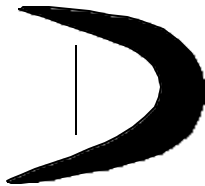


# AOSD 2002 Tutorial: Demeter

## Aspect-Oriented Programming of Traversal-Related Concerns in Java



4/7/02

Demeter Research Group

Karl Lieberherr, Doug Orleans,  
Johan Ovinger, John Sung,  
Mitchell Wand, Pengcheng Wu  
Aspects and Demeter



1



## Overview

- DJ introduction using AOP concepts
- How we got to AOP
- AspectJ and DJ

© 2002, by authors

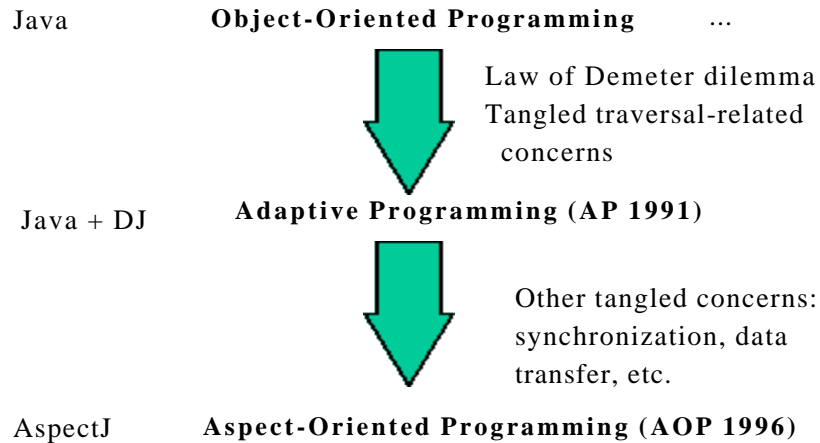
4/7/02

Aspects and Demeter

2

# Technology Evolution view of Demeter Research Group

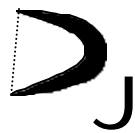
Xerox, Twente,  
IBM, Austin,



4/7/02

AOO / Demeter / NU

3



## Overview

- Aspect-oriented Programming in pure Java using the DJ library
  - AP concepts = DJ classes
  - Meaning of a traversal
  - Case study: semantic checking
  - Patterns for programming in DJ style
  - Generic programming with DJ
  - Adding traversals to AspectJ (with AspectJ team)

4/7/02

Aspects and Demeter

4

# Modularization of crosscutting concerns

```
interface Shape 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;
    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 Shape {
    implements AdjustableLocation
    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(x);
    }
    double get_y() throws RemoteException {
        return loc.y;
    }
    void set_y(int y) throws RemoteException {
        loc.set_y(y);
    }
    double get_width() throws RemoteException {
        return dim.width;
    }
    void set_width(int w) throws RemoteException {
        dim.set_w(w);
    }
    double get_height() throws RemoteException {
        return dim.height;
    }
    void set_height(int h) throws RemoteException {
        dim.set_h(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 = longCalculation();
        y = longCalculation();
    }
}

class AdjustableDimension {
    protected double width, height;
    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 = longCalculation();
        height = longCalculation();
    }
}

```

```
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 = longCalculation();
        y = longCalculation();
    }
    void adjustDimensions() {
        width = longCalculation();
        height = longCalculation();
    }
}

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

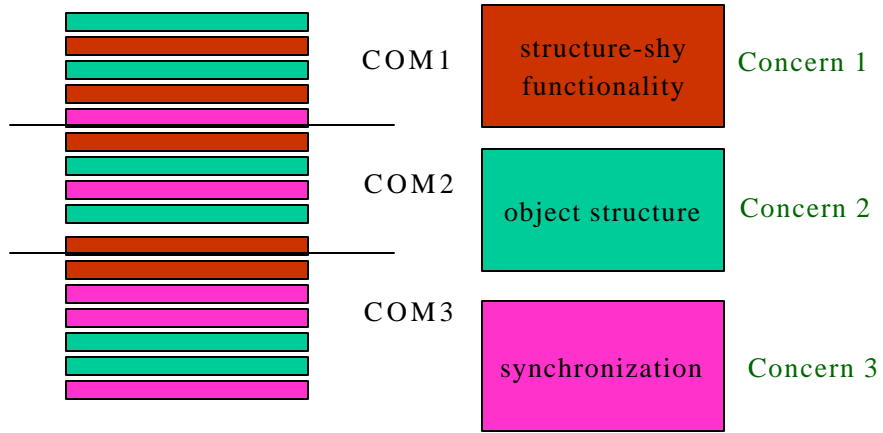
portal Shape {
    double get_x() {}
    void set_x(int x) {}
    double get_y() {}
    void set_y(int y) {}
    double get_width() {}
    void set_width(int w) {}
    double get_height() {}
    void set_height(int h) {}
    void adjustLocation() {}
    void adjustDimensions() {}
}

```

Write this

Instead of writing this

# Scattering: count number of components to which color goes





## Problem addressed

- Encapsulation of traversal-related concerns
  - Those are concerns that involve a group of collaborating objects that are connected by has-a relationships or zero-argument methods.
  - Control scattering of traversal-related concerns and tangling with other concerns
- Construct useful programs without knowing exactly what data types are involved

4/7/02

Aspects and Demeter

7



## How is the problem solved?

### Use an AOP System

- Need to talk about traversal join points:
  - sets of points (called hooks or **pointcuts**) in the execution of a traversal where additional semantics will be specified
  - a means of specifying the semantics at those join points (called an enhancement or **advice**)

```
public void before(Person host){ r++; }  
public void before(Object host){ host.foo() }
```

4/7/02

Aspects and Demeter

8

## An AOP System

- has 3 critical elements
  - what are the join points
  - means of **identifying** join points (**pointcuts**)
  - means of **specifying semantics at** join points (**advice**)

4/7/02

Aspects and Demeter

9

all, **pointcuts**, **advice**, aspects, weaves

## An AOP System (in more detail)

- what is the set of all join points
- means of **identifying** join points (**pointcuts**)
- means of **specifying semantics at** join points (**advice**)
- encapsulated units combining **pointcuts** and **advice** (aspects)
- method of attachment of units (weaves)

Pointcuts and advice are sometimes overlapping. Pointcuts might define an initial behavior plus a set of join points in that behavior.

4/7/02

Aspects and Demeter

10

*all*, pointcuts, advice, aspects, weaves

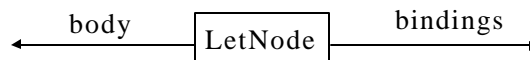
## DJ: set of all join points

- All points during the execution of a traversal algorithm on an object
- All nodes and edges of object graph slices for a fixed traversal algorithm
- An object graph slice is a subgraph of an object graph (selected by a traversal specification)

4/7/02

Aspects and Demeter

11



## DJ pointcuts

*all*, *pointcuts*, advice, aspects, weaves

- **visitor method signatures**
- the signatures define the points during the traversal when additional behavior needs to be executed.

```
Object around(LetNode l, Subtraversal st) {
    Object[] bg=st.applyElement("bindings");
    Object[] bd=st.applyElement("body");
    return checkIfDeclared(bg,bd);
}
```

4/7/02

Aspects and Demeter

12

all, pointcuts, *advice*, aspects, weaves

## DJ *advice*

- each visitor method body is *advice* on the *pointcut* specified by the method signature.

```
Object around(LetNode l, Subtraversal st) {  
    Object[] bg=st.applyElement("bindings");  
    Object[] bd=st.applyElement("body");  
    return checkIfDeclared(bg,bd);  
}
```

4/7/02

Aspects and Demeter

13

all, pointcuts, *advice*, *aspects*, weaves

## DJ *aspects*

- a visitor class is a package of pointcuts and *advice*, i.e., an *aspect*
- when you use a visitor in a traversal of an object graph (in *traverse*) then each pointcut is intersected with the traversal pointcut

```
class MyVisitor extends Visitor { int r;  
    public void before(Person host){ r++; }  
    public void start() { r = 0; }  
    public Object getReturnValue() {return new Integer (r);}}
```

4/7/02

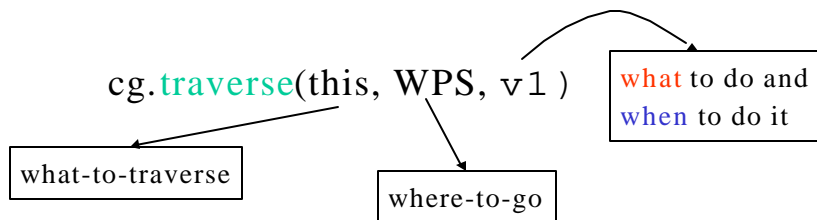
Aspects and Demeter

14

all, pointcuts, advice, aspects, *weaves*

## DJ weaves

- to attach an aspect you call traverse with an aspect (visitor).
- traverse expression attaches the aspect to an object graph slice (a subgraph of an object)



4/7/02

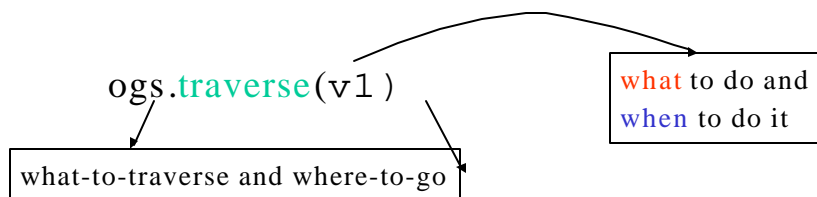
Aspects and Demeter

15

all, pointcuts, advice, aspects, *weaves*

## DJ weaves

- to attach an aspect you call traverse with an aspect (visitor).
- traverse expression attaches the aspect to an object graph slice (a subgraph of an object)



4/7/02

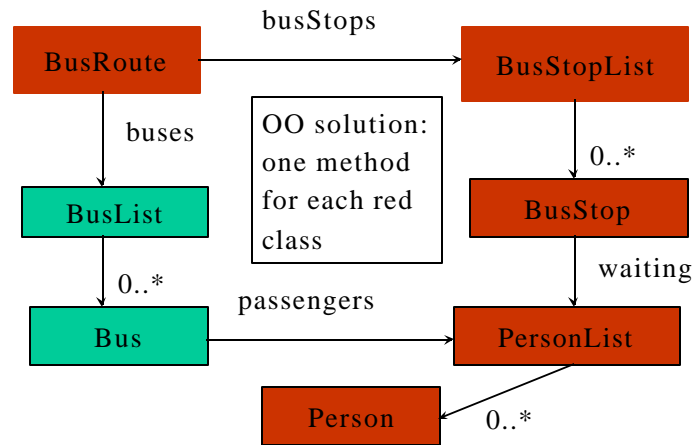
Aspects and Demeter

16



## Collaborating Classes

find all persons waiting at any bus stop on a bus route



4/7/02

Aspects and Demeter

17

all, **pointcuts**, **advice**, aspects, weaves

## DJ complete aspect example

```
class BusRoute {
    int countPersons(ClassGraph cg) {
        String WPS="from BusRoute via BusStop to Person"
        Integer result = (Integer)
        cg.traverse(this, WPS, new Visitor(){ int r;
        public void before(Person host){ r++; }
        public void start() { r = 0; }
        public Object getReturnValue()
            {return new Integer (r);}
        }); return result.intValue();
    }
}
```

4/7/02

Aspects and Demeter

18



continued

```
// Prepare the class graph
ClassGraph classGraph = new ClassGraph();
BusRoute aBusRoute = ...;
int r = aBusRoute.countPersons(classGraph);
```

4/7/02

Aspects and Demeter

19

## How we got to Aspect-Oriented Programming

- Started simple: traversal-seeded programs:  
Law of Demeter dilemma.
- Talk generically about points in the execution  
of a traversal program.
- Generically means: parameterize program by  
an abstraction of its execution (class graph).

4/7/02

Aspects and Demeter

20



## Why Traversal Strategies?

- Law of Demeter: a method should talk only to its friends:
  - arguments and part objects (computed or stored)
  - and newly created objects



- Dilemma:
  - Small method problem of OO (if followed) or
  - Unmaintainable code (if not followed)
- Traversal strategies are the solution to this dilemma

4/7/02

Aspects and Demeter

21



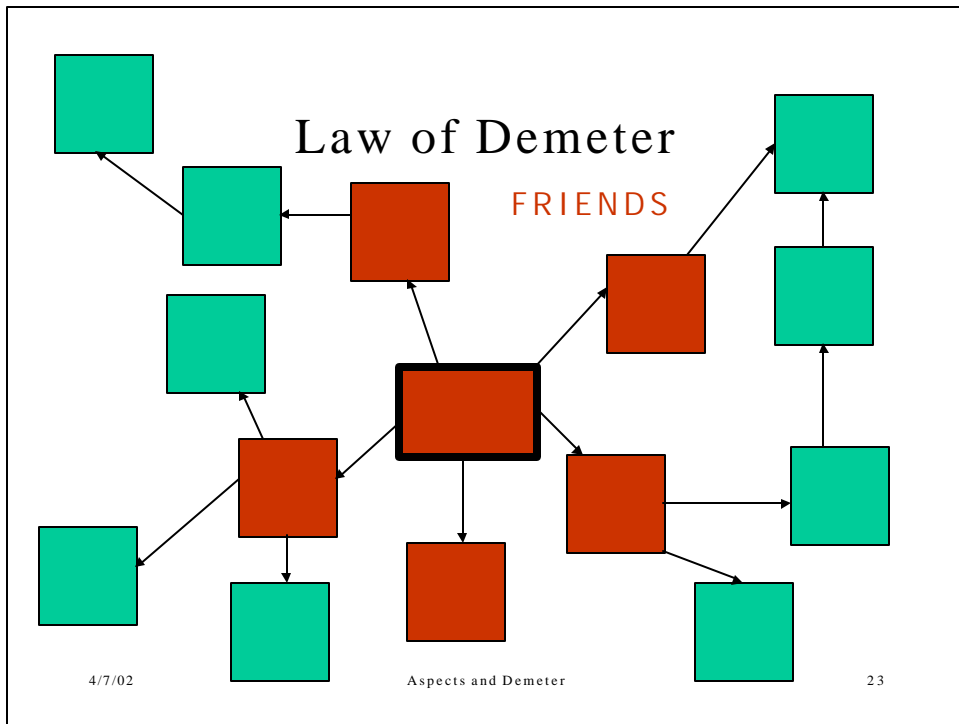
## Law of Demeter Principle

- **Each unit should only communicate with a limited set of other units: only units “closely” related to the current unit.**
- “Each unit should only talk to its friends.”  
“Don’t talk to strangers.”
- Main Motivation: Control information overload. We can only keep a limited set of items in short-term memory.

