

A Short Introduction to Adaptive Programming (AP) for Java Programmers with AOP Interest

Karl Lieberherr
Doug Orleans

10/7/00

DJ Lieberherr/Orleans

1

Overview

- DJ introduction
- AspectJ and DJ
- Aspect-oriented Programming in pure Java using the DJ library

10/7/00

DJ Lieberherr/Orleans

2

AP

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

10/7/00

DJ Lieberherr/Orleans

3

Concepts needed (DJ classes)

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

10/7/00

DJ Lieberherr/Orleans

4

Adaptive Programming

Bold names refer to DJ classes.

Strategy



is use-case based abstraction of

ClassGraph



defines family of

ObjectGraph

10/7/00

DJ Lieberherr/Orleans

5

Adaptive Programming

Strategy



defines traversals of

ObjectGraph



plus Strategy defines

ObjectGraphSlice

10/7/00

DJ Lieberherr/Orleans

6

Adaptive Programming

Strategy



guides and informs

Visitor

10/7/00

DJ Lieberherr/Orleans

7

Software Design and Development with DJ (very brief)

- Functional decomposition into generic behavior
 - Decomposition into methods
 - Decomposition into formal traversal graphs
 - Decomposition into visitors
- Adaptation of generic behavior
 - Identify class graph
 - Identify traversal strategies

10/7/00

DJ Lieberherr/Orleans

8

AspectJ

DJ

- Abstract pointcut
 - set of execution points
 - where to watch
- advice
 - what to do
- Concrete pointcut
 - set notation using regular expressions

- Abstract object slice
 - set of entry/exit points
 - where to go
- visitor
 - what to do
- Actual object slice
 - path set notation using traversal strategies

10/7/00

DJ Lieberherr/Orleans

9

AspectJ: Observer Pattern: Abstract Pointcut

```
public abstract aspect Subject { ...
  abstract pointcut stateChanges();
  after(): stateChanges() {
    for (int i = 0; i < observers.size(); i++)
      { ((Observer)observers.elementAt(i)).update(); } }
  ... }
```

10/7/00

DJ Lieberherr/Orleans

10

AspectJ: Observer Pattern: Concrete Pointcut

```
aspect ColoredNumberAsSubject extends Subject
of eachobject(instanceof(ColoredNumber)) {
  pointcut stateChanges():
    (receptions(void setValue(..)) ||
     receptions(void setColor(..))); ... }
```

10/7/00

DJ Lieberherr/Orleans

11

DJ: Counting Pattern: Abstract Pointcut

```
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(); }
}
```

10/7/00

DJ Lieberherr/Orleans

12

DJ: Counting Pattern: Concrete Pointcut

```
// Prepare the traversal for the current class graph
ClassGraph classGraph = new ClassGraph();
TraversalGraph WPTraversal = new TraversalGraph
("from BusRoute via BusStop to Person", classGraph);

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

10/7/00

DJ Lieberherr/Orleans

13

DJ Example: TBR

- Terminal Buffer Rule:
 - Terminal classes (i.e., classes not defined in the present class graph), if used as a part class, must be the only part part class.
 - Address = String String Number.
 - Address = Streetname CityName ZipCode.
StreetName = String. CityName = String.
ZipCode = Number.



10/7/00

DJ Lieberherr/Orleans

14

Terminal Buffer Rule (TBR)

```
//class Cd_graph
public void TBRchecker(
  TraversalGraph definedClassNamesT,
  TraversalGraph allPartsT,
  TraversalGraph fetchIdentT) {
  // definedClassNamesT defines
  // the part of the object graph
  // that is relevant for finding
  // all defined classes.
  // allPartsT defines
  // the part of the object graph
  // that is relevant for checking
  // the TerminalBufferRule
}
```

10/7/00

DJ Lieberherr/Orleans

15

TBR: generic behavior

```
// find defined classes
DefinedClassVisitor v1 =
  new DefinedClassVisitor
(fetchIdentT);
Vector definedClasses =
  (Vector) definedClassNamesT.
  traverse(this, v1);
// check for violations
TBRVisitor v2 =
  new TBRVisitor(definedClasses);
allPartsT.traverse(this, v2);
}
```

10/7/00

DJ Lieberherr/Orleans

16

TBR: DefinedClassVisitor

```
public class DefinedClassVisitor
  extends Visitor { ...
  private Vector vNonTerminals = new Vector();
  public void before(Adj o) {
    Ident idCurrentAdj = fetchIdentT.fetch(o);
    vNonTerminals.addElement(idCurrentAdj);}
  public Object getReturnValue() {
    return vNonTerminals;
  }
}
```

10/7/00

DJ Lieberherr/Orleans

17

TBR: Adaptation to a concrete Structure 1

```
ClassGraph cg = new ClassGraph();//reflection
TraversalGraph tgl = new TraversalGraph(
  "from Cd_graph to Adj", cg);
// The purpose of traversal tg2
// is to visit all parts of all classes
TraversalGraph tg2 = new TraversalGraph(
  "from Cd_graph via Construct to Vertex", cg);
TraversalGraph tg3 = ...
  "from Adj through Vertex to Ident" ...;
CdGraph cdGraph = new CdGraph(...);
cdGraph.TBRchecker(tgl,tg2,tg3);
```

10/7/00

DJ Lieberherr/Orleans

18

AP history

- Programming with partial data structures = propagation patterns
- Programming with participant graphs
- Programming with object slices
 - partial data structures = all the constraints imposed by visitors

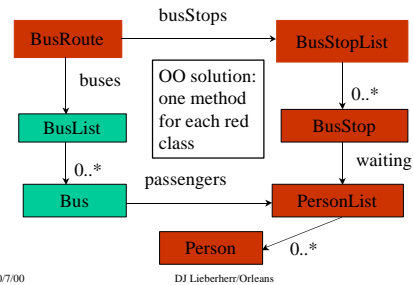
10/7/00

DJ Lieberherr/Orleans

25

Collaborating Classes

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



10/7/00

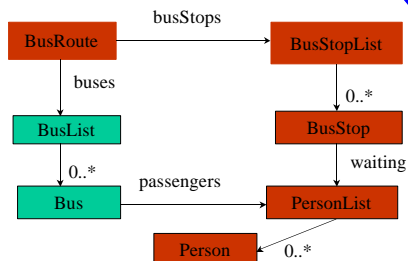
DJ Lieberherr/Orleans

26

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

Traversal Strategy

from BusRoute through BusStop to Person



10/7/00

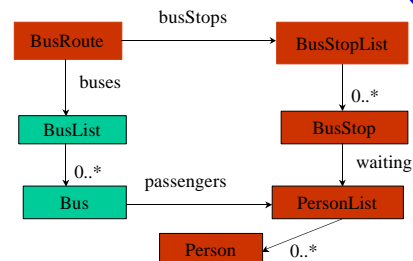
DJ Lieberherr/Orleans

27

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

Traversal Strategy

from BusRoute through BusStop to Person



10/7/00

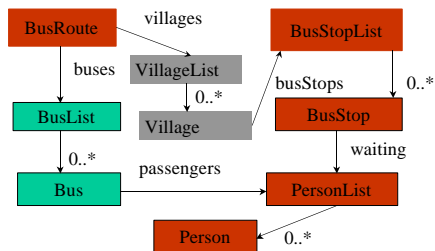
DJ Lieberherr/Orleans

28

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

Robustness of Strategy

from BusRoute through BusStop to Person



10/7/00

DJ Lieberherr/Orleans

29

Writing Adaptive Programs with Strategies (DJ=pure Java)

String WPStrategy="from BusRoute through 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();
    }
}
```

10/7/00

DJ Lieberherr/Orleans

30

Writing Adaptive Programs with Strategies (DJ=pure Java)

String WPStrategy="from BusRoute through BusStop to Person"

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

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

10/7/00

DJ Lieberherr/Orleans

31

Writing Adaptive Programs with Strategies (DJ=pure Java)

String WPStrategy="from BusRoute through BusStop to Person"

```
class BusRoute {
    int countPersons(TraversalGraph WP) {
        Integer result = (Integer)
        WP.traverse(this, new Visitor(){...});
        return result.intValue();
    }
}
```

WP.traverse(this, visitor) <====>

```
ObjectGraph objectGraph =
new ObjectGraph(this, classGraph);
ObjectGraphSlice objectGraphSlice =
new ObjectGraphSlice(objectGraph, WP);
objectGraphSlice.traverse(visitor);
```

10/7/00

DJ Lieberherr/Orleans

32

ObjectGraph

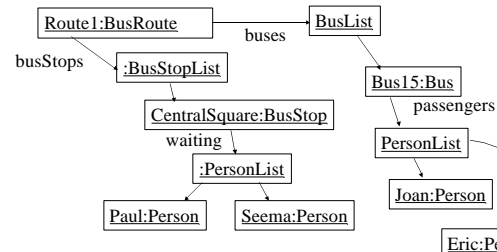
```
BusRoute (
    <busStops> BusStopList {
        BusStop(
            <waiting> PersonList {
                Person()
                Person()
            }
        }
    }
    <buses> BusList{
        Bus(
            <passengers> PersonList {
                Person()
                Person()
            }
        )
    }
}
```

10/7/00

DJ Lieberherr/Orleans

33

ObjectGraph: in UML notation



10/7/00

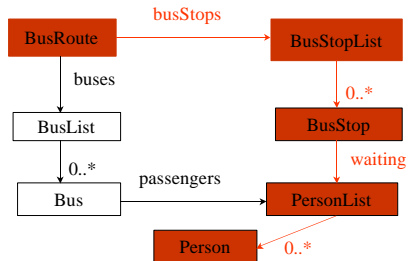
DJ Lieberherr/Orleans

34

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

TraversalGraph

from BusRoute through BusStop to Person



10/7/00

DJ Lieberherr/Orleans

35

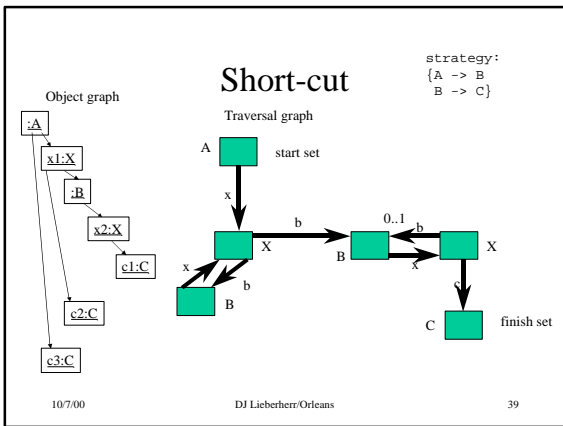
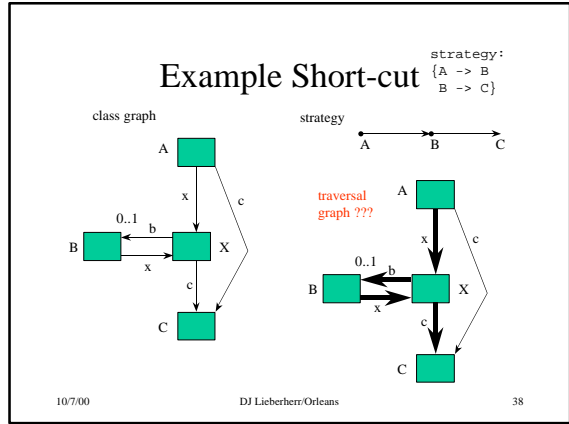
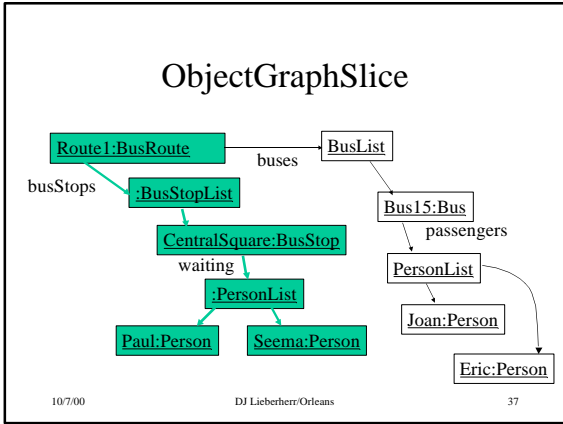
ObjectGraphSlice

```
BusRoute (
    <busStops> BusStopList {
        BusStop(
            <waiting> PersonList {
                Person()
                Person()
            }
        }
    }
    <buses> BusList{
        Bus(
            <passengers> PersonList {
                Person()
                Person()
            }
        )
    }
}
```

10/7/00

DJ Lieberherr/Orleans

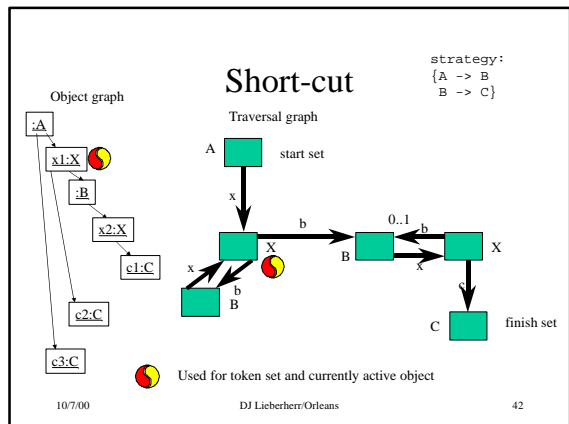
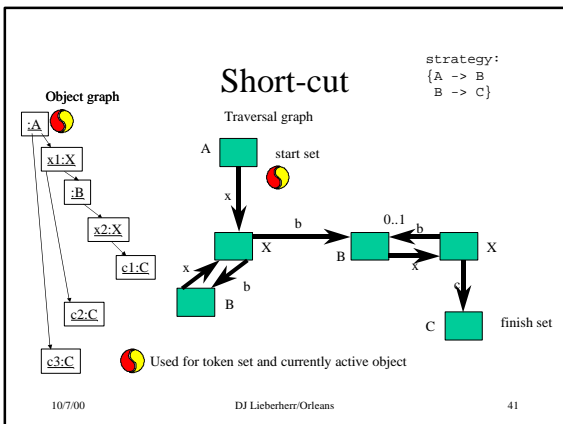
36

