Modularization of crosscutting concerns

```
interface ShapeI extends Remote {
  double get_x() throws RemoteException ;
  void   set_x(int x) throws RemoteException ;
  double get_y() throws RemoteException ;
  void   set_y(int y) throws RemoteException ;
  double get_width() throws RemoteException ;
  void   set_width(int w) throws RemoteException ;
  double get_height() throws RemoteException ;
  void   set_height(int h) throws RemoteException ;
  void   adjustLocation() throws RemoteException ;
  void   adjustDimensions() throws RemoteException ;
}
public class Shape
       implements ShapeI {
  protected AdjustableLocation loc;
  protected AdjustableDimension dim;
  public Shape() {
    loc = new AdjustableLocation(0, 0);
    dim = new AdjustableDimension(0, 0);
  }
  double get_x() throws RemoteException {
    return loc.x(); }
  void   set_x(int x) throws RemoteException {
    loc.set_x(); }
  double get_y() throws RemoteException {
    return loc.y(); }
  void   set_y(int y) throws RemoteException {
    loc.set_y(); }
  double get_width() throws RemoteException {
    return dim.width(); }
  void   set_width(int w) throws RemoteException {
    dim.set_w(); }
  double get_height() throws RemoteException {
    return dim.height(); }
  void   set_height(int h) throws RemoteException {
     dim.set_h(); }
  void   adjustLocation() throws RemoteException {
    loc.adjust();
  }
  void   adjustDimensions() throws RemoteException {
    dim.adjust();
  }
}
class AdjustableLocation {
  protected double x_,  y_;
  public AdjustableLocation(double x, double y) {
    x_ = x; y_ = y;
  }
  synchronized double get_x() { return x_; }
  synchronized void   set_x(int x) {x_ = x;}
  synchronized double get_y() { return y_; }
  synchronized void   set_y(int y) {y_ = y;}
  synchronized void adjust() {
    x_ = longCalculation1();
    y_ = longCalculation2();
  }
}
class AdjustableDimension {
  protected double width_=0.0, height_=0.0;
  public AdjustableDimension(double h, double w) {
    height_ = h; width_ = w;
  }
  synchronized double get_width() { return width_; }
  synchronized void   set_w(int w) {width_ = w;}
  synchronized double get_height() { return height_; }
  synchronized void   set_h(int h) {height_ = h;}
  synchronized void adjust() {
    width_ = longCalculation3();
    height_ = longCalculation4();
  }
}
```

