Modularization of crosscutting concerns

Instead of writing this

```java
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(); }
}
```

Write this

```java
public class Shape {
    
    public 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();
        }
    }
    
    public class AdjustableDimension {
        protected double width_ = 0.0, height_ = 0.0;
        public AdjustableLocation(double w, double h) {
            width_ = w; height_ = h;
        }
        synchronized double get_width() { return width_; }
        synchronized void set_width(int w) { width_ = w; }
        synchronized double get_height() { return height_; }
        synchronized void set_height(int h) { height_ = h; }
        synchronized void adjust() {
            width_ = longCalculation3();
            height_ = longCalculation4();
        }
    }
}
```
Scattering: count number of components to which color goes.

ordinary program, aspect-oriented prog.

Concern 1: structure-shy functionality

Concern 2: object structure

Concern 3: synchronization
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

• $D ::= [A,B] \mid \text{join}(D1,D2) \mid \text{merge}(D1,D2)$
• 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.

\begin{equation}
\text{merge}(\text{join}([A,X],[X,C]), \text{join}([A,Y],[Y,C]))
\end{equation}

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

• \( D ::= [A,B] \mid \text{join}(D1,D2) \mid \text{merge}(D1,D2) \)
• \( \text{Source}([A,B]) = A \)
• \( \text{Target}([A,B]) = B \)
• \( \text{Source}(\text{join}(D1,D2)) = \text{Source}(D1) \)
• \( \text{Target}(\text{join}(D1,D2)) = \text{Target}(D2) \)
• \( \text{Source}(\text{merge}(D1,D2)) = \text{Source}(D1) \)
• \( \text{Target}(\text{merge}(D1,D2)) = \text{Target}(D1) \)
Well-formed Flow Expressions

- $D ::= [A,B] \mid \text{join}(D1,D2) \mid \text{merge}(D1,D2)$
- $WF([A,B]) = true$  // well-formed
- $WF(\text{join}(D1,D2)) = WF(D1) \land WF(D2) \land \text{Target}(D1) = \text{Source}(D2)$
- $WF(\text{merge}(D1,D2)) = WF(D1) \land WF(D2) \land \text{Source}(D1)=\text{Source}(D2) \land \text{Target}(D1)=\text{Target}(D2)$
Interpretation of traversal strategies

• $D ::= [A,B] \mid \text{join}(D1,D2) \mid \text{merge}(D1,D2)$
• A and B are pointcut designators.
• $[A,B]$: the set of B-nodes reachable from A-nodes.
• $\text{join}(D1,D2)$: the set of Target(D2)-nodes reachable from Source(D1)-nodes following D1 and then following D2.
Interpretation of traversal strategies

• \text{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

- $D$
- $[A,B]$
- $\text{join}(D_1,D_2)$
- $\text{merge}(D_1,D_2)$

- $\text{PathSet}(D)$
- $\text{Paths}(A,B)$
- $\text{PathSet}(D_1) . \text{PathSet}(D_2)$
- $\text{PathSet}(D_1) \parallel \text{PathSet}(D_2)$
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.

flow expressions are called traversal strategies
Object tree

- $D$
- $\text{join}(D_1, D_2)$
- subgraph of $O$
- subgraph of $O$ consisting of all paths following $D_1$ and those reaching $\text{Target}(D_1)$ concatenated with all paths following $D_2$. 
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 \textit{dcg} with sg + am
    • extract node labels
Correspondences

- D1
- from A to B
- from A to *
- from A via B to C
- from A via B via C to E
- merge(from A via B1 to C, from A via B2 to C)
- merge(D1,D2)
- join(D1,D2)
- join (from A to B, from B to C)

- t(D1)
- flow(A) && B
- flow(A)
- flow(flow(A) && B) && C
- flow(flow(flow(A) && B) && C) && E
- (flow(flow(A) && B) && C) || (flow(flow(A) && B2) && C)
- t(D1) || t(D2)
- flow(t(D1)) && t(D2)
- 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.