

Pattern Language for Adaptive Programming (AP)

Karl Lieberherr

Northeastern University

Introduction

Four Patterns

- Structure-shy Traversal
- Selective Visitor
- Structure-shy Object
- Class Graph

On-line information

- \$D = www.ccs.neu.edu/research/demeter
- \$D is Demeter Home Page
- \$AOO = [\\$D/course/f98/](http://www.ccs.neu.edu/research/demeter/course/f98/)
- Lectures are in: [\\$AOO/lectures](http://www.ccs.neu.edu/research/demeter/course/f98/lectures)
- This lecture is in [powerpoint/PLAP.ppt](http://www.ccs.neu.edu/research/demeter/course/f98/powerpoint/PLAP.ppt) and [powerpoint/PLAP.ppt](http://www.ccs.neu.edu/research/demeter/course/f98/powerpoint/PLAP.ppt)

Summary

- Present ideas of AP at a high-level of abstraction.
- Explain concepts independent of tools and languages.
- Prepare you for homework 1.

Vocabulary

- Pattern: Reusable solution to a problem in a context.
- Class graph = Class diagram: Graph where nodes are classes and edges are relationships between the classes.
- Design pattern book: Gamma, Helm, Johnson, Vlissides: 23 design patterns

Vocabulary

- Visitor pattern: Define behavior for classes without modifying classes.
- Parser: Takes a sequence of tokens and creates a syntax tree or object based on a grammar.
- Grammar: a class graph annotated with concrete syntax.

Overview

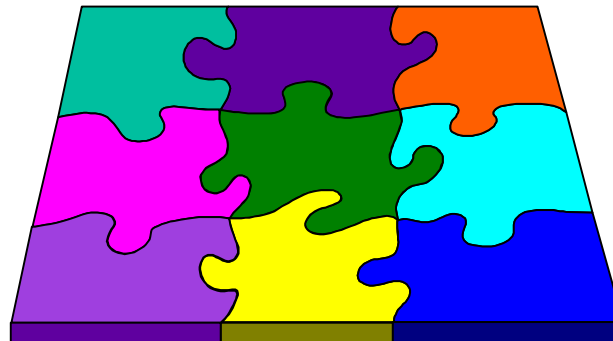
- Patterns a useful way to write down experience.
- Use a standard format: Intent, Motivation, Applicability, Solution, Consequences, etc.
- Patterns are connected and refer to each other.
- Extended version at: [SD/adaptive-patterns/pattern-lang-conv](#)

Connections

- There are several connections between the AP patterns and other design patterns.
- Class Graph is the basis for Structure-shy Traversal, Selective Visitor and Structure-shy Object.

Structure-shy Traversal

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



Solve Law of Demeter Dilemma



Small Method Goat

Big Method Goat

Structure-shy Traversal

- Could also be called:
 - Adaptive Traversal
 - Structure-shy Walker
 - Adaptive Visitor (significantly improves the Visitor pattern)

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)

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.

Structure-shy Traversal

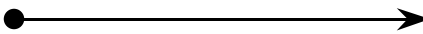

- Solution
 - Use succinct subgraph specifications
 - Use succinct path set specifications

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
 - *Third stage*: Encapsulated strategies. Use symbolic elements and map to class graph.

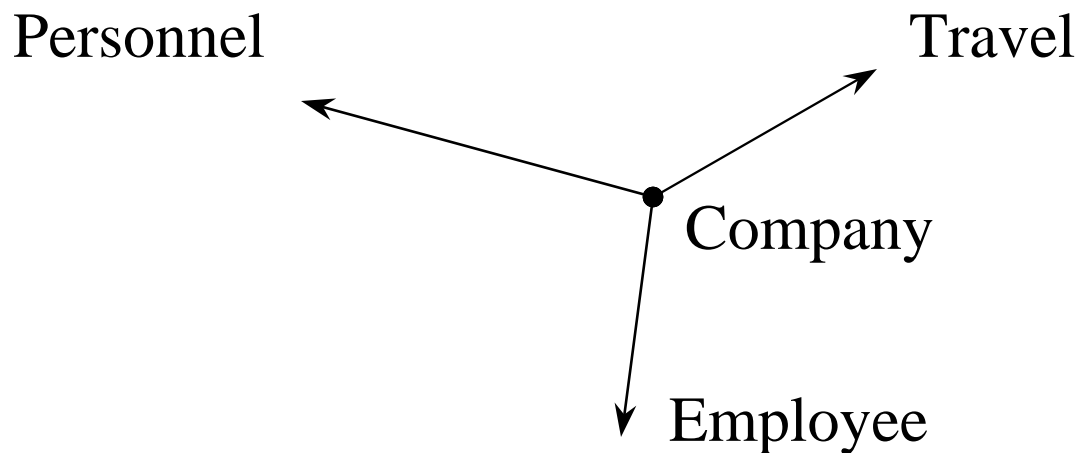


Structure-shy Traversal: Solution

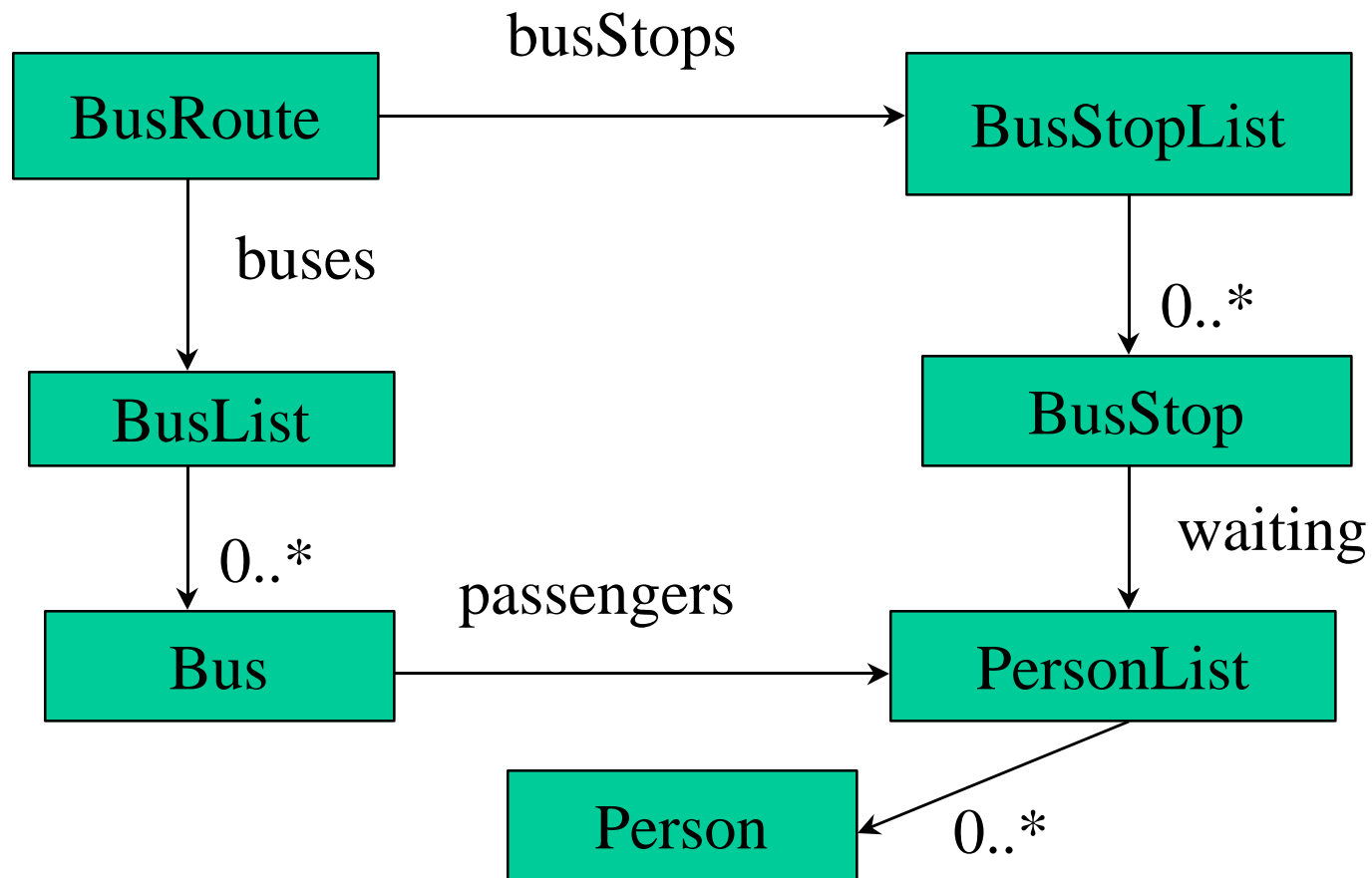
- Traversal Strategy Graphs (**Strategies**)
 - Simplest useful strategy: One Edge. Possible syntax:
 - 
 - from Company to Salary or
 - {Company -> Salary}
 - Line graph. Several edges in a line. Possible syntax:
 - 
 - From Company via Employee to Salary
 - {Company -> Employee, Employee -> Salary}

Structure-shy Traversal: Solution

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



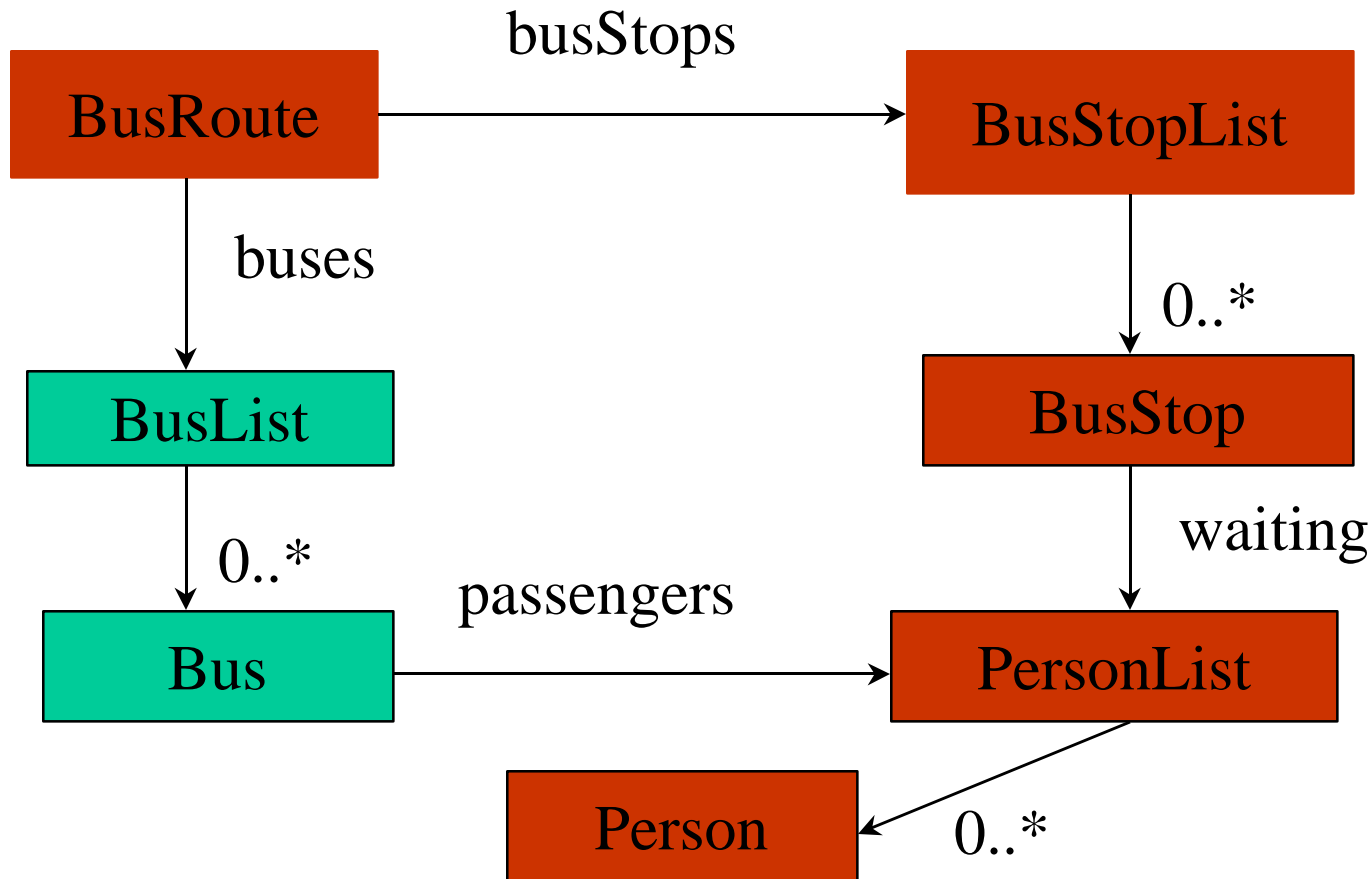
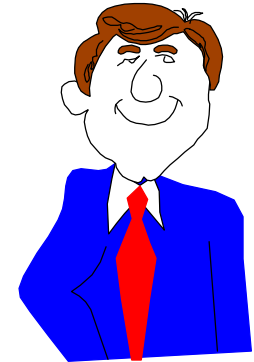
UML Class Diagram



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

Traversal Strategy

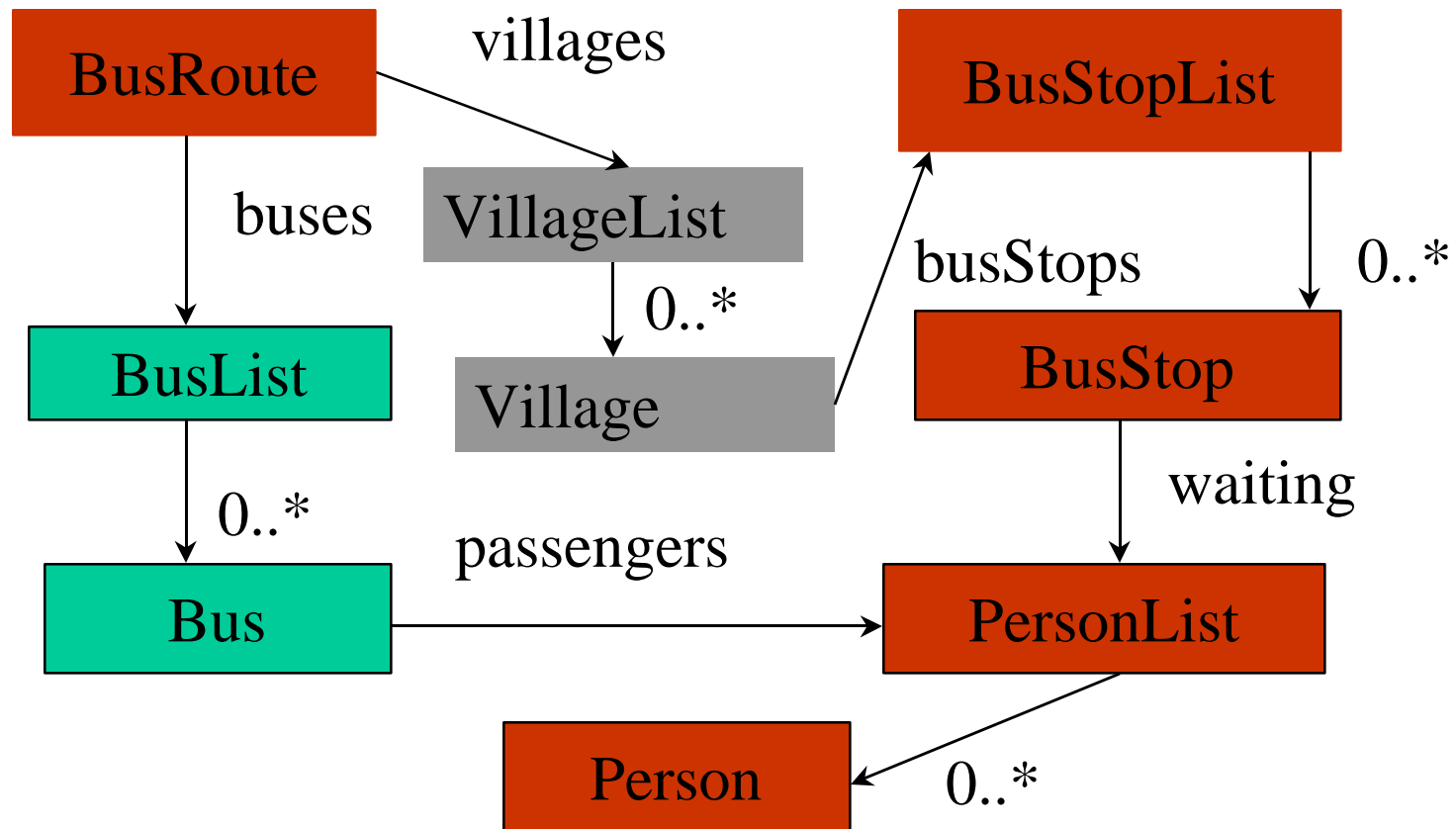
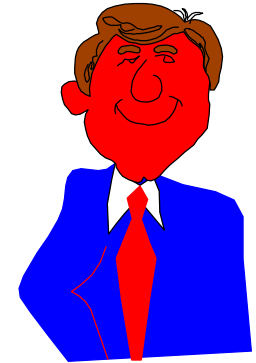
from BusRoute through BusStop to Person



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

Robustness of Strategy

from BusRoute through BusStop to Person



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.

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.

Structure-shy Traversal

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

Nature Analogy

same strategy in different class graphs: similar traversals
same seeds in different climates: similar trees



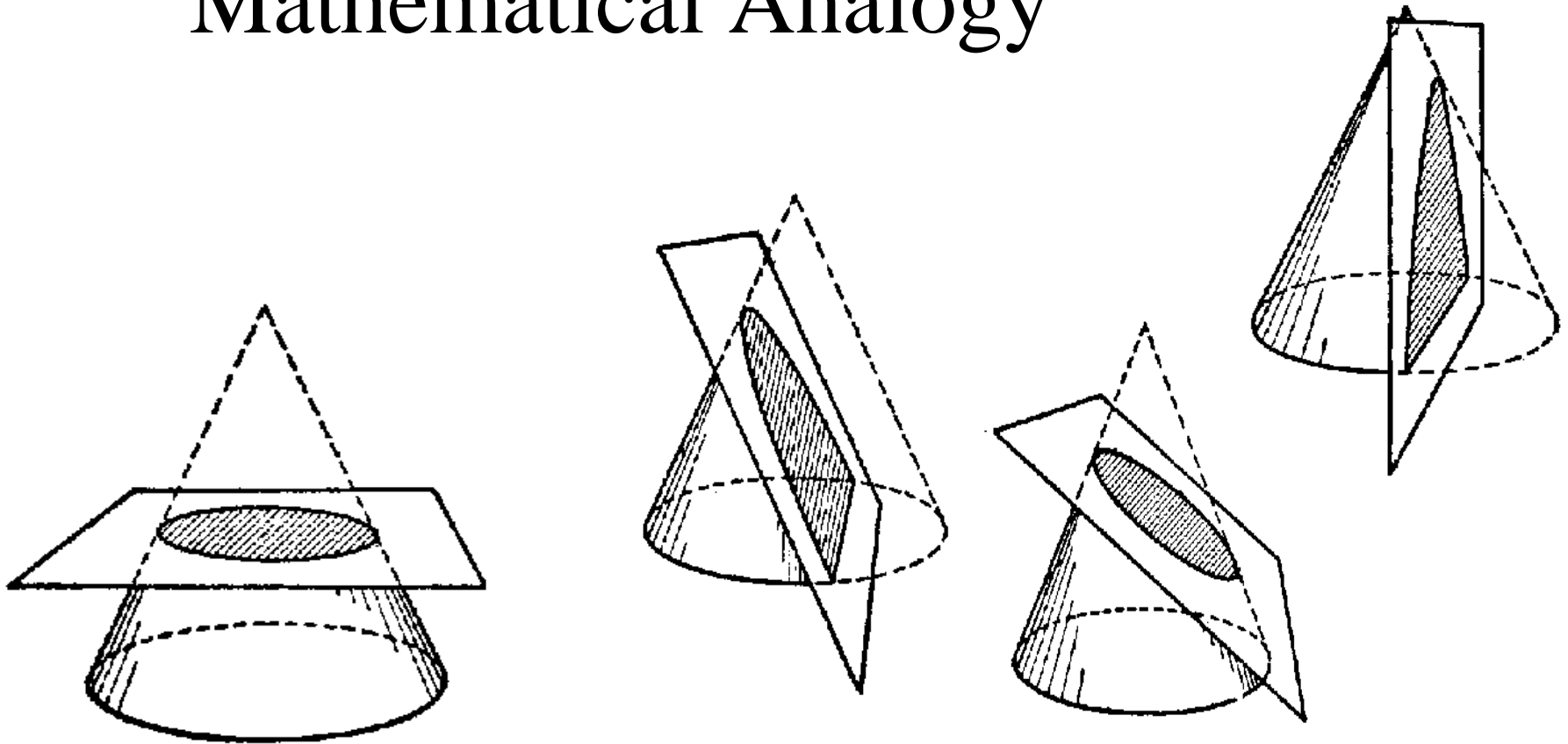
warm climate



cold climate

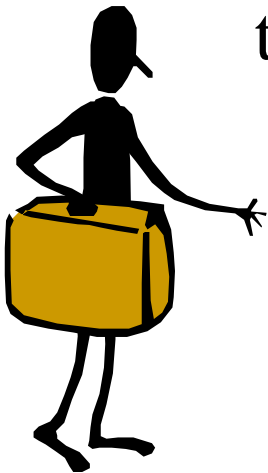
same cone different planes define different point sets
same strategy different class graphs define different path sets

Mathematical Analogy



Selective Visitor

- Intent
 - Loosely couple behavior modification to behavior and structure.
 - Would like to write an editing script to modify traversal code instead of modifying the traversal code manually.



Selective Visitor

- Could also be called:
 - Structure-shy Behavior Modification
 - Event-based Coupling

Selective Visitor

- Motivation:
 - Avoid tangling of code for one behavior with code for other behaviors.
 - Localize code belonging to one behavior.
 - Compose behaviors.
 - Modify the behavior of a traversal call (traversals only traverse).

Selective Visitor

- Applicability:
 - Need to add behavior to a traversal.

