Enterprise Aspect-Oriented Programming

Ron Bodkin  rbodkin@newaspects.com
New Aspects of Software, AspectMentor

Ramnivas Laddad  ramnivas@aspectivity.com
Aspectivity, AspectMentor

Made available under the EPL v1.0.
This talk is about...

• AOP for the Enterprise
  – AspectJ/AOP Foundational Example
  – Enterprise Java Integration

• AOP Examples
  – Development-Time
  – Infrastructure
  – Domain-Specific

• Conclusion
The Crosscutting Problem

• Auxiliary concerns are scattered and tangled
  – data security
  – audit trail and logging
  – business rules
  – error handling

• 80% of problems come from this 20% of code
  – inflexibility
  – redundancy … inconsistency
  – incomprehensibility … unmaintainability
  – complexity
The AOP Solution

• **Crosscutting is natural**
  – can’t decompose requirements in one-dimension

• **The problem is a lack of support**

• **So AOP uses aspects:**
  – modular support for crosscutting
  – language and tools support (AOP)
  – formalism for design, analysis, … (AOSD)

• **Evolutionary step for software development:**
  – structured → objects → components → aspects
Overview: AOP Concepts

- Aspects
  - the unit of modularity for crosscutting concerns

- Advice
  - block of code executed at specified points in the program execution

- Join points
  - well-defined points in the program flow
Overview: AOP Concepts

- **Pointcuts**
  - join point ‘queries’

- **Inter-type declarations**
  - ‘open classes’ through type extension
  - are also crosscutting
AOP Language Structure

- Aspect
  - contains
  - precedes
  - extends
  - 0..1
- Crosscutting Element
  - contains
  - <<use>>
  - World
    - exposes
- Dynamic Crosscutting
- Static Crosscutting
- Pointcut
- JoinPoint
  - captures
  - contains
- Advice
- Weave-time Declaration
- Inter-type Declaration
AspectJ

• The leading AOP implementation for Java
  – Language extension, @AspectJ annotation, and XML definition options
  – Java platform compatible

• Tool support
  – Compiler, linker, and classloader-based weaving
  – IDE support: Eclipse, JBuilder, JDeveloper, NetBeans, , IDEA (in development)
  – Ant, Maven, ajdoc, Java debugger

• Open source: http://eclipse.org/aspectj
AspectJ Mechanisms

- Relies on bytecode modification of aspect-affected classes
- Weave can happen at compile, post-compile, or load time
AspectJ 5.0 and Java 5.0

- **JDK 5.0 (aka 1.5) includes metadata and generics**
- **Metadata is complementary to AOP**
  - Pointcuts without explicit tags is best
  - But annotations useful for exceptions or inherent properties
  - Supplant marker interface idiom
- **AspectJ 5.0 support**
  - Pointcuts for annotations
    - advice is a great way to handle annotations
  - Declare annotation
  - Pointcuts for generics
  - Generic pointcuts, methods, advice
Example: Online Music Service

- Online music streaming
- Playlists have Songs
- Both Songs and Playlists can be played by a User

Based on “Implementing Observer in .NET” example at MSDN with inspiration from the original AspectJ tutorial
Nick TODO: update for ant integration, classloading, generics & annotations

Ron Bodkin, 1/20/2005
public class Song implements Playable {
    private String name;

    public Song(String name) {this.name = name;}
    public String getName() {return name;}

    public void play() {..}
    public void showLyrics() {..}
}

Music Service Code
public class Playlist implements Playable {
    private String name;
    private List<Playable> songs;

    public Playlist(String name) {this.name = name;}
    public String getName() {return name;}

    public void setSongs(List<Playable> songs) {
        this.songs = songs;
    }

    public void play() {
        for (Song song : songs) {
            song.play();
        }
    }
}
Goal: Correctly Billing User Activity

- When using titles
  - Individual songs
    - Including lyrics
  - play lists
- Should charge user account
- Exact billing may vary on a number of factors
Join Points

key points in dynamic call graph
Join Points: terminology

- **Types**
  - Method and constructor call
  - Method and constructor execution
  - Not shown:
  - Field *get* and *set*
  - Exception *handler*
  - Static and dynamic *initialization*
### Pointcuts: Queries over Join Points

**Pointcut Definition:**

```java
pointcut useTitle() :
    execution(public void Song.play()) ||
    execution(public void Song.showLyrics());
```

