Aspect-Oriented Software Development

Johan Fabry - jfabry @ dcc

Overview

- Introduction to AOSD
- AspectJ Overview
  - General introduction
  - Language description
- AspectJ Examples
- Hot Research Topics

Introduction to AOSD

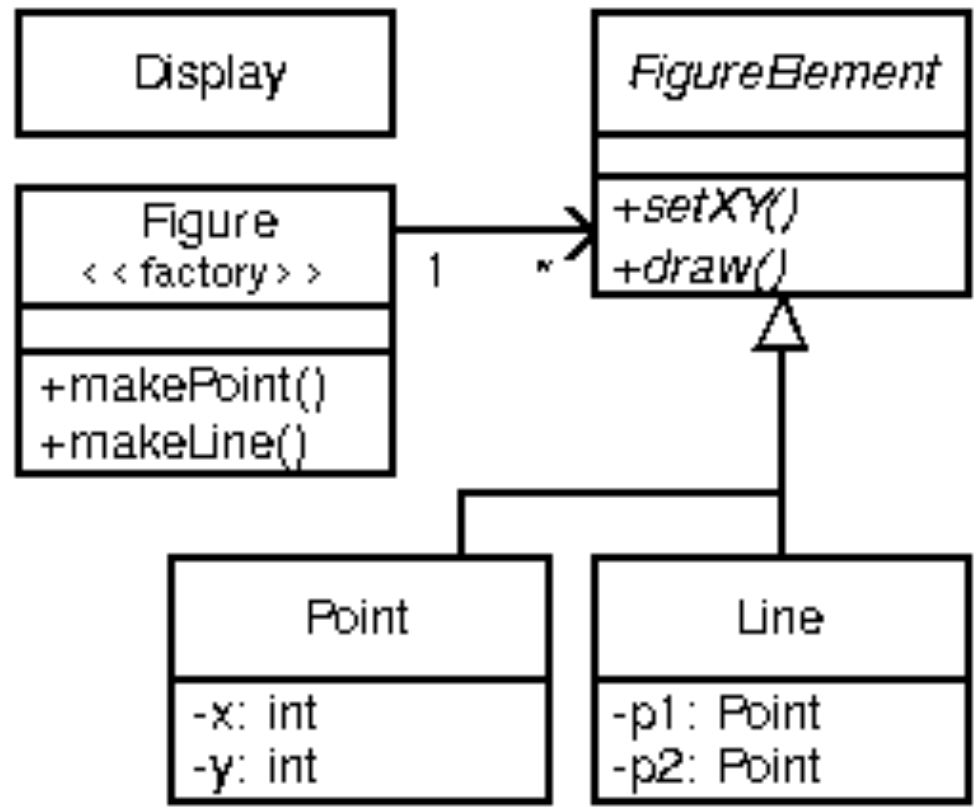
- Presentation from AOSD-Europe
  - http://www.aosd-europe.net
  - Teaching section: Introduction To AOSD

Overview

- Introduction to AOSD
- AspectJ Overview
  - General introduction
  - Language description
- AspectJ Examples
- Hot Research Topics

AspectJ Introduction

- Running example: FigureEditor



Join Point Model

- For now only method call join points
- Considered dynamically
- Contains a dynamic context

Pointcuts

```
call(void Point.setX(int))
```

```
call(void Point.setX(int)) ||
call(void Point.setY(int))
```

&& , || , !

```
call(void FigureElement.setXY(int,int)) ||
call(void Point.setX(int))              ||
call(void Point.setY(int))              ||
call(void Line.setP1(Point))            ||
call(void Line.setP2(Point))
```

```
pointcut move():
    call(void FigureElement.setXY(int,int)) ||
   [...]
```

Pointcuts (II)

```
call(void Figure.make*(..))
```

```
call(public * Figure.* (..))
```

Property-based x-cutting

```
cflow(move())
```

Dynamic Context

Advice

```
before(): move() {
  System.out.println("about to move");
}
```

```
after() returning: move() {
    System.out.println("just successfully
moved");
}
```

```
after() throwing: move() {...}
after(): move() {...}
```

```
around(): move () {...}
```

