AJDT: getting started with aspect-oriented programming in Eclipse

Andy Clement
AJDT & AspectJ Committer, IBM UK

Mik Kersten
AJDT & AspectJ Committer, UBC
Agenda

- What is Aspect-Oriented Programming (AOP)?
  - A brief overview of AspectJ

- Demos demos demos…
  - AspectJ Development Tools (AJDT) for Eclipse

- Future of AJDT
good modularity

- socket creation in Tomcat
  - colored lines show relevant lines of code
  - fits nicely into one package (3 classes)
pretty good modularity

- class loading in Tomcat
  - colored lines show relevant lines of code
  - mostly in one package (9 classes)
not so good modularity

- logging in Tomcat
  - scattered across the packages and classes
  - error handling, security, business rules, …
the cost of tangled code

- redundant code
  - same fragment of code in many places
- difficult to reason about
  - non-explicit structure
  - the big picture of the tangling isn’t clear
- difficult to change
  - have to find all the code involved
  - and be sure to change it consistently
the aop idea

- crosscutting is inherent in complex systems
- crosscutting concerns
  - have a clear purpose
  - have a natural structure
- so, let’s capture the structure of crosscutting concerns explicitly...
  - in a modular way
  - with linguistic and tool support
- aspects are
  - well-modularized crosscutting concerns
a simple figure editor

Display

Figure
- makePoint(..)
- makeLine(..)

FigureElement
- moveBy(int, int)

Point
- getX()
- getY()
- setX(int)
- setY(int)
- moveBy(int, int)

Line
- getP1()
- getP2()
- setP1(Point)
- setP2(Point)
- moveBy(int, int)

Factory methods

Operations that move elements
join points

imagine `l.move(2, 2)`

key points in the dynamic call graph

- a Line
  - dispatch
  - a method execution returning or throwing
  - a method call returning or throwing

- a Point
  - dispatch
  - a method execution returning or throwing
join point terminology

- several kinds of join points
  - method & constructor execution
  - method & constructor call
  - field get & set
  - exception handler execution
  - static & dynamic initialization
pointcuts: naming join points

each execution of the

\texttt{<void Line.setP1(Point)>} or
\texttt{<void Line.setP2(Point)>} method

name and parameters

pointcut \texttt{move()}:\n\begin{align*}
\text{execution(} \texttt{void Line.setP1(Point))} & \quad \text{or} \\
\text{execution(} \texttt{void Line.setP2(Point))} & \\
\end{align*}

a “void Line.setP1(Point)” execution

or

a “void Line.setP2(Point)” execution
advice: action under joinpoints

pointcut move():
  execution(void Line.setP1(Point)) ||
  execution(void Line.setP2(Point));

after() returning: move() {
  <code here runs after each move>
}

after advice runs “on the way back out”
a simple aspect

an aspect defines a special class that can crosscut other classes

```java
aspect HistoryUpdating {
  pointcut move():
    execution(void Line.setP1(Point)) ||
    execution(void Line.setP2(Point));

  after() returning: move() {
    <code here runs after each move>
  }
}
```
Eclipse is #1 for AOP

- AspectJ Development Tools (AJDT) for Eclipse
  - Open Source
  - Developed in Hursley
  - Partnership with AspectJ team

- AspectJ
  - Originally a PARC project, now on eclipse.org
AJDT Demo - Figures
Web Services Invocation Framework (WSIF)

- Middleware component
  - Simple Java API for invoking web services, no matter how or where they are provided

- Released to Apache
  - But IBM wants a version tightly coupled to IBM’s normal ‘qualities of service’
    - IBM tracing/monitoring/management

- How do we manage this?
  - Manage an IBM internal of the apache codebase?
  - Put the IBM facilities into the open source codebase?
AJDT Demo - WSIF
Exploring Re-Use: The WSIF Story

org.apache.wsif → WSIF for Open Source Community

+ WebSphere RAS

WebSphere FFDC

WebSphere PMI → Composition → WSIF for WebSphere
Demo conclusions

- Capabilities of AOSD technology look promising
  - Code size reduction
  - No tangling
  - Faster development times
  - Product-line engineering
Eliminating tangling

BEFORE

```java
try {
    if (!removed)
        entityBean.ejbPassivate();
    setState( POOLED );
} catch (RemoteException ex) {
    FFDCEngine.processException(ex, "EBean.passivate()");
    destroy();
    throw ex;
} finally {
    if (!removed && statisticsCollector != null)
        statisticsCollector.recordPassivation();
    removed = false;
    beanPool.put( this );
    if (Logger.isEnabled)
        Logger.exit(tc, "passivate");
}
```

AFTER

```java
try {
    if (!removed)
        entityBean.ejbPassivate();
    setState( POOLED );
} catch (RemoteException ex) {
    destroy();
    throw ex;
} finally {
    removed = false;
    beanPool.put( this );
}
```

