a <u>consistent</u> way: entering the methods and exiting the methods

aspectj.org

90

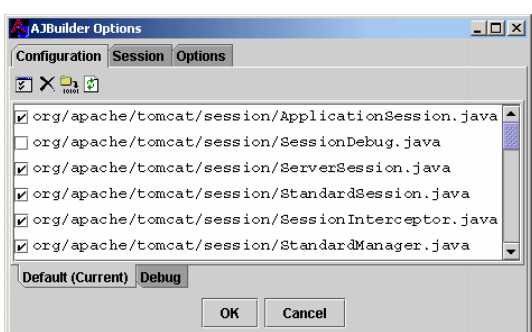
tracing as an aspect




```
aspect PointTracing {

  pointcut trace():
      within(com.bigboxco.boxes.*) &&
      execution(* *(..));

  before(): trace() {
    TraceSupport.traceEntry(tjp);
  }
  after(): trace() {
    TraceSupport.traceExit(tjp);
  }
}
```


plug and debug

- **plug in:** `ajc Point.java Line.java TraceSupport.java PointTracing.java`
- **unplug:** `ajc Point.java Line.java`
- **or…**



plug and debug

```
//From ContextManager

public void service( Request rrequest, Response rresponse ) {
 // log( "New  request " + rrequest );
 try {
     //     System.out.print("A");
     rrequest.setContextManager( this );
     rrequest.setResponse(rresponse);
     rresponse.setRequest(rrequest);

     // wront request - parsing error
     int status=rresponse.getStatus();

     if( status < 400 )
  status= processRequest( rrequest );

     if(status==0)
  status=authenticate( rrequest, rresponse );
     if(status == 0)
  status=authorize( rrequest, rresponse );
     if( status == 0 ) {
  rrequest.getWrapper().handleRequest(rrequest,
          rresponse);
     } else {
  // something went wrong
  handleError( rrequest, rresponse, null, status );
     }
 } catch (Throwable t) {
     handleError( rrequest, rresponse, t, 0 );
 }
 // System.out.print("B");
 try {
     rresponse.finish();
     rrequest.recycle();
     rresponse.recycle();
 } catch( Throwable ex ) {
     if(debug>0) log( "Error closing request " + ex);
 }
 // log( "Done with request " + rrequest );
 // System.out.print("C");
 return;
}
```

```
// log( "New  request " + rrequest );

// System.out.print("A");

// System.out.print("B");

if(debug>0)
   log("Error closing request " + ex);

// log("Done with request " + rrequest);

// System.out.print("C");
```

93 aspectj.org

plug and debug

- turn debugging on/off without editing classes
- debugging disabled with no runtime cost
- can save debugging code between uses
- can be used for profiling, logging
- easy to be sure it is off

94 aspectj.org

aspects in the design

have these benefits

- **objects are no longer responsible for using the trace facility**
  - trace aspect encapsulates that responsibility, for appropriate objects
- **if the Trace interface changes, that change is shielded from the objects**
  - only the trace aspect is affected
- **removing tracing from the design is trivial**
  - just remove the trace aspect

95 aspectj.org

aspects in the code

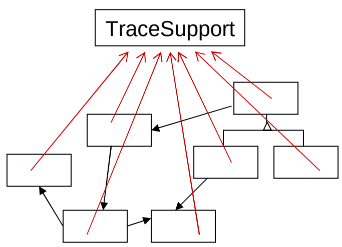
have these benefits

- **object code contains no calls to trace functions**
  - trace aspect code encapsulates those calls, for appropriate objects
- **if the Trace interface changes, there is no need to modify the object classes**
  - only the trace aspect class needs to be modified
- **removing tracing from the application is trivial**
  - compile without the trace aspect class

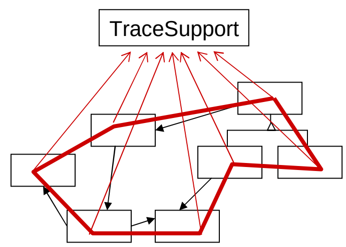
96 aspectj.org

tracing: object vs. aspect

- using an object captures tracing support, but does not capture its consistent usage by other objects