- **Name and Parameters:**
  - Execution of `Song.play()` method
  - Execution of `Song.showLyrics()` method

- **Or:**
  - Name and Parameters

**Explanation:**

- This pointcut captures the method execution join points of `play()` and `showLyrics()`.
Advice

- Code that runs before, after, or instead of a join point

```
pointcut useTitle() :
    execution(void Song.play()) ||
    execution(void Song.showLyrics());

after() returning :
    useTitle()
    { //code to run after using a title
    }
```
An aspect is a special type
  – Like a class
  – That crosscuts other types
  – Can contain constructs like pointcuts and advice
Expanding the Aspect

```
public aspect BillingPolicy {

  pointcut useTitle() :
     execution(public void Playable.play()) ||
     execution(public void Song.showLyrics());

  after() returning : useTitle() {
    BillingService.generateCharge();
  }
}
```

- Aspect now applies to Playlist and any other Playables (including Song)
Exposing Context

Billing 3

```java
public aspect BillingPolicy {

  pointcut useTitle(Playable playable) :
    this(playable) &&
    (execution(public voidPlayable.play()) ||
    execution(public void Song.showLyrics()));

  after(Playable playable) returning :
    useTitle(playable) {
      BillingService.generateCharge(playable);
    }
}
```

- This version exposes the currently executing object at each join point (i.e. the Playlist or Song) using `this()`
class PlayList {
    private String name;
    private List<Songs> songs = new ArrayList<Songs> ();
    
    public void play() {
        for (Song song : songs) {
            song.play();
        }
    }
}

class Song {
    private String name;
    
    public void play() {
        // play song
        BillingService.generateCharge();
    }
    
    public void showLyrics() {
        // show lyrics
        BillingService.generateCharge();
    }
}
Java Implementation

class PlayList {
    private String name;
    private List<Songs> songs =
        new ArrayList<Songs>();

    public void play() {
        for (Song song : songs) {
            song.play();
        }
        BillingService.generateCharge();
    }
}

class Song {
    private String name;

    public void play() {
        // play song
        BillingService.generateCharge();
    }

    public void showLyrics() {
        // show lyrics
        BillingService.generateCharge();
    }
}

 Billing 2
Java Implementation

Billing 3

- Billing code scattered through domain objects
- No module captures intent and implementation of billing policy
- Evolution of billing behavior cumbersome
  - Each caller must be changed
  - Easy to introduce bugs
AspectJ Implementation

```java
class PlayList {
    private String name;
    private List<Songs> songs =
        new ArrayList<Songs>();

    public void play() {
        for (Song song : songs) {
            song.play();
        }
    }
}

class Song{
    private String name;

    public void play() {
        // play song
    }

    public void showLyrics(){
        // show lyrics
    }
}
```
```java
class PlayList {
    private String name;
    private List<Songs> songs = new ArrayList<Songs>();

    public void play() {
        for (Song song : songs) {
            song.play();
        }
    }
}

class Song{
    private String name;

    public void play() {
        // play song
    }

    public void showLyrics() {
        // show lyrics
    }
}

aspect BillingPolicy {
    pointcut useTitle() :
        execution(public void Song.play()) ||
        execution(public void Song.showLyrics());

    after() returning : useTitle() {
        BillingService.generateCharge();
    }
}
```
class PlayList {
    private String name;
    private List<Songs> songs = new ArrayList<Songs>;

    public void play() {
        for (Song song : songs) {
            song.play();
        }
    }
}

class Song {
    private String name;

    public void play() {
        // play song
    }

    public void showLyrics() {
        // show lyrics
    }
}

aspect BillingPolicy {
    pointcut useTitle():
        execution(public void Playable.play()) ||
        execution(public void Song.showLyrics());

    after() returning : useTitle() {
        BillingService.generateCharge();
    }
}
AspectJ Implementation

 Billing 3

class PlayList {
    private String name;
    private List<Songs> songs =
        new ArrayList<Songs>();

    public void play() {
        for (Song song : songs) {
            song.play();
        }
    }
}

class Song{
    private String name;

    public void play() {
        // play song
    }

    public void showLyrics(){
        // show lyrics
    }
}

aspect BillingPolicy {
    pointcut useTitle(Playable playable) :
        this(playable) &&
        execution(public void Playable.play()) ||
        execution(public void Song.showLyrics());

    after(Playable playable) returning : useTitle(playable) {
        BillingService.generateCharge(playable);
    }
}