Advice & Context

```
pointcut setXY(FigureElement fe, int x, int y):
  call(void FigureElement.setXY(int, int))
  && target(fe) && args(x, y);
```

this()
target()
args()

```
after(FigureElement fe, int x, int y) returning:
setXY(fe, x, y) {
    System.out.println(fe +" moved "+x+" "+y);}
```

Advice & Context (II)

```
after(FigureElement fe, int x, int y) returning:
        call(void FigureElement.setXY(int, int))
        && target(fe) && args(x, y) {
  System.out.println(fe +" moved "+x+" "+y);}
```

Aspects

```
aspect Logging {

  pointcut move():
  call(void FigureElement.setXY(int,int)) ||
  [...]

  before(): move() {
      logStream.println("about to move");
  }

}
```

Inter-Type Declarations

```
  aspect PointObserving {
    private Vector Point.observers
      = new Vector();

    public static void addObserver(Point p,
Screen s) { p.observers.add(s);}

    public static void removeObserver(Point p,
Screen s) { p.observers.remove(s);}

    pointcut changes(Point p): target(p) &&
call(void Point.set*(int));
...
}
```

Static mechanism

Language Description

- Pointcuts
- Advice
- Inter-Type declarations
- Other declarations
- Aspects

Pointcuts (1)

Methods & Constructors

```
call(Signature)
execution(Signature)
```

Fields

```
get(Signature)
set(Signature)
```

Exception Handlers

```
handler(TypePattern)
```

Advice Executions

```
adviceexecution()
```

Pointcuts (II)

Initialisation

```
staticinitialization(TypePattern)
initialization(Signature)
```

Lexical stuff

```
within(TypePattern)
withincode(Signature)
```

Instanceof & Context Exposure

```
this(Type or Id)
target(Type or Id)
args(Type or Id, ...)
```

Pointcuts (II)

```
pointcut testEquality(Point p):
    target(Point) &&
    args(p) &&
    call(boolean equals(Object));
```

Pointcuts (III)

Control Flow

```
cflow(Pointcut)
cflowbelow(Pointcut)
```

Conditional

```
if(Expression)
```

Combination

```
! Pointcut
Pointcut0 && Pointcut1
Pointcut0 || Pointcut1
( Pointcut )
```

Advice

```
[strictfp] AdviceSpec [ throws TypeList ] :
    Pointcut { Body }
```

AdviceSpec

```
before( Formals )

after( Formals ) returning
[ ( Formal ) ]after( Formals ) throwing
[ ( Formal ) ]after( Formals )

Type around( Formals )
```

use **`proceed()`**

Inter-Type Declarations

Methods

```
Modifiers ReturnType OnType . Id ( Formals )
[ throws TypeList ] { Body }
abstract Modifiers ReturnType OnType . Id
( Formals ) [ throws TypeList ] ;
```

Constructors

```
Modifiers OnType . new ( Formals ) [ throws
TypeList ] { Body }
```

Fields

```
Modifiers Type OnType . Id [ =
Expression ] ;
```

Inter-Type Declarations (II)

Hierarchy Changing

```
declare parents : TypePattern extends Type ;

declare parents : TypePattern implements
TypeList ;
```

Declarations

Warnings and Errors

```
declare warning : Pointcut : String ;
declare error : Pointcut : String ;
```

Make exception soft

```
declare soft : Type : Pointcut ;
```

Aspect precedence

```
declare precedence : TypePatternList ;
```

Aspects

```
[ privileged ] Modifiers aspect Id
 [ extends Type ] [ implements TypeList ]
 [ PerClause ] { Body }
```

PerClause

```
[ issingleton() ]
perthis(Pointcut)
pertarget(Pointcut)
percflow(Pointcut)
percflowbelow(Pointcut)
```

Overview

- Introduction to AOSD
- AspectJ Overview
  - General introduction
  - Language description
- AspectJ Examples
- Hot Research Topics