- using an aspect captures the consistent usage of the tracing support by the objects



97

aspectj.org

tracing

using a library aspect

```
aspect BigBoxCoTracing {

  pointcut trace():
      within(com.bigboxco.*)
      && execution(* *(..));

  before(): trace() {
    TraceSupport.traceEntry(
      tjp);
  }
  after(): trace() {
    TraceSupport.traceExit(
      tjp);
  }
}
```

```
abstract aspect Tracing {
  abstract pointcut trace();

  before(): trace() {
    TraceSupport.traceEntry(tjp);
  }
  after(): trace() {
    TraceSupport.traceExit(tjp);
  }
}
```

```
aspect BigBoxCoTracing
    extends Tracing {

  pointcut trace():
      within(com.bigboxco.*)
      && execution(* *(..));
}
```

98

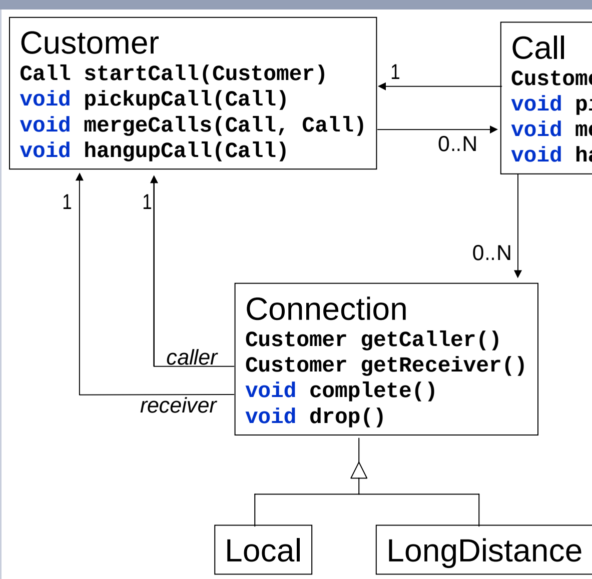
aspectj.org

example

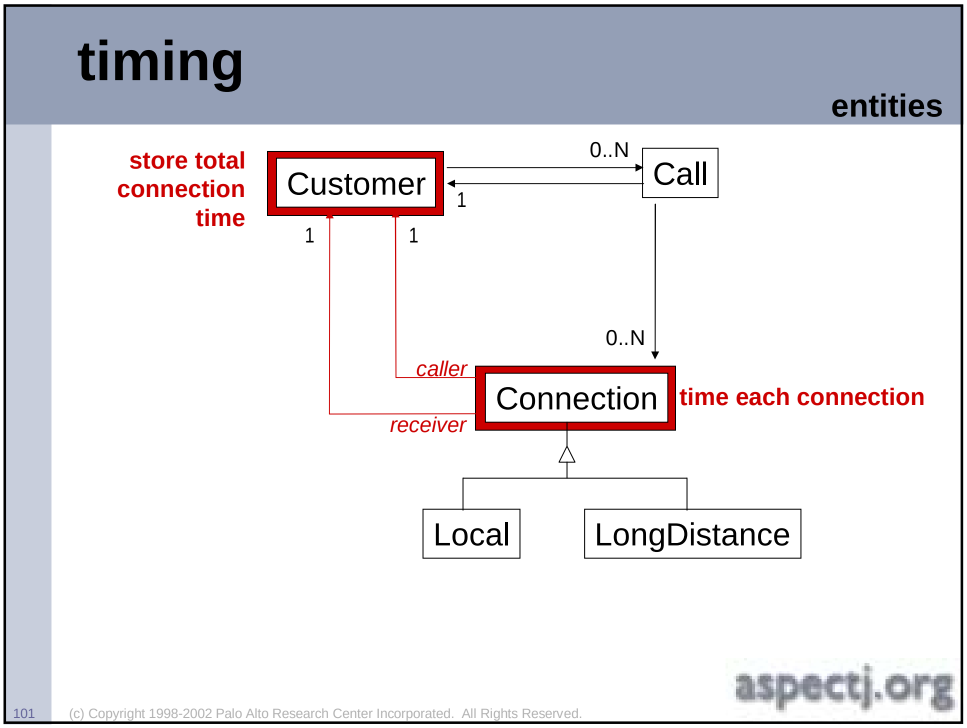
layers of functionality

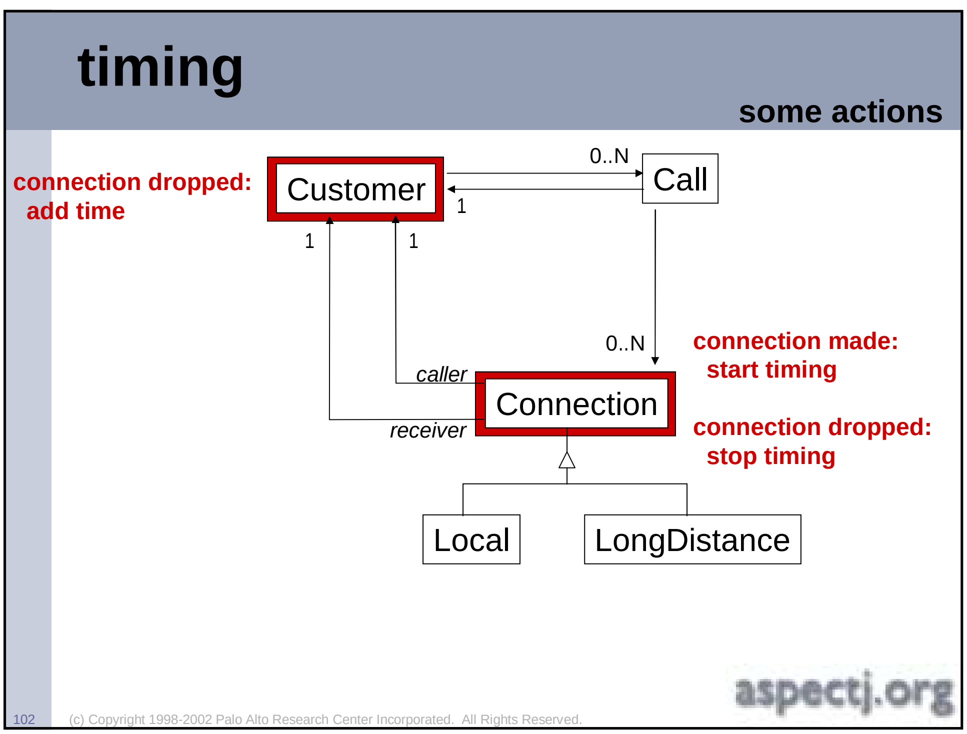
