Programming with Aspects

Venkat Subramaniam

venkats@agiledeveloper.com

http://www.agiledeveloper.com/download.aspx

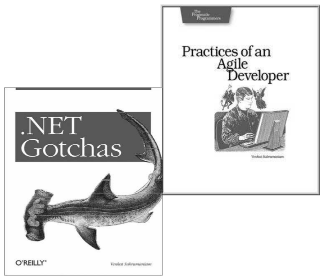
Abstract

Abstract OOP is currently the most popular and practical software development approach. One of the reasons for its popularity is the ability to separate concerns, focusing on behaviors as they relate to business or technical issues. But this very same capability reaches its limits in OOP when it comes to global and crosscutting concerns. Aspect Oriented Programming is receiving attention for its ability to address these concerns. How is it similar and different from OOP? What are the traits of AOP and what are the limitations of utilizing it in projects? In this interactive presentation, the speaker will introduce AOP, discuss its capabilities and benefits, and share his cautious optimism on how to put it to use in your projects.

About the Speaker *Dr. Venkat Subramaniam*, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada and Europe. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat has been a frequent speaker at No Fluff Just Stuff Software Symposium since Summer 2002.



Programming with Aspects

- **Limitation of OO**
- Separation of Concerns
- Aspect Oriented programming
- AspectJ
- PointCut
- Advice
- Pitfalls
- Conclusion

Software Development

- Various methodologies have evolved
- Object-Oriented Paradigm is the most popular and practical in effect currently
- System composed of objects/ entities
- Used in all kinds of application development

Reasons to use OO

- Helps us manage complexity
- If done well, easier to make change
- Component based approach to developing systems
- But, what are the limitations of OO?

Limitations of OO

- OO advocates decomposing a system into entities
- As complexity increases, the limitations surface
- Breaking system into objects helps manage complexity
- However, can all the system concerns be decomposed into an object?
  - Not really
  - Move commonality into a base class?
  - How about spreading them across several objects?
  - Makes it harder to keep up with the change

Programming with Aspects

- Limitation of OO
- **Separation of Concerns**
- Aspect Oriented programming
- AspectJ
- PointCut
- Advice
- Pitfalls
- Conclusion

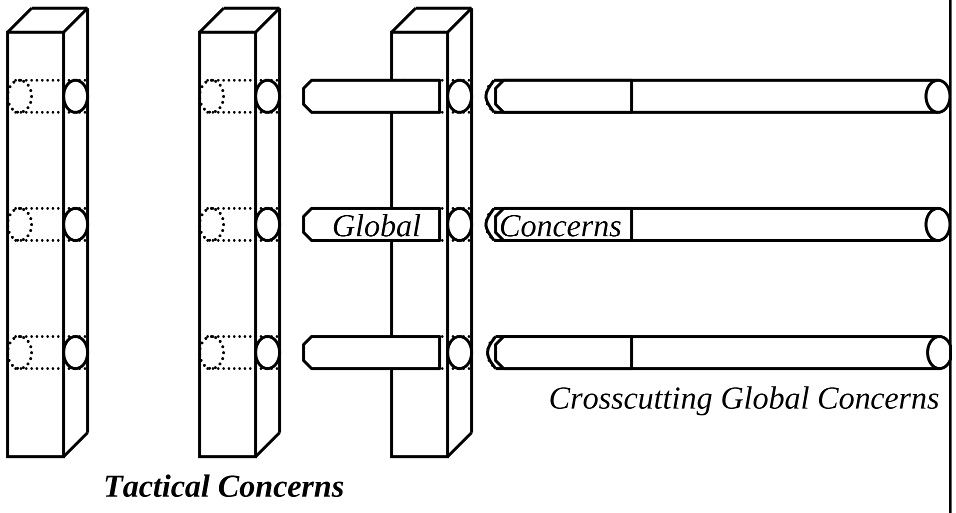
Separation of Concerns

- We have heard this in OOP
- We want to separate the concerns in our system into manageable pieces
- OO does this to a certain extent
- But what about concerns at global level
- Concerns like security, transaction, tracing, logging, error handling, etc.