• Billing code centralized in BillingPolicy
  – Intent of billing behavior is clear
  – Changes to policy only affect aspect
  – Modular evolution
Aspects Crosscut Classes

- Aspect modularity cuts across class modularity

```
BillingService

<<interface>>
Playable
+ play()

BillingPolicy

Playlist
+ play()

Song
+ play()
+ showLyrics()
```
Double Billing

• Don’t want to bill twice for songs played within the context of playing a Playlist
• Can accomplish using cflowbelow pointcuts
Refresher : Control Flow

All these join points are in the control flow of the original play call
Using the cflowbelow() Pointcut

- `cflowbelow()` selects only join points within the control flow of another join point.
One Billing Service Per Playable

- Allows different songs to have different price models

```java
public aspect SingleBillingPolicy {
    private BillService Playable.billService;
    public void Playable.setBillService(BillService s) {
        billService = s;
    }

    pointcut useTitle(Playable playable) :
        <as before>

    pointcut topLevelUse(Playable playable) :
        <as before>

    after(Playable playable) returning :
        topLevelUse(playable) {
            playable.billService.generateChargeFor(playable);
        }
}
```

Private ... to the enclosing aspect

Public ITD method
Several Services Per Playable

```java
public aspect GeneralBillingPolicy {

  private List Playable.billServices = new ArrayList();
  public void Playable.addBillService(BillService s) {
    billServices.add(s);
  }

  <as before..>

  after (Playable playable) returning :
  topLevelUse(playable) {
    for (BillService service : playable.billServices) {
      service.generateChargeFor(playable);
    }
  }
}
```

- One service for each copyright holder perhaps?
Client Code

```java
...

Song song = new Song("Yellow Submarine");
BillService john = Publisher.find("ABC").getBillService();
BillService ringo = Publisher.find("XYZ").getBillService();

song.addBillingService(song, john);
song.addBillingService(song, ringo);

song.play();
```
A Reusable Aspect

```java
public abstract aspect ObserverProtocol {
    protected interface Subject {}
    protected interface Observer {}

    private WeakHashMap<Subject, List<Observer>> perSubjectObservers;
    protected List<Observer> getObservers(Subject subject) {<retrieve from map>}
    public void addObserver(Subject subject, Observer observer) { ... }
    public void removeObserver(Subject subject, Observer observer) { ... }

    protected abstract pointcut subjectChange(Subject s);

    after(Subject subject): subjectChange(subject) {
        for(Observer o : getObservers(subject)) {
            updateObserver(subject, o);
        }
    }

    protected abstract void updateObserver(Subject subject, Observer observer);
}
```

Define Roles within the pattern
Aspect Inheritance

• Aspects can be declared abstract and extended by subaspects
• Abstract aspects can define abstract methods
• Abstract aspects can define abstract pointcuts and apply concrete advice to them
  – It’s up to the subaspect to concretize the pointcut
Concrete Reuse

```java
public aspect ObserverBillingPolicy extends ObserverProtocol {

    declare parents : Playable extends Subject;
    declare parents : BillingService implements Observer;

    pointcut topLevelUse(Playable playable) :
        <as before>

    public pointcut subjectChange(Subject subject):
        topLevelUse(Playable) && this(subject);

    public void updateObserver(Subject s, Observer o){
        BillingService service = (BillingService)o;
        service.generateChargeFor((Playable)s);
    }
}
```

- Apply roles to participants
- Override pointcut to define where a change has occurred (a top-level use)
Wiring Up Observers

- Last piece of puzzle
- Billing Service needs to begin “watching” the various Playables
  - And to perhaps stop at some point (old songs become free?)
- Accomplished through calls to add/removeObserver on the aspect
  - Can be done programatically
  - Or through advice
  - Examples during demo in next section
Part II. Enterprise Integration

- Build Integration
- Debugging support
- Library generation
- Load Time Weaving
- View Layer Integration
Compiling With Ant

• **Minute-to-minute development done in IDE**
  – Exploit integrated incremental compilation for rapid development

• **Deployment builds can integrate with Ant and Maven**
  – Use iajc task to invoke ajc compiler
  – Various options to control bytecode weaving, jar generation, source and bytecode aspects, etc.
Basic War Generation With Ant

- iajc similar to javac task
  - Source options more complex
  - Need aspectjrt.jar on the classpath
Hot Deployment Strategy