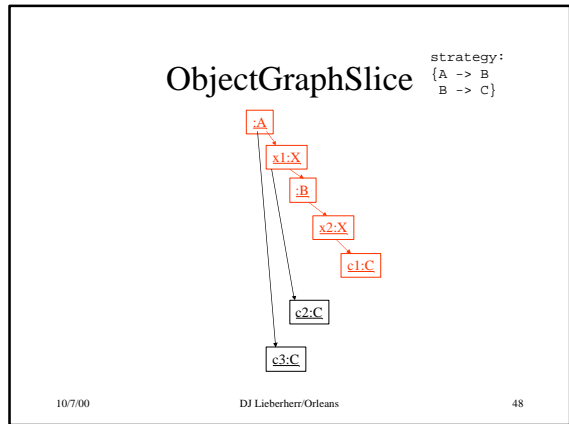
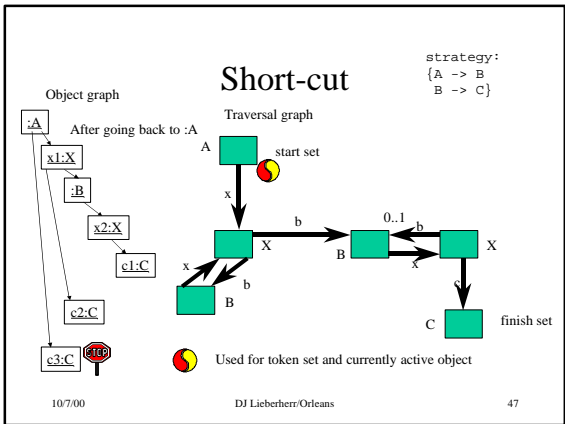
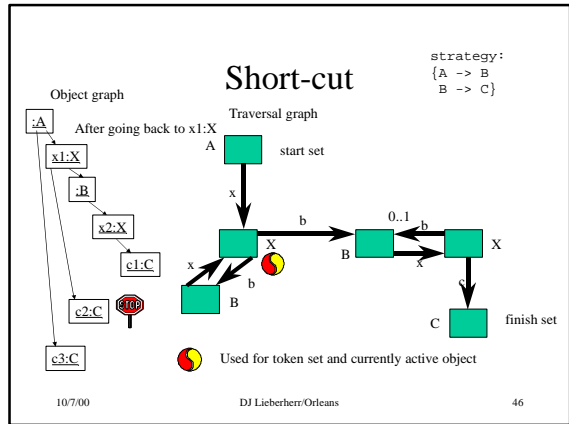
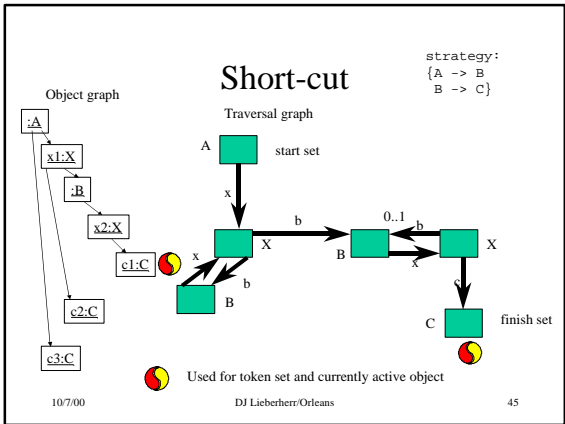
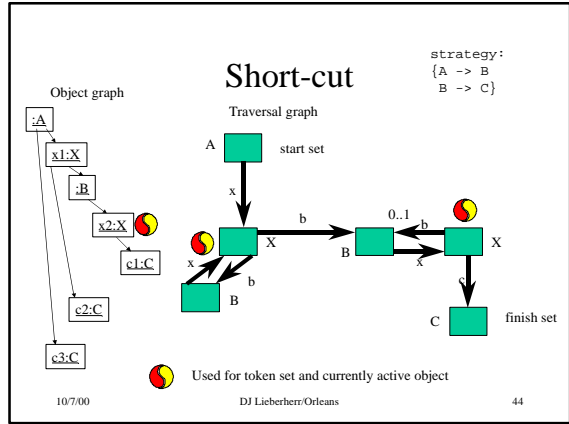
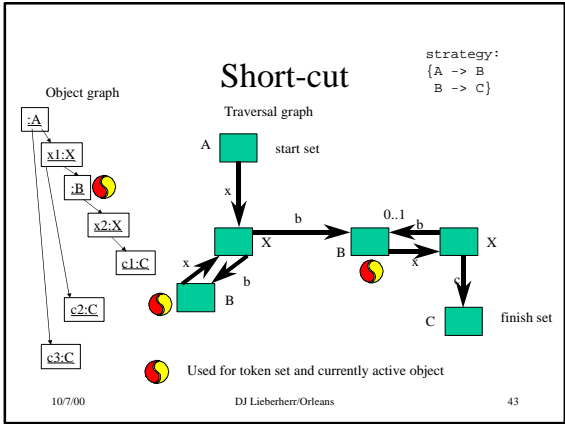


Short-cut

- ObjectGraphSlice
 - static view
 - ObjectGraph, TraversalGraph, start nodes, finish nodes
 - dynamic view
 - when traverse method gets called
 - traversal history: node sequence of nodes in object graph
 - subgraph of object graph (but only part of the story)
 - visualize it by a movie of traverse

10/7/00 DJ Lieberherr/Orleans 40





Graphs and paths

- Directed graph: (V, E) , V is a set of nodes, $E \subseteq V \times V$ is a set of edges.
- Directed labeled graph: (V, E, L) , V is a set of nodes, L is a set of labels, $E \subseteq V \times L \times V$ is a set of edges.
- If $e = (u, l, v)$, u is source of e , l is the label of e and v is the target of e .

10/7/00

DJ Lieberherr/Orleans

49

Graphs and paths

- Given a directed labeled graph: (V, E, L) , a node-path is a sequence $p = \langle v_0 v_1 \dots v_n \rangle$ where $v_i \in V$ and $(v_{i-1}, l_i, v_i) \in E$ for some $l_i \in L$.
- A path is a sequence $\langle v_0 l_1 v_1 l_2 \dots l_n v_n \rangle$, where $\langle v_0 \dots v_n \rangle$ is a node-path and $(v_{i-1}, l_i, v_i) \in E$.

10/7/00

DJ Lieberherr/Orleans

50

Graphs and paths

- In addition, we allow node-paths and paths of the form $\langle v_0 \rangle$ (called trivial).
- First node of a path or node-path p is called the source of p , and the last node is called the target of p , denoted $Source(p)$ and $Target(p)$, respectively. Other nodes: interior.

10/7/00

DJ Lieberherr/Orleans

51

Strategy definition:

embedded, positive strategies

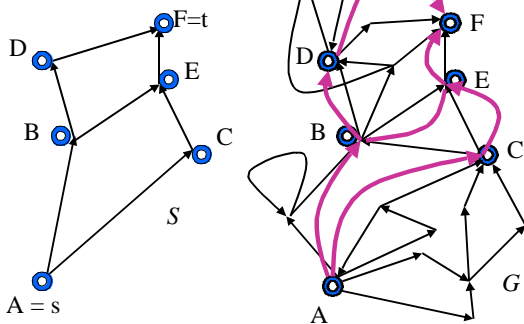
- Given a graph G , a strategy graph S of G is any subgraph of the transitive closure of G .
- The transitive closure of $G=(V, E)$ is the graph $G^*=(V, E^*)$, where $E^* = \{(v, w) : \text{there is a path from vertex } v \text{ to vertex } w \text{ in } G\}$.

10/7/00

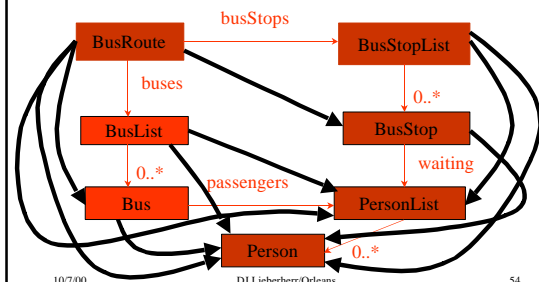
DJ Lieberherr/Orleans

52

S is a strategy for G



Transitive Closure



10/7/00

DJ Lieberherr/Orleans

54

Strategy graph and base graph are directed graphs

Key concepts

- Strategy graph S with source s and target t of a base graph G . $Nodes(S)$ subset $Nodes(G)$ (Embedded strategy graph).
- A path p is an **expansion** of path p' if p' can be obtained by deleting some elements from p .
- S defines **path set** in G as follows:
 $PathSet_{st}(G,S)$ is the set of all $s-t$ paths in G that are expansions of any $s-t$ path in S .

10/7/00

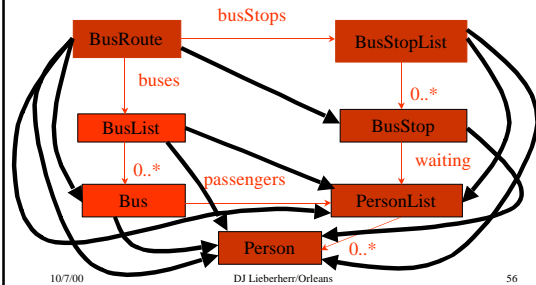
DJ Lieberherr/Orleans

55

$S = \text{from BusRoute through BusStop to Person}$

Expansion

$(BR BSL BS PL P)$ is an expansion of $(BR BS P)$



10/7/00

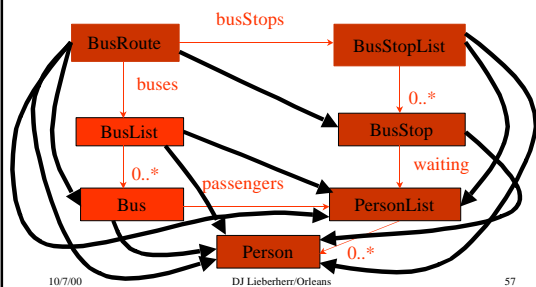
DJ Lieberherr/Orleans

56

$S = \text{from BusRoute through BusStop to Person}$

PathSet

$PathSet_{BusRoute,Person}(S,G) = \{(BR BL B PL P), (BR BSL BS PL P)\}$



10/7/00

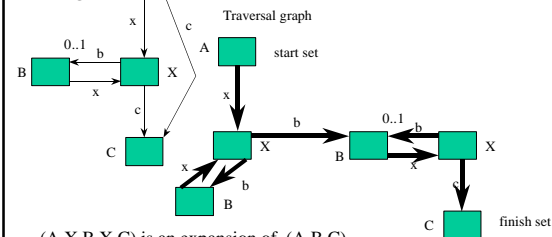
DJ Lieberherr/Orleans

57

class graph

Expansion PathSet

strategy:
 $\{A \rightarrow B$
 $B \rightarrow C\}$ S



$(A X B X C)$ is an expansion of $(A B C)$
 $PathSet_{A,C}(S,G) = \{(A X B X (B X)^* C)\}$

10/7/00

DJ Lieberherr/Orleans

58

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

10/7/00

DJ Lieberherr/Orleans

59

Integration of Generic and Adaptive Programming

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

10/7/00

DJ Lieberherr/Orleans

60

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
```

10/7/00

DJ Lieberherr/Orleans

61

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)

10/7/00

DJ Lieberherr/Orleans

62

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
```

10/7/00

DJ Lieberherr/Orleans

63

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).

10/7/00

DJ Lieberherr/Orleans

64

Gather Method

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

10/7/00

DJ Lieberherr/Orleans

65

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.

10/7/00

DJ Lieberherr/Orleans

66

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});
```

10/7/00

DJ Lieberherr/Orleans

67

DJ binaryconstruction operations

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

10/7/00

DJ Lieberherr/Orleans

68

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						*

10/7/00

DJ Lieberherr/Orleans

69

Methods returning an ObjectGraphSlice

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

Blue: constructors

10/7/00

DJ Lieberherr/Orleans

70

Traverse method arguments

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

10/7/00

DJ Lieberherr/Orleans

71

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

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

10/7/00

DJ Lieberherr/Orleans

72

Where is collection framework used?

- `ObjectGraphSlice.asList()`
 - a fixed-size `List` backed by the object graph slice.

10/7/00

DJ Lieberherr/Orleans

73

DJ unary construction operations

- Class graph from `TraversalGraph`
- Class graph from all classes in package

10/7/00

DJ Lieberherr/Orleans

74

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,sg,o)->ogs  
(cg,sg)->tg  
(cg,o)->og  
(tg,o)->ogs
```

```
cg    class graph  
s     strategy  
tg    traversal graph  
o     object  
og    object graph  
ogs   object graph slice
```

Abbreviations

10/7/00

DJ Lieberherr/Orleans

75

ClassGraph construction

- make a class graph from all classes in default package
 - `ClassGraph()`
 - include all fields and non-void no-argument methods. **Static?**
 - `ClassGraph(boolean f, boolean m)`
 - If `f` is true, include all fields; if `m` is true, include all non-void no-argument methods.

10/7/00

DJ Lieberherr/Orleans

76

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.

10/7/00

DJ Lieberherr/Orleans

77

Dynamic features of DJ ClassGraph construction

- `ClassGraph.addPackage(String p)`
 - adds the classes of package `p` to the class graph. The package is search for in the `CLASSPATH`.
- Java has no reflection for packages. Motivates above solution.

10/7/00

DJ Lieberherr/Orleans

78

Adding Nodes and Edges to ClassGraph

- `addClass(Class cl)`
 - add `cl` and all its members to the class graph, if it hasn't already been added.
- `addClass(Class cl, boolean aF, boolean aM)`
 - add `cl` 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.

10/7/00

DJ Lieberherr/Orleans

79

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` ???
- And also a `Class` argument called `target` ???

10/7/00

DJ Lieberherr/Orleans

80

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 ???

10/7/00

DJ Lieberherr/Orleans

81

Design and Implementation

- Until summer 1999: Josh Marshall
- Since then: Doug Orleans
- Available on the Web from DJ home page
- Quite complex
- Viewing an object as a list is done through coroutines to simulate continuation

10/7/00

DJ Lieberherr/Orleans

82

Problem with DJ

- 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.

10/7/00

DJ Lieberherr/Orleans

83

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 = List(B)` which we cannot express in Java. Instead we have: `A = Vector(Object)`. `Object : A | B`. Let's assume we also have a class `X=B`.

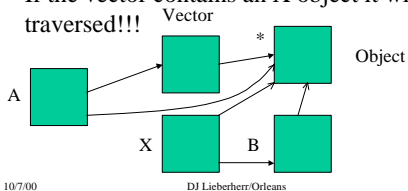
10/7/00

DJ Lieberherr/Orleans

84

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!!!