Tracing Example

```
public abstract class TwoDShape {[...]}
public class Circle extends TwoDShape{[...]}
public class Square extends TwoDShape{[...]}

public class Trace {

 public static void traceEntry
  (String str) { [...] }
 public static void traceExit
  (String str) { [...] }
}
```

Tracing Output

```
  --> double tracing.Square.area()
  <-- double tracing.Square.area()
  --> double
tracing.TwoDShape.distance(TwoDShape)
    --> double tracing.TwoDShape.getX()
    <-- double tracing.TwoDShape.getX()
    --> double tracing.TwoDShape.getY()
    <-- double tracing.TwoDShape.getY()
  <-- double
tracing.TwoDShape.distance(TwoDShape)
```

Tracing Aspect

```
aspect TraceMyClasses {

 pointcut myClass():
      within(TwoDShape) ||
      within(Circle) ||
      within(Square);

 pointcut myConstructor():
      myClass() && execution(new(..));
 pointcut myMethod():
      myClass() && execution(* *(..));
```

Tracing Aspect

```
  before (): myConstructor() {
   Trace.traceEntry("" +
thisJoinPointStaticPart.getSignature());}
  after(): myConstructor() {
   Trace.traceExit("" +
thisJoinPointStaticPart.getSignature());}

  before (): myMethod() {
   Trace.traceEntry("" +
thisJoinPointStaticPart.getSignature());}
  after(): myMethod() {
   Trace.traceExit("" +
thisJoinPointStaticPart.getSignature());}
}
```

Tracing Aspect V2

```
abstract aspect Trace {
 abstract pointcut myClass();

 pointcut myConstructor():
      myClass() && execution(new(..));
 pointcut myMethod():
      myClass() && execution(* *(..));
```

Tracing Aspect V2

```
 before (): myConstructor() {
  Trace.traceEntry("" +
thisJoinPointStaticPart.getSignature());}
 after(): myConstructor() {
  Trace.traceExit("" +
thisJoinPointStaticPart.getSignature());}

 before (): myMethod() {
  Trace.traceEntry("" +
thisJoinPointStaticPart.getSignature());}
 after(): myMethod() {
  Trace.traceExit("" +
thisJoinPointStaticPart.getSignature());}
```

Tracing Aspect V2

```
 protected static void traceEntry
   (String str) { [...] }
 protected static void traceExit
   (String str) { [...] }

}
```

Tracing Aspect V2

```
public aspect TraceMyClasses extends Trace {

 pointcut myClass():
      within(TwoDShape) ||
      within(Circle) ||
      within(Square);
}
```

Tracing Aspect V3

```
abstract aspect Trace {

before(Object obj): myConstructor(obj) {
 traceEntry(
  obj.toString() + ">>" +
  thisJoinPointStaticPart.getSignature());
 }

 after(Object obj): myConstructor(obj) {
  traceExit(
  obj.toString() + ">>" +
  thisJoinPointStaticPart.getSignature());
 }
[...]
```