Instead of writing this

Write this

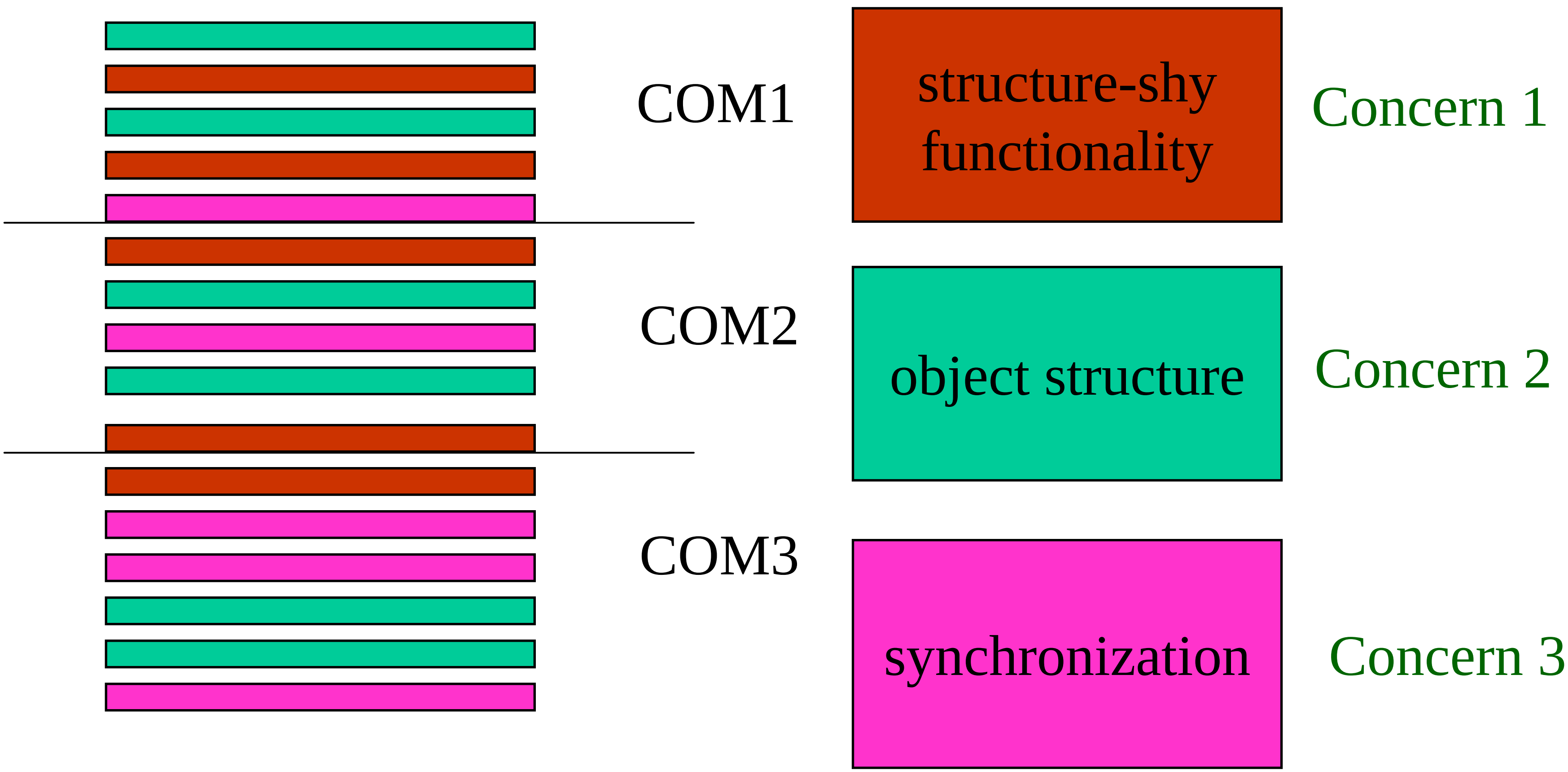
```
public class Shape {
  protected double x_= 0.0,  y_= 0.0;
  protected double width_=0.0, height_=0.0;

  double get_x() { return x_(); }
  void   set_x(int x) { x_ = x; }
  double get_y() { return y_(); }
  void   set_y(int y) { y_ = y; }
  double get_width(){ return width_(); }
  void   set_width(int w) { width_ = w; }
  double get_height(){ return height_(); }
  void   set_height(int h) { height_ = h; }
  void   adjustLocation() {
    x_ = longCalculation1();
    y_ = longCalculation2();
  }
  void   adjustDimensions() {
    width_ = longCalculation3();
    height_ = longCalculation4();
  }
}
```

```
coordinator Shape {
  selfex adjustLocation, adjustDimensions;
  mutex {adjustLocation, get_x, set_x,
                         get_y, set_y};
  mutex {adjustDimensions, get_width, get_height,
                           set_width, set_height};
}
```

```
portal Shape {
  double get_x() {} ;
  void   set_x(int x) {};
  double get_y() {};
  void   set_y(int y) {};
  double get_width() {};
  void   set_width(int w) {};
  double get_height() {};
  void   set_height(int h) {};
  void   adjustLocation() {};
  void   adjustDimensions() {};
}
```

Scattering: count number of components to which color goes

ordinary program | aspect-oriented prog.



AspectJ

- Xerox PARC: Gregor Kiczales et al.: lingua franca of AOP.
- First version: Crista Lopes (member of Demeter group): implementing both COOL and RIDL in a general purpose AO language (early AspectJ version).
- Model: join points, pointcuts, advice.

From Demeter to AspectJ

Demeter (for Scheme or C++ or Java or Flavors)

- Visitor method sig.
  - set of execution points of traversals
  - specialized for traversals (nodes, edges)
  - where to enhance
- Visitor method bodies
  - how to enhance

AspectJ

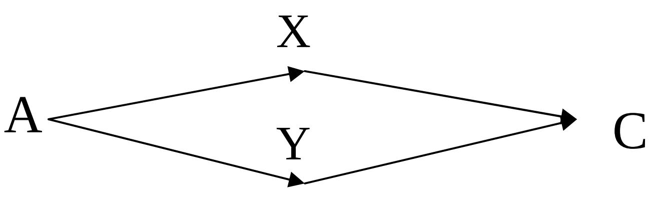
- Pointcut
  - set of execution points of any method, …
  - rich set of primitive pointcuts: this, target, call, … + set operations
  - where to enhance
- Advice
  - how to enhance

Flow Expressions

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- We can use them in three different graphs relevant to programming:
  – call trees: subset of nodes
  – class graphs: subset of nodes
  – object trees: subgraph

Flow Expressions and AOP

- They are a basic cross-cutting construct for defining subgraphs.