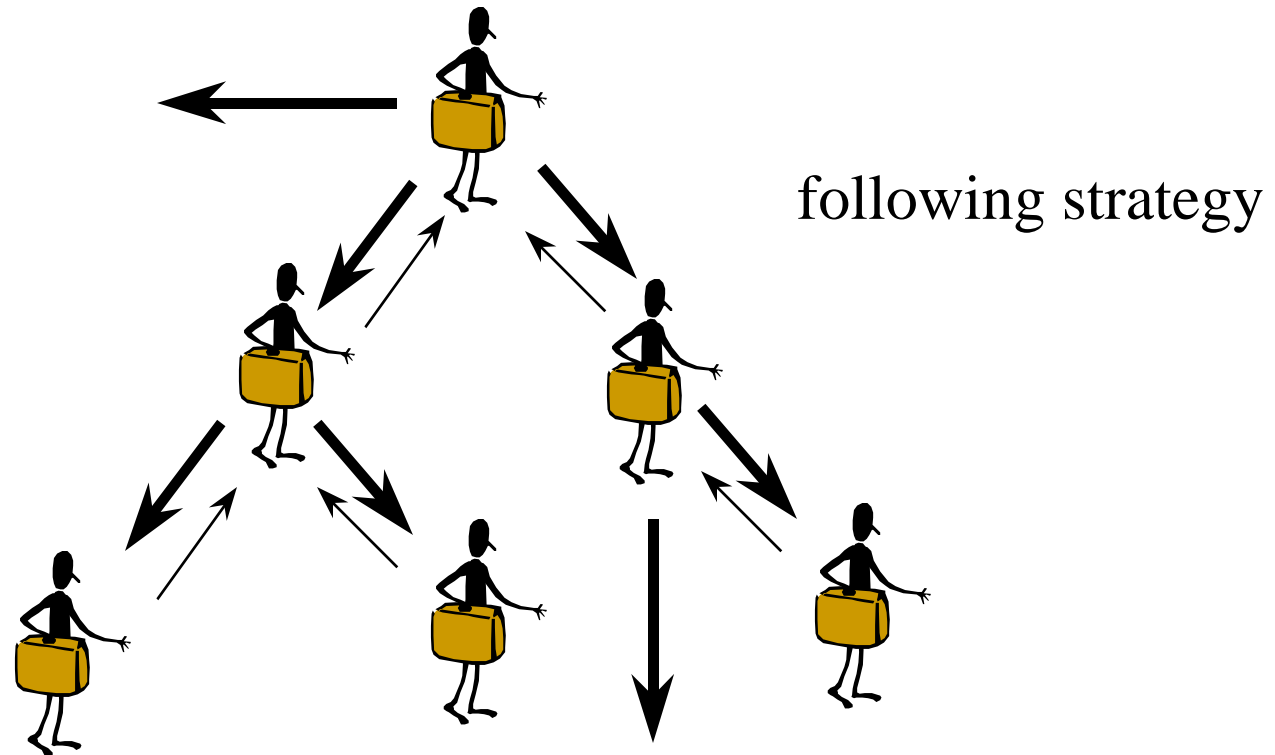
Selective Visitor

- Solution:
 - Use visitor classes and objects.
 - Pass visitor objects as arguments to traversals.
 - Either use naming conventions for visitor methods (e.g., `before_A()`) or extend object-oriented language (e.g. **before** A, **before** is a new key word).

Selective Visitor

- Solution:
 - before, after methods for nodes and edges in the class graph
 - Activated during traversal as follows:
 - Execute before methods
 - Traverse
 - Execute after methods

Visitor visits objects



Visitor collects information in suitcase (variables)

Selective Visitor

- Solution: Focus on what is important.

```
SummingVisitor {  
    (@ int total; @)  
    init (@ total = 0; @)  
    before Salary (@ total = total + host.get_v(); @)  
    return (@ total @)  
}
```

host is object visited

Code between (@ and @) is Java code

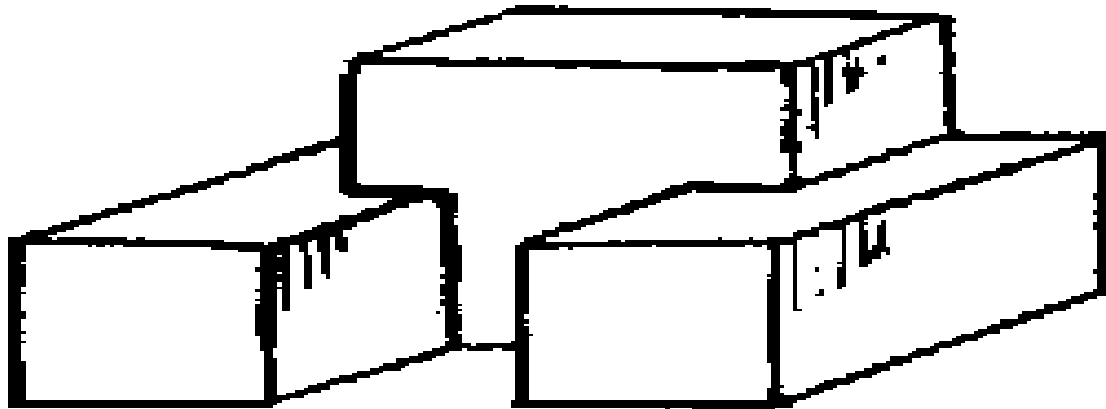
Selective Visitor

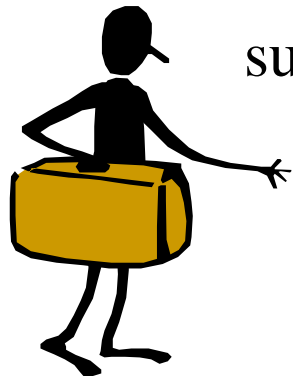
- Solution: Use of visitor

```
Company {
    traversal allSalaries(UniversalVisitor) {do S;}
    (@ int sumSalaries() {
        SummingVisitor s = new SummingVisitor();
        this.allSalaries(s);
        return s.get_return_val();
    } @)
}
```

Selective Visitor

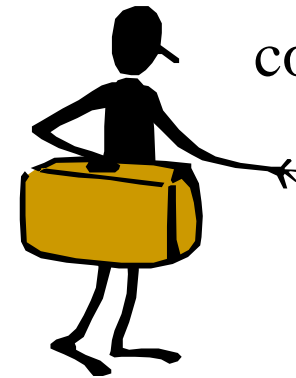
- Consequences
 - Easy behavior adjustments: Add visitor
 - Reuse of visitors





summing

Selective Visitor



counting

- Consequences: Easy behavior enhancement

```
Company { // enhancements in red  
  traversal allSalaries(UniversalVisitor, UniversalVisitor)  
    {do S;}  
  (@ float averageSalaries() {  
    SummingVisitor s = new SummingVisitor();  
    CountingVisitor c = new CountingVisitor();  
    this.allSalaries(s, c);  
    return s.get_return_val() / c.get_return_val();  
  } @)  
}
```

Writing Programs with Strategies

Example of Adaptive Program

strategy: from BusRoute through BusStop to Person

```
BusRoute {
  traversal waitingPersons(PersonVisitor) {
    through BusStop to Person; } // from is implicit
  int printWaitingPersons() // traversal/visitor weaving instr.
    = waitingPersons(PrintPersonVisitor);
PrintPersonVisitor {
  before Person (@ ... @) ... }
PersonVisitor {init (@ r = 0; @) ... }
```

Extension of Java: keywords: traversal init
through bypassing to before after etc.