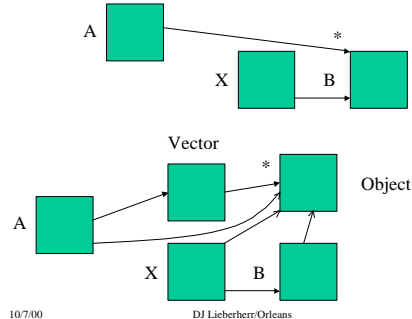


10/7/00

DJ Lieberherr/Orleans

85

No X-object is allowed to be in vector



10/7/00

DJ Lieberherr/Orleans

86

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.
- However, if the Collection objects contain additional objects (like an X-object) they might be traversed accidentally.

10/7/00

DJ Lieberherr/Orleans

87

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.

10/7/00

DJ Lieberherr/Orleans

88

Technical Details

- Using DJ and DemeterJ

10/7/00

DJ Lieberherr/Orleans

89

Combining DJ and DemeterJ

- DJ is a 100% Java solution for adaptive programming.
- DemeterJ has
 - XML style data binding facilities: code generation from schema (class dictionary).
 - Its own adaptive programming language.
- We attempt an optimal integration giving us the strong advantages of both and only few small disadvantages.

10/7/00

DJ Lieberherr/Orleans

90

Optimal DJ and DemeterJ Integration

- Take all of DJ
- Take all of DemeterJ class dictionary notation
- Take a very tiny bit of DemeterJ adaptive programming language (basically only part that allows us to weave methods).

10/7/00

DJ Lieberherr/Orleans

91

Combining DJ and DemeterJ

- Pros (advantages)
 - Java class generation from class dictionary (getters, setters, constructors).
 - Parser generation.
 - Better packaging of Java code into different files.
 - MUCH MORE POWERFUL.
- Cons (disadvantages)
 - No longer pure Java solution.
 - need to learn Demeter notation for class dictionaries (similar to XML DTD notation).
 - need to learn how to call DemeterJ and how to use project files.

10/7/00

DJ Lieberherr/Orleans

92

Combining DJ and DemeterJ

- What do we have to learn about DemeterJ?
 - Class dictionaries: * .cd files
 - Behavior files: * .beh files. Very SIMPLE!
 - A { {{ ... }} } defines methods ... of class A
 - Project files: * .prj
 - list behavior files * .beh that you are using
 - Commands:
 - demjava new, demjava test, demjava clean

10/7/00

DJ Lieberherr/Orleans

93

Combining DJ and DemeterJ

- What you might forget in class dictionaries that are used with DJ:
 - `import edu.neu.ccs.demeter.dj.*;`
 - visitors need to inherit from `Visitor`
 - parts of visitors need to be defined in class dictionary

10/7/00

DJ Lieberherr/Orleans

94

Combining DJ and DemeterJ

- Structuring your files
 - put reusable visitors into separate behavior files.
 - put each new behavior into a separate behavior file. Visitors that are not planned for reuse should also go into same behavior file. Update the class dictionary with required structural information for behavior to work.
 - List all * .beh files in .prj file.

10/7/00

DJ Lieberherr/Orleans

95

Plans for DJ

- Continued use and refinement: 65+ students this quarter
- Write compile-time checker and partial evaluator for adaptive part

10/7/00

DJ Lieberherr/Orleans

96

Example : Count Aspect Pattern

```
collaboration Counting {
  participant Source {
    expect TraversalGraph getT();
    public int count () { // traversal/visitor weaving
      getT().traverse(this, new Visitor(){ int r;
        public void before(Target host){ r++; }
        public void start() { r = 0; ... } } }
  participant Target { }
}
```

Base:
Meta variable: bold
Keywords: underscore

10/7/00

DJ Lieberherr/Orleans

97

Adapter 1

classGraph1 is
fixed and therefore
the traversal is fixed

```
adapter CountingForBusRoute1 {
  BusRoute is Counting.Source
  with {
    TraversalGraph getT() {
      ClassGraph classGraph1 = new ClassGraph();
      return
        new TraversalGraph(classGraph1,
          new Strategy("from BusRoute via BusStop to Person"));
    }
  }
  Person is Counting.Target { }
}
```

10/7/00

DJ Lieberherr/Orleans

98