4/7/02

Aspects and Demeter

22



D

## Application to OO: generic application

- Unit = method
  - closely related =
    - methods of class of `this/self` and other argument classes
    - methods of immediate part classes (classes that are return types of methods of class of `this/self`)

4/7/02 Aspects and Demeter 24



Law of Demeter: talk only to your friends

object traversal

**when:** select points in  
object traversal

**what:** actions  
to perform before and after  
node and edge visits

Scattering / Tangling



## Other aspect systems from our Research Group

- COOL (1993) and RIDL (1994) by Crista Lopes
- Starting summer 1995: in collaboration with Xerox PARC (Gregor Kiczales)
- COOL and RIDL were the breeding ground for the first AspectJ weaver and the DemeterJ weaver.



thread synchronization

**when:** select method calls

**what:** actions to perform before and after method calls to maintain synchronization variables

Scattering / Tangling

COOL: Crista Lopes: 1993

4/7/02

Aspects and Demeter

27

all, **pointcuts**, **advice**, aspects, weaves

## COOL example

```
coordinator BoundedBuffer {  
  selfex put, take;  
  mutex {put, take}  
  condition empty=true, full=false;  
  
  put requires (!full) {  
    on exit {empty=false;  
      if (usedSlots==array.length)  
        full=true; }}  
  take requires (!empty) {  
    on exit {full=false;  
      if (usedSlots==0)  
        empty=true; }}  
}
```

4/7/02

Aspects and Demeter

28



selective marshaling

**when:** select method calls

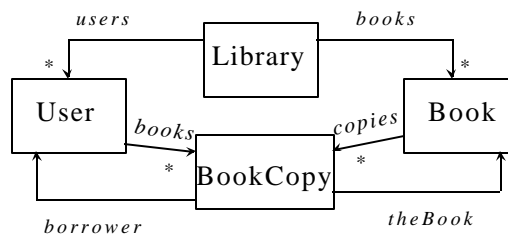
**what:** actions to perform to transmit arguments and result

Scattering / Tangling

RIDL: Crista Lopes: 1994

all, **pointcuts**, **advice**, aspects, weaves

## RIDL example: Selective Marshaling



```
portal Library {
  BookCopy getBook(User u, String title) {
    return: copy {BookCopy bypass borrower,
                  Book bypass copies;}
    u: copy {User bypass books;}
  }
  Book findBook(String title) {
    return: copy {Book bypass copies, ps;}
  }
}
```

# Modularization of crosscutting concerns

```
interface Shape 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 Shape {
    implements AdjustableLocation {
    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(x);
    }
    double get_y() throws RemoteException {
        return loc.y;
    }
    void set_y(int y) throws RemoteException {
        loc.set_y(y);
    }
    double get_width() throws RemoteException {
        return dim.width;
    }
    void set_width(int w) throws RemoteException {
        dim.set_w(w);
    }
    double get_height() throws RemoteException {
        return dim.height;
    }
    void set_height(int h) throws RemoteException {
        dim.set_h(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 = longCalculation();
        y = longCalculation();
    }
}

class AdjustableDimension {
    protected double width, height;
    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 = longCalculation();
        height = longCalculation();
    }
}

```

```
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 = longCalculation();
        y = longCalculation();
    }
    void adjustDimensions() {
        width = longCalculation();
        height = longCalculation();
    }
}

coordinator Shape {
    setloc adjustable, adjustDimensions
    mutex [adjustLocation, get_x, set_x,
           get_y, set_y];
    mutex [adjustDimensions, get_width, get_height,
           set_width, set_height];
}

portal Shape {
    double x, y;
    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() {}
}

```

Write this

Instead of writing this

Aspects and Demeter

31

## What is the problem? Tangling!

During implementation separate issues are mixed together

During maintenance individual issues need to be factored out of the tangled code

4/7/02

Aspects and Demeter

32



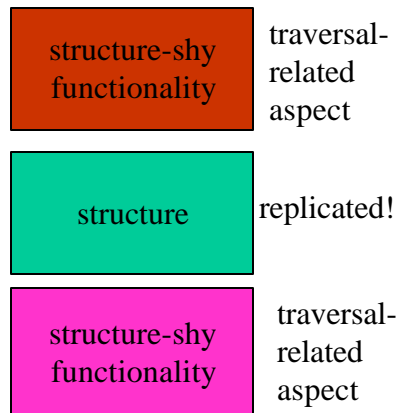
## Crosscutting in Java with DJ

generated  
Java program



4/7/02

DJ program



Aspects and Demeter

33

◆ Many functional concerns involve multiple objects that need to be traversed from a starting object. Such concerns are called traversal-related concerns.

◆ Subconcerns expressing

◆ the traversal and

◆ **when** to take action during the traversal and

◆ **what** action to take

◆ Traversal related concerns deserve a traversal-specific aspect language which happens to fit into Java.

4/7/02

Aspects and Demeter

34

subcomputation = join points related to traversing through the objects guided by traversal specification and class graph.

	AspectJ	DJ
Kind	Dynamic	Dynamic
On What	Dynamic call graph of base program	Dynamic call graph of a subcomputation of base program
<b>When</b>	Pointcuts	Signatures of visitor methods
<b>What</b>	Before / around / after advice	Before / around / after visitor method body

4/7/02

Aspects and Demeter

35



## Comparison of pointcuts: AspectJ / **DJ**

- `target(a) && call(* t(*));`
- **DJ: A a**
- `this(a) && target(b) && call(* t1(*));`
- **DJ: A a, B b**

4/7/02

Aspects and Demeter

36

# AspectJ

- Xerox PARC: Gregor Kiczales et al.: lingua franca of AOP.
- One of the first versions: 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.

4/7/02

Aspects and Demeter

37



## From Demeter to AspectJ

Demeter (for C++ or Java)

AspectJ

- Visitor method sig.

- set of execution points of *traversals*

- **specialized for traversals (nodes, edges)**

- where to enhance

- Visitor method bodies

- how to enhance

- 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

4/7/02

Aspects and Demeter

38



### From Demeter to AspectJ

## Java+DJ

## AspectJ

aspect name

```
class S{
  void collect(ClassGraph cg,
               String constraint){
    String s = "from S" +
              constraint + "to T";

    cg.traverse(this, s,
                new Visitor(){ ... ;
                public void before(T h){...}
                public void start() {...}});
  }
}
```

```
aspect Traversal_t { // t()
  //declare traversal t :
  // "from S" + c + "to T"; ...
}
aspect Collecting {
  pointcut start() : ... ;
  pointcut visitingT(T h):
    call(void t())
    && target(h);
  before():visitingT() { ... }
  before():start() { ... }
}
```

4/7/02

green: pointcut  
purple: advice

Aspects and Demeter

39



## Notice Difference

- In DJ
  - weave is done in Java code
  - `cg.traverse(this,s,v);`
- In AspectJ
  - weave is done on command line:
  - `ajc`
  - `Traversal_t.java`
  - `Collecting.java`
  - ...

but can use visitor in traversal specification!

4/7/02

Aspects and Demeter

40

AspectJ: refers to existing join points

Demeter: defines new join points in traversal

AspectJ

Java+DJ

aspect name

```
aspect SimpleTracing{
  pointcut traced():
    call(void D.update()) ||
    call(void D.repaint());
  before() : traced(){
    println("Entering:" +
      thisJoinPoint);}
}
```

blue: pointcut  
red: advice

```
class Source{
  HashSet collect(ClassGraph cg,
    String constraint){
    String s = "from Source" +
      constraint + "to Target";
    return (HashSet)
      cg.traverse(this, s,
        new Visitor(){ ... ;
          public void before
            (Target h) { ... }
          public void start() {...}});
  }
```

4/7/02

Aspects and Demeter

41



## AP

- Late binding of data structures
- Programming without accidental data structure details yet handling all those details on demand without program change
- Reducing representational coupling

4/7/02

Aspects and Demeter

42



## Concepts needed (DJ classes)

- *ClassGraph*
- *Strategy*
- *Visitor*
- TraversalGraph
- ObjectGraph
- ObjectGraphSlice

*important*

for advanced programming

4/7/02

Aspects and Demeter

43

AOSD: not every concern fits  
into a component: **crosscutting**

	CM1	CM2	CM3	CM4	CM5	CM6	
CR1	x						components
CR2		x					
CR3			x				
CR4	x	x		x	x		
concerns							

Goal: encapsulate "crosscutting" concerns

4/7/02

Aspects and Demeter

44



## AP history

- Programming with partial data structures = propagation patterns
- Programming with participant graphs = high-level class graphs
- Programming with object slices
  - partial objects determined by traversal specifications

4/7/02

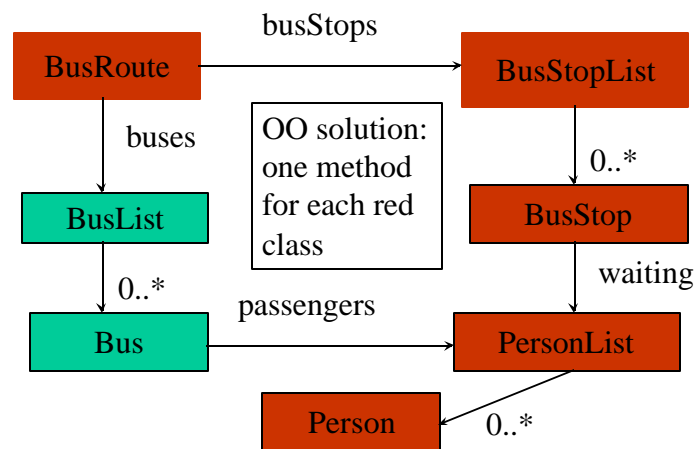
Aspects and Demeter

45



## Collaborating Classes

find all persons waiting at any bus stop on a bus route



4/7/02

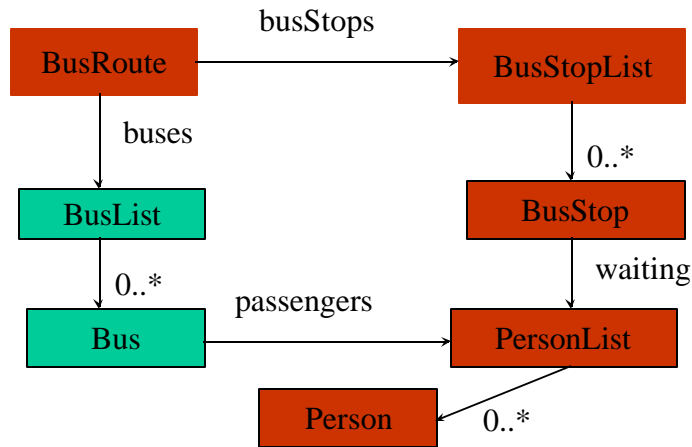
Aspects and Demeter

46

find all persons waiting at any bus stop on a bus route

# Traversal Strategy

from BusRoute via BusStop to Person



4/7/02

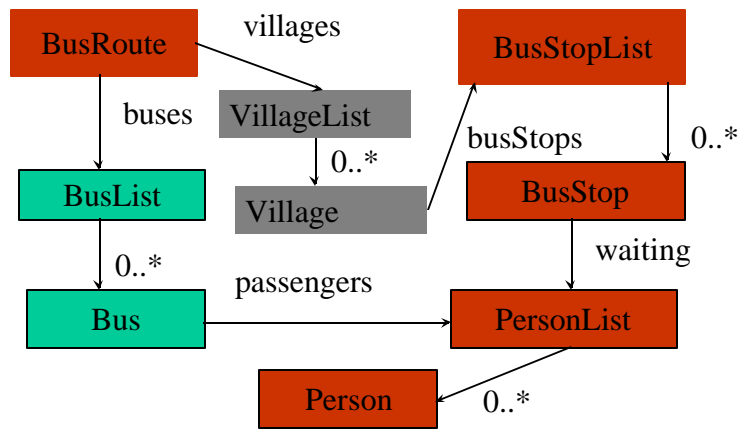
Aspects and Demeter

47

find all persons waiting at any bus stop on a bus route

# Robustness of Strategy

from BusRoute via BusStop to Person

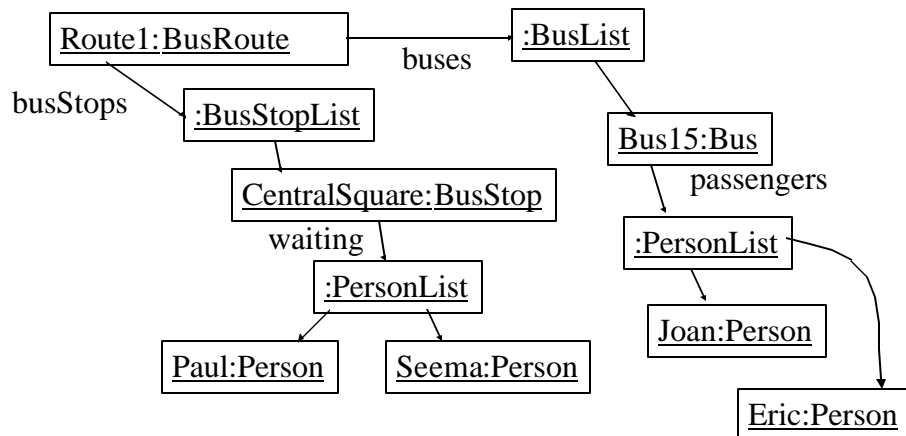


4/7/02

Aspects and Demeter

48

## ObjectGraph: in UML notation

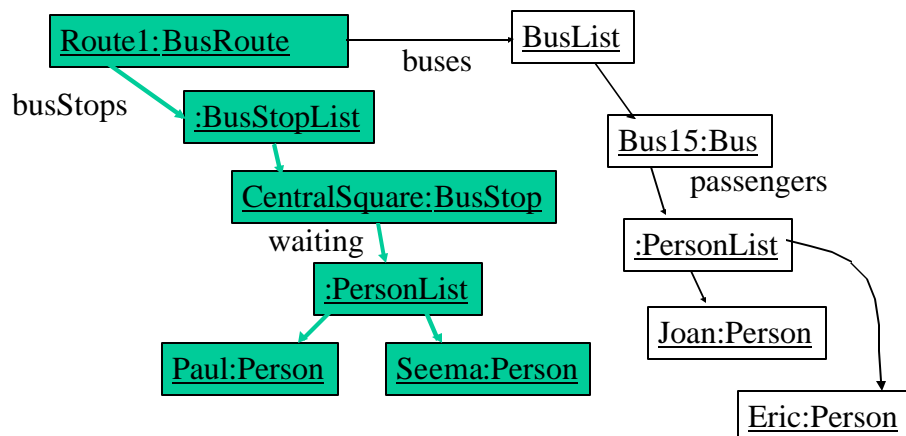


4/7/02

Aspects and Demeter

49

## ObjectGraphSlice



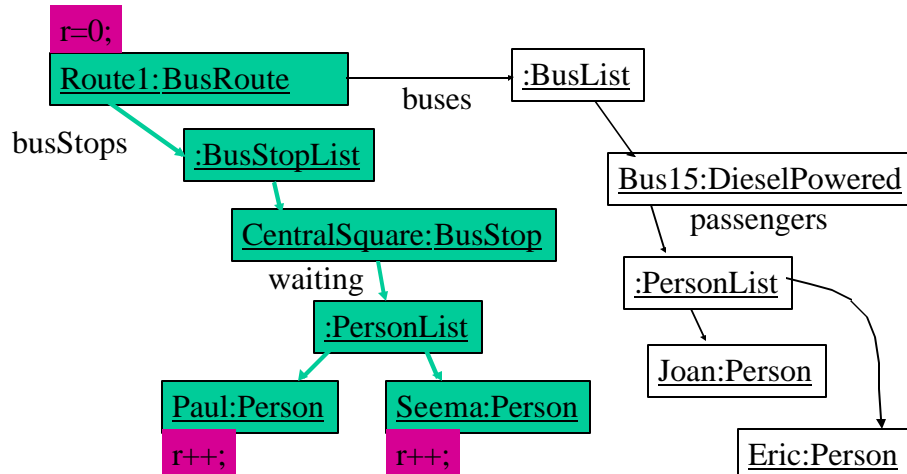
4/7/02

Aspects and Demeter

50

overall graph: object structure; green graph: traversal; purple: advice

## Why crosscutting?



4/7/02

Aspects and Demeter

51

## Writing Adaptive Programs with Strategies (DJ=pure Java)

String WPS="from BusRoute via BusStop to Person"

```
class BusRoute {
    int countPersons(ClassGraph cg) {
        String WPS="from BusRoute via BusStop to Person"
        Integer result = (Integer)
        cg.traverse(this, WPS, new Visitor(){ int r;
        public void before(Person host){ r++; }
        public void start() { r = 0;}
        public Object getReturnValue()
            {return new Integer ( r);}
        }); return result.intValue();}
}
```

4/7/02

Aspects and Demeter

52

## Writing Adaptive Programs with Strategies (DJ=pure Java)

```
// Prepare the class graph
ClassGraph classGraph = new ClassGraph();
BusRoute aBusRoute = new BusRoute(...);
int r = aBusRoute.countPersons(classGraph);
```

4/7/02

Aspects and Demeter

53



### Goal of DJ

- Focus on crosscutting traversal-related concerns: involve a group of collaborating objects which are manipulated to implement a behavior.
- Provide a Java library to cleanly encapsulate crosscutting traversal-related concerns whose ad hoc implementation would be scattered across many classes.

4/7/02

Aspects and Demeter

54



## Solves Open Problem in AOP for Behavioral Aspects in Java

- The ClassGraph-Aspect-Freezing problem
  - When we have n aspects and the class graph changes, we potentially need to update all n aspects.
  - DJ allows us to loosely couple behavioral aspects to the class graph.
  - And this is all done in Java.

4/7/02

Aspects and Demeter

55

## Writing Adaptive Programs with Strategies (DJ=pure Java)

String WPStrategy="from BusRoute via BusStop to Person"

```
class BusRoute {
    int countPersons(TraversalGraph WP) {
        Integer result = (Integer)
        WP.traverse(this, new Visitor(){ int r;
        public void before(Person host){ r++; }
        public void start() { r = 0;}
        public Object getReturnValue()
            {return new Integer ( r);}
        }); return result.intValue();}
}
```

4/7/02

Aspects and Demeter

56

## Writing Adaptive Programs with Strategies (DJ=pure Java)

String WPStrategy="from BusRoute via BusStop to Person"

```
// Prepare the traversal for the current class graph
ClassGraph classGraph = new Y();
TraversalGraph WPTraversal = new TraversalGraph
(WPStrategy, classGraph);

int r = aBusRoute.countPersons(WPTraversal);
```

4/7/02

Aspects and Demeter

57

## Writing Adaptive Programs with Strategies (DJ=pure Java)

```
class Utility {
    static
    int countPersons(ObjectGraphSlice countSlice){
        Integer result = (Integer)
        countSlice.traverse(new Visitor(){ int r;
        public void before(Person host){ r++; }
        public void start() { r = 0;}
        public Object getReturnValue()
        {return new Integer ( r);}
        });
        return result.intValue();
    }
}
```

4/7/02

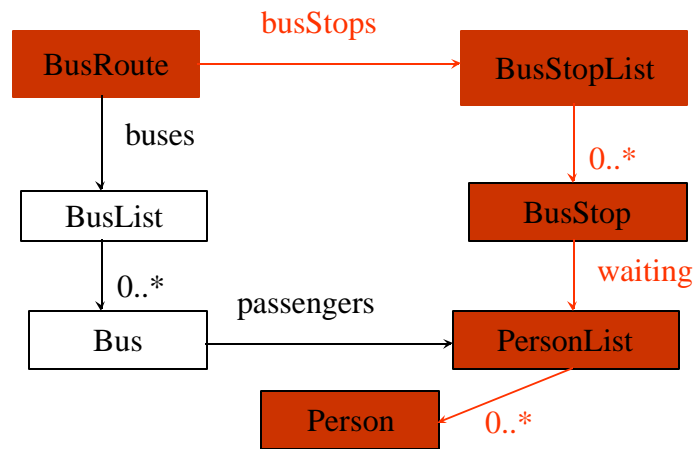
Aspects and Demeter

58

find all persons waiting at any bus stop on a bus route

## TraversalGraph

from BusRoute via BusStop to Person



4/7/02

Aspects and Demeter

59



## Applications of Traversal Strategies

- Program Kinds in DJ
  - AdaptiveProgram<sub>Traditional</sub>(ClassGraph)
    - strategies are part of program: DemeterJ, Demeter/C++
  - AdaptiveProgram<sub>Dynamic</sub>(Strategies, ClassGraph)
    - strategies are a parameter. Even more adaptive.
  - AdaptiveProgram<sub>TraditionalOptimized</sub>(TraversalGraphs)
    - strategies are a parameter. Reuse traversal graphs.
  - AdaptiveProgram<sub>DJ</sub>(ObjectGraphSlices)
    - strategies are a parameter. Reuse traversal graph slices.

4/7/02

Aspects and Demeter

60



## Example

- For data member access:
- `C c = (C) Main.cg.fetch(this, "from A via B to C");`

4/7/02

Aspects and Demeter

61

## Understanding the meaning of a strategy

- Classes involved: Strategy, ObjectGraph, ObjectGraphSlice, ClassGraph
- We want to define the meaning of a Strategy-object for an ObjectGraph-object as an ObjectGraphSlice-object (a subgraph of the ObjectGraph-object). Minimal attention necessary will be given to ClassGraph-object.

4/7/02

Aspects and Demeter

62



## Searching for Reachable Objects

- Task: Given an object o1 of class c1 in an object graph, find all objects of type c2 that are reachable from o1.
- Assumptions: we know the class structure that describes the object graph, but we know nothing else about the object graph except the class of the current object.

4/7/02

Aspects and Demeter

63

## Search using meta information

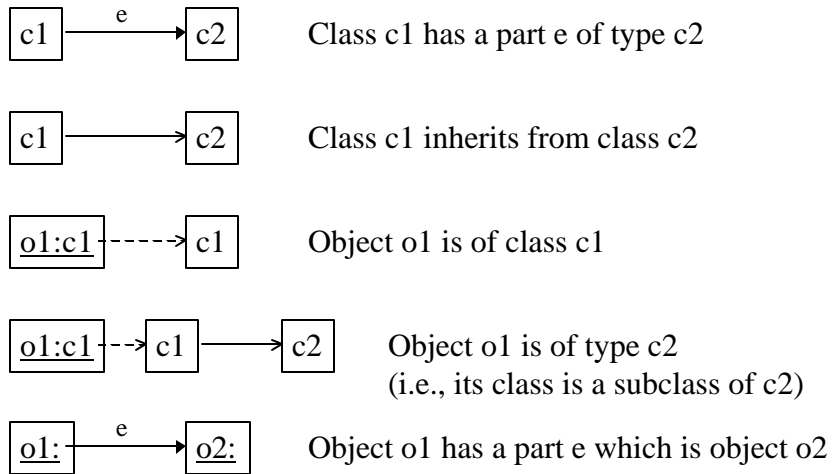
- we could visit the entire object but that
  - would be wasteful or
  - might lead to wrong results

4/7/02

Aspects and Demeter

64

## Classes and Objects: Basic Notations



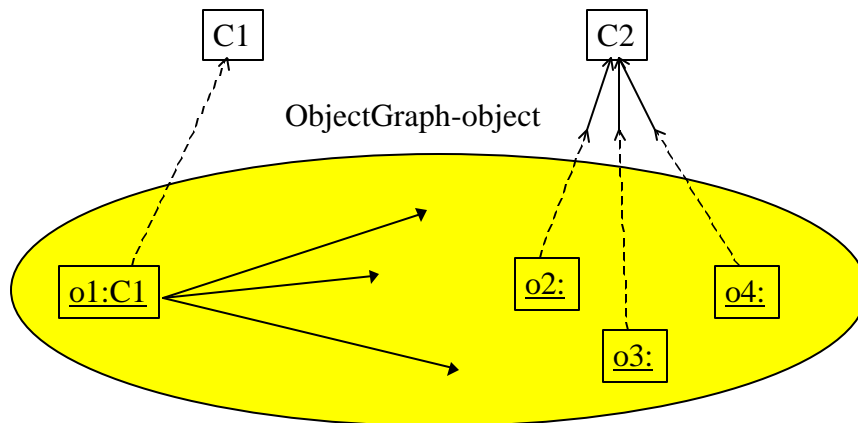
4/7/02

Aspects and Demeter

65



## Finding the first step for the search



Which arrows might lead to an object of type C2?

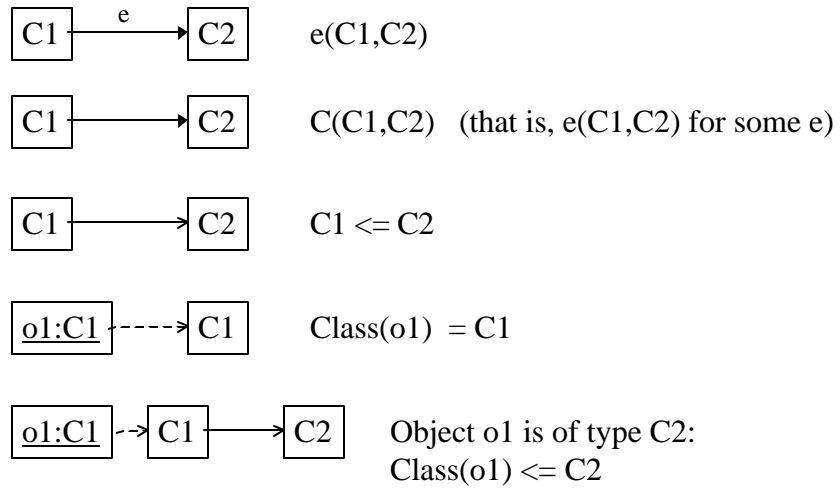
Traversal Strategy:  
from C1 to C2

4/7/02

Aspects and Demeter

66

## Relations between Classes

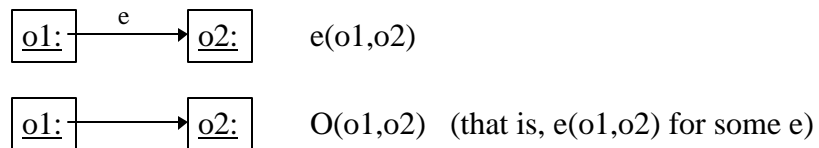


4/7/02

Aspects and Demeter

67

## Relations between Objects



4/7/02

Aspects and Demeter

68

## Operations on Relations

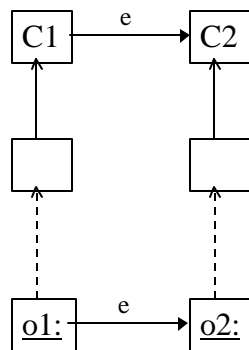
- $R.S = \{(x,z) \mid \text{exists } y \text{ s.t. } R(x,y) \text{ and } S(y,z)\}$
- $R^* = \text{reflexive, transitive closure of } R$

4/7/02

Aspects and Demeter

69

## Possible edges in the object graph



$e(o1, o2)$  implies  
 $\text{class}(o1) (<= .e .=>) \text{class}(o2)$   
in the class graph

“up, over, and down”

$O(o1, o2)$  implies  
 $\text{class}(o1) (<= .C .=>) \text{class}(o2)$   
in the class graph

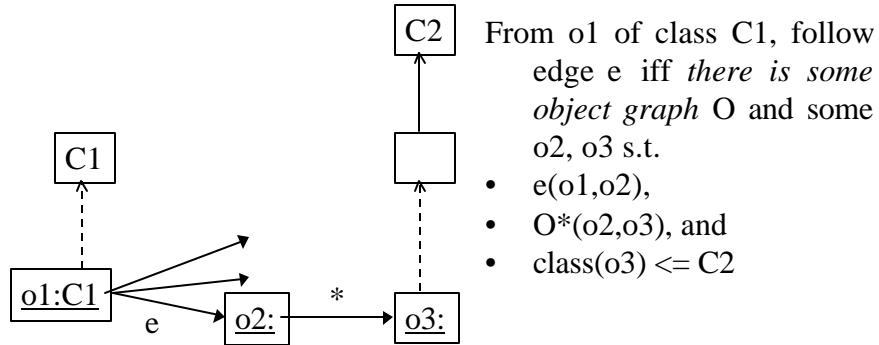
4/7/02

Aspects and Demeter

70



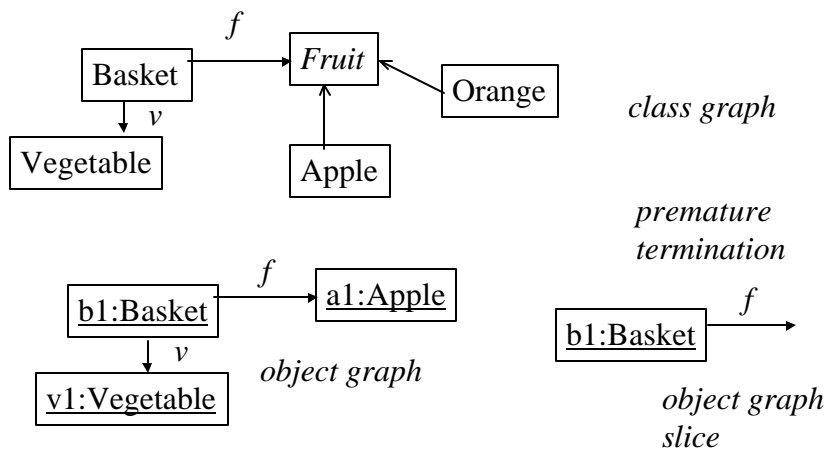
# Which edges to follow to C2?



The existential quantifier “there is some object graph” represents our lack of knowledge about the rest of the object graph

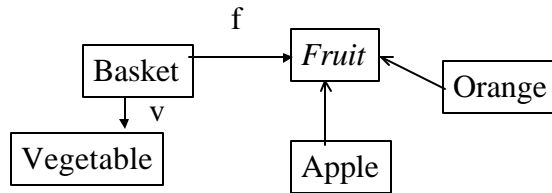
from Basket to Orange

## Example



from Basket to Orange

## Example



*mapping:*

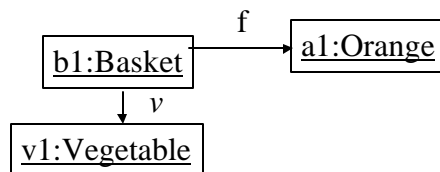
o1 b1

o2 a1

o3 a1

e f

*class graph*



*object graph*

4/7/02

Aspects and Demeter

73

## Lack of Knowledge

- Objects of a given class may be very different.
- We want to go down edges without looking ahead!
- We don't want to go down edges that are guaranteed to be unsuccessful (never reaching a target object).

4/7/02

Aspects and Demeter

74

## Object graph conforms to class graph

- The object graph  $O$  must follow the rules of the class graph: the object graph cannot contain more information than the class graph allows.

For all edges  $e(o1,o2)$  in the object graph:  
 $e(o1, o2)$  implies  
 $class(o1) \langle = .e . => \rangle class(o2)$   
in the class graph

4/7/02

Aspects and Demeter

75



## From dynamic to static characterization

From  $o1$  of class  $c1$ , follow edge  $e$  iff there is some object graph  $O$  and some  $o2, o3$  s.t.

- $e(o1,o2)$ ,
- $O^*(o2,o3)$ , and
- $class(o3) \leq c2$



From  $o1$  of class  $c1$ , follow edge  $e$  iff there are classes  $c', c''$  s.t.

- $c1 \langle = .e . => \rangle c'$
- $c' \langle = .C . => \rangle^* c''$  and
- $c'' \leq c2$

Let  $c'$  be  $class(o2)$ ,  $c''$  be  $class(o3)$



4/7/02

Aspects and Demeter

76



from c1 to c2

## Relational Formulation

From object o of class c1, to get to c2, follow edges in the set

$$\text{POSS}(c1,c2,o)=\{e \mid c1 \leq.e.\Rightarrow (\leq.C.\Rightarrow)^* \leq c2 \}$$

Can easily compute these sets for every c1, c2 via transitive-closure algorithms.

POSS = abbreviation for: following these edges it is still **possible** to reach a c2-object for some c1-object rooted at o.