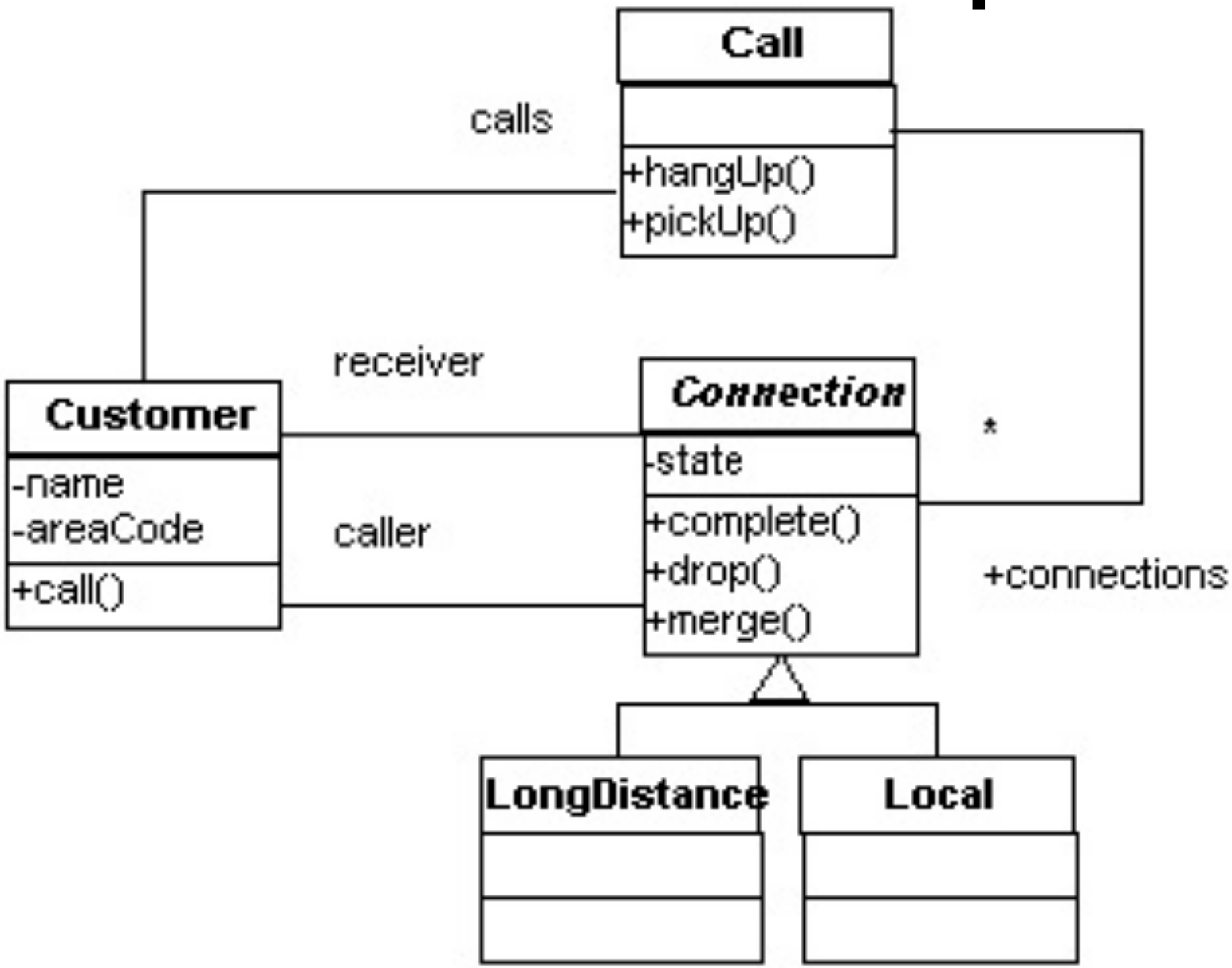
Tracing Aspect V3

```
pointcut myConstructor(Object obj):
  myClass() && this(obj) &&
  execution(new(..));

pointcut myMethod(Object obj):
  myClass() && this(obj) &&
  execution(* *(..)) &&
  !cflow(execution(String toString()));

[...]
}
```

Telecom Example



Timer Class

```
class Timer {
  long startTime, stopTime;
  public void start() {
   startTime = System.currentTimeMillis();
   stopTime = startTime;}

  public void stop() {
   stopTime = System.currentTimeMillis();}

  public long getTime() {
   return stopTime - startTime;}
}
```

Timer Logging

```
public aspect TimerLog {

 after(Timer t): target(t) &&
  call(* Timer.start())  {
      System.err.println(
       "Timer started: " + t.startTime);
    }

 after(Timer t): target(t) &&
  call(* Timer.stop()) {
      System.err.println(
       "Timer stopped: " + t.stopTime);
    }
}
```

Timing aspect

```
public aspect Timing {
 public long Customer.totalConnectTime = 0;

 public long getTotalConnectTime(Customer c){
  return cust.totalConnectTime;
  }
```