- **given a basic telecom operation, with customers, calls, connections**
- **model/design/implement utilities such as**
  - timing
  - consistency checks
  - ...

telecom basic design



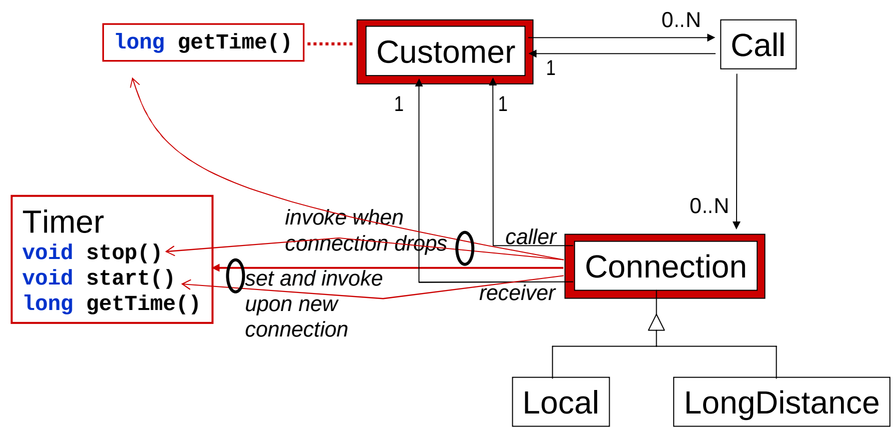
These classes define the protocols for setting up calls (includes conference calling) and establishing connections


timing
entities
store total connection time
Customer
Call
0..N
1
1
1
0..N
caller
receiver
Connection
time each connection
Local
LongDistance
aspectj.org
101
(c) Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.


timing
some actions
connection dropped: add time
Customer
Call
0..N
1
1
1
0..N
connection made: start timing
caller
receiver
Connection
connection dropped: stop timing
Local
LongDistance
aspectj.org
102
(c) Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

timing

additional design elements



aspectj.org

timing

what is the nature of the crosscutting?

- **Connections and Customers are involved**
- **well defined protocols among them**
- **pieces of the timing protocol must be triggered by the execution of certain basic operations. e.g.**
  - when connection is completed,
    set and start a timer
  - when connection drops,
    stop the timer and add time to customers' connection time

aspectj.org

timing

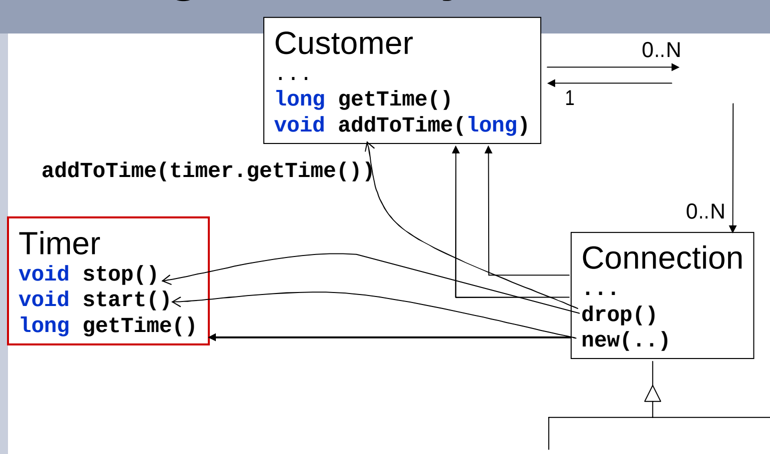
an aspect implementation

```
aspect Timing {
  private Timer Connection.timer = new Timer();

  private long Customer.totalConnectTime = 0;
  public static long getTotalConnectTime(Customer c) {
    return c.totalConnectTime;
  }

  pointcut startTiming(Connection c): target(c) && call(void c.complete());
  pointcut   endTiming(Connection c): target(c) && call(void c.drop());

  after(Connection c) returning: startTiming(c) {
    c.timer.start();
  }

 after(Connection c) returning: endTiming(c) {
    Timer timer = c.timer;
    timer.stop();
    long currTime = timer.getTime();
    c.getCaller().totalConnectTime += currTime;
    c.getReceiver().totalConnectTime += currTime;
  }
}
```

aspectj.org

105

timing as an object

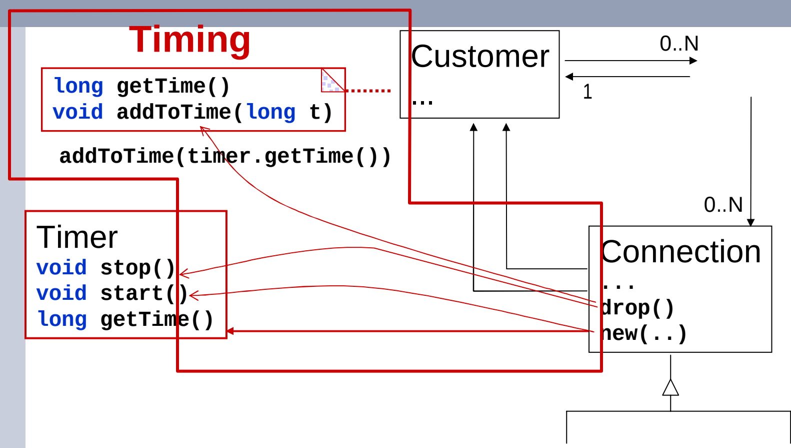


timing as an object captures timing support, but does not capture the protocols involved in implementing the timing feature

aspectj.org

106

timing as an aspect



timing as an aspect captures the protocols involved in implementing the timing feature

aspectj.org

107

timing

interface change

- **Consider a change to the timer interface**

```
Timer
void start()
long stopAndGetTime()
```

- **What changes are necessary in the program?**

```
aspect Timing {
 ...
 after(Connection c): endTiming(c) {
    Timer timer = c.timer;
    long currTime = timer.stopAndGetTime();
    c.getCaller().totalConnectTime += currTime;
    c.getReceiver().totalConnectTime += currTime;
  }
}
```

aspectj.org

108

timing as an aspect

has these benefits

- **basic objects are not responsible for using the timing facility**
  - timing aspect encapsulates that responsibility, for appropriate objects
- **if requirements for timing facility change, that change is shielded from the objects**
  - only the timing aspect is affected
- **removing timing from the design is trivial**
  - just remove the timing aspect

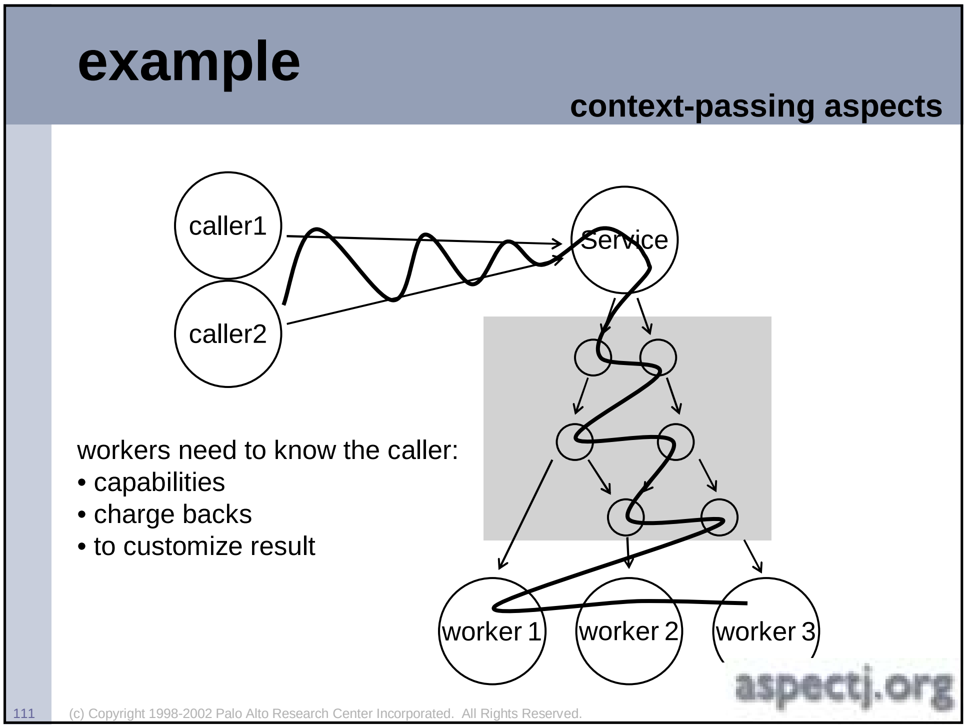
109 aspectj.org

timing with AspectJ

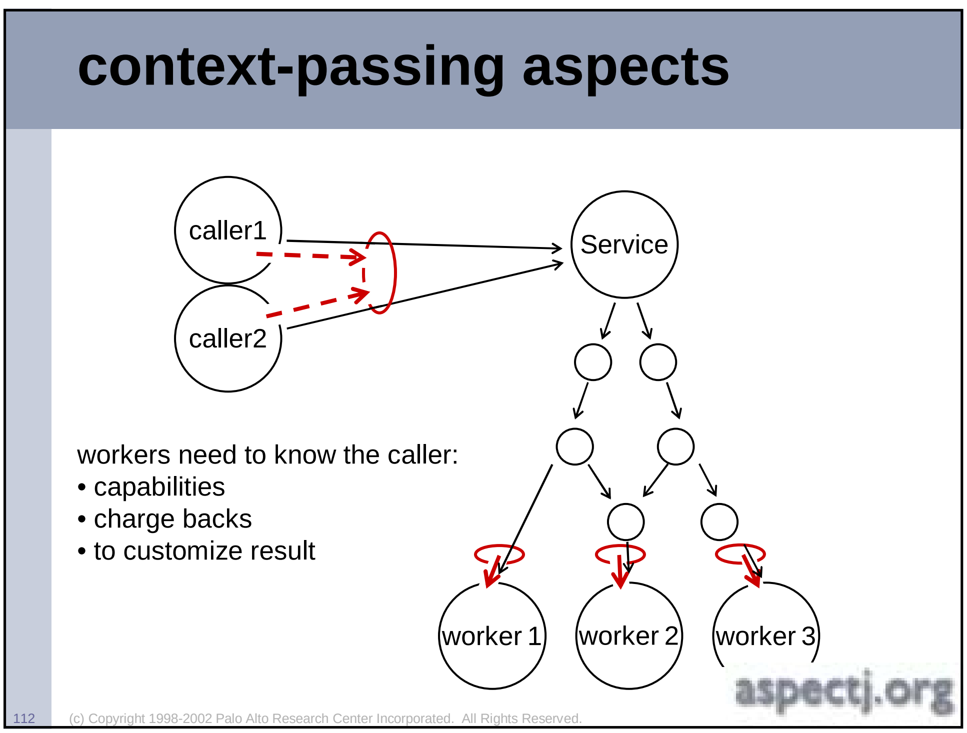
has these benefits

- **object code contains no calls to timing functions**
  - timing aspect code encapsulates those calls, for appropriate objects
- **if requirements for timing facility change, there is no need to modify the object classes**
  - only the timing aspect class and auxiliary classes needs to be modified
- **removing timing from the application is trivial**
  - compile without the timing aspect class

110 aspectj.org



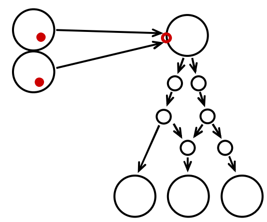
example
context-passing aspects
caller1
caller2
Service
workers need to know the caller:
• capabilities
• charge backs
• to customize result
worker 1
worker 2
worker 3
aspectj.org
111
(c) Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.



context-passing aspects
caller1
caller2
Service
workers need to know the caller:
• capabilities
• charge backs
• to customize result
worker 1
worker 2
worker 3
aspectj.org
112
(c) Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

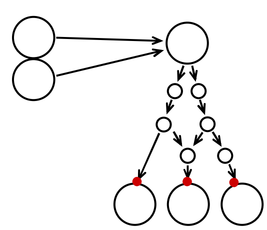
context-passing aspects

```
pointcut invocations(Caller c):
  this(c) && call(void Service.doService(String));
```



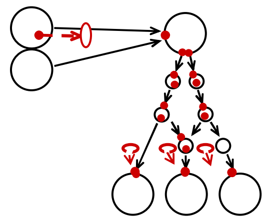
context-passing aspects

```
pointcut invocations(Caller c):
  this(c) && call(void Service.doService(String));

pointcut workPoints(Worker w):
  target(w) && call(void Worker.doTask(Task));
```



context-passing aspects

```
pointcut invocations(Caller c):
  this(c) && call(void Service.doService(String));

pointcut workPoints(Worker w):
  target(w) && call(void Worker.doTask(Task));

pointcut perCallerWork(Caller c, Worker w):
  cflow(invocations(c)) && workPoints(w);
```

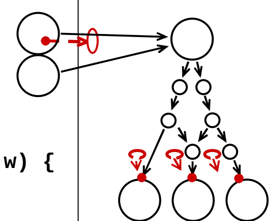


115

aspectj.org

context-passing aspects

```
abstract aspect CapabilityChecking {

  pointcut invocations(Caller c):
    this(c) && call(void Service.doService(String));

  pointcut workPoints(Worker w):
    target(w) && call(void Worker.doTask(Task));

  pointcut perCallerWork(Caller c, Worker w):
    cflow(invocations(c)) && workPoints(w);


  before (Caller c, Worker w): perCallerWork(c, w) {
    w.checkCapabilities(c);
  }
}
```



116

aspectj.org

summary so far

- **presented examples of aspects in design**
  - intuitions for identifying aspects
- **presented implementations in AspectJ**
  - how the language support can help
- **raised some style issues**
  - objects vs. aspects

117

aspectj.org

when are aspects appropriate?

- **is there a concern that:**
  - crosscuts the structure of several objects or operations
  - is beneficial to separate out

118

aspectj.org

… crosscutting

- **a design concern that involves several objects or operations**
- **implemented without AOP would lead to distant places in the code that**
  - do the same thing
    - e.g. traceEntry(“Point.set”)
    - try grep to find these [Griswold]
  - do a coordinated single thing
    - e.g. timing, observer pattern
    - harder to find these

aspectj.org

… beneficial to separate out

- **exactly the same questions as for objects**
- **does it improve the code in real ways?**
  - separation of concerns
    - e.g . think about service without timing
  - clarifies interactions, reduces tangling
    - e.g. all the traceEntry are really the same
  - easier to modify / extend
    - e.g. change the implementation of tracing
    - e.g. abstract aspect re-use
  - plug and play
    - tracing aspects unplugged but not deleted

aspectj.org

good designs

summary

- **capture "the story" well**
- **may lead to good implementations, measured by**
  - – code size
  - – tangling
  - – coupling
  - – etc.

learned through experience, influenced by taste and style

expected benefits of using AOP

- **good modularity, even in the presence of crosscutting concerns**
  - – less tangled code, more natural code, smaller code
  - – easier maintenance and evolution
    - easier to reason about, debug, change
  - – more reusable
    - more possibilities for plug and play
    - abstract aspects

Part III

conclusion

aspectj.org

AOSD future

- **language design**
  - more dynamic crosscuts, type system …
- **tools**
  - more IDE support, aspect discovery, re-factoring, re-cutting, crosscutting views…
- **software engineering**
  - UML extension, finding aspects, …
- **metrics**
  - measurable benefits, areas for improvement
- **theory**
  - type system for crosscutting, faster compilation, advanced crosscut constructs, modularity principles
- **see also aosd.net**

aspectj.org

124

AspectJ possible features

continue building language, compiler & tools

- **user demand driven**
- **specialized support**
  - J2EE (servlet, JSP, EJB, JMS), J2ME
- **flexible weaving**
  - bytecodes
  - intermediate form for aspect libraries
  - load time
- **tools**
  - incremental compiler
  - re-factoring, structure-aware editing, design …
- **language**
  - aspect configuration
  - extensible pointcuts
  - generic types (Java 1.5)
  - structure-shy XML support
  - 2.0?: new dynamic crosscut constructs

aspectj.org

125

AspectJ technology

- **AspectJ is a small extension to Java™**
  - valid Java programs are also valid AspectJ programs
- **AspectJ has its own compiler, ajc**
  - runs on Java 2 platform (Java 1.2 - 1.4)
  - produces Java platform-compatible .class files (Java 1.1 - 1.4)
- **AspectJ tools support**
  - IDE extensions: Emacs, JBuilder, Forte4J, Eclipse
  - ajdoc to parallel javadoc
  - ant tasks
  - JPDA debugger integration (JSR 45 support)
- **license**
  - compiler, runtime and tools are free for any use
  - compiler and tools are Open Source

aspectj.org

126

AspectJ on the web

- **aspectj.org**
  - documentation
  - downloads
  - users@aspectj.org
  - support@aspectj.org

aspectj.org

127

summary

- **OOP → AOP**
  - handles greater complexity, provides more flexibility…
  - crosscutting modularity
- **AspectJ**
  - incremental adoption package → revolutionary benefits
  - free AspectJ tools
  - community
  - training, consulting, and support for use

aspectj.org

128

credits

AspectJ.org is a PARC project:

Erik Hilsdale, Jim Hugunin, Wes Isberg,
Mik Kersten, Gregor Kiczales

slides, compiler, tools & documentation are available at aspectj.org

partially funded by DARPA under contract F30602-97-C0246

aspectj.org

129