- Many J2EE containers support exploded and/or hot redeployment
- Leverage incremental compiler to avoid time-consuming full builds
- Copy IDE output to deployment directory

```xml
<target name="dirty-deploy">
  <copy todir="${deploy.dir}/songs/">
    <fileset dir="webapp" includes="**/*.jsp"></fileset>
  </copy>
  <copy todir="${deploy.dir}/songs/WEB-INF/classes">
    <fileset dir="bin" />
  </copy>
</target>
```
Sample Aspect: Error Logging

```java
// problem: poor error handling
public void problematicMethod() {
    try {
        int i = 1/0;
    } catch (Exception e) {
        throw new RuntimeException("masks the real problem");
    }
}

// Solution: log all too-wide handler blocks
public aspect ErrorLogger {
    public pointcut suspiciousHandlers(Throwable t) :
        (handler(Exception) || handler(Throwable))
        && args(t) && scope();
    before(Throwable t) : suspiciousHandlers(t) {
        // ...log error
    }
}
```
Aspect Library Jars

- AspectJ can create binary libraries for reuse: stored in standard jar file format
- Typically use Ant to generate output jars
- Can also use AJDT, Maven, …
- Consume output jars with Ant, AJDT, …
Consuming bytecode aspects

\[\text{\texttt{<!-- compile aspects into JAR -->}}\]
\[\text{\texttt{<target name=\"compileJar\">}}\]
\[\text{\texttt{<iajc source="1.5" outjar="antError.jar" sourceroots="src">}}\]
\[\text{\texttt{<classpath> // i n c l u d e a s p e c t j r t . j a r}}\]
\[\text{\texttt{</iajc>}}\]
\[\text{\texttt{<copy todir="../ObserverDemo/webapp/WEB-INF/lib" file="antError.jar"/>}}\]
\[\text{\texttt{</target>}}\]

\[\text{\texttt{<!-- modify previous target to put jar on aspectpath -->}}\]
\[\text{\texttt{<iajc source="1.5" destDir="target/classes" verbose="true"}}\]
\[\text{\texttt{sourceroots="src" aspectpath="${webLib}/antError.jar">}}\]

\[\text{\texttt{<!-- Or modify already compiled JARs-useful for logging errors in third-party code -->}}\]
\[\text{\texttt{<iajc outjar="lib/struts.jar" injars="outlib/struts.jar"}}\]
\[\text{\texttt{aspectpath="${webLib}/antError.jar">}}\]

• To weave a classfile more than once, use -Xreweavable
Load Time Integration

- AspectJ 5 has robust support for weaving classes at load-time
- Allows easy deployment of auxiliary aspects
- Configuration files customize which aspects are woven
- LTW enabled through -javaagent in JDK 5.0, custom classloader in JDK 1.4
  - "aj" launch script simplifies
<aspectj>
    <aspects>
        <!-- declare existing aspects to the weaver -->
        <aspect name="errorlogging.ErrorLogger"/>
        <!-- Can include aspects based on name patterns and annotations. -->
        <include within="errorLogging..*"/>
        <exclude within="@AdvancedLogging *"/>
    </aspects>
    <weaver options="-verbose">
        <!-- can include or exclude affected classes on similar criteria -->
        <include within="org.apache.struts..*"/>
        <include within="(!@NoWeave songplayobserver.*) AND songplayobserver.*"/>
    </weaver>
</aspectj>

- File placed in META-INF on classloader search path
Concrete Aspects in XML

- Allows for customization of infrastructure pointcuts without full-subclassing

```xml
<aspectj>
  <aspects>
    <aspect name="errorlogging.ErrorLogger"/>
    <!-- Supposing that ErrorLogger was abstract and that loggedErrors was an abstract pointcut -->
    <concrete-aspect name="com.foo.SongAppLogger"
      extends="errorlogging.ErrorLogger">
      <pointcut name="scope"
        expression="within(songplayobserver.Application)"/>
    </concrete-aspect>
  </aspects>
  ...
</aspectj>
```
View Layer Integration

• Challenge of the View Layer
  – “Code” in view layer not written in Java
    • Templating languages (Velocity, JSP)
    • Component specs (JSF, Tapestry)
    • Often uses reflection to call methods on application code
  – Pointcuts cannot detect join points outside the "code the implementation controls"
View Layer Example

- How to write advice or start a cflow on the call to `playCurrentSong()`?
- Or on call to `setCurrentSongName()`?

```jsp
<jsp:useBean id="songApp" scope="session" class="songplayobserver.Application" />
<c:set target="${songApp}" property="currentSongName" value="${param.song}" />

<% songApp.playCurrentSong(); %>
```
View Layer Options

• **Weave into JSPs**
  – Tomcat… allows pluggable JSP compilation
  – LTW, but are there classloader issues?
  – Both solutions depend on JSP->Java translation
  – Neither addresses \texttt{c:set}!

