UNIVERSITÄT SALZBURG

Proseminar Software Engineering II

AspectJ

Emilia Farcas

emilia.farcas@cs.uni-salzburg.at

CS, University of Salzburg, Austria

SOFTWARE RESEARCH LAB

Resources

- http://eclipse.org/aspectj/
- http://aosd.net/

AspectJ

- an extension to the Java programming language that adds aspect-oriented programming (AOP) capabilities
- captures the structure of cross-cutting concerns explicitly, in a modular way

Aspects

- Constructs that dynamically affect program flow
  - A join point is a well-defined point in the program flow. A pointcut picks out certain join points and values at those points.
  - A piece of advice is code that is executed when a join point is reached

Aspects

- Constructs that statically affects a program's class hierarchy
  - inter-type declarations - allow the programmer to modify the members of the classes and the relationship between classes.

Aspects

- behave somewhat like Java classes
- but may also include pointcuts, advices, and inter-type declarations.

Join Points

- method & constructor call
- method & constructor execution
- field get & set
- exception handler execution
- static & dynamic initialization

- Each method call at runtime is a different join point

Pointcuts

- A pointcut can be built out of other pointcuts with &&, ||, and !
- we can use wildcards.

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

pointcut move():
      call(void Point.setX(int))  ||
      call(void Point.setY(int))

execution(!static * *(..))
call(* setY(long))
within(MyClass)
```

Pointcuts

- cflow - identifies join points that occur in the dynamic context of other join points.

- cflow(P) && cflow(Q)
- cflow(P && Q)

Pointcuts

- primitive pointcuts **this, target** and **args**
  - **this** picks out each join point where the currently executing object is an instance of a particular type
  - **target** picks out each join point where the target object (the object on which a method is called or a field is accessed) is an instance of a particular type.

```
!this(Point) && call(int *(..))
```

```
args(int, .., String)
```

Advices

- Before, after, or around

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

- An advice declaration has a parameter list
- The advice's pointcut publishes the values for the advice's arguments

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

Advices

- Within all advice bodies the special variable **thisJoinPoint** is bound to an object that describes the current join point.

```
before (): myMethod() {
        Trace.traceEntry(thisJoinPointStaticPart.getSignature());
    }
```

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));

    after(Point p): changes(p) {
        Iterator iter = p.observers.iterator();
        while ( iter.hasNext() ) {
            updateObserver(p, (Screen)iter.next());
        }
    }

    static void updateObserver(Point p, Screen s) {
        s.display(p);
    }
}
```