4/7/02

Aspects and Demeter

77

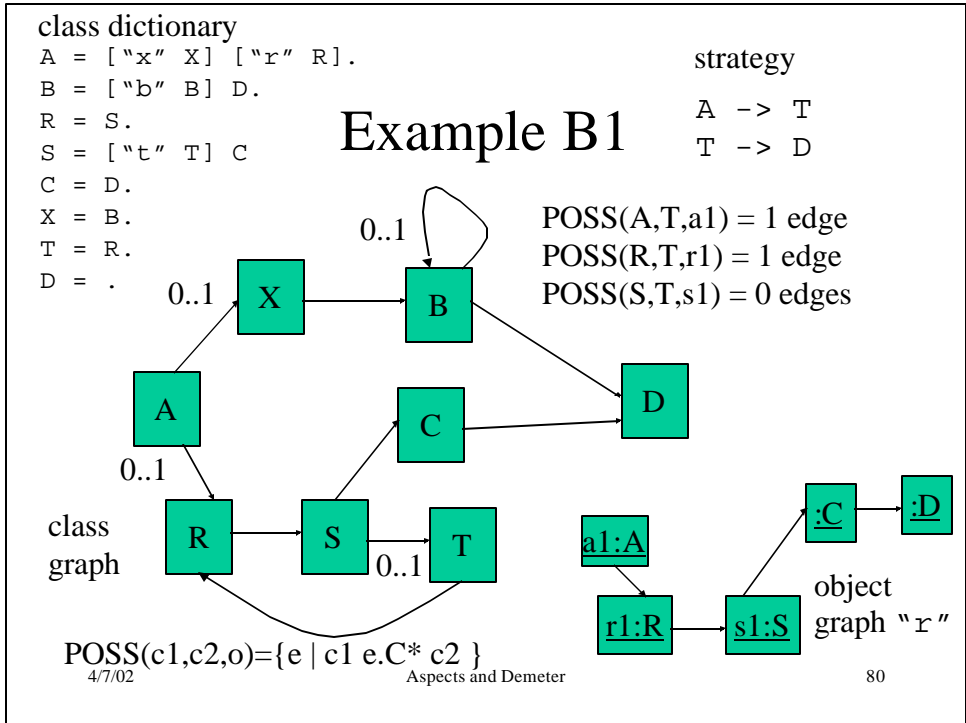
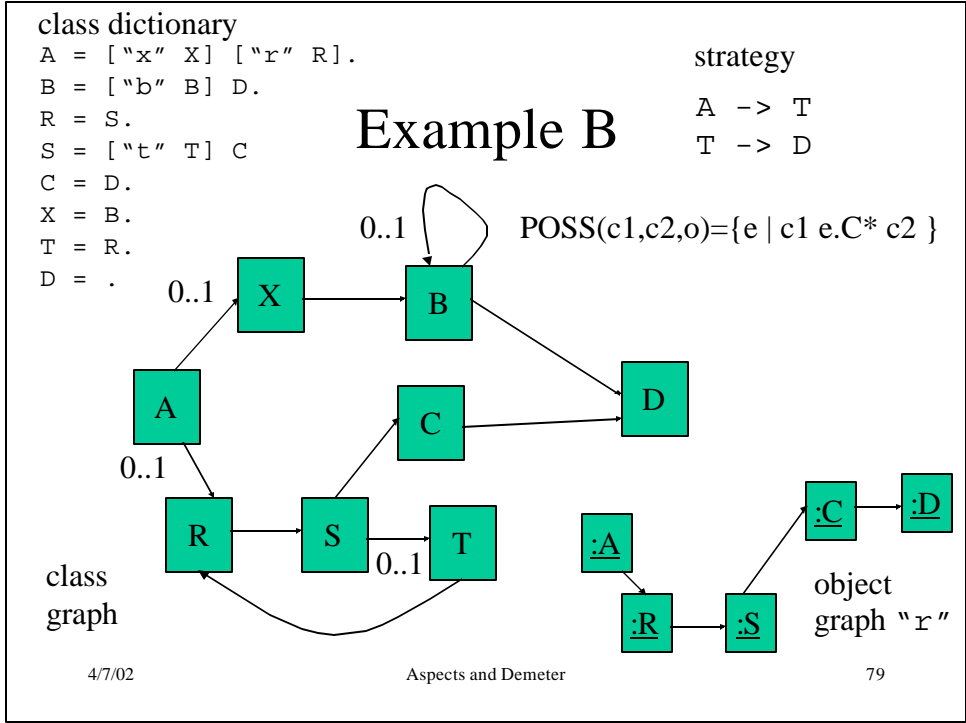
## Generalizations

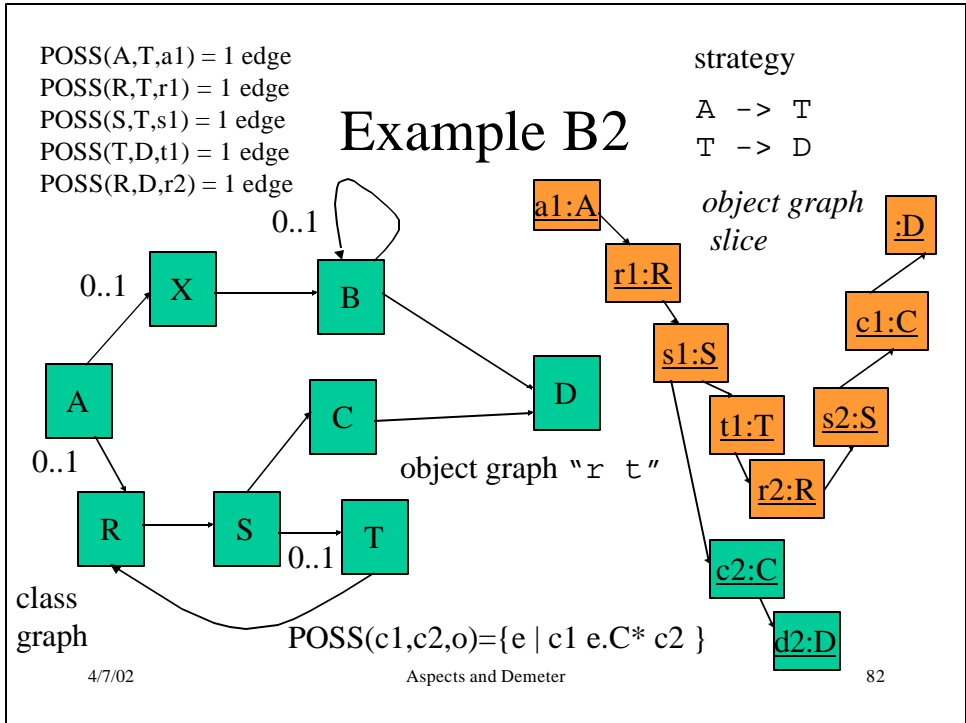
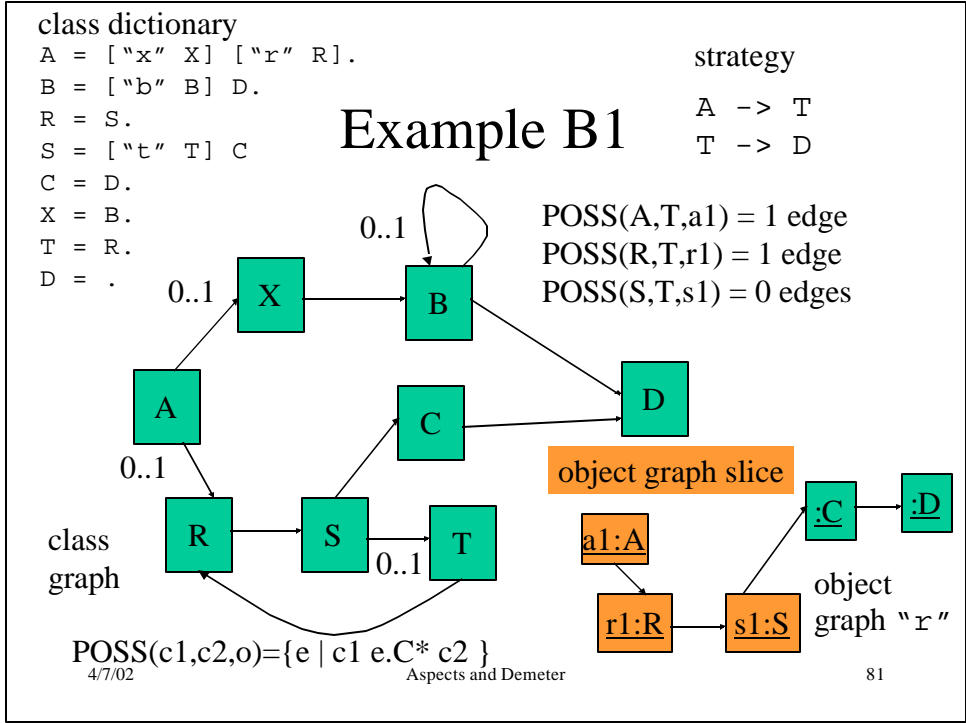
- More complex strategies
- “from c1 via c2 to c3”
  - Use “waypoint navigation”; get to a c2 object, then search for a c3 object.
- More complex strategy graphs also doable in this framework

4/7/02

Aspects and Demeter

78



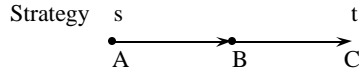
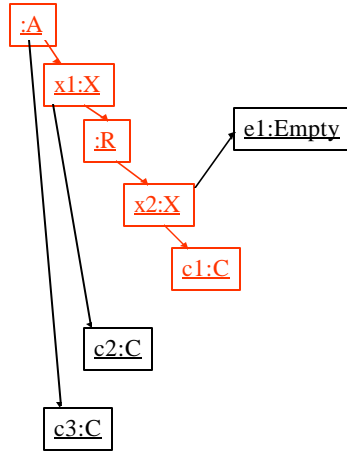


Only node paths shown for space reasons

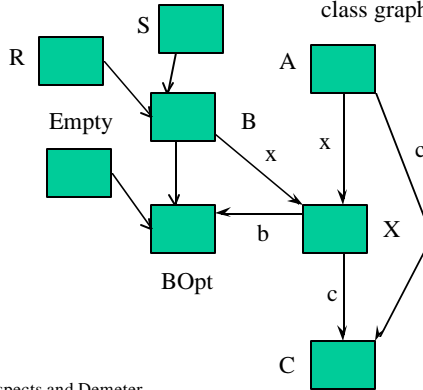
# Example C

strategy SG:  
{A -> B  
B -> C}

Object graph



class graph



4/7/02

Aspects and Demeter

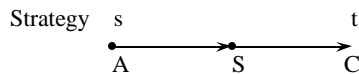
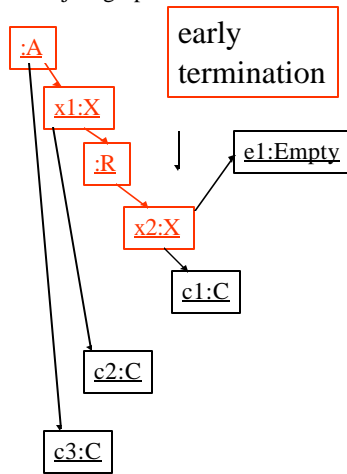
83

Only node paths shown for space reasons

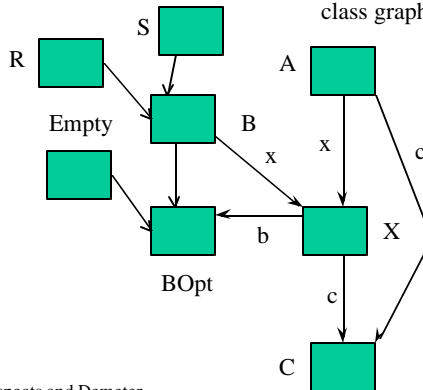
# Example C1

strategy SG:  
{A -> S  
S -> C}

Object graph



class graph



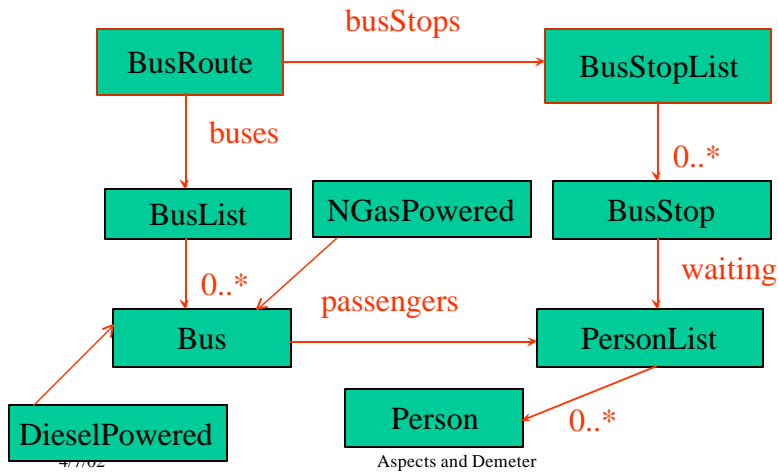
4/7/02

Aspects and Demeter

84

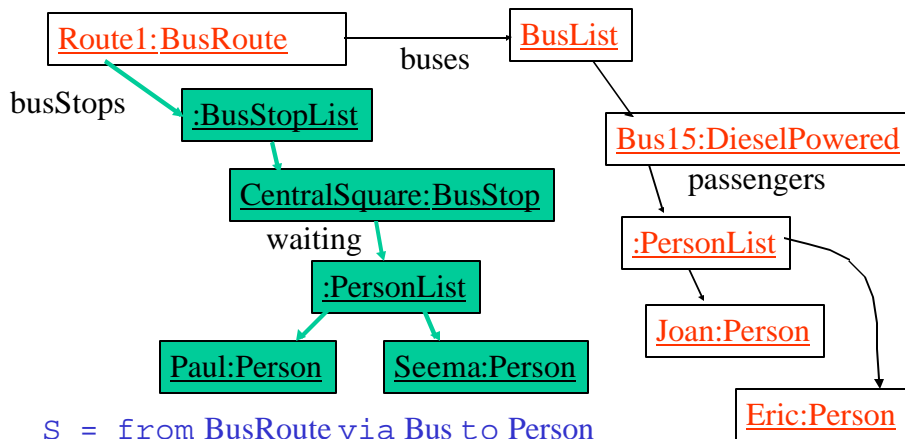
S = from BusRoute via Bus to Person

## Example D



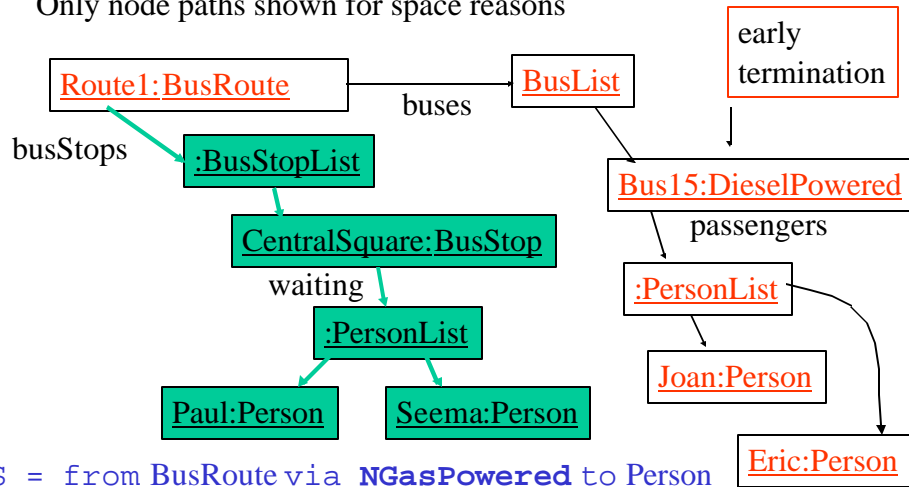
## Example D1

Only node paths shown for space reasons



## Example D2

Only node paths shown for space reasons



4/7/02

Aspects and Demeter

87

from c1 **bypassing** {x1,x2, ... ,xn} to c2

## Relational Formulation

From object o of class c1, to get to c2, follow edges in the set

$$\text{POSS}(c1,c2,o) = \{e \mid c1 \leq e \Rightarrow (\leq C \Rightarrow)^* \leq c2\}$$

POSS = abbreviation for: following these edges it is still possible to reach a c2-object for some c1-object rooted at o.

**Delete x1,x2, ... ,xn and all edges incident with these nodes from the class graph (unless they are c1, c2).**

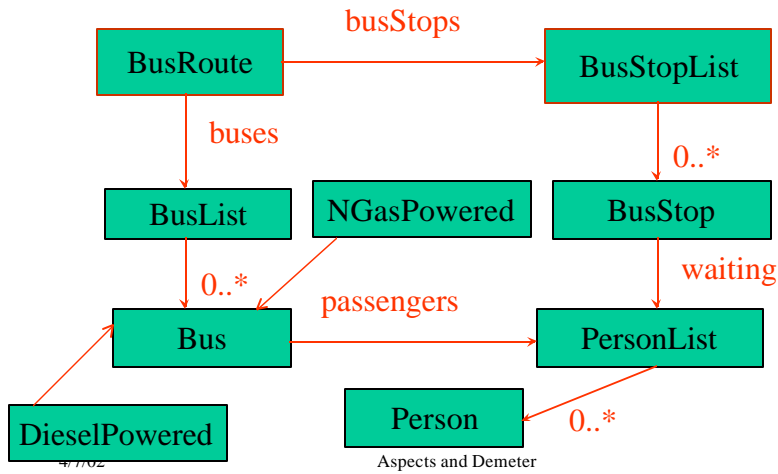
4/7/02

Aspects and Demeter

88

S = from BusRoute bypassing Bus to Person

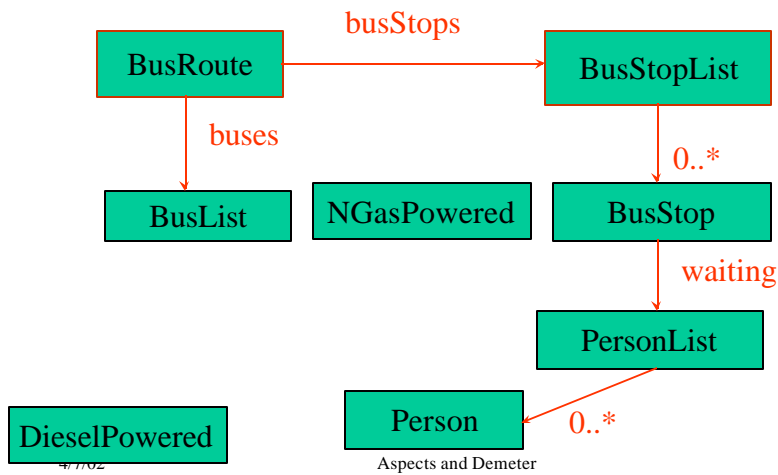
## Example D



89

S = from BusRoute bypassing Bus to Person

## Example D



90

## Note

- Separation of concerns is also useful for defining programming language elements
  - separate subgraph selected from
  - how the subgraph is traversed (depth-first etc.)
- In earlier works: meaning of a traversal strategy for an object graph
  - was a traversal history
  - now it is a subgraph of the object graph. A traversal history can be defined ...

4/7/02

Aspects and Demeter

91



## Programming Exercise

- Check whether all used entities are defined.
- Object structure, traversal (basically an introduction of methods), advice on traversal.

4/7/02

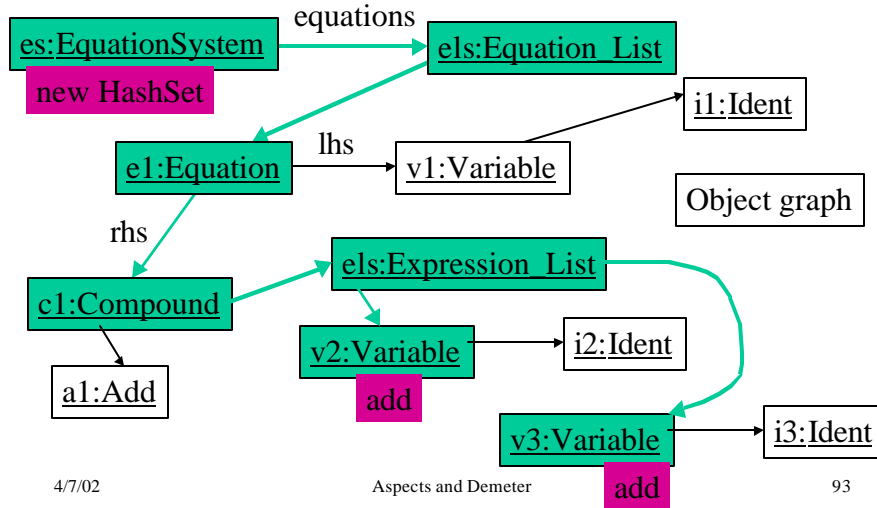
Aspects and Demeter

92

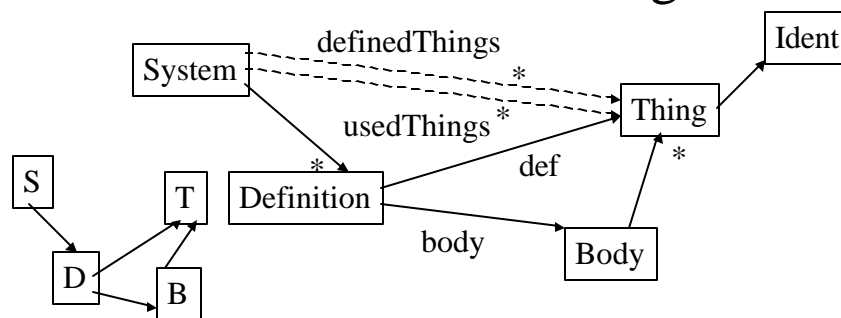
overall graph: object structure; green graph: traversal; purple: advice

usedThings = from EquationSystem via Expression to Variable

## Crosscutting in Equation System



## Class graph: Find undefined things



definedThings = from System via  $\rightarrow^*$ , def,  $\rightarrow^*$  to Thing

usedThings = from System via  $\rightarrow^*$ , body,  $\rightarrow^*$  to Thing

4/7/02

Aspects and Demeter

94

## Java Program: Adaptive Method with DJ

```
class System{
  String id = "from Thing to edu.neu.ccs.demeter.Ident";
  void repUndef(ClassGraph cg){
    checkDefined(cg, getDefThings(cg));
    HashSet getDefThings(ClassGraph cg){
      String definedThings =
        "from System via ->*,def,* to Thing";
      Visitor v = new Visitor(){
        HashSet return_val = new HashSet();
        void before(Thing v1){
          return_val.add(cg.fetch(v1, id) );
        }
        public Object getReturnValue(){return return_val;};
      };
      return (HashSet)
        cg.traverse(this, definedThings, v);
    }
  }
}
```

repUndef is a modular unit of crosscutting implementation. Ad-hoc implementation may cut across 100 classes.

green: traversal  
black bold: structure  
purple: advice  
red: parameters <sup>95</sup>

4/7/02

Aspects and Demeter

## Java Program: Adaptive Method with DJ

```
void checkDefined(ClassGraph cg, final HashSet classHash){
  String usedThings =
    "from System via -> *,body,* to Thing";
  cg.traverse(this, usedThings, new Visitor(){
    void before(Thing v){ Ident vn = cg.fetch(v, vi);
      if (!classHash.contains(vn)){
        System.out.println("The object " + vn
          + " is undefined.");
      }
    }
  });
}
```

4/7/02

Aspects and Demeter

96

# Reengineer

- Reuse collection behavior twice.
- Much simpler to reengineer in this abstract form.
- Structure of program is not hardened yet by details of class graph.

4/7/02

Aspects and Demeter

97

```
class System{
    String id = "from Thing to edu.neu.ccs.demeter.Ident";
    HashSet collect(ClassGraph cg, String constraint){
        Visitor v = new Visitor(){
            HashSet return_val = new HashSet();
            void before(Thing v1){
                return_val.add(cg.fetch(v1, id) );}
            public Object getReturnValue(){return return_val;}};
    cg.traverse(this, "from System"+constraint+"to Thing", v);
    return (HashSet)v.getReturnValue();
}

HashSet defined(ClassGraph cg) {
    return (HashSet) this.collect(cg, "via ->*,def,*" );}
HashSet used(ClassGraph cg) {
    return (HashSet) this.collect(cg, "via ->*,body,*" );}}
```

pure Java using DJ

green: traversal  
black bold: structure  
purple: advice  
red: parameters

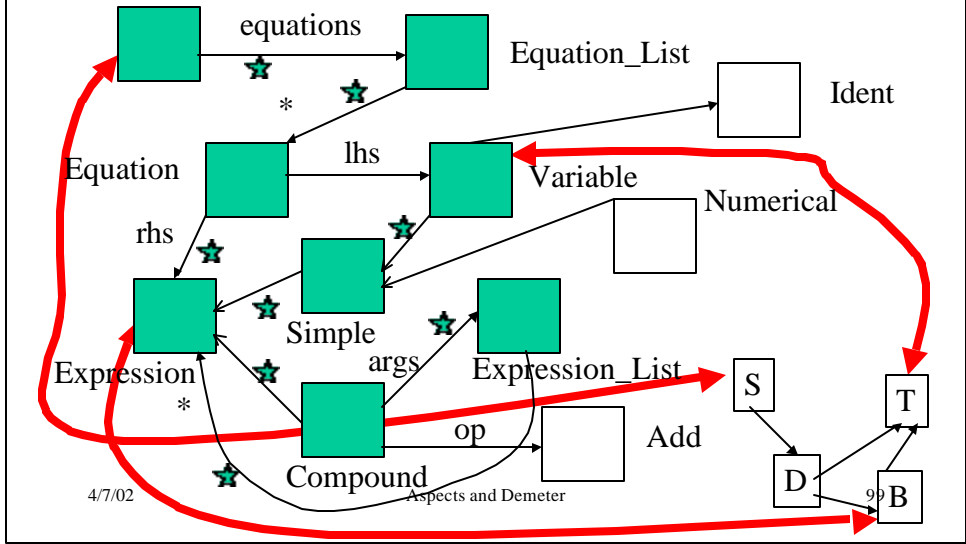
4/7/02

Aspects and Demeter

98

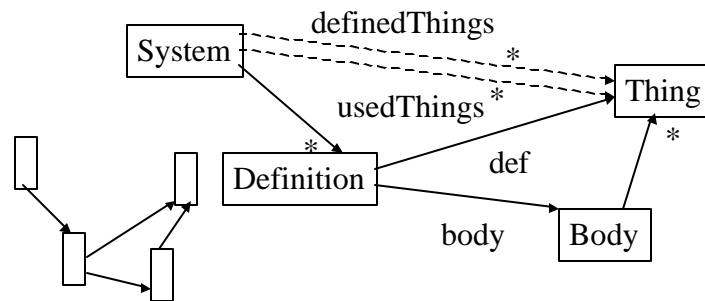
usedThings = from EquationSystem via Expression to Variable Fig. Eq3

# M1: Equation System



System	EquationSystem
Definition	Equation
Body	Expression
Thing	Variable

## Collect Things



definedThings = from System via -> \*,def,\* to Thing  
 usedThings = from System via -> \*,body,\* to Thing  
 = **from System + constraint + to Thing**

4/7/02

Aspects and Demeter

100

## Example of Aspect-Oriented Programming

- Separating the following crosscutting concerns:
  - traversal-related concerns, for each one separate
    - Object Structure (detailed meta information)
    - Traversals via Objects (where to go)
    - Advice on Traversals (what to do)
- Traversal-related concerns are common.

4/7/02

Aspects and Demeter

101

## Loose Coupling

- Object Structure
  - does not have to know about traversals and advice on traversals
- Traversals
  - don't have to know about advice on traversals
- Advice on Traversals
  - has to know minimally about object structure and traversals

4/7/02

Aspects and Demeter

102

## Ad-hoc Implementation of three concerns

- Leads to lots of tangled code with numerous disadvantages
- The question is not how to eliminate the tangling but how to reduce it
- AOP is about tangling control for the implementation of crosscutting concerns
- Crosscutting will always lead to some tangling at code level

4/7/02

Aspects and Demeter

103

## Name map

<i>Roles</i>	<i>CS1</i>	<i>CS2</i>	<i>MI</i>
<i>System</i>	ClassG	Grammar	Equation-System
<i>Body</i>	Body	Body	Expression
<i>Thing</i>	ClassName	NonTerm	Variable
<i>Definition</i>	ClassDef	Production	Equation

4/7/02

Aspects and Demeter

104

## High-level description

- It is useful to have a high-level description of the collaboration besides the Java source code. Useful documentation.
- Ultimately we are interested in the executable form of the collaboration (Java source code).

4/7/02

Aspects and Demeter

105

## Collaboration with strategies

```
collaboration checkDef {
  role System {
    out repUndef(){(uses getDefThings, checkDefined)};
    getDefThings(){(uses definedThings)};
    checkDefined(){(uses usedThings)};
    in definedThings =
      from System bypassing Body to Thing;
    in usedThings =
      from System via Body to Thing; }
  role Body { }
  role Thing { }
  role Definition { }
}
```

4/7/02

Aspects and Demeter

106

## Collaboration with strategies

```
collaboration COLLECT {
  role System {
    out HashSet collect(){};
    defined(){(uses collect, definedThings)};
    used(){(uses collect, usedThings)};
    in definedThings =
      from System bypassing Body to Thing;
    in usedThings =
      from System via Body to Thing; }
  role Body { }
  role Thing { }
  role Definition { }
}
```

4/7/02

Aspects and Demeter

107

## Use of collaboration: Adapter

Need to provide the expected methods (**in** methods) and provide name map.

- name map:
  - System : EquationSystem
  - Definition : Equation
  - Body : Expression
  - Thing : Variable
- expected methods:
  - **in** definedThings // use default
  - **in** usedThings // use default

4/7/02

Aspects and Demeter

108

## Ad-hoc Implementation of traversal-related concerns

- Leads to lots of tangled and scattered code with numerous disadvantages
- The question is not how to eliminate the tangling but how to reduce it
- AOP is about tangling control of the implementation of crosscutting concerns
- Crosscutting will always lead to some tangling at code level

4/7/02

Aspects and Demeter

109

## Need more than localization of crosscutting concerns

- If we localize a crosscutting traversal-related concern in the standard way, we get a method that violates the Law of Demeter: it duplicates much class graph information
- In addition: Use traversal strategies to eliminate accidental noise in class graph
- Need AP to improve AOP

4/7/02

Aspects and Demeter

110

## What is an aspect?

- An aspect is a modular unit of crosscutting implementation.
- A Java method is a modular unit.
- Can we make a Java method an aspect?
- Yes, we call such methods adaptive methods.
- They cut across many classes in an ad-hoc implementation.

4/7/02

Aspects and Demeter

111

## Java Program: Adaptive Method

```
class System{
  String id = "from Thing to edu.neu.ccs.demeter.Ident";
  void repUndef(ClassGraph cg) {
    checkDefined(cg, getDefThings(cg));
  }
  HashSet getDefThings(ClassGraph cg) {
    String definedThings =
      "from System bypassing Body to Thing";
    Visitor v = new Visitor(){
      HashSet return_val = new HashSet();
      void before(Thing v1){
        return_val.add(cg.fetch(v1, id) );
      }
      public Object getReturnValue(){return return_val;}};
    cg.traverse(this, definedThings, v);
    return (HashSet)v.getReturnValue();
  }
}
```

4/7/02

Aspects and Demeter

green: traversal  
black bold: structure  
purple: advice  
red: parameters<sup>112</sup>

## Java Program: Adaptive Method

```
void checkDefined(ClassGraph cg, final HashSet classHash){
    String usedThings =
        "from System via Body to Thing";
    cg.traverse(this, usedThings, new Visitor(){
        void before(Thing v){ Ident vn = cg.fetch(v, id);
            if (!classHash.contains(vn)){
                System.out.println("The object "+ vn
                    + " is undefined.");
            }
        }
    });
}
```

4/7/02

Aspects and Demeter

113

## After applying name map

- For now we manually edit the Java program.

4/7/02

Aspects and Demeter

114

## Java Program with less tangling

```
class EquationSystem{
    String id = "from Variable to edu.neu.ccs.demeter.Ident";
    void repUndef(ClassGraph cg){
        checkDefined(cg, getDefThings(cg));
    }
    HashSet getDefThings(ClassGraph cg){
        String definedThings =
            "from EquationSystem bypassing Expression to Variable";
        Visitor v = new Visitor(){
            HashSet return_val = new HashSet();
            void before(Variable v1){
                return_val.add(cg.fetch(v1, id) );
            }
            public Object getReturnValue(){return return_val;};
        };
        cg.traverse(this, definedThings, v);
        return (HashSet)v.getReturnValue();
    }
}
```

4/7/02

Aspects and Demeter

green: traversal  
black bold: structure  
purple: advice  
red: parameters<sup>115</sup>

## Java Program with less tangling

```
void checkDefined(ClassGraph cg, final HashSet classHash){
    String usedThings =
        "from EquationSystem via Expression to Variable";
    cg.traverse(this, usedThings, new Visitor(){
        void before(Variable v){ Ident vn = cg.fetch(v, id);
            if (!classHash.contains(vn)){
                System.out.println("The object "+ vn
                    + " is undefined.");
            }
        }
    });
}
```

4/7/02

Aspects and Demeter

116

# Tangling is localized Scattering eliminated

- Instead of having code spread across several classes, it is localized in one class.
- Java program is describing the abstract pattern behind the computation of interest: checking whether used entities are defined.
- Tangling control through abstraction of patterns. We abstract away from structure.

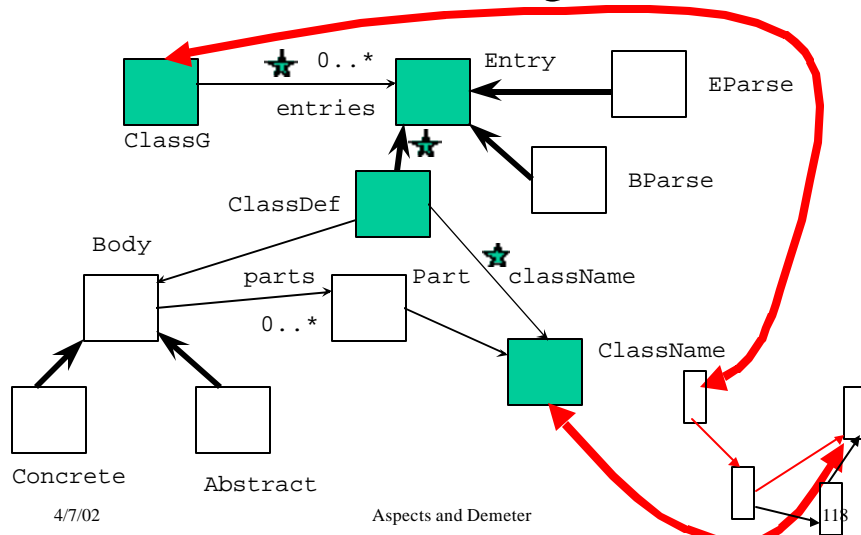
4/7/02

Aspects and Demeter

117

definedThings = from ClassG bypassing Body to ClassName

## CS1: UML class diagram ClassG



4/7/02

Aspects and Demeter

118

usedThings = from ClassG via Body to ClassName

## CS1:UML class diagram ClassG

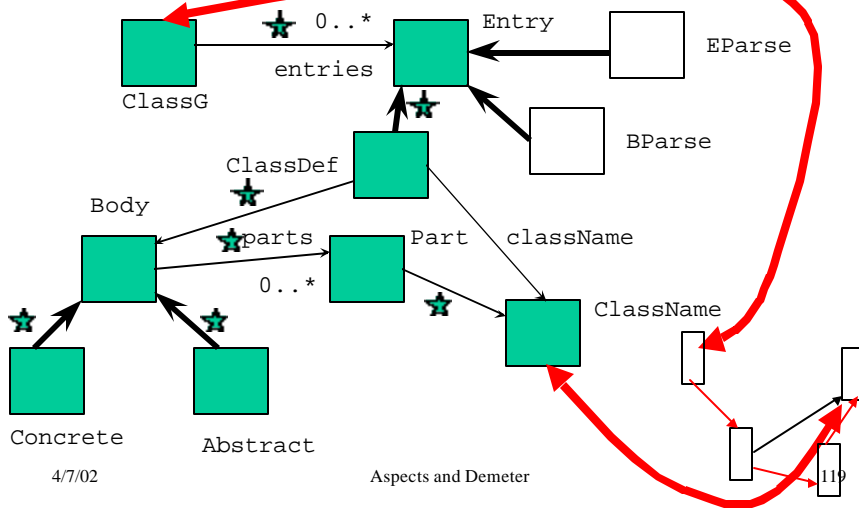


Fig. Eq1

## M1: Equation System

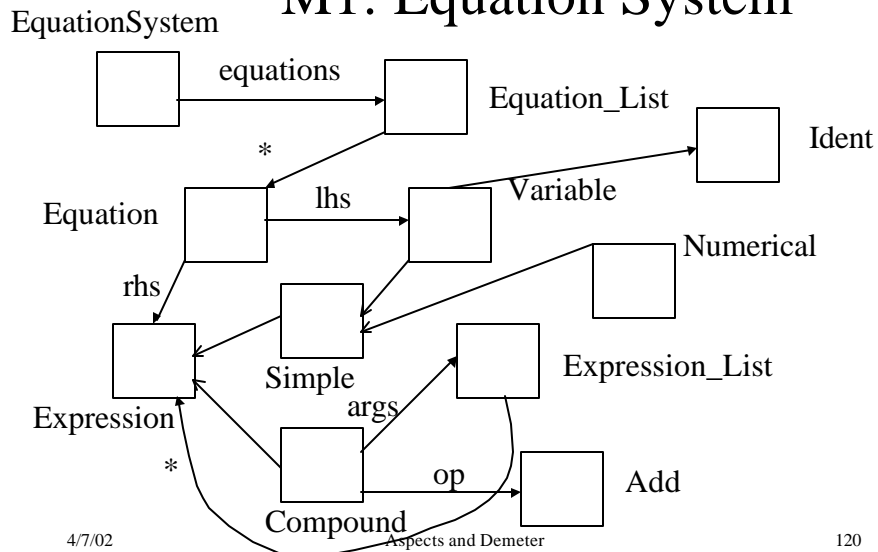
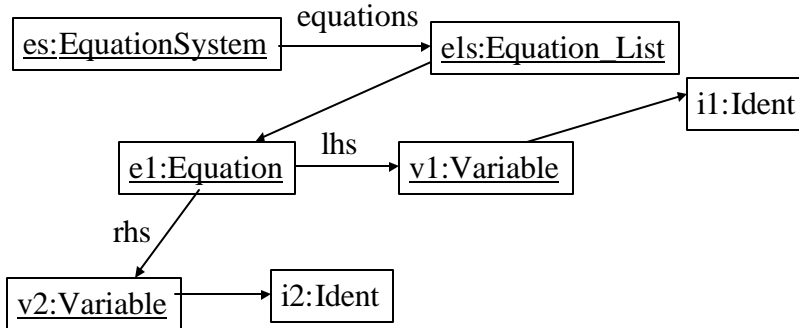




Fig. Eq4

## Equation System Object



4/7/02

Aspects and Demeter

123

Example:  
 $x = 1.0$  .  
 $y = (+ x 4.0)$  .  
 $z = (* x y)$  .

usedThings = from EquationSystem  
 via Expression to Variable

## M1: Equation System

```

EquationSystem = <equations> List(Equation).
Equation = <lhs> Variable "=" <rhs> Expression ".".
Variable = Ident.
Expression : Simple | Compound.
Simple : Variable | Numerical.
Compound = "(" Op <args> List(Expression) ")".
Op : Add | Mul.
Add = "+".
Mul = "*".
Numerical = float.
  
```

4/7/02

Aspects and Demeter

124

Example:  
 x = 1.0 .  
 y = (+ x 4.0).  
 z = (\* x y).

definedThings= from EquationSystem  
 bypassing Expression to Variable  
**M1: Equation System**

```

EquationSystem = <equations> List(Equation).
Equation = <lhs> Variable "=" <rhs> Expression ".".
Variable = Ident.
Expression : Simple | Compound.
Simple : Variable | Numerical.
Compound = "(" Op <args> List(Expression) ")".
Op : Add | Mul.
Add = "+".
Mul = "*".
Numerical = float.
  
```

4/7/02

Aspects and Demeter

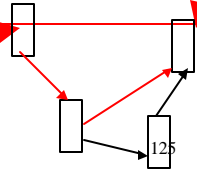
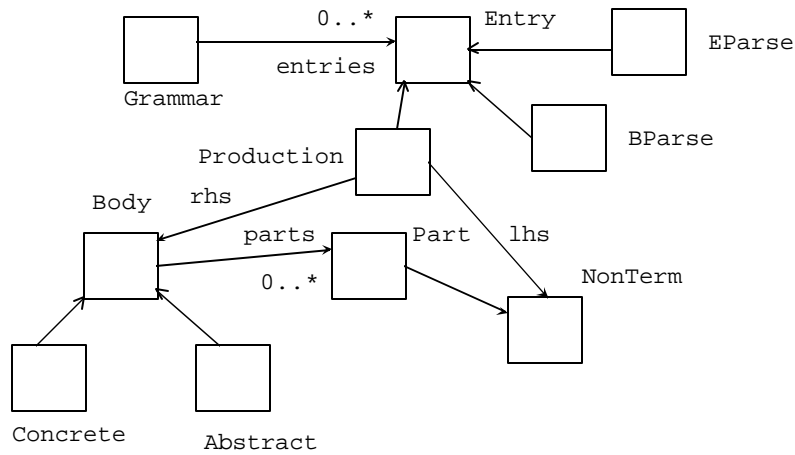


Fig. G1

**CS1: UML class diagram Grammar**



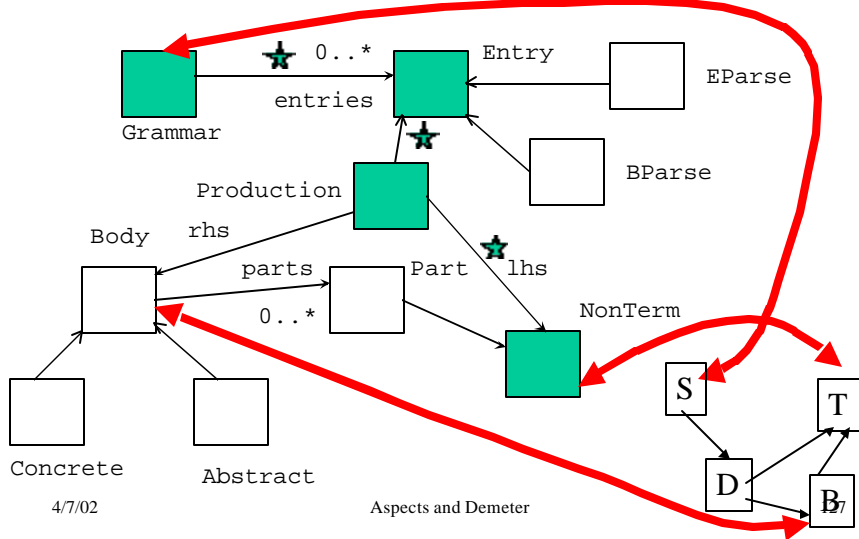
4/7/02

Aspects and Demeter

126

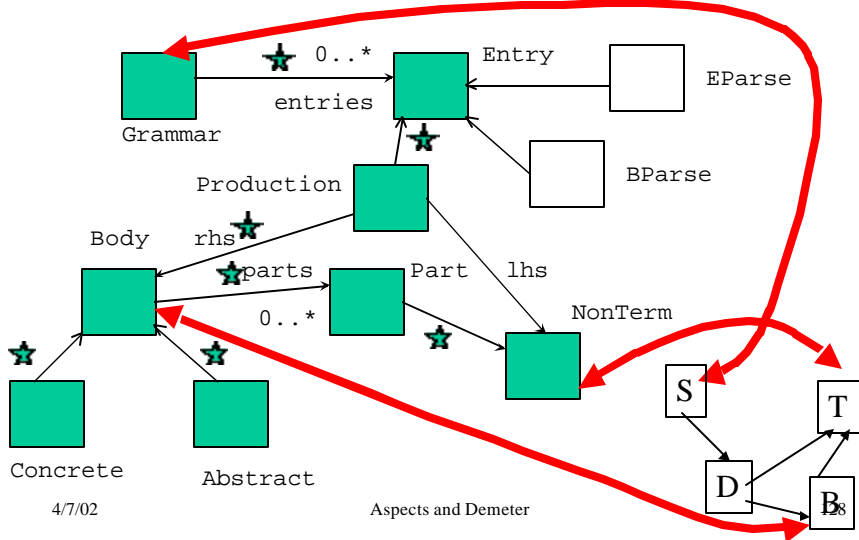
definedThings = from Grammar bypassing Body to NonTerm Fig. G2

## CS1: UML class diagram Grammar



usedThings = from Grammar via Body to NonTerm Fig. G3

## CS1:UML class diagram Grammar



## What DJ adds to AspectJ

- Pointcut definitions based on connectivity in class graph.
- Pointcut reduction (high-level point cut designator): free programmer from details of class graph.

4/7/02

Aspects and Demeter

129

## Discussion with Gregor Kiczales at UBC

- Ontology of AOP
- Ontology is the study of what there is, an inventory of what exists. An ontological commitment is a commitment to an existence claim for certain entities.

4/7/02

Aspects and Demeter

130

## basis of crosscutting

- a *join point model* (JPM) has 3 critical elements
  - what are the join points
  - means of identifying join points
  - means of specifying semantics at join points

4/7/02

Aspects and Demeter

131

## Range of AOP languages

JPM	join points	means of ... join points	
		identifying	specifying semantics at
AspectJ dynamic JPM	points in execution call, get, set...	signatures w/ wildcards & other properties of JPs	advice
static JPM	class members	signatures	add members
DemeterJ, Demeter/C++ dynamic JPM 1	when traversal reaches object or edge	visitor method signatures	visitor method bodies
static JPM 1	class members	traversal spec. s class graph g	s + g (result = traversal implementation)
static JPM 2	class members	class names	add members
static JPM 3	class members	class graph	class graph with tokens=grammar (result = parsing and printing implementation)

4/7/02

Aspects and Demeter

132

## Range of AOP languages

JPM	join points	means of ... join points	
		identifying	specifying semantics at
AspectJ dynamic JPM	points in execution call, get, set...	signatures w/ wildcards & other properties of JPs	advice
static JPM	class members	signatures	add members
DJ dynamic JPM 1	when traversal reaches object or edge (method traverse)	visitor method signatures	visitor method bodies
dynamic JPM 2	when traversal reaches object (methods fetch, gather, asList)	source and targets of traversal	method name (fetch, gather, asList)
static JPM 4	nodes in object graph o	trav. spec. s class graph g	s+g (result = traversal impl. = edges to traverse at nodes in object graph o)
4/7/02	Aspects and Demeter	133	

## Combining two join point models

- **Static JPM 1**: In Demeter we use traversal specifications and the class graph to define a traversal implementation (either static or dynamic)
- **Visitor JPM 1**: The result of static JPM 1 is used to define a second JPM:
  - The traversal implementation defines nodes and edge visits.
  - Visitor signatures define the nodes and edges where additional advice is needed: they are the means of identifying join points.
  - The means of specifying semantics at join points are the visitor bodies.

4/7/02

Aspects and Demeter

134



# Pattern Language for Adaptive Programming (AP)

Structure-shy Traversal Pattern only

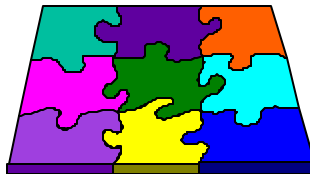
4/7/02

Aspects and Demeter

135

## Structure-shy Traversal

- Intent
  - Succinctly represent a traversal to be performed on objects
  - Commit only to navigation *strategy* and specify navigation details later



4/7/02

Aspects and Demeter

136

## Structure-shy Traversal

- Could also be called:
  - Adaptive Traversal
  - Structure-shy Walker

4/7/02

Aspects and Demeter

137

## Structure-shy Traversal

- Motivation
  - Noise in objects for specific task
  - Focus on long-term intent
  - Don't want to attach every method to a specific class explicitly. Leads to brittle programs.
  - Small methods problem (example: 80% of methods are two lines long or shorter)

4/7/02

Aspects and Demeter

138

## Structure-shy Traversal

- Applicability
  - Need collaboration of at least two classes.
  - In the extreme case, each data member access is done through a succinct traversal specification.
  - Some subgraphs don't have a succinct representation, for example a path in a complete graph. More generally: avoid well connected, dense graphs.

4/7/02

Aspects and Demeter

139

## Structure-shy Traversal

- Solution
  - Use succinct subgraph specifications
  - Use succinct path set specifications

4/7/02

Aspects and Demeter

140

## Structure-shy Traversal: Solution

- Traversal Strategy Graphs (Strategies)
  - *First stage*: A strategy is a graph with nodes and edges. Nodes are labeled with nodes of a class graph. Edges mean: all paths.
  - *Second stage*: label edges with constraints excluding edges and nodes in class graph



4/7/02

Aspects and Demeter

141

## Structure-shy Traversal: Solution

- Traversal Strategy Graphs (**Strategies**)
  - Simplest useful strategy: One Edge. Possible syntax:
    - from Company to Salary or
    - {Company -> Salary}
  - Line graph. Several edges in a line. Possible syntax:
    - From Company via Employee to Salary
    - {Company -> Employee Employee -> Salary}

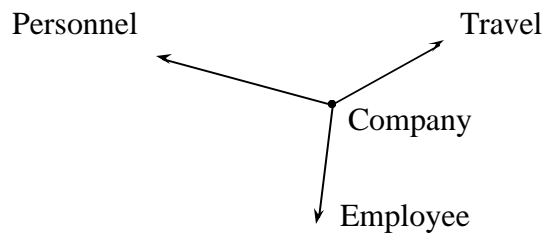
4/7/02

Aspects and Demeter

142

## Structure-shy Traversal: Solution

- Traversal Strategy Graphs (Strategies)
  - Star graph
    - From Company to {Personnel, Travel, Employee}

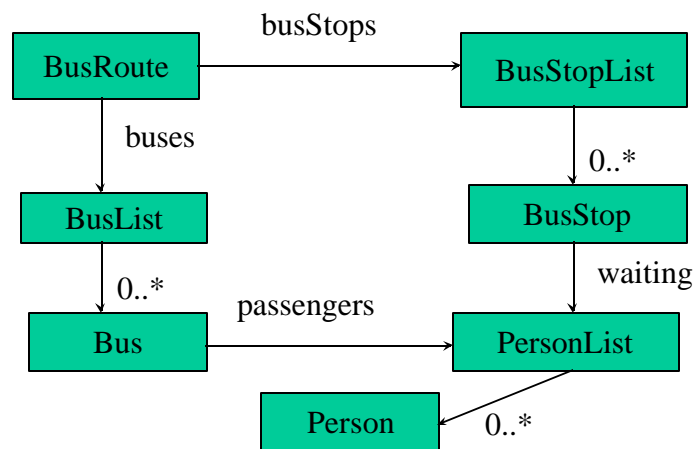


4/7/02

Aspects and Demeter

143

## UML Class Diagram



4/7/02

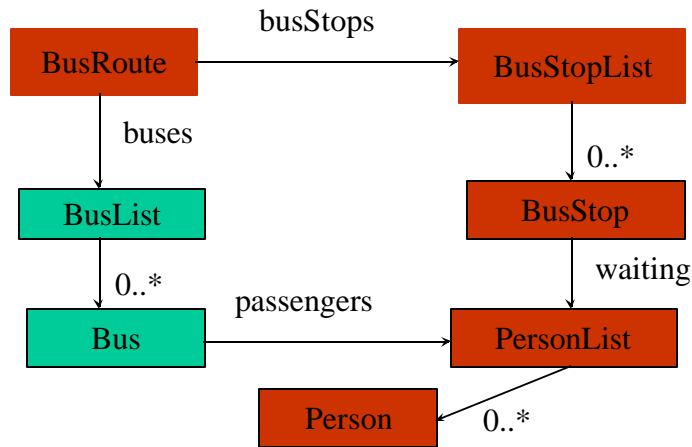
Aspects and Demeter

144

find all persons waiting at any bus stop on a bus route

# Traversal Strategy

from BusRoute via BusStop to Person



4/7/02

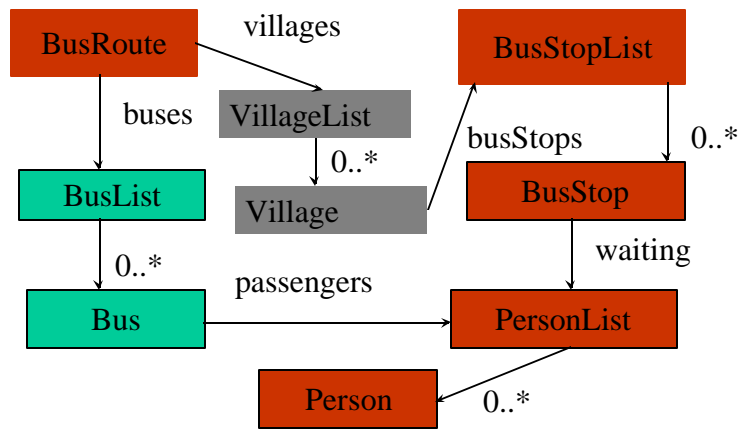
Aspects and Demeter

145

find all persons waiting at any bus stop on a bus route

# Robustness of Strategy

from BusRoute via BusStop to Person



4/7/02

Aspects and Demeter

146

## Structure-shy Traversal

- Consequences
  - Programs become shorter and more powerful. A paradox. With less work we achieve more. Polya's inventor paradox.
  - Program will adapt to many changes in class structure.

4/7/02

Aspects and Demeter

147

## Structure-shy Traversal

- Implementation
  - Many different models for succinct traversal specifications.
  - Best one: Strategies
  - Correct implementation of strategies is tricky. See paper by Lieberherr/Patt-Shamir strategies.ps in my FTP directory.

4/7/02

Aspects and Demeter

148

## Structure-shy Traversal

- Known Uses
  - *Adaptive Programming*: Demeter/C++, DemeterJ, Dem/Perl, Dem/CLOS etc.
  - *Databases* (limited use): Structure-shy queries: See Cole Harrison's Master's Thesis (Demeter Home Page)
  - XML: XPath
  - *Artificial Intelligence* (limited use): Minimal ontological commitment

4/7/02

Aspects and Demeter

149

## More on DJ

- Including the connection to generic programming

4/7/02

Aspects and Demeter

150



## ObjectGraphSlice

- The object graph slice starting with `o1` is the slice built by following the edges `POSS(Class(o1), t, o1)` starting at `o1` and continuing until every path terminates (at an object of type `t` or it terminates prematurely).

4/7/02

Aspects and Demeter

151

## DJ

- An implementation of AP using only the DJ library (and the Java Collections Framework)
- All programs written in pure Java
- Intended as prototyping tool: makes heavy use of introspection in Java
- Integrates Generic Programming (a la C++ STL) and Adaptive programming

4/7/02

Aspects and Demeter

152

## Integration of Generic and Adaptive Programming

- A traversal specification turns an object graph into a list.
- Can invoke generic algorithms on those lists. Examples: contains, containsAll, equals, isEmpty, contains, etc. add, remove, etc. throws operation not supported exception.
- What is gained: genericity not only with respect to data structure implementations but also with respect to class graph

4/7/02

Aspects and Demeter

153

## Sample DJ code

```
// Find the user with the specified uid
List libUsers =
    classGraph.asList(library,
        "from Library to User");
ListIterator li =
    libUsers.listIterator();
// iterate through libUsers
```

4/7/02

Aspects and Demeter

154

## Methods provided by DJ

- On ClassGraph, ObjectGraph, TraversalGraph, ObjectGraphSlice: traverse, fetch, gather
- traverse is the important method; fetch and gather are special cases
- TraversalGraph
  - Object traverse(Object o, Visitor v)
  - Object traverse(Object o, Visitor[] v)

4/7/02

Aspects and Demeter

155

## Traverse method: excellent support for Visitor Pattern

```
// class ClassGraph
Object traverse(Object o,
               Strategy s, Visitor v);
```

traverse navigates through Object o following traversal specification s and executing the before and after methods in visitor v

ClassGraph is computed using introspection

4/7/02

Aspects and Demeter

156

## Fetch Method

- If you love the Law of Demeter, use fetch as your shovel for digging:
  - Part k1 = (K) classGraph.fetch(a,"from A to K");
- The alternative is (digging by hand):
  - Part k1 = a.b().c().d().e().f().g().h().i().k();
- DJ will tell you if there are multiple paths to the target (but currently only at run-time).

4/7/02

Aspects and Demeter

157

## Gather Method

- Returns a list of objects.
- Object ClassGraph.gather(Object o, String s)
  - List ks = classGraph.gather(a,"from A to K");  
returns a list of K-objects.

4/7/02

Aspects and Demeter

158

## Using DJ

- `traverse(...)` returns the `v[0]` return value. Make sure the casting is done right, otherwise you get a run-time error. If “`public Object getReturnValue()`” returns an `Integer` and `traverse(...)` casts it to a `Real`: casting error at run-time.
- Make sure all entries of `Visitor[]` array are non-null.

4/7/02

Aspects and Demeter

159

## Using multiple visitors

```
// establish visitor communication
aV.set_cV(cV);
aV.set_sV(sV);
rV.set_aV(aV);

Float res = (Float) whereToGo.
    traverse(this,
        new Visitor[] {rV, sV, cV, aV});
```

4/7/02

Aspects and Demeter

160



## DJ binary construction operations

	cg	s	tg	o	og	ogs
cg	*	tg,cg	*	og	*	*
s		*	*	*	ogs	*
tg			*	*	ogs	*
o				*	*	*
og					*	*
ogs						*

4/7/02

Aspects and Demeter

161

Who has **traverse, fetch, gather?**  
(number of arguments of traverse)

	cg(3)	s	tg(2)	o	og(2)	ogs(1)
cg	*	tg,cg	*	og	*	*
s		*	*	*	ogs	*
tg			*	*	ogs	*
o				*	*	*
og					*	*
ogs						*

4/7/02

Aspects and Demeter

162

## Methods returning an ObjectGraphSlice

- ClassGraph.slice(Object, Strategy)
- ObjectGraph.slice(Strategy)
- TraversalGraph.slice(Object)
- ObjectGraphSlice(ObjectGraph, Strategy)
- ObjectGraphSlice(ObjectGraph, TraversalGraph)

Blue: constructors

4/7/02

Aspects and Demeter

163

## Traverse method arguments

- ClassGraph
  - Object, Strategy, Visitor
- TraversalGraph
  - Object, Visitor
- ObjectGraph
  - Strategy, Visitor
- ObjectGraphSlice
  - Visitor

4/7/02

Aspects and Demeter

164

## Traverse method arguments. Where is **collection framework** used?

- ClassGraph
  - gather(Object, Strategy) / asList(Object, Strategy)
- TraversalGraph
  - gather(Object) / asList(Object)
- ObjectGraph
  - gather(Strategy) / asList(Strategy)
- ObjectGraphSlice
  - gather() / asList()

4/7/02

Aspects and Demeter

165

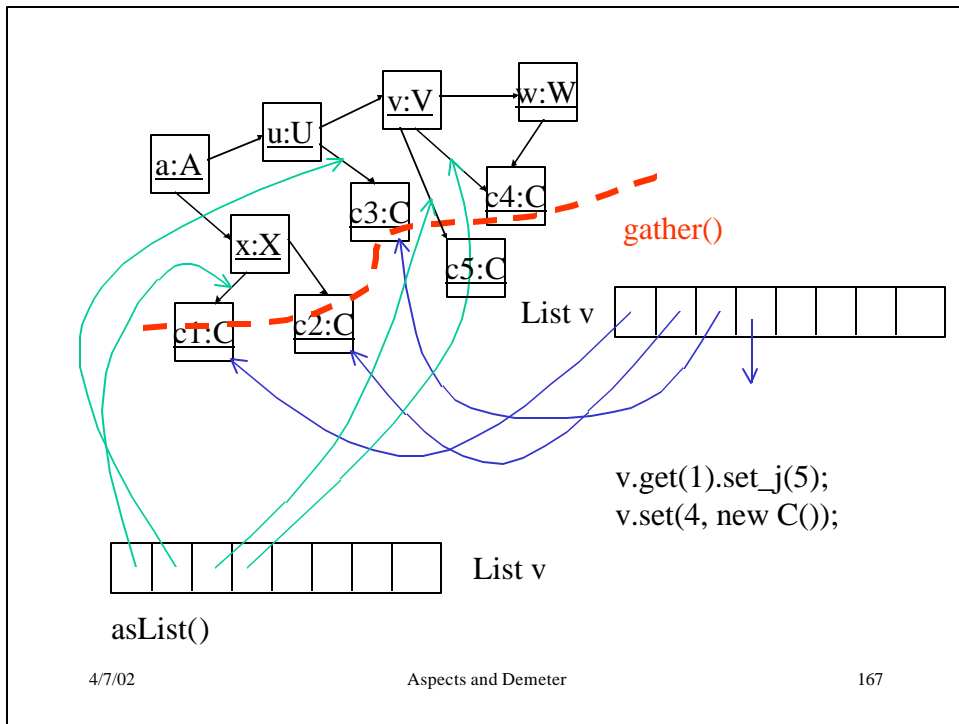
## Where is collection framework used?

- ObjectGraphSlice.asList()
  - a fixed-size List backed by the object graph slice. Is write-through: modifying list will modify object and modifying object will modify list. (Similar to Arrays.asList() in Java.)
- gather copies the pointers to the objects.

4/7/02

Aspects and Demeter

166



## Interfaces

- Interface List
  - ListIterator listIterator()
  - Object set(int index, Object element)
- Interface ListIterator
  - void set(Object o): replaces the last element returned by next or previous with the specified element.

## Why is asList useful?

- from Application to X: want to change all F-objects to D-objects.
- From Application to “predecessors” of D:  
put in a new object.
  - This is not structure-shy: all the predecessors of X also need to be mentioned.

4/7/02

Aspects and Demeter

169

## Why is asList useful?

- from Application to X: want to change all X-objects to D-objects.

Application = <es> List(E).

E = X D.

D = X F.

X : F | D.

F = .

D = List(X).

List(S) ~ {S}.

Sketch:

from Application to E:

aE.set\_x(new D());

aE.get\_d().set\_x(new D());

from E to D: update list elements

from Application to X: set(new D())

4/7/02

Aspects and Demeter

170

## Why is asList useful?

- From A to B: want to change a B-object that satisfies some property. Does not matter whether it is in a list.

A = B C.

C = List(B).

4/7/02

Aspects and Demeter

171

## More on semantics of DJ with abstract classes

- What is the meaning of a visitor on an abstract class?

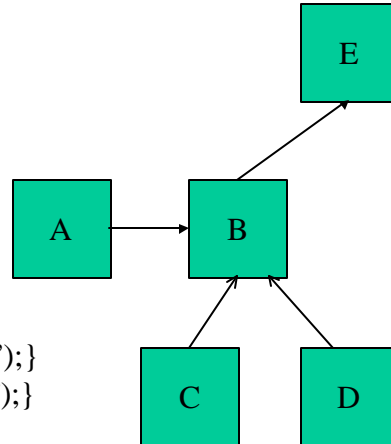
4/7/02

Aspects and Demeter

172

## Traversals to Abstract Classes

- Class graph



visitor:

```
void before (B){p("b");}  
void before (C){p("c");}
```

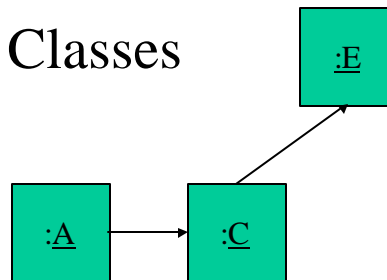
4/7/02

Aspects and Demeter

173

## Visitor Methods on Abstract Classes

- from A to E: **b, c**
- from A to B: **b, c**
- from A to C: **b, c**
- visitor:



```
– void before (B){p("b");}  
– void before (C){p("c");}
```

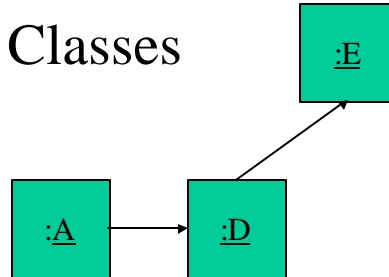
4/7/02

Aspects and Demeter

174

## Visitor Methods on Abstract Classes

- from A to E: **b**
- from A to B: **b**
- from A to C:
- visitor:
  - void before (B){p(**“b”**);}
  - void before (C){p(**“c”**);}



4/7/02

Aspects and Demeter

175

## Visitor Rule

- When an object of class X is visited, all visitors of ancestor classes of X will be active.
- The before visitors in the downward order and the after visitors in the upward order.

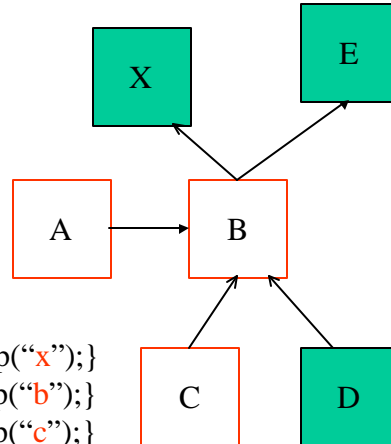
4/7/02

Aspects and Demeter

176

# Traversals to Abstract Classes

- From A to C



visitor:

```

void before (X host){p("x");}
void before (B host){p("b");}
void before (C host){p("c");}
  
```

4/7/02

Aspects and Demeter

177



# Guidelines

IF you use the combination of the following pairs and triples for multiple traversals, fetch or gather, introduce the following computation saving objects:

```

(CG,S,O)->OGS
(CG,S)->TG
(CG,O)->OG
(TG,O)->OGS
  
```

In principle can express programs only with ClassGraph and Strategy and Visitor:

```

cg      class graph      cg.traverse(o,s,v);
s       strategy         cg.fetch(o,s);
tg      traversal graph  cg.gather(o,s);
o       object          cg.asList(o,s);
og      object graph
ogs     object graph slice
v       visitor
  
```

4/7/02

Abbreviations

Aspects and Demeter

178

## DJ unary construction operations

- Class graph from Strategy
  - ClassGraph(Strategy): make a class graph with just the classes and edges in traversal graph determined by strategy. Interpret path set as a graph. Maintain several class graphs cg1, cg2, cg3 that are suitable for expressing what you need.
- Class graph from all classes in package

4/7/02

Aspects and Demeter

179

## ClassGraph construction

- make a class graph from all classes in default package
  - ClassGraph()
    - include all fields and non-void no-argument methods. **Static members are not included.**
  - ClassGraph(boolean f, boolean m)
    - If f is true, include all fields; if m is true, include all non-void no-argument methods.