Crosscutting Concerns

- Some concerns are fairly localized within entities
- Other concerns cut across multiple elements in the system
- How about keeping these cross cutting concerns separately and weaving them horizontally into the system?

Weaving the system



Advantages

- Could we not write these as functions & call?
  - Results in code permeating though the system at various places – hard to maintain
  - Harder to express concerns this way
  - intrusive – you modify your code to invoke these concerns
    - requires understanding at each level
- In this approach
  - You can focus on the concerns at one place
  - Easier to add and remove concerns
  - Easier to modify or fine tune concerns
  - Easier to understand
  - Efficient to implement
  - More efficient

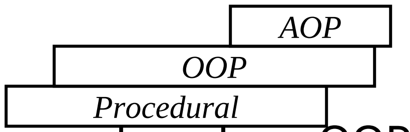
Programming with Aspects

- Limitation of OO
- Separation of Concerns
- **Aspect Oriented programming**
- AspectJ
- PointCut
- Advice
- Pitfalls
- Conclusion

What is an Aspect?

- Aspects are
  - Collection of crosscutting concerns in a system
    - the crosscutting implementations
  - These are generally present among several layers or levels of class hierarchy in a OO system
  - Concerns that are orthogonal to the system

AOP vs. OOP



- AOP does not replace OOP
- It handles separation of concerns better than OOP
- Much like how OOP still uses concepts that are procedural, AOP uses concepts that are OOP
- It extends OOP
- AOP has
  - Functions, Classes and Aspects

Goals of AOP

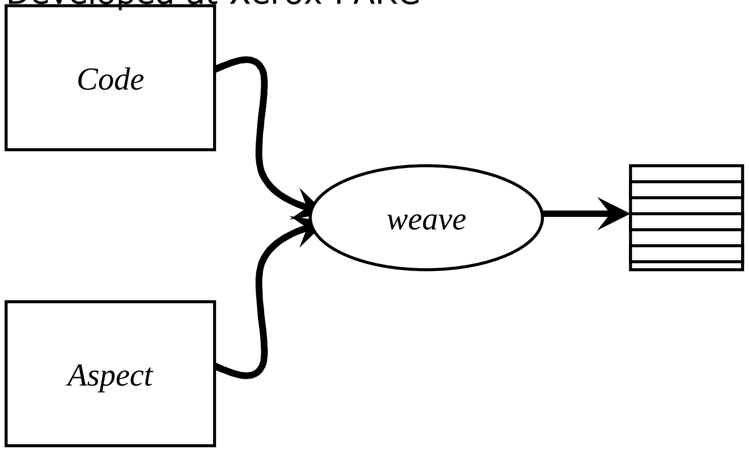
- To
  - separate expression of behavioral concerns from structural ones
  - make design and code more modular
    - not scatter the concerns though out your code
  - isolate the concerns for separate development and
  - be able to plug and unplug these concerns at will

Programming with Aspects

- Limitation of OO
- Separation of Concerns
- Aspect Oriented programming
- **AspectJ**
- PointCut
- Advice
- Pitfalls
- Conclusion

What does AspectJ do?

- General purpose aspect oriented extension to Java
  - Developed at Xerox PARC

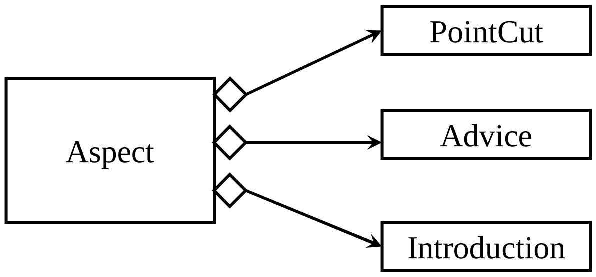


AspectJ Concepts & Constructs

- Join Point
  - well defined points in the execution flow of the code
    - method calls
      - constructor invocation
    - field access
- PointCut
  - selects certain join points and values at those points
- Advice
  - defines code that is executed when a pointcut is reached
- Introduction
  - modifies static structure –classes relationship

Aspect in AspectJ

- Module of crosscutting concerns



```
public aspect MenuEnabling     {
            pointcut CreationOfMenuItem() : call(JMenuItem.new(..));

            after() returning(JMenuItem item) :  CreationOfMenuItem()  {
                        // advice definition code goes here
            }

            after() returning(JMenuItem item) : CreationOfMenuItem() {…}
}
```

Programming with Aspects

- Limitation of OO
- Separation of Concerns
- Aspect Oriented programming
- AspectJ
- **PointCut**
- Advice
- Pitfalls
- Conclusion

Pointcuts

- Defines arbitrary number of points in a program
- However, defines finite number of kinds of points
  - method invocation
  - method execution
  - exception handling
  - object instantiation
  - constructor execution
  - field reference

PointCut Designators

- execution
  - execution(void X.foo()) – when X.foo's body executes
- call
  - call(void X.foo()) – when method X.foo is called
- handler
  - handler(OutOfMemoryException) – execution of the exception handler
- this
  - this(X) – object currently executing is of type X
- within
  - within(X) – executing code belongs to class X
- target
  - target(X) – target object is of type X
- cflow
  - cflow(void X.foo()) - This special pointcut defines all joint points between receiving method calls for the method and returning from those calls, i.e., points in the control flow of the call to X.foo()

PointCut Examples

- name-based crosscutting
  - call (void MyClass.foo(int))
    - any call to foo(int) on any object of MyClass
  - call (void MyClass1.f1(int)) ||
    call (void MyClass2.f2(double))
    - any call to either f1 on object of MyClass1 or f2 on object of MyClass2
  - pointcut pc1() : call (void MyClass.foo(int))
    - named pointcut with name pc1
- property-based crosscutting (not exact name)
  - call (void MyClass.f*(..)) || call (* MyClass2.*(..))
    - void methods of MyClass starting with f or any method of MyClass2

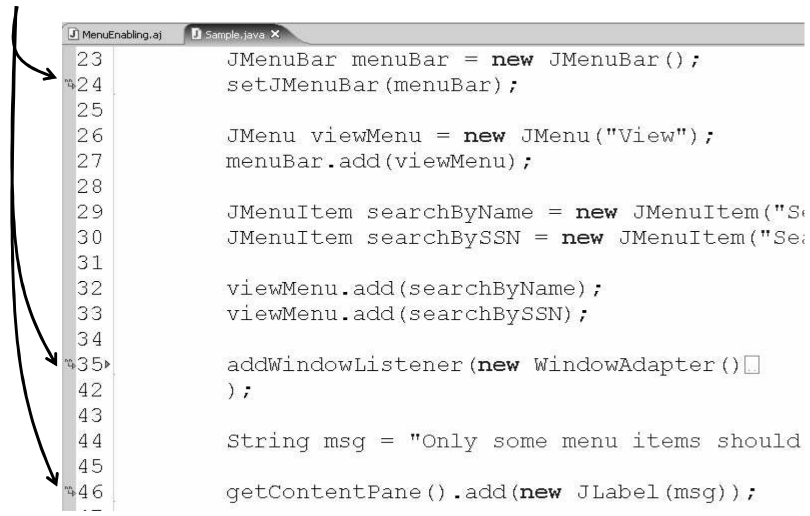
PointCut Examples…

- pointcut pc3(X ref) : target(ref) && call(public * *(..))
  - calls to any methods, on an object of X, with any args
- I want to find which methods of my class are invoked during a certain execution of my program

Quiz Time

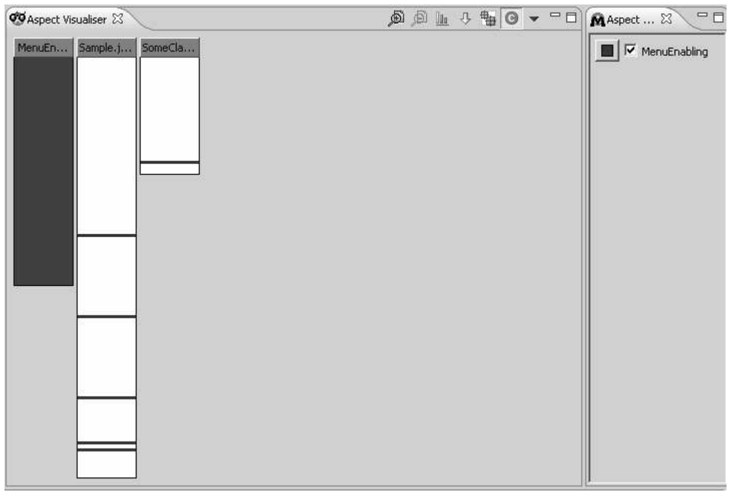
Eclipse Plugin Support

- You can find out crosscutting visually



```
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);

JMenu viewMenu = new JMenu("View");
menuBar.add(viewMenu);

JMenuItem searchByName = new JMenuItem("S
JMenuItem searchBySSN = new JMenuItem("Sea

viewMenu.add(searchByName);
viewMenu.add(searchBySSN);

addWindowListener(new WindowAdapter()
);

String msg = "Only some menu items should

getContentPane().add(new JLabel(msg));
```

Eclipse Plugin Support…



call vs. execution

- In the case of a call, the context is in the caller of the method
- In the case of execution, the context is within the method of interest
- call will not capture super calls to non-static methods of the base, execution will

- Use call if you want an advice to run when the call is made. Use execution if you want an advice to run when ever a code is executed

Pointcut Context

- Execution context at the join point
- advice declarations may use these values

- pointcut pc2(MyClass obj, int a) :

```
     call (void MyClass.foo(int)) && target(obj)
     && args(a);
```

- after(MyClass obj, int a) : pc2(obj, a)

```
{
     System.out.println("method foo called on "
+ obj +
             " with arg " + a);
}
```

Programming with Aspects

- Limitation of OO
- Separation of Concerns
- Aspect Oriented programming
- AspectJ
- PointCut
- **Advice**
- Pitfalls
- Conclusion

Advice

- Defines code that should run at join points
- Types of Advices:
  - Before
    - runs when joint point is reached, but before computation proceeds
  - After
    - runs after computation finishes and before the control returns to the caller
  - Around
    - controls if the computation under joint point is allowed to run
- Example
  - before() : pc1()

```
{
 the code to run
}
```

Advice and call execution

- after() : call(int X.foo(int) {…}
  - executes after the call to X.foo(int), irrespective of successful completion or not
- after() returning(int result) : call(int X.foo(int){…}
  - executes after the successful completion of the call. The returned result may be accessed by advice definition
- after() throwing(Exception e) : call(int X.foo(int))
  - executes only if foo throws exception of type Exception. After the advice runs, the exception is re-thrown.

Bypassing calls

- Using around you may bypass calls to methods
- You may check for conditions and let the call go though or simply refuse to allow the call as well
- int around(X ref, int a) : call(int X.foo(int) && args(a) && target(ref)

```
{
    if (a > 2) return proceed(ref, a);
    return 4;
}
```

Programming with Aspects

- Limitation of OO
- Separation of Concerns
- Aspect Oriented programming
- AspectJ
- PointCut
- Advice
- **Pitfalls**
- Conclusion

Pitfalls

- While concept is very simple, syntax is confusing
- Has some learning curve, especially to implement some complex cross cuttings
- Easy to write a pointcut that puts your code in recursive calls – StackOverflowException
- Different tools for different languages

Quiz Time

Programming with Aspects

- Limitation of OO
- Separation of Concerns
- Aspect Oriented programming
- AspectJ
- PointCut
- Advice
- Pitfalls
- **Conclusion**

Conclusion

- AOP seems to be the next logical step in handling
  - complexity
  - separation of concerns
- Has a lot of promise
- This is a beginning and not the end to the next phase of refinement

References

1. Aspect-Oriented Software Development: http://www.aosd.net

2. Aspect J: http://www.eclipse.org/aspectj/

3. Aspect J Development Tools (Eclipse Plugin): http://www.eclipse.org/ajdt/

4. AOP Focus issue: Communications of the ACM, October 2001- Volume 44, Number 10.

5. Examples, slides are for your download at

http://www.agiledeveloper.com/download.aspx

Thanks!