Timing aspect

```
private Timer Connection.timer =
 new Timer();

public Timer getTimer(Connection c){
 return c.timer;
 }

after (Connection c):
 target(c) &&
 call(void Connection.complete()) {
  getTimer(c).start();
  }
```

Timing aspect

```
 pointcut endTiming(Connection c):
  target(c) &&
  call(void Connection.drop());

  after(Connection c): endTiming(c) {
    getTimer(c).stop();
    c.getCaller().totalConnectTime +=
      getTimer(c).getTime();
    c.getReceiver().totalConnectTime +=
      getTimer(c).getTime();
    }
}
```

Billing aspect

```
public aspect Billing {
 declare precedence: Billing, Timing;

 public Customer Connection.payer;

 public Customer getPayer(Connection conn)
{ return conn.payer; }

 after(Customer cust)
  returning (Connection conn):
   args(cust, ..) &&
   call(Connection+.new(..)) {
        conn.payer = cust;    }
```

Billing aspect

```
public static final long LOCAL_R = 3;
public static final long LONG_DIST_R = 10;

public abstract long Connection.callRate();

public long LongDistance.callRate(){
 return LONG_DIST_R;
 }

public long Local.callRate(){
 return LOCAL_R;
 }
```

Billing aspect

```
after(Connection conn):
 Timing.endTiming(conn) {
  long time = Timing.aspectOf().
   getTimer(conn).getTime();
  long rate = conn.callRate();
  long cost = rate * time;
  getPayer(conn).addCharge(cost);
  }
```

Billing aspect

```
 public long Customer.totalCharge = 0;
 public long getTotalCharge(Customer cust){
  return cust.totalCharge;
  }

 public void Customer.addCharge(long charge)
{
  totalCharge += charge;
  }
}
```

Getting inter-type state

```
public class TimeReporter(){

[...]

 protected void report(Customer c){
  Timing t = Timing.aspectOf();
  System.out.println(c + " spent " +
   t.getTotalConnectTime(c));
  }

}
```

Dependencies & Interactions

- Billing uses Timing.endTiming
- Precedence: Billing > Timing
- Billing gets timer for connection
- `declare precedence`
  - global (= static scope)
  - compile time

Overview

- Introduction to AOSD
- AspectJ Overview
  - General introduction
  - Language description
- AspectJ Examples
- Hot Research Topics

Aspects, Dependencies and Interactions

- How do we define / detect semantic interaction? How do we specify the semantics of a concern?
- What taxonomy, categories, granularity and kind of interactions do we have?
- What scope and binding time/binding mode do we have for composition rules? How do we do interaction detection in this context?

Aspects, Dependencies and Interactions

- How are interactions detected, propagated and resolved through different stages of development process?
- Are there specific problems from pointcut models / languages in terms of aspect interactions?
- How can we deal with ad-hoc constraints, e.g. expert intuition?

Aspects, Dependencies and Interactions

- What abstraction granularity do we need to define precedence or ordering constrains?
- How do we deal with interference, e.g. use total order or partial order?
- What language mechanisms or new operators beyond aspect precedence do we need to specify resolution?

Aspects, Dependencies and Interactions

- How do we detect interactions when an aspect suppresses a join-point needed by another aspect?
- How do we analyze aspect interactions without having the base code?
- How can refactoring techniques be used to simplify dependencies and interactions?
- ...

Aspects, Dependencies and Interactions

- Early research
- Many questions, wide scope
- Many opportunities !