4/7/02

Aspects and Demeter

180

## Dynamic features of DJ ClassGraph construction

- When a class is defined dynamically from a byte array (e.g., from network) `ClassGraph.addClass(Class cl)` has to be called explicitly. Class `cl` is returned by class loader.
- `ClassGraph()` constructor examines class file names in default package and uses them to create class graph.

4/7/02

Aspects and Demeter

181

## Dynamic features of DJ ClassGraph construction

- `ClassGraph.addPackage(String p)`
  - adds the classes of package `p` to the class graph. The package is searched for in the `CLASSPATH`. How can we control (f,m) options? Uses the same rule as used for the original class graph construction.
- Java has no reflection for packages. Motivates above solution.

4/7/02

Aspects and Demeter

182

## Adding Nodes and Edges to ClassGraph

- `addClass(Class c1)`
  - add `c1` and all its members to the class graph, if it hasn't already been added.
- `addClass(Class c1, boolean aF, boolean aM)`
  - add `c1` to the class graph. If `aF`, add all its non-static fields as construction edges. If `aM`, add all its non-static non-void methods with no arguments as derived construction edges.

4/7/02

Aspects and Demeter

183

## Adding Nodes and Edges to ClassGraph

- Part `addConstructionEdge(Field f)`
  - add `f` as a construction edge.
- Part `addConstructionEdge(Method m)`
  - add a no-args method as a construction edge.
- `addConstructionEdge` may have in addition a `String` argument called `source`. For `Impl`.
- And also a `Class` argument called `target`. Also for `Impl`. Should not be public.

4/7/02

Aspects and Demeter

184

## Add other repetition edges

- void ClassGraph.addRepetitionEdge(String source, String target)
  - add a repetition edge from source to target
- Questions
  - what about subclass and inheritance edges
  - what happens if class graph contains edges not in program. Error will occur.

4/7/02

Aspects and Demeter

185



## Problem with Java

- What is coming is not about a problem of DJ but about a problem with Java: the lack of parameterized classes.
- The lack of parameterized classes forces the use of class Object which, as the mother of all classes, is too well connected.
- This leads to unnecessary traversals and traversal graphs that are too big.

4/7/02

Aspects and Demeter

186

## Lack of parameterized classes in Java makes DJ harder to use

- Consider the traversal: from A to B
- Let's assume that in the class graph between A and B there is a Java collection class. The intent is:  $A = \text{List}(B)$  which we cannot express in Java. Instead we have:  $A = \text{Vector}(\text{Object})$ .  $\text{Object} : A \mid B$ . Let's assume we also have a class  $X=B$ .

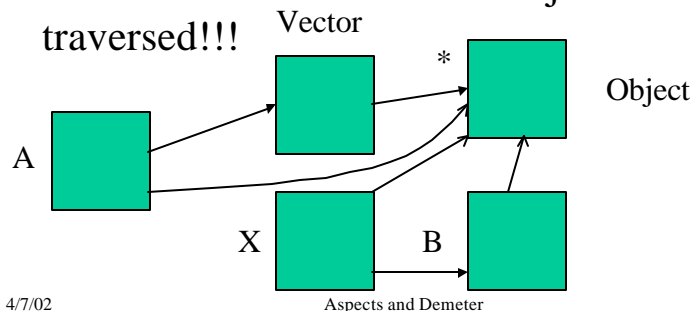
4/7/02

Aspects and Demeter

187

## Lack of parameterized classes in Java makes DJ harder to use

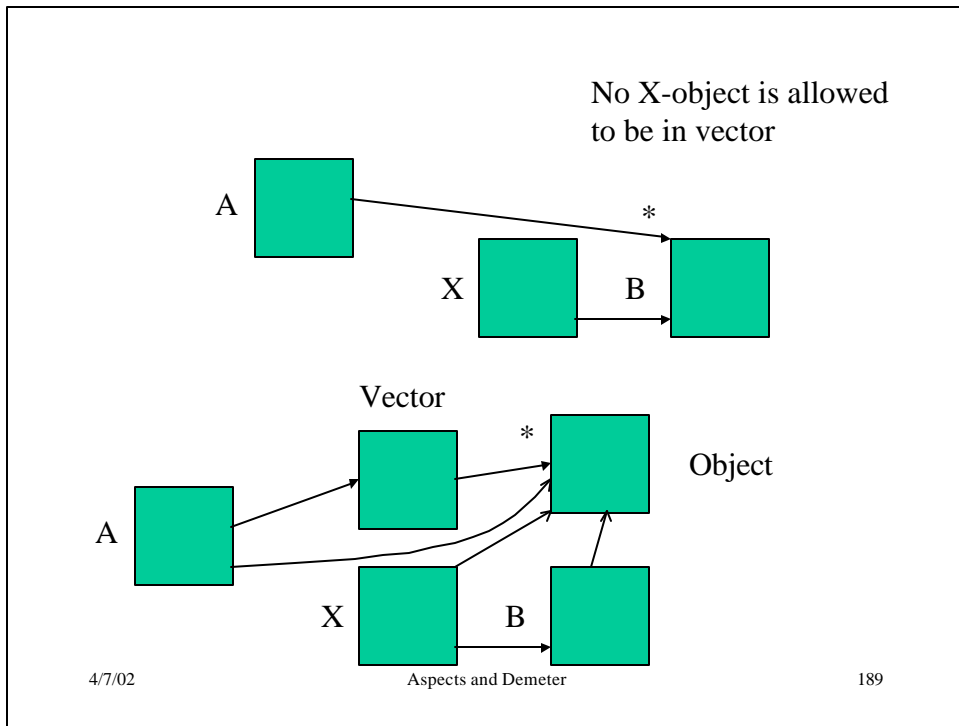
- We have:  $A = \text{Vector}(\text{Object})$ .  $\text{Object} : A \mid B \mid X$ .  $X = B$ .
- If the vector contains an X object it will be traversed!!!



4/7/02

Aspects and Demeter

188



## Moral of the story

- If the Collection objects contain only the objects advertised in the nice class graph of the application the traversal done by DJ will be correct. But unnecessary traversals still happen.
- However, if the Collection objects contain additional objects (like an X-object) they will be traversed accidentally.

## Moral of the story

- Java should have parameterized classes.
- Workaround: Use a JSR (Java Specification Request) 31 approach: use a schema notation with parameterization to express class graph and generate Java code from schema. For traversal computation, the schema will be used.

4/7/02

Aspects and Demeter

191

## Size of traversal graph

- DJ might create big traversal graphs when collection classes are involved. DJ will plan for all possibilities even though only a small subset will be realized during execution.
- To reduce the size of the traversal graph, you need to use bypassing. In the example: from A bypassing {A,X} to B.

4/7/02

Aspects and Demeter

192



## Adding Demeter Traversals to AspectJ

- Because the AspectJ join point model is a generalization of Demeter's join point model, we can use the AspectJ pointcuts to express the traversal pointcuts.
- However, this would expose the traversal implementation.
- Extend AspectJ with two new pointcut primitives: `traversal(t)` and `crossing(e)`.

4/7/02

Aspects and Demeter

193

## Traversal(t) pointcut

- picks out all join points between the start and the end of the traversal that are needed for the traversal only, i.e. the node visits and the edge visits.

4/7/02

Aspects and Demeter

194

## Note

- AspectJ is well-aligned with Demeter. Need only a small extension to add the Demeter traversals.
- Pointcuts become more verbose. Could also use traversals with visitors and extend them with AspectJ (use the args pointcut).

4/7/02

Aspects and Demeter

195

## DAJ (temporary name for AspectJ with traversals)

- planned to be added to AspectJ
- provides an efficient implementation of traversals using the AP library
- we have also added traversals to Java but implementation is very slow (using reflection)
- AspectJ + traversals > Java + traversals because AspectJ > Java

4/7/02

Aspects and Demeter

196

## Adding Demeter Traversals to AspectJ

- declare ClassGraph
- declare TraversalGraph
- declare Behavior

4/7/02

Aspects and Demeter

197

## Self-Contained Example

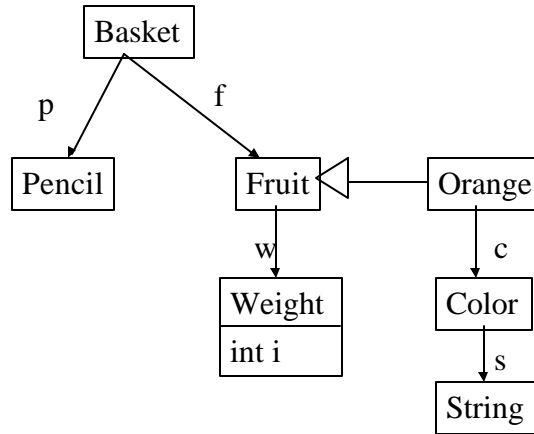
- Java classes
- Traversal file
- Use traversal defined

4/7/02

Aspects and Demeter

198

## Basket Example



4/7/02

Aspects and Demeter

199

## BasketMain.java

```
class Basket {
    Basket(Fruit _f, Pencil _p) { f = _f; p = _p; }
    Fruit f;
    Pencil p;
}
class Fruit {
    Fruit(Weight _w) { w = _w; }
    Weight w;
}
class Orange extends Fruit {
    Orange(Color _c) { super(null); c = _c; }
    Orange(Color _c, Weight _w) { super(_w); c = _c; }
    Color c;
}
class Weight{
    Weight(int _i) { i = _i; }
    int i;
    int get_i() { return i; }
}
class Pencil {}
class Color {
    Color(String _s) { s = _s; }
    String s;
}
```

4/7/02

Aspects and Demeter

200

## BasketMain.java

```
class BasketMain {  
    static public void main(String args[]) throws Exception {  
        Basket b = new Basket(  
            new Orange(  
                new Color("orange"),  
                new Weight(5)),  
            new Pencil());  
        int totalWeight = b.totalWeight();  
        System.out.println("Total weight of basket = " +  
            totalWeight);  
    }  
}
```

4/7/02

Aspects and Demeter

201

## Traversals

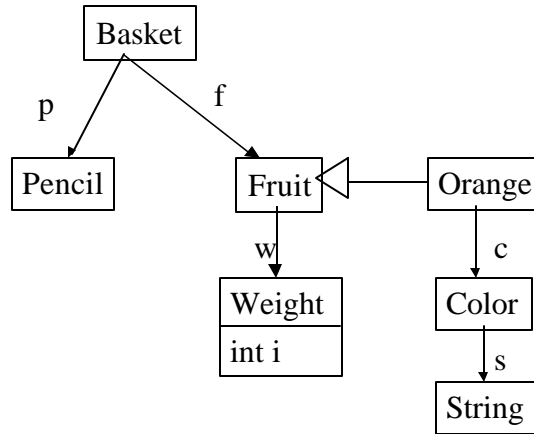
- Count the total weight within the basket
- Traversal Strategy: “From Basket to Weight”
- Visitor: Add up all the values within Weight

4/7/02

Aspects and Demeter

202

## Basket Example Traversal Graph



4/7/02

Aspects and Demeter

203

## Two versions

- define visitor using
  - AspectJ pointcuts and advice
  - Java visitor class complete with initialization, finalization and return value processing

4/7/02

Aspects and Demeter

204

## BasketTraversal.trv

```
// traversals for basket
aspect BasketTraversal {
    declare ClassGraph default;
    declare ClassGraph myClassGraph : default,
        "from Basket to * bypassing {->*,*.java.lang.String }");
    declare TraversalGraph t2 : myClassGraph, "from Basket to Weight";
}
```

4/7/02

Aspects and Demeter

205

## generated: BasketTraversal.java

```
public aspect BasketTraversal {
    // traversal t2 : {source: Basket -> target: Weight} with { }
    public void Basket.t2(){
        if (f != null) t2_crossing_f();
    }
    public void Basket.t2_crossing_f() { f.t2();}
    public void Fruit.t2(){
        if (w != null) t2_crossing_w();
    }
    public void Fruit.t2_crossing_w() { w.t2();}
    public void Weight.t2(){
    }
}

public void Orange.t2(){
    super.t2();
}

pointcut pointcut_t2() : call(public void t2*());
before () : pointcut_t2 () {
    System.out.println(thisJoinPoint);
}
} //BasketTraversal
```

for testing  
traversal code

4/7/02

Aspects and Demeter

206

## BasketMainCount.java

```
// the aspect for counting the total weight of the basket
aspect BasketMainCount {
    static int returnVal;
    int Basket.totalWeight() {
        returnVal = 0;
        t2();
        return returnVal;
    }
    pointcut t2WeightPC(Weight weight) : call(* *t2*()) && target(weight);
    before(Weight weight) : t2WeightPC(weight) {
        returnVal += weight.get_i();
    }
}
4/7/02
```

Aspects and Demeter

207

## BasketTraversal.trv

```
// traversals for basket
aspect BasketTraversal {
    declare ClassGraph default;
    declare ClassGraph myClassGraph : default,
        "from Basket to * bypassing {->*,*.java.lang.String }";
    declare TraversalGraph tg : myClassGraph, "from A to B"; // tg()
    declare Behavior void b1 : tg, Vis; //b1()
    declare Behavior Integer summing : myCg, "from A to B", Vis; //summing()
    declare traversal Integer t2(myClassGraph, SumVis) :
        "from Basket to Weight";
}
4/7/02
```

Aspects and Demeter

208

## BasketMainCount.java

```
// the aspect for counting the total weight of the basket
aspect BasketMainCount {
    int Basket.totalWeight() {
        Integer retVal = summing();
        return retVal.intValue();
    }
}
class SumVis {
    int retVal = 0;
    void before (Weight w) {retVal += weight.get_i();}
    Object returnVal() {return new Integer(retVal);}
}
```

4/7/02

Aspects and Demeter

209

## Related work

- [www.ccs.neu.edu/research/demeter](http://www.ccs.neu.edu/research/demeter)
- Or use [google.com](http://google.com) and search for “DJ DemeterJ”
- [aosd.net](http://aosd.net)

4/7/02

Aspects and Demeter

210

## Specific DJ references

- Reflection 2001 paper
- Special issue on AOP of Comm. ACM Oct. 2001
- DAJ: Demeter AspectJ: John Sung's Master's Thesis 2002: provides fast implementation for AspectJ



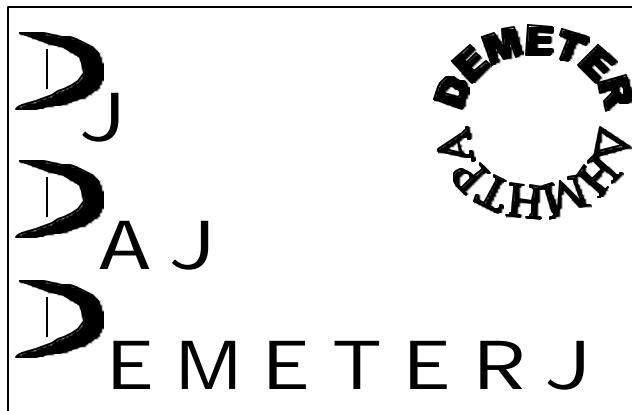
## Conclusions

- Traversal-related concerns are common
- Learning about traversal-related concerns is a good path to enter the world of AOP by using only Java
- Once the pointcuts and advice of traversals are mastered, it is easy to generalize to more pointcuts and the powerful world of AspectJ.

4/7/02

Aspects and Demeter

213



4/7/02

Aspects and Demeter

214