Selective Visitor

- Consequences:
 - Can reuse SummingVisitor and CountingVisitor in other applications.

Selective Visitor

- Implementation
 - Translate to object-oriented language.
 - See Demeter/Java, for example.

Selective Visitor

- Known uses
 - Propagation patterns use inlined visitor objects (see AP book).
 - Demeter/Java.
 - The Visitor Design Pattern from the design pattern book uses a primitive form of Selective Visitor.

Differences to Visitor pattern

- Focus selectively on important classes. Don't need a method for each traversed class.
- Finer control: not only one accept method but before and after methods for both nodes and edges.

Structure-shy Object

- Intent
 - Make object descriptions for tree objects robust to changes of class structure.
 - Make object descriptions for tree objects independent of class names.

Structure-shy Object

- Could also be called:
 - Object Parsing
 - Grammar
 - Abstract=Concrete Syntax

Structure-shy Object

- Motivation
 - Data maintenance a major problem when class structure changes
 - Tedious updating of constructor calls
 - The creational patterns in the design pattern book also recognize need
 - Concrete syntax is more abstract than abstract syntax!

Structure-shy Object

- Applicability
 - Useful in object-oriented designs of any kind.
 - Especially useful for reading and printing objects in user-friendly notations. Ideal if you control notation.
 - If you see many constructor calls: think of Structure-shy Object.

Structure-shy Object

- Solution
 - Extend the class structure definitions to define the syntax of objects.
 - Each class will define a parse function for reading objects and a print visitor for printing all or parts of an object.

Structure-shy Object

- Solution
 - Start with familiar grammar formalism and change it to make it also a class definition formalism. In the Demeter group we use Wirth's EBNF formalism.
 - Use a parser generator (like YACC or JavaCC) or a generic parser.

Structure-shy Object Parsers weave sentences into objects

Problem in OO programs: Constructor calls for compound objects are brittle with respect to structure changes.

Solution: Replace constructor calls by calls to a parser. Annotate class diagram to make it a grammar.

Benefit: reduce size of code to define objects, object descriptions are more robust

Correspondence: Sentence defines a **family of** objects. Adaptive program defines **family of** object-oriented programs. In both cases, family member is selected by (annotated) class diagram.

Structure-shy Object

Run-time weaving: Description

Sentence

* 3 + 4 5

Grammar

Compound ...

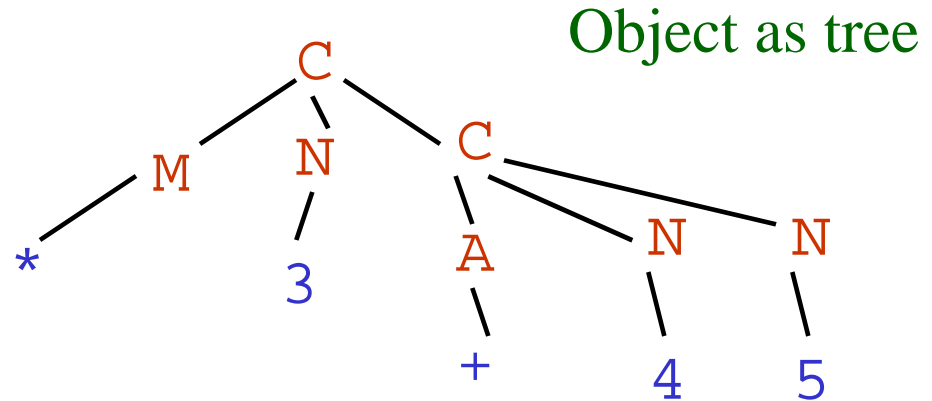
Simple ...

Number ...

Multiply ...

Add ...

etc.



Object in linear form (Constructor calls)

C M * N 3 C A + N 4 N 5

SENTENCE IS MORE ROBUST THAN OBJECT

Grammar defined by annotating UML class diagram

Structure-shy Object

- Consequences
 - more robust and shorter object descriptions
 - Need to deal with *unique readability with respect to an efficient parsing algorithm*
 - Can guarantee unique readability by adding more syntax
 - debug class structures by reading objects

Structure-shy Object

- Related patterns
 - Creational patterns in design pattern book.
 - Interpreter pattern uses similar idea but fails to propose it for general object-oriented design.
 - Structure-shy Object useful in conjunction with Prototype pattern.

Structure-shy Object

- Known uses
 - Demeter Tools since 1986, T-gen, applications of YACC, programming language Beta and many more.

Structure-shy Object

- References
 - Chapters 11 and 16 of AP book describe details.
- Exercise
 - Use your favorite grammar notation and modify it to also make it a class graph notation.

Class Graph

- Intent
 - Write class relationships once and reuse them many times.
 - Generate a visitor library from class graph for copying, displaying, printing, checking, comparing and tracing of objects.

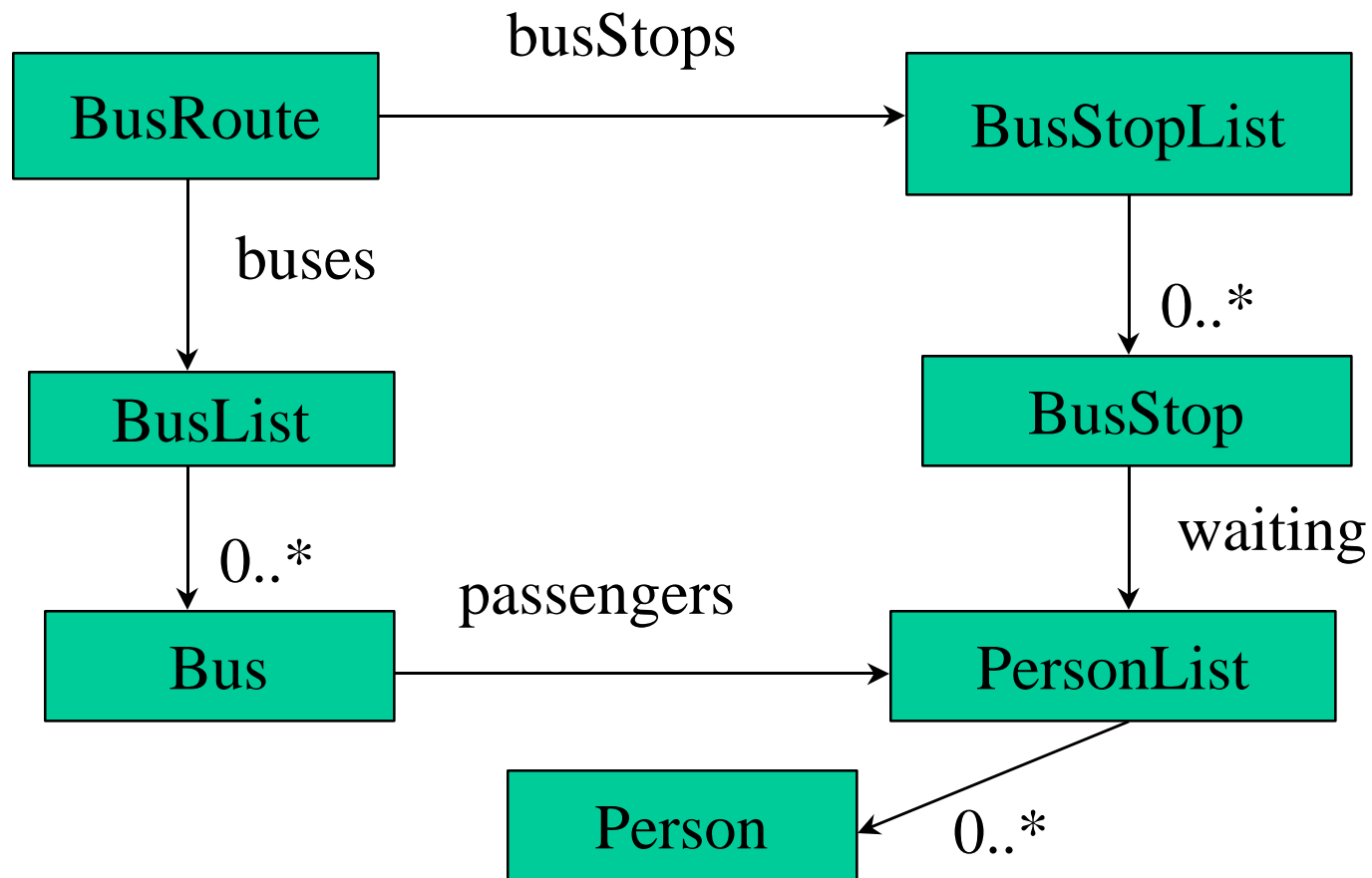
Class Graph

- Could also be called:
 - Class diagram
 - Class dictionary

Class Graph

- Applicability
 - For every application having more than one class.
- Implementation
 - Preferred: Use UML class graph model and notation (becoming a standard)
 - Use tool to generate visitor library (see Demeter/Java).

UML Class Diagram



Class Graph

- Known uses:
 - Almost all object-oriented design methods use some form of class diagram. Only Demeter/Java generates visitor library and allows strategies to refer to the class graph.
- References
 - UML class graphs, see UML books
 - Demeter class graphs, see chapter 6 of AP book

Summary

- State what has been learned: Principles of AP in high-level form.
- How to apply: Do homework one and recognize those patterns in the thousands of lines Java code. See [\\$D/course/f97/hw/1](#)

Where to get more information

- Those patterns will be discussed in much more detail.
- AP book covers the concepts.
- UML Distilled discusses UML class diagrams.
- See \$D for more information.

Feedback

- Please see me after class or send me email if you have improvements to those patterns.
- lieberherr@ccs.neu.edu