• **Refactor or use existing design to rely on join points you do control**
  – Use
    \begin{verbatim}
    execution(Application.setCurrentSongName(...))
    \end{verbatim}
  – More widely useful (tag libraries, alternative view layers)
Part III: Development Time Aspects

- Tracing remote calls
- Architectural enforcement
- Testing
public abstract aspect TraceRemoteRequests {
    pointcut remoteCall(Object caller, Remote recipient) :
        call(public * Remote+.*(..) throws RemoteException) &&
        this(caller) && target(recipient);
    
    before(Object caller, Remote recipient) :
        remoteCall(caller, recipient) && inScope() {
            logger.info("Calling "+thisJoinPoint+" from "+
                toTraceString(caller)+" to "+toTraceString(recipient));
        }
    
    after() returning (Object result) :
        remoteCall(*, *) && inScope() {
            logger.info("Returned "+toTraceString(result));
        }
    
    after() throwing (Throwable throwable) :
        remoteCall(*, *) && inScope() {
            logger.info("Threw "+toTraceString(throwable));
        }
    ...
}
Tracing Remote Service Calls

```java
...

public abstract pointcut scope();

private pointcut inScope():
    scope() &&
    !within(TraceRemoteRequests) && !withincode(* toString());

private String toTraceString(Object obj) {
    ...
}

private Logger logger = Logger.getLogger(TraceRemoteRequests.class);
```
Architectural Enforcement

• Compile time checking
  - declare error, declare warning

• Only for statically determinable pointcut designators
  - execution, initialization, static initialization, within, within code, get, set, call

• Dynamic cases addressed later (in testing)

• Prevent code rot and enforce consistent policy
  - E.g., enforcing EJB restrictions

• Often a first step; even better is using aspects to consistently implement …
Use Logger, not printing

• 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(* printStackTrace());

    declare warning: scope() && printing():
        "don't print, use the logger";
}
```
Architectural Layering

Layer 1

Layer 2

Layer 3

Layer 4

Persistence

Model

View

Controller
Enforcing Layering: Mapping packages to layers...

aspect Architecture {

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

    ...
}
Enforcing layering: PCD’s for external calls into layers

...  

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

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

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

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

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

...
Enforcing layering: Compiler warnings for illegal calls

```java
...

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";

}
Subsystem Testing: Problems

- Complex state
- Complex dependencies
- Error handling
- Sequence of actions
- Refactor for testability!?
Subsystem Testing: Solutions

- Use mock objects to simulate some components
- Using aspects
  - Virtual Mocks
  - Fault injection
  - Count automatically

1. Leverage JUnit
2. Don’t compromise design for testability
3. Clean mock processing
Testing Database Errors

```java
public void testDatabaseFailure() {
    try {
        from.transfer(2.25, to);
        fail("no exception");
    } catch (ModelException e) {
        // success
    }
}
```

• Account should convert a SQLException to a ModelException
  – How to simulate a SQLException?
public aspect AccountTestFailureInjection {
    pointcut inFailureTest():
    cflow(execution(* Failure(..)) && this(TestAccount));

    before() throws SQLException:
    call(* save()) && target(DAO) && inFailureTest() {
        throw new SQLException("can't reach database");
    }
}
public void testTransfer() {
    from.transfer(2.25, to);

    assertEquals(from.getBalance(), 7.75, TOLERANCE);
    assertEquals(to.getBalance(), 3.25, TOLERANCE);
    // verify values persisted
    assertEquals(1, from.getExecutions("save"));
    assertEquals(1, to.getExecutions("save"));
    assertEquals(1, transaction.getCallsTo("commit"));

