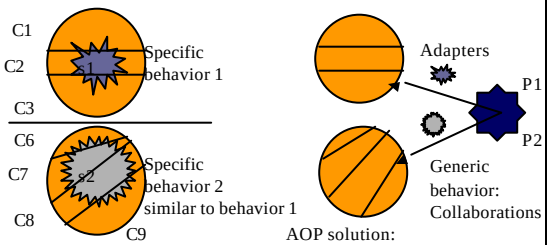
Aspect-Oriented Programming: Fad or the Future

Karl Lieberherr
Northeastern University
UBS AG

AOP: Factoring out similarities that cut across abstractions

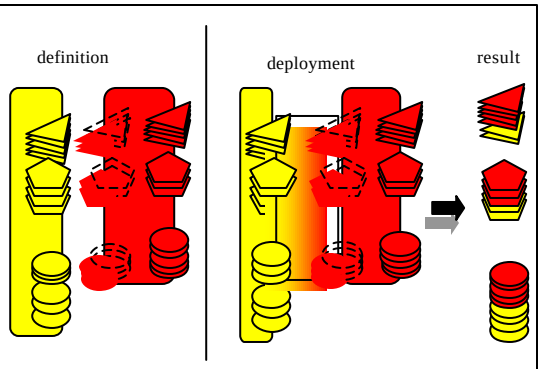


Traditional solution: redundancy

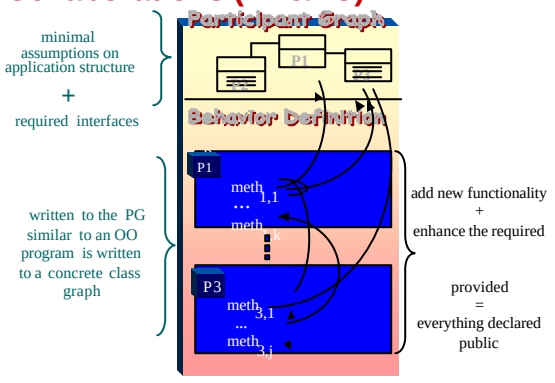
AOP solution: less redundancy, requires tool support

The goal: Tangling control

- **Tangling cannot be eliminated, it can only be reduced and localized.**
- **The goal is to encapsulate program enhancements and minimize dependencies between them (loose coupling).**



Collaborations (AP&PC)



Deployment/Composition of Collaborations

- **Specified by adapters separately from collaborations**
- **Adapters specify crosscutting using**
  - regular-expressions to express sets of method names and class names and interface names
  - code where simple method name mapping is not enough
  - graphs and regular expression-like constructs for mapping graphs

Design issues

- **What goes into collaborations and what goes into adapters?**
- **Depends on reuse of collaboration C.**
- **Consider n reuses of the same collaboration: $A_1(C)$, $A_2(C)$, … ,$A_n(C)$**
- **The goal is to put enough information into collaboration C so that in the adapters contain minimal code duplication.**

Rough correspondences

AOP (generic)	enhancement	crosscutting
Hyper/J	Hyperslice	Hypermodule
AspectJ	Advice Introduction	Crosscut
Collab/ Adapters	Collaboration	Adapter

Fad or Future

- **Future!**
- **Many issues to be resolved**
  - Correctness
    - aspect
    - composition of aspects
      - conflicts between aspects
  - Trying to unify the existing approaches: taxonomy of the design space

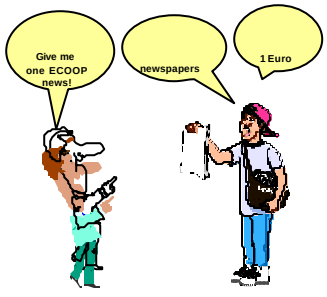
ECOOP'2000 Panel:

Aspect-Oriented Programming

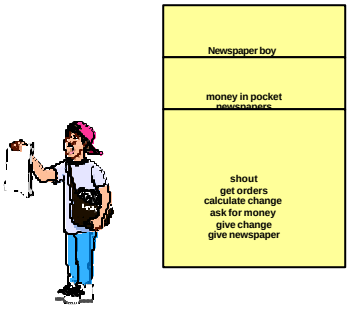
Fad or the future

Mehmet Aksit
Department of Computer Science
P.O. Box 217
7500 AE Enschede, The Netherlands
aksit@cs.utwente.nl
http://www.cs.utwente.nl/~aksit

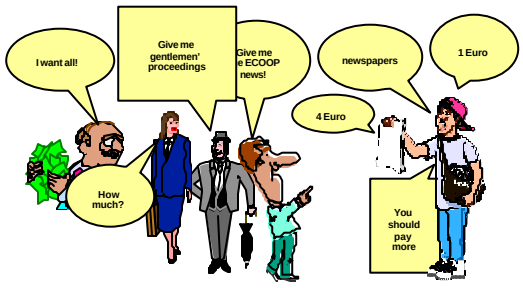
A tale of a newspaper boy



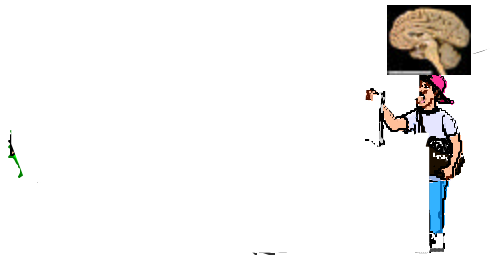
An UML model for the newspaper boy



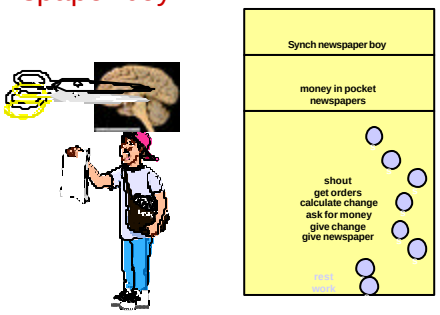
A tale of a popular newspaper boy without synchronization



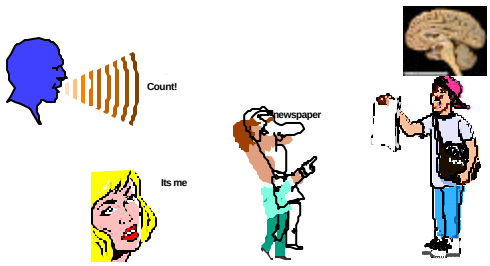
A tale of a popular newspaper boy without synchronization



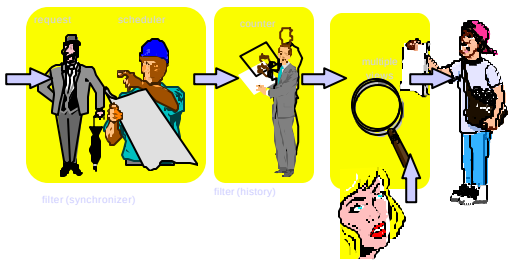
An UML model for the synchronized newspaper boy



Multiple concerns and a tale of a newspaper boy who cannot count & adapt



Modular and orthogonal extensions



A composition-filters model for the newspaper boy

```
diff: Dispatch = {isLover=>*}
syn: Wait = {notActive=> *}
syn: Wait = {isRest=> work, not.isRest=> *}
aCount: Meta = {[*]counter.count}
```

Lessons learned from the newspaper boy

- Aspects should be defined as modular extensions
  (example: composition filters)
- Aspects must be composable within and among aspect domains
  (example: multiple views, mutual exclusion, locking, counting)
- Aspects must have well-defined semantics
  (example, dispatch, wait, meta filters)
- Aspects must be declarative specifications
  (example filter specifications)

AOP: Future or Fad?

Gregor Kiczales
University of British Columbia

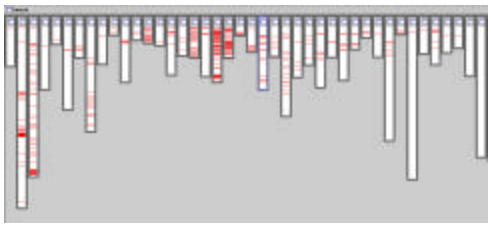
why OOP is a lasting fad

- **a powerful modularity principle**
  - encapsulated objects, hierarchical inheritance, polymorphic operations ("message as goal")
- **supported by language & tool technology**
- **time-proven to**
  - improve modularity of real system designs
  - improve modularity of real programs

but…

problems like…

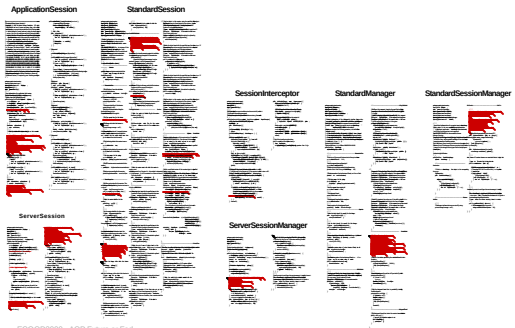
logging is not modularized



- **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

problems like…

session caching is not modularized



AOP is…

- **a modularity principle**
  - crosscutting concerns are important
  - should get explicit support
- **supported by language & tool technology**
  - aspects are crosscutting modules
  - join points are guideposts to coordinate crosscutting
    - points in program text
    - points in dynamic call graph
    - points in dynamic data flow graph
    - …
  - tools can navigate the crosscutting structure

explicit crosscutting structure

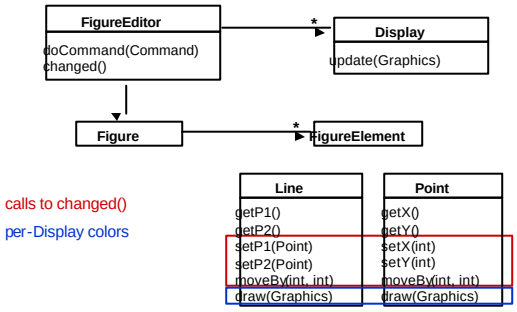
neatly captures public interface of mypackage

```
aspect PublicCallLogging {

  pointcut publicInterface ():
    instanceof(mypackage..*) & receptions(public * *(..));

  static before(): publicInterface() {
    System.out.println(thisJoinPoint + " called.");
  }
  static after() returning (Object r): publicInterface() {
    System.out.println(thisJoinPoint + " returned " + r);
  }
}
```

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 (the structure of the crosscutting)

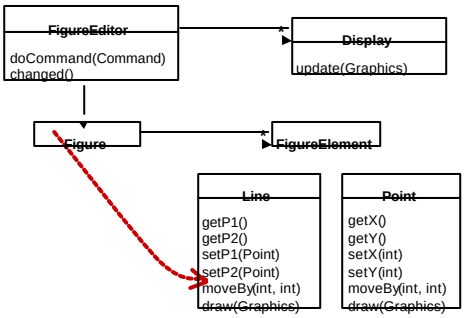
FigureEditor



calls to changed()

per-Display colors

context dependence




```
aspect DeferUpdates {

  // whenever anyone calls a method that moves a figure element
  pointcut movecalls():
      calls(Line, void moveBy(int, int))   |
      calls(Line, void setP1(Point))       |
      calls(Line, void setP2(Point))       |
      calls(Point, void moveBy(int, int))  |
      calls(Point, void setX(int))         |
      calls(Point, void setY(int));

  // this is a call from outside of FigureElememts
  pointcut deferTo():
      !withinall(FigureElement) & movecalls();

  static after(): deferTo() {
    FIGUREEDITOR.update();
  }
}
```


contribution of AOP

✓ **attention to crosscutting concerns**
- crosscutting concerns are inherent
- deserve explicit support in design and implementation

➢ **language support**
- aspects, join points…

? **better programs**
- time will tell
- AspectJ is empirical language & SE research