Crosscutting concerns extracted

Example: Code to handle EJB Entity bean passivation
Applications of AOP

- Problem determination
  - Logging, FFDC, performance monitoring

- Architectural rule enforcement
  - Contracts, encapsulation, separation (no “up calls”)

- Other concerns
  - Security, transactions, persistence, caching/pooling, locking

- Open source integration
  - Removal of value-add when submitting
Adoption Curve

- enforcement
- testing
- debugging
- performance

- error handling
- management
- timing
- caching

- security
- domain aspects
- persistence
- feature management

- reusable libraries
- aspects and classes for:
  - development
  - infrastructure
  - business logic

- beyond OO
- AOP redefines services, middleware

- development time
- infrastructure
- business logic
- enterprise libraries
- AO architecture

© 2003, 2004 IBM Corporation and New Aspects Of Security
Simple but powerful enforcement aspect

- Warn developers using System.out, System.err and printStackTrace

```java
public aspect EnforceLogging {
    pointcut scope():
        within(com.example..*) &&
        within(ConsoleDebugger);

    pointcut printing():
        get(* System.out) ||
        get(* System.err) ||
        call(* Throwable+.printStackTrace());

    declare warning: scope() && printing():
        "don't print, use the logger";
}
```
AJDT Status - February 2003

- 1.1.4 released (AspectJ v1.1.1 inside)
- Features
  - Auto-configuration
    - Red squiggles
    - AspectJEditor
    - Ignore unused imports
  - Incremental Compilation
  - Improved Structure View
  - Editor templates
  - Integrated help and user guide
  - Better performance and memory usage
Current AJDT engineering challenges

- Commercial Quality
  - Used for large scale projects
- Improve scalability
  - Increase performance, reduce memory usage
- View integration
  - Package Explorer, Type Hierarchy, …
- Improved editor support
  - Code assist, code formatting, organize imports
- Incremental Structure Model
- Eager Outline Updating
Surface more aspect structure

- show inheritance
  - abstract aspects
  - declare parents

- show dynamic info
  - aspect precedence
  - cflow call graphs

- crosscutting
  - navigator
AJDT v2.0.0

- Complete Restructure
- AJDT as an AspectJ project
- Solid Unit Test Foundation
  - Test Driven Development
    (See ‘Contributing to Eclipse’ book…)
- Plugins coordinate via defined extension points and public APIs (open for other contributions)
  - E.g. Visualiser will be extensible
- And more features, for example:
  - Aspect monitor – shows how pointcut matches are varying across compilations
  - Pointcut wizard – helps you easily construct pointcuts
Current AspectJ Status

- v1.1.1 released
- Features
  - Binary weaving
  - Incremental compilation
  - Improved Structure Model
    - new API
    - better performance
  - Resource copying with injars
  - Enabling support for Mac OS X AJDT
  - Many bug fixes and quality enhancements
    - aspect libraries
AspectJ v1.2

- No backwards-incompatible language changes
- Release ahead of Eclipse Tiger support
  - prepare the way for a Tiger focused AspectJ follow-on release
- Important to support users who are applying AspectJ on large-scale projects
  - Performance
  - Robustness
  - Concentrating on defect fixing
- Enterprise Application Support
  - Classloader support
    - for application integration
    - command-line
  - Documentation / Samples
    - for Tomcat, JBoss, WebLogic, WebSphere
Thank You!

- AspectJ 0.1-1.1 was developed at Xerox PARC
- To learn more about AspectJ:
  - “AspectJ in Action” by Ramnivas Laddad

- Useful web links:
  - http://eclipse.org/ajdt (join our mailing list)
  - http://eclipse.org/aspectj
  - http://aosd.net

- Get in touch…
  Andy Clement (clemas@uk.ibm.com)
  Mik Kersten (beatmik@cs.ubc.ca)
Additional material follows…
Other examples …

- Enforcement
  - Architectural Layering
Architectural Layering
Map packages to components

aspect Architecture {

  pointcut inView() : within(view..*);
  pointcut inModel() : within(model..*);
  pointcut inController() : within(controller..*);
  pointcut inPersistence() : within(persistence..*);

...
Pointcuts for ‘external’ calls into components

... 

pointcut viewCall(): call(* view...() ) && !inView();

pointcut modelCall(): call(* model...() ) && !inModel();

pointcut controllerCall(): call(* controller...() ) && !inController();

pointcut persistenceCall(): call(* persistence...() ) && !inPersistence();

pointcut jdbcCall(): call(* javax.sql...() );

...
Compile warnings for illegal calls

...  

declare warning : controllerCall() :  
    "No calls into controller";

declare warning : viewCall() && inController() :  
    "Only controller can call view";

declare warning : modelCall() && !(inController() || inView()) :  
    "Only view and controller can call model";

declare warning : persistenceCall() && !inModel() :  
    "Only model can access persistence layer";

declare warning : jdbcCall() && !inPersistence() :  
    "Persistence layer handles all db access";

}