    // verify commit happened AFTER saves
    int saveFromSeq = from.getMethodSeq("save");
    int saveToSeq = to.getMethodSeq("save");
    int commitSeq = transaction.getMethodSeq("commit");
    assertTrue(saveFromSeq < commitSeq);
    assertTrue(saveToSeq < commitSeq);
}
...
Extending Library Aspect

```java
aspect AccountVirtualMocks extends VirtualMocks {
  declare parents: (AccountDAO || Provider || Transaction)
    implements Replaced;

  declare parents: Account
    implements MonitoredItem

  public pointcut replacementScope():
    cflow(execution(public void TestAccount.test*(..)));
}
```

- **Makes collaborators into virtual mock objects**
  - Counts methods and parameters
  - Mock return values (even exception injection)
  - Method counting and mock values
- **Also tracks operations on the tested class (Account)**
Part IV: Infrastructure Aspects

- Performance
- Error Handling
- Transaction Management
Performance Management

• Performance is also a crosscutting concern
• AOP can help to
  – Monitor performance
  – Identify bottlenecks (when optimizing/tuning)
  – Improve performance
• AOP complements:
  – Good system architecture
  – Good (improved) algorithms
  – Profilers
Monitor Performance

- **Goal:** capturing key system statistics like
  - How many requests?
  - How long do they take?
  - How much time is spent in database calls?
  - Which requests are slow?

- **Can be used for**
  - Real-time alerts
  - Management consoles
  - Offline analysis
  - Low-overhead tuning
Basic Request Monitoring

```java
public aspect MonitorRequests {
    void around() : monitoredRequest() {
        PerfStats stats = getPerfStats(thisJoinPointStaticPart);
        long start = System.currentTimeMillis();
        proceed();
        stats.counter++;
        stats.time += System.currentTimeMillis() - start;
    }

    pointcut monitoredRequest() :
        execution(void HttpServlet.do*(..)) && if(enabled);

    // can expose stats via JMX, dump method, getStats etc.
    public static class PerfStats {
        ...
        private static final String[]...}
    private Map<StaticPart, PerfStats> perfStatMap = //...
    private boolean enabled;
}
```
Refresher: percflow() aspects

- Instantiated at each join point in specified pointcut
- Keep state for length of control flow
- To jump back to our first example, the next slide shows the lifecycle of an aspect declared as

```java
percflow(execution(public void Playlist.play()))
```
Refresher: percflow() aspects

Aspect instantiated

Aspect eligible for garbage collection

The aspect may run at any of these join points

playlist

song

song
Database Monitoring Per Request

public aspect MonitorDatabaseRequests
  per cflow (monitoredRequest() && !cflowbelow(monitoredRequest())) {
    void around() : monitoredRequest() {
      PerfStats stats = getPerfStats(thisJoinPointStaticPart);
      long time = System.currentTimeMillis();

      proceed();

      stats.counter++;
      stats.databaseTime += accumulatedDatabaseTime;
      stats.time += System.currentTimeMillis()-time;
    }

    ...}

- Captures statistics at top level request entry & exit
- !cflowbelow() prevents the aspect from double counting requests invoked by this request
Database Monitoring Per Request

```java
Object around() : databaseOperation() {
    long time = System.currentTimeMillis();
    Object ret = proceed();
    accumulatedDatabaseTime += System.currentTimeMillis()-time;
    return ret;
}

public pointcut databaseOperation() : call(* java*.sql..*()..);

private long accumulatedDatabaseTime = 0L;
```

- Accumulates database time for each request
Identify Bottlenecks

Common stages in optimizing:
• Find & eliminate top-level bottlenecks
• Find & tune expensive “leaf” operations
• Now your performance profile will be many moderately expensive calls… with no “low hanging fruit”
• *How do you proceed?*
AOP for Active Analysis

- Collect selective statistics (\(\text{sum}(x+y+z)\) when \(a\) or \(b\) is true)
- Test hypotheses ... non-invasively
- Like our example for monitoring database, but tailored
  - e.g., track string or XML operations only for
    - \(\text{cflow(execution(..) \&\& this(UpdateServlet))}\)
    or
    - \(\text{args(list)} \&\& \text{if(list.length > 10)}\);
AOP to *Improve* Performance

- **Caching**
  - Storing previously calculated values

- **Prefetching**
  - Retrieving information before it’s required

- **Changing Concurrency**
  - Altering locking or conflict detection for parallel use

- **Changing query strategy**
  - Using indexes or simpler queries in a common case