Domain-Specific Aspect Languages

```
private void transfer
     (BankAccount from_orig, BankAccount to_orig, int amount)
     throws TxException
{
     TransactionManager txmgr = TransactionManager.getCurrent();
     Integer self = txmgr.newID();
     txmgr.addTransaction(self);
     Integer RCS = txmgr.lookup(Thread.currentThread());
     txmgr.addToGroup("RCS"+ RCS + "Step",self);

     final Integer comp_id = txmgr.newID(); //for compensation
     txmgr.addTransaction(comp_id);
     txmgr.addToGroup("RCS"+ comp_id+ "Comp",comp_id);
     txmgr.bind("RCS"+ comp_id+ "Comp",comp_id);

     final BankAccount compfrom = from_orig; //for inner class
     final BankAccount compto = to_orig; //for inner class
     final int compamount = amount; //for inner class

     Runnable compensator = new Runnable()
{
     public void run(){
          undoTransfer(compfrom, compto, compamount, comp_id);
     }
};
     txmgr.addDependency(RCS, "ad", self);
     txmgr.addDependency(self, "wd" ,RCS);
     txmgr.addDependency(comp_id, "bcd" ,self);

     new Thread(compensator).run();
```

```
Forcing bf = txmgr.mayBegin(self);
if (bf == null){
     Object preView = txmgr.lookupGroupBinding("RCS"+ RCS + "V
     txmgr.begin(self);
     txmgr.removeViewGroup(RCS, preView);
     txmgr.delegate(RCS, self);
}
else {
     txmgr.rollback(self);
     return;
}
try {
     proceed()
     Forcing cf = txmgr.mayCommit(self);
     if (cf != null)
          throw new TxAbortedException();

     txmgr.addDependency(comp_id, "cmd" ,RCS);
     txmgr.addDependency(comp_id, "bad" ,RCS);

     txmgr.bindGroup("transferGroup","RCS"+ RCS + "View")
     Object newView = txmgr.lookupGroupBinding("RCS"+ RCS + "V
     txmgr.addViewGroup(RCS, newView);
     txmgr.delegate(self, RCS);

     txmgr.commit(self);
}
catch (TxException ex){
     txmgr.mayAbort(self);//will always succeed
     txmgr.rollback(self);
     throw ex;
}}
```

Domain-Specific Aspect Languages

```
Cashier.transfer(BankAccount, BankAccount, int) {
  alias (Saga <Thread.currentThread()>);
  groupAdd(self <""+Saga+"Step">);}
  autostart (transfer(BankAccount, BankAccount, int)
    <dest, source, amount> {
      name(self <""+Saga+"Comp">);
      groupAdd(self <""+Saga+"Comp">);});
  begin {
    alias (Comp <""+Saga+"Comp">);
    dep(Saga ad self, self wd Saga, Comp bcd self); }
  commit {
    alias (Comp <""+Saga+"Comp">);
    dep(Comp cmd Saga, Comp bad Saga); }}
```

Reuse Progression

- Low-Level Code
- Abstraction & Abstract Data Types
- Modules
- Objects
- Frameworks
- Domain-Specific Languages

DSLs

- Small (declarative)
  - Focussed on the domain
- Code = concepts
  - Concise code
- Constrain the programmer
  - No hacking!

DSL Advantages

- Problem domain abstraction level & idioms
  - usable for domain experts
  - conserve & reuse domain knowledge
  - validation & optimization at domain level
- Self-documenting, reusable programs
- + productivity, + reliability, + maintainability

DSALs

- DSL for a Crosscutting concern.
- Cool:

```
per_class coordinator A, B {
    selfex A.f, A.g, B.f;
    mutex {A.f, B.h, B.i};
 }
```

DSALs

```
coordinator BoundedBuffer {
 selfex put, take; mutex {put, take};
 condition empty = true, full = false;
 put: requires !full;
   on_exit {
     if(empty) empty = false;
     if(usedSlots == capacity)full = true;}
 take: requires !empty;
   on_exit {
      if (full) full = false;
      if (usedSlots == 0) empty = true;}
 }
```

Why not DSAL?

- First priority: explore
  - Examine Aspect Language Design Space
- Language and Weaver Effort
  - Language scope
  - Need infrastructure (Reflex)
- User education

Why DSALs?

- Bring DSL advantages to the aspect world
- Domain info useful for aspect interaction!

DSAL Research

- Many questions:
  - pointcut models?
  - reuse of (partial) DSL definitions?
  - DSAL Engineering?
  - DSAL & Interactions ?
- Many opportunities!