Presentation Assumptions

• Already know the concepts of AOP
• Know the problem AOP is attempting to solve.
• Have some knowledge of Java
• Familiar with OOD
Outline

• Introduction to AspectJ
• Structure Shyness with Context Passing
• Implementing Branches with AspectJ
• Conceptual Analysis of DJ, AspectJ, Fred
• Master’s Thesis
Introduction to AspectJ

- Developed at Xerox PARC
- Launched in August 1998
- Provide AOP tool to programmers
- Build and support AspectJ community
- Improve AspectJ and AOP
Concepts of AspectJ

- Join Points
- Pointcuts
- Advice
- Aspect
Highlevel View of AspectJ

- **Pointcut**
- **Advice**
- **Join point**

AspectJ

Java Program
Join Points

- Transition points or boundaries
  - Method Call
  - Class Boundary
  - Method Execution
  - Access or Modification of member variable
  - Exception Handlers
  - Static and Dynamic Initialization
Graphical View of Join Points

bar1.foo();

bar
int x;

foo()
{x = 0;
if (x < 0)
throw KException;
}
Specifying Join Points with Designators

- `call` (method signature)
- `handler` (exception name)
- `cflow` (join point designator)
- `this` (type name)
- `target` (type name)
- `within` (class name)
- `execution` (method signature)
- `get` (signature), `set` (signature)
- `initialization` (signature), `staticinitialization` (type name)
Designators with Wildcards

- call(* foo())
- call(public bar.*(..))
- call(void foo(..))
- call(* *(..))
- call(*.new(int, int))
- handler(File*Exception)
Pointcuts

- Set of join points
- Treats pointcut designators as predicates
- Can use &&, ||, and ! predicate modifiers
- Allows parameter passing from pointcut to the advice
Generic Pointcuts

pointcut name(args) : pointcut_designators;
Pointcut Examples

- pointcut MemberRead() : call(* *get*(..)) || call( * *Get*(..));
- pointcut MemberReadOrWrite(): MemberRead() || call(* *set*(..)) || call(* *Set*(..));
- pointcut A1(int t2, String m1): !target(t2) && (call(* c*.foo*(..)) || * get(m1));
Advice

• Action to take when pointcut holds

• Advice Forms
  • before(param) : pointcut(param) {body}
  • after(param) : pointcut(param) {body}
  • after(param) returning []: pointcut(param) {body}
  • after(param) throwing []: pointcut(param) {body}
  • type around(param) [throws typelist] : pointcut(param) {body}
Advice Examples

- `before() : MemberRead() { ... }
- `after() : MemberReadOrWrite() { .. }
- `before(): MemberRead() && within(Lock) { .. }

Parameter Information Flow

- Advice
- Pointcut
- Join Point Designator
- Advice Body
Parameter Example

```java
before(int index) : MemberRead(index) {
    System.out.println(index);
}

pointcut MemberRead(int index) :
    args(index) &&
    (call(*get*(..)) || call(*Get*(..)));```
Introduction

• Additions to a class
  • Data members
  • Methods
  • Inheritance and Interface Specification
• Easy to group different concerns within a program into aspects
  • Structure
  • Collaborations
Introduction Examples

• public int foo.bar(int x);
• private int foo.counter;
• declare parents: mammal extends animal;
• declare parents: MyThread implements MyThreadInterface;
Aspect

- Encapsulates an aspect
- Looks like a class definition in Java
- `aspect aspect_name { ... }`
- Abstract aspects
Reflection in AspectJ

- `thisJoinPoint`
- Similar to “this” in Java
- Able to find various information about the join point that triggered the advice.
thisJoinPoint Example

- thisJoinPoint.toString()
- thisJoinPoint.getArgs()

aspect TraceNonStaticMethods {
    before(Point p): target(p) && call(* *(..)) {
        System.out.println("Entering " + thisJoinPoint + " in " + p);
    }
}
Structure Shyness with Context Passing

- Need to add functionality to a method
- Need to add context to do it
- Traditional method
  - change signature of method or overload
  - change method body or add method body
example – context passing

workers need to know the caller:
• capabilities
• charge backs
• to customize result
context-passing aspects

workers need to know the caller:
• capabilities
• charge backs
• to customize result
context-passing aspects

\texttt{pointcut invocations(Caller c):}
\hspace{1em}this(c) && call(\texttt{void Service.doService(String)});
context-passing aspects

pointcut invocations(Caller c):
    this(c) && call(void Service.doService(String));

pointcut workPoints(Worker w):
    target(w) && call(void Worker.doTask(Task));
context-passing aspects

pointcut invocations(Caller c):
   this(c) && call(void Service.doService(String));

pointcut workPoints(Worker w):
   target(w) && call(void Worker.doTask(Task));

pointcut perCallerWork(Caller c, Worker w):
   cflow(invocations(c)) && workPoints(w);
context-passing aspects

abstract aspect CapabilityChecking {

    pointcut invocations(Caller c):
        this(c) && call(void Service.doService(String));

    pointcut workPoints(Worker w):
        target(w) && call(void Worker.doTask(Task));

    pointcut perCallerWork(Caller c, Worker w):
        cflow(invocations(c)) && workPoints(w);

    before (Caller c, Worker w): perCallerWork(c, w) {
        w.checkCapabilities(c);
    }
}
Branches using AspectJ

- AspectJ have similar concepts as Fred
  - Predicates on method calls and arguments
  - Bodies to be executed
(define-msg fact)

(define-branch (and (eq? (dp-msg dp) fact)
  (= (car (dp-args dp)) 1))
  1)

(define-branch (eq? (dp-msg dp) fact)
  (let (((x (car (dp-args dp)))))
    (* x (fact (- x 1)))))
Branches in AspectJ

Java Program

Initial Call

FactClass.fact(int x)

if (x==1)

if (x>1)

x-1

Base Advice

Recursion Advice

AspectJ
Branches in AspectJ

class FactClass {}
aspect FactExample {
    static int factReturn;
    static int FactClass.fact(int x) {
        return factReturn;
    }
    pointcut factRecursion(int x):
        args(x)
        && call(static int FactClass.fact(int))
        && if (x > 1);
    pointcut factBase(int x):
        args(x)
        && call(static int FactClass.fact(int))
        && if(x==1);
    before(int x) : factRecursion(x) {
        System.out.print("recurse x = ");
        System.out.println(x);
        factReturn = x * FactClass.fact(x-1);
    }
    before(int x) : factBase(x) {
        System.out.print("base x = ");
        System.out.println(x);
        factReturn = 1;
    }
}
Conceptual Analysis

- Why can I implement Branches in AspectJ?
- Taking a look at DJ, AspectJ and Branches
  - Describe different concepts used
  - Relationships between concepts
  - Possible future directions
Conceptual Breakdown

- DJ
  - Traversals, Advice, Strategy Graph, Class Graph, Visitor, Incremental Behavior
- AspectJ
  - Introduction, Predicated Execution, Advice, Incremental Behavior
- Branches
  - Incremental Behavior, Predicated Execution
DJ Conceptual Application

- Concepts applied to data structure
- Concepts have bigger scope
  - Things are applied to bigger groups
- Incrementally add visitors
AspectJ Conceptual Application

- Introduction of almost anything within a java class
  - Allows very flexible grouping of program
- Predicated execution on control flow
- Incrementally add aspects
Branches Conceptual Application

- Each node is a function call
- Each exit from a node is a branch
- Predicated execution based on arguments on a branch
- Introduction of Branches and nodes
- Incrementally add functions and branches
Master’s Thesis

• Attempt to implement AOP concepts using different AOP tools
• Mix different AOP tools together
• Generalized view of AOP tools and concepts
AspectJ Resources

• www.AspectJ.org
  • downloads, documentation, articles, examples
• aosd.net
  • collection of AO software development