- merge(join([A,X],[X,C]), join([A,Y],[Y,C])) defines a subgraph of a larger graph whose ad-hoc description cuts across many nodes or edges.
- succinct encapsulations of subgraphs related to some aspect.

Flow expressions

- are abstractions of some aspect whose ad-hoc description would cut across many nodes or edges of a graph.
- define sets of join points based on connectivity in graph.
- offer pointcut designator reduction (high-level pointcut designator): free programmer from details of some graph representing some aspect.

Definitions for Flow Expressions

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- Source([A,B]) = A
- Target([A,B]) = B
- Source(join(D1,D2))=Source(D1)
- Target(join(D1,D2))=Target(D2)
- Source(merge(D1,D2))=Source(D1)
- Target(merge(D1,D2))=Target(D1)

Well-formed Flow Expressions

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- WF([A,B]) = true // well-formed
- WF(join(D1,D2))=WF(D1) && WF(D2) && Target(D1) = Source(D2)
- WF(merge(D1,D2))= WF(D1) && WF(D2) && Source(D1)=Source(D2) && Target(D1)=Target(D2)

Interpretation of traversal strategies

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- A and B are pointcut designators.
- [A,B]: the set of B-nodes reachable from A-nodes.
- join(D1,D2): the set of Target(D2)-nodes reachable from Source(D1)-nodes following D1 and then following D2.

Interpretation of traversal strategies

- merge(D1,D2): the union of the set of Target(D1)-nodes reachable from Source(D1)-nodes following D1 and the set of Target(D2)-nodes reachable from Source(D2)-nodes following D2.

Meaning in Class graph

- D
- [A,B]
- join(D1,D2)
- merge(D1,D2)

- PathSet(D)
- Paths(A,B)
- PathSet(D1).PathSet(D2)
- PathSet(D1) || PathSet(D2)

flow expressions are called traversal strategies

Demeter/C++
DemeterJ
DJ

Object tree

- D
- [A,B]

- subgraph of O
- subgraph of O consisting of all paths from an A-node to a B-node, including prematurely terminated paths.

Object tree

- D
- join(D1,D2)

- subgraph of O
- subgraph of O consisting of all paths following D1 and those reaching Target(D1) concatenated with all paths following D2.

Object tree

- D
- merge(D1,D2)

- subgraph of O
- subgraph of O consisting of all paths following D1 or following D2.

Purposes of strategies

- DJ
- Traversal
  - strategy graph
  - class graph
  - object graph
- Purposes
  - select og with sg
    - extract node labels
  - select cg with sg

- AspectJ
- General computation
  - strategy call graph
  - static call graph
  - dynamic call graph
- Purposes
  - select dcg with sycg
    - extract node labels
  - select scg with sycg

Purposes of strategies

- DJ + method edges
- Traversal
  - strategy graph
  - class graph
  - object graph
  - argument map
- Purposes
  - select dcg with sg + am
    - extract node labels

Correspondences

• D1	• t(D1)
• from A to B	• flow(A) && B
• from A to *	• flow(A)
• from A via B to C	• flow(flow(A) && B) && C
• from A via B via C to E	• flow(flow(flow(A) && B) && C) && E
• merge(from A via B1 to C, from A via B2 to C)	• (flow(flow(A) && B1) && C) || (flow(flow(A) && B2) && C)
• merge(D1,D2)	• t(D1) || t(D2)
• join(D1,D2)	• flow(t(D1)) && t(D2)
• join (from A to B, from B to C)	• flow(flow(A) && B) && (flow(B) && C)
	• = flow(flow(A) && B) && C

subset(flow(B)) && flow(B) = subset(flow(B))

Theme

- Defining high-level artifact in terms of a low-level artifact without committing to details of low-level artifact in definition of high-level artifact. Low-level artifact is parameter to definition of high-level artifact.
- Exploit structure of low-level artifact.

AspectJ adds

- Generalizes from join points of specialized methods to join points of any method, field access, field modification, etc.
- Uses set operations && and ! combined with a rich set set of primitive pointcuts.
- Generalizes from a flow construct for traversals to a flow construct for arbitrary join points.