- **Batching distributed requests**
  - Sending many updates in one message
A Constant Cache

```java
public aspect CacheTaxCalculation {
    private Object Account.cachedTaxRate;

    Object around(Account account) :
        execution(double calcTaxRate(..)) && this(account) {
            Object value;
            if (account.cachedTaxRate == null) {
                value = proceed(account);
                account.cachedTaxRate = value;
            } else {
                value = account.cachedTaxRate;
            }
            return value;
        }
}
```

- Using Object allows for null as a cached value
- AspectJ auto-boxes/unboxes
An Event-Based Invalidating Cache

```java
public aspect CacheTaxCalculation {
    private Object Account.cachedTaxRate;

    Object around(Account account) :
        // same as before...

    before(Account account) : set(* Account+.*) &&
        target(account) && !within(CacheTaxCalculation) {
            account.cachedTaxRate = null;
        }
}
```

- Invalidate the cache if Account changes state
  - i.e. if a field within Account changes
- Could can extend to use observer (e.g., if TaxRegulation changes)
A Time-Based Invalidating Cache

```java
public aspect CacheTaxCalculation {
    private Object Account.cachedTaxRate;
    private long Account.cacheWriteTime;
    private static int INVALIDATION_TIME = 5000; // 5s

    Object around(Account account) :
        execution(double calcTaxRate(..)) && this(account) {
            Object value;
            if (!account.hasValidCache()) {
                account.cachedTaxRate = proceed(account);
                account.cacheWriteTime = System.currentTimeMillis();
            }
            return account.cachedTaxRate;
        }

    ...
```
Checking Cache Expiry

```java
... public boolean Account.hasValidCache() {
    return account.cachedTaxRate != null &&
    account.cacheWriteTime + INVALIDATION_TIME >
    System.currentTimeMillis();
}
}```
A Reusable Caching Aspect

- Example inspired by Adrian Colyer’s blog

- Two parts:
  - abstract aspect
  - concrete reuse
public abstract aspect BaseCache {

    private Map cache = new WeakHashMap();
    public void invalidate(Object key) { ... }

    abstract pointcut cachedOperation(Object key);

    Object around(Object key) : cachedOperation(key) {
        Object value;
        if (cache.containsKey(key)) {
            value = cache.get(key);
        } else {
            value = proceed(key);
            cache.put(key, value);
        }
        return value;
    }
}
Concrete Reuse

```java
public aspect CacheTaxCalculation extends BaseCache {

  pointcut cachedOperation(Object key) :
    execution(double calcTaxRate(..)) &&
    this(Account) && this(key);

  // invalidate as before...
}
```

- Abstract aspect could be further extended to add an “invalidation” pointcut
Monitoring Example: RUBiS

Currently available categories

- Antiques & Art
- Books
- Business, Office & Industrial
- Clothing & Accessories
- Coins
- Collectibles
- Computers
- Consumer Electronics
- Dolls & Bears
- Home & Garden
- Jewelry, Gems & Watches
- Movies & Television
- Music
- Photo
- Pottery & Glass
- Sports
- Stamps
- Tickets & Travel
- Toys & Hobbies
- Everything Else

RUBiS (C) Rice University/INRIA
Page generated by BrowseCategories in 195 milliseconds
AOP Performance Studies

• **Micro-benchmarks**
  – AOP Benchmark from AspectWerkz
  – abc compiler benchmarks
  – AspectJ internal benchmark suite
  – Haupt & Mezini: micro-measurements

• **System Benchmarks**
  – Hugunin & Hilsdale: XSLTMark tracing
    • 22% slower when tuned for basic policy
    • 24% faster with static field optimization
  – Zhang & Jacobsen: ORBacus refactoring
    • 8% slower with all aspects included
    • 6% faster without unnecessary aspects
Performance Results Summary

• AspectJ (& AspectWerkz) performance usually close (within 5%) of hand-written alternatives
  – Typical advice overhead 10-80 ns/invocation
  – If it’s important, test and tune

• Small caveats
  – Pre-AspectJ 1.5: no pertypewithin
  – Pre-AspectJ 1.2: thisJoinPoint eager creation issue
  – Pre-AspectJ 1.2: cflow overhead
  – Use of inefficient methods (e.g., Class.getName())
  – Generally performance issues can be resolved (and AOP also enables better optimization)
  – Build times significantly slower
Error Handling Challenges

• Confusing/wrong exception handling
• Swallowing exceptions
• Repetitious logic
• Inconsistent exception types
• Added as an afterthought
• Hard to change

• Failing to log errors
• Logging errors too many times
• Presenting error information to users
• Determining root causