

Adaptive Plug-and-Play Components

Karl Lieberherr

Northeastern University

College of Computer Science

Demeter Group

Joint work with Mira Mezini, David
Lorenz, Linda Seiter, Mitchell Wand,
Johan Ovlinger, Doug Orleans

Outline of talk

- Overall Message: Aspect-Oriented Programming (AOP) makes software more sustainable.
- AOP idea with collaborations and adapters.
- Example: Synchronization aspect: ReadersWriters pattern.
- Some definitions (collaborations, adapters, etc.)

Outline of talk

- Example 2: Behavior slice aspect: Bus simulation query
- Explaining Demeter with XML terminology
- Summary

1/11/00

Demeter

3

Example: From Crista Lopes PhD thesis (NU/Xerox)

```
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 {
    protected AdjustableLocation loc;
    protected AdjustableDimension dim;
    public Shape() {
        loc = new AdjustableLocation(0, 0);
        dim = new AdjustableDimension(0, 0);
    }
    double get_x() throws RemoteException {
        return loc.x();
    }
    void set_x(int x) throws RemoteException {
        loc.set_x(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 double get_y() { return y_; }
    synchronized void set_x(int x) { x_ = x; }
    synchronized void set_y(int y) { y_ = y; }
    synchronized void adjust() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
}

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_ = longCalculation3();
        height_ = longCalculation4();
    }
}

```

Write this

Instead of writing this

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

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

coordinator Shape {
    self: AdjustableLocation, adjustableDimension
    mutex: adjustableLocation, get_x, set_x,
        get_y, set_y;
    mutex: adjustableDimension, 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() {}
}

```

1/11/00

Demeter

4

AOP example: inspired by AspectJ

```
collaboration ShowWAccesses {  
  participant DataToAccess {  
    expect void writeOp(*);  
    replace void writeOp(*) {  
      System.out.println("W");  
      expected(*);  
    }  
  }  
}
```

aspect description

```
adapter AddShowWAccesses {  
  connects appl, ShowWAccesses ...  
  Point is DataToAccess {  
    ... writeOp = set* ...  
  }  
}
```

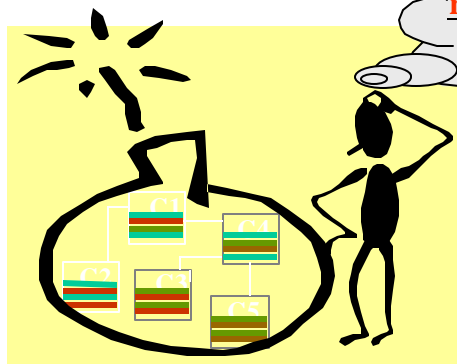
```
class Point {  
  int _x = 0;  
  int _y = 0;  
  
  void set(int x, int y) {  
    _x = x; _y = y;  
  }  
  
  void setX(int x)  
  { _x = x; }  
  
  void setY(int y)  
  { _y = y; }  
  
  int getX() {  
    return _x; }  
  
  int getY() {  
    return _y; }  
}
```

1/11/00

Demeter

5

Effects of scattering



bad, because the unit of reuse is generally not a class, but a slice of behavior affecting several classes

- “tangled code”
 - difficult to reason about
 - why is this here?
 - what does this connect to?
 - difficult to change
 - difficult to evolve
- code duplication

essentially, lack of modularity

Idea of Aspect-Oriented Programming (AOP)

- How can we make changing easier?

Localize changes!

- But: A change might intrinsically influence many methods or classes. The code needed for the change might be spread over a good fraction of the program.
- Solution: Allow programmer to refer to many points in execution of program in a localized manner. Allow additional actions to be executed at those points.

1/11/00

Demeter

7

Two things are important!

- Specify
 - additional or modified actions.
 - Actions come first!
 - when to call those actions.
 - Specification may cut across many methods. Crosscutting comes second.
- We use separate constructs to specify crosscuts and actions on crosscuts.

1/11/00

Demeter

8

UML: Unified Modeling Language: industry standard

Constructs to use: collaborations and adapters

- UML collaborations (adapted) for expressing actions decoupled from crosscuts. New: allow rewriting of methods.
- Adapters to express the crosscutting that maps actions to execution points. May contain arbitrary Java code to implement required interface of collaboration with provided interface.

1/11/00

Demeter

9

Why the title: AP&PC

- OOPSLA '98 paper (with Mezini) with follow-on technical report (with Mezini and Lorenz)
- Now we like the term collaborations and adapters better than AP&PC.
- Adapters studied in detail in conference proceeding paper (with Mezini and Seiter).

1/11/00

Demeter

10

Terminology comparison with AspectJ and HyperJ

- A collaboration is similar in nature to a hyperslice in Hyper/J.
- An adapter is similar in nature to a hypermodule in Hyper/J.
- A collaboration is similar in nature to the actions attached to a crosscut in AspectJ.
- An adapter is similar in nature to a crosscut in AspectJ.

1/11/00

Demeter

11

Rough correspondence

AOP (generic)	enhancement	crosscutting
Hyper/J	Hyperslice	Hypermodule
AspectJ	Actions Weaves	Crosscut
Demeter	Collaboration	Adapter

1/11/00

Demeter

12

Example 1: outline

- ReadersWriters synchronization pattern
- Have a data structure with read and write methods that are used in a multi-threaded environment
- Rules
 - one writer and no reader or
 - no writer and several readers

1/11/00

Demeter

13

Example 1: plan

- Describe synchronization pattern as a UML (Unified Modeling Language) collaboration.
- Describe instantiation of pattern using an adapter.

1/11/00

Demeter

14

Example 1: purpose

- Show simple (one class collaboration)
- Show benefit of separating aspect code from crosscutting instantiation.
- Show method rewriting capability.

1/11/00

Demeter

15

Example 1: collaboration

```
collaboration ReadersWriters {  
  protected int activeReaders_ = 0; ...  
  participant DataToProtect {  
    expect Object read(Object[] args);  
    expect void write(Object[] args);  
    replace Object read(Object[] args){  
      beforeRead(); Object r = expected(args);  
      afterRead(); return r;}  
    replace write(Object[] args){  
      beforeWrite(); expected(args);  
      afterWrite();}
```

1/11/00

Demeter

16

Example 1: collaboration

```
protected boolean allowReader() {
    return waitingWriters_ == 0 &&
        activeWriters_ == 0;}
protected boolean allowWriter() {
    return activeReaders_==0 &&
        activeWriters_ == 0;}
protected synchronized void beforeRead() {
    ++ waitingReaders_;
    while (!allowReader())
        try {wait();} catch (...) {}
    -- waitingReaders_; ++ activeReaders_; } ... }
```

1/11/00

Demeter

17

Example 1: adapter

```
adapter ReadersWritersUse {
    MyHashtable is ReadersWriters.DataToProtect
    with {
        read = {clone(), get(Object),
            contains(Object), elements(), isEmpty(),
            ...}
        write = {clear(), put(Object,Object),
            remove(Object), ... }
    }
}
```

1/11/00

Demeter

18

Example 1: discussion

- Simple case: one class only. Typical case: one collaboration affects many classes.
- One collaboration may affect program at many different join points. Collaboration localizes many changes cutting across one or several classes.
- Adapter describes the crosscutting: mapping of participants/methods to classes.

1/11/00

Demeter

19

What is an aspect language?

- An aspect language is a domain-specific language for specifying a collaboration pattern.
- An adapter language is a gluing language to instantiate collaboration patterns by expressing how the patterns crosscut the application.

1/11/00

Demeter

20

What is an aspect language (continued)?

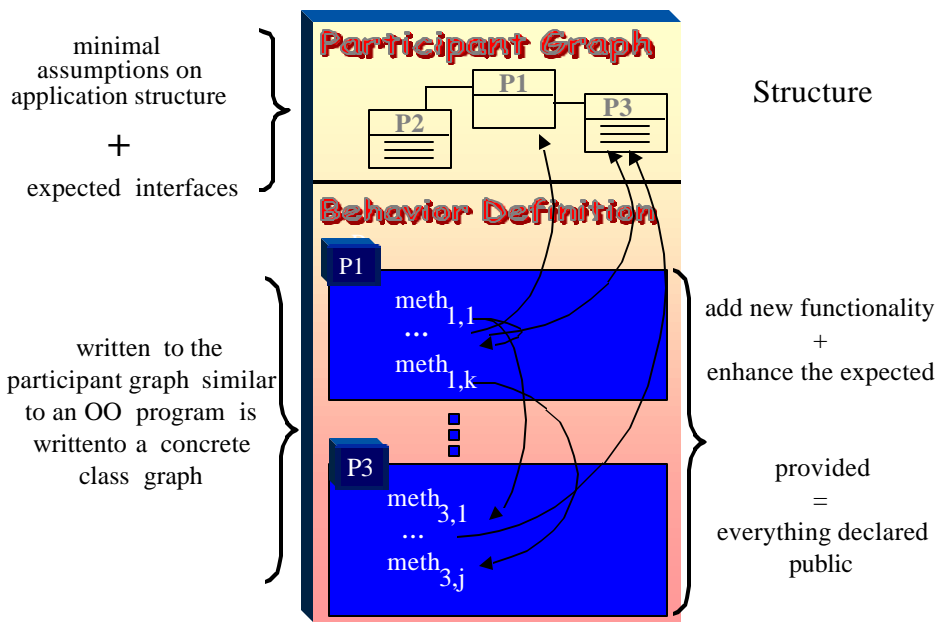
- Adaptation
 - one participant may be mapped to a set of otherwise not structurally related classes.
 - two neighboring participants may be mapped to classes that are “far apart” (many intermediate classes)
 - Adaptation crosscuts method/object structure.

1/11/00

Demeter

21

Collaborations



Adapters

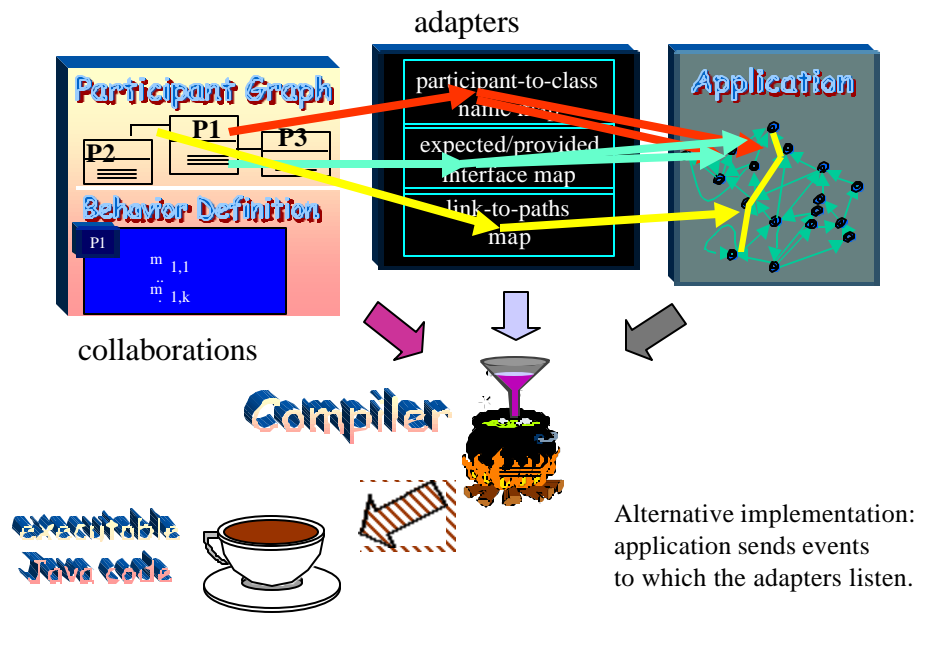
- Implement required interface in terms of provided interface.
- The adaptation bodies are written in terms of three self variables: The environment of the participant, the base environment and the adapter environment.
- Adapters express the crosscutting.

1/11/00

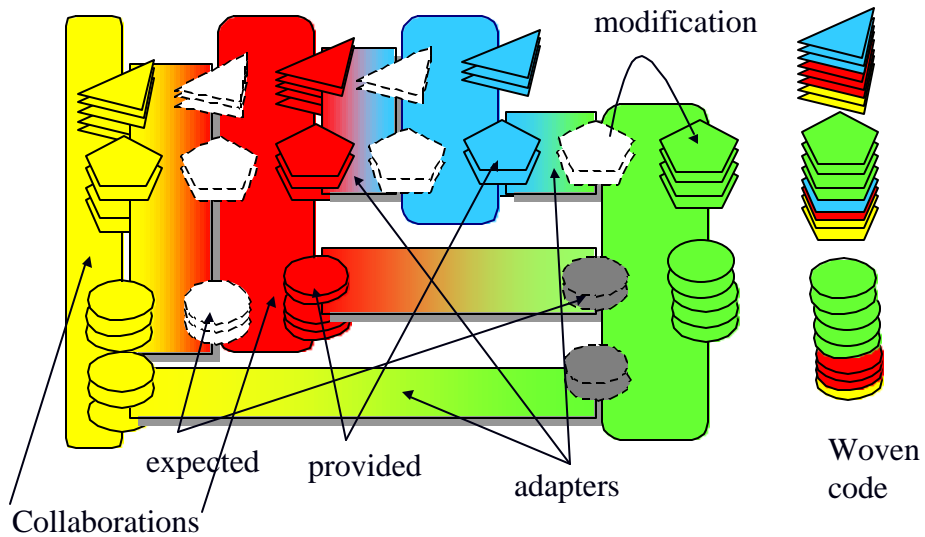
Demeter

23

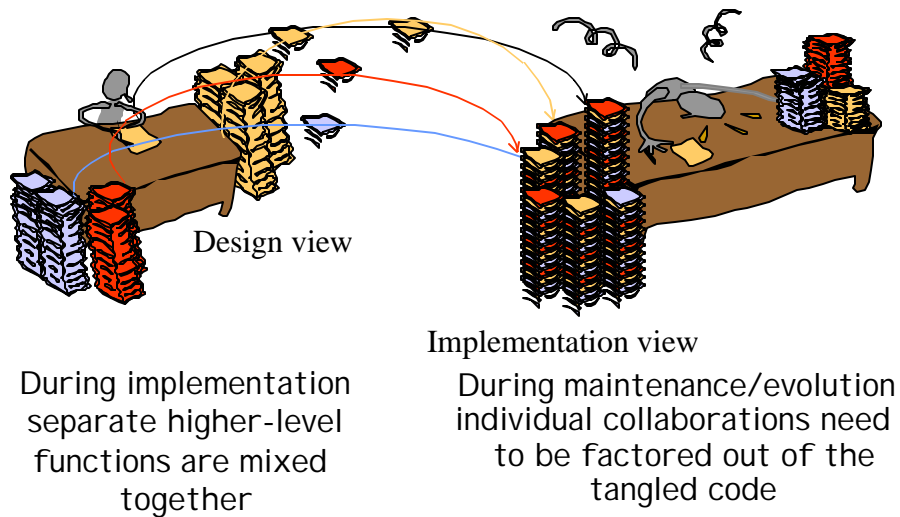
One Implementation Approach: insertion



The intuition behind collaborations/adapters



Problems without Aspects: Effects of Uncontrolled Tangling



Example 2: A simple multi-class collaboration

- Solve simple counting problems
- Define Count collaboration and use it twice
- Apply it to a bus system simulation problem

1/11/00

Demeter

27

Example 2: purpose

- Example of a functional aspect
- Demonstrates concept of adaptive programming used in Demeter.
 - Adaptive programming is good to express certain kinds of crosscuts in a robust way

1/11/00

Demeter

28

Example 2: Count Collaboration

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

1/11/00

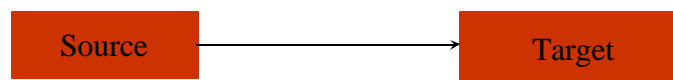
Demeter

29

Example 2: Count Collaboration

```
participant Target {}
}
```

Participant Graph:



Use in Bus simulation example:

Source -- BusRoute

Target -- Person

1/11/00

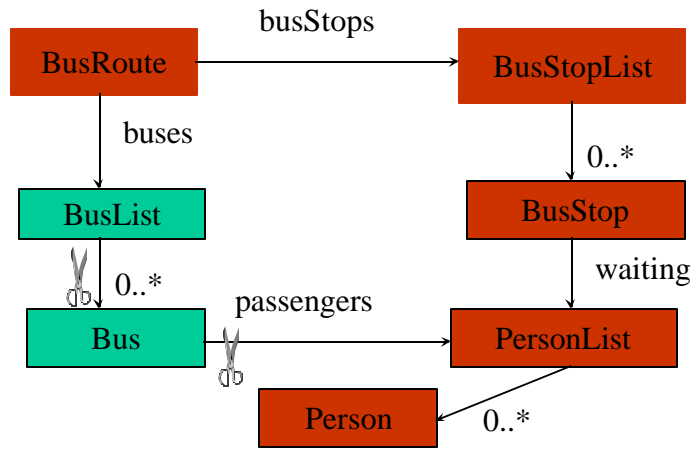
Demeter

30

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

Use 1

from BusRoute via BusStop to Person



1/11/00

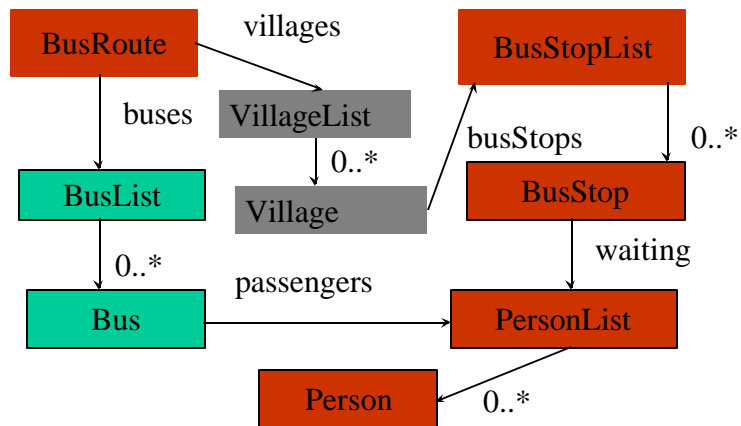
Demeter

31

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

Use 2

from BusRoute via BusStop to Person



1/11/00

Demeter

32

Adapter 1

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

1/11/00

Demeter

33

Adapter 2

```
adapter CountingForBusRoute2 {
  BusRoute is Counting.Source
  with {
    TraversalGraph getT() {return
      new TraversalGraph(classGraph2,
        new Strategy(
          "from BusRoute via BusStop to Person"));}
  }
  Person is Counting.Target { }
}
// ClassGraph classGraph2 = new ClassGraph();
```

1/11/00

Demeter

34

Discussion

- Program (collaboration and adapter) adapts to changing class graph. Adaptive Programming (AP) applied to AOP.
- Collaborations work well both for non-functional aspects (like synchronization) as well as functional aspects (like counting).
- Use 2: crosscutting goes through seven classes.

1/11/00

Demeter

35

Discussion

- Adaptive Programming (AP): Programming with traversals that are represented succinctly.
- Adapters are good clients of AP because one edge in a participant graph crosscuts often several classes in application.
- AP gives AOP relief from the structural concern: important crosscutting represented succinctly.

1/11/00

Demeter

36

Crosscut

- Crosscut refers to certain times during the execution of the program. Examples:
 - when a method is called on an object in a specified set of objects
 - events of interest: method t called on objects in a specified set of classes
 - whenever a method in a specified method set is called

1/11/00

Demeter

37

Demeter explained with XML or: how can XML use OO/AOP?

Schema (similar to an XML schema)

Object descriptions (e.g., XML documents)

Demeter produces:

Java classes with basic capabilities to process descriptions: parser, various kind of visitor classes for printing, copying, comparing, etc. Java objects e.g., produced by the parser from object descriptions.

Behavior (Java with support for traversal/visitor programming)

Synchronization descriptions (COOL)

Remote invocation/data transfer descriptions (RIDL)

1/11/00

Demeter

38

Separated concerns in Demeter/Java

- Object structure (.cd)
- Object description (.input)
- Behavior (.beh)
- Navigation (where do you want to go?)
- Synchronization (.cool)
- Remote invocation/data transfer (.ridl)

1/11/00

Demeter

39

AOP idea in XML

- Java server page contains
 - XML description (concern: structure of info.)
 - Java code (concern: how to compute info.)
- XSL description (concern: how to display information)
- Three integrated concerns: Structure, Retrieval, Display

1/11/00

Demeter

40

AP idea in XML

- XPath: navigation language of XML
- A subset of the Demeter navigation language (for adaptive navigation) is contained in Xpath
- Can write robust XML code using same techniques as in Demeter
- `//Chapter//Paragraph` = from Root via Chapter to Paragraph

1/11/00

Demeter

41

How to learn more

- <http://www.ccs.neu.edu/research/demeter>
 - free book, software, papers, comparisons etc.

1/11/00

Demeter

42

Some Related Work

- Other groups (see Demeter home page)
 - Xerox PARC: influential AOP paper applies AOP to other areas than OO. AspectJ tool.
 - IBM Watson Research Lab. Subject-Oriented Programming. Multidimensional separation of concerns. Hyper/J tool.
 - University of Twente (Netherlands): Composition Filters
 - etc.

1/11/00

Demeter

43

Summary

Software becomes more sustainable if we use Aspect-Oriented Programming (AOP) ideas

- localize enhancements that crosscut application
- express enhancements using collaborations and adapters with crosscuts in adapters.
- main goal: control tangling between concerns

1/11/00

Demeter

44

Summary

- express enhancements with two constructs
 - UML collaborations: use traversal/visitor programming when more than two objects are traversed.
 - adapters: use Adaptive Programming (AP) to express collaboration instantiations that involve graph embeddings. Use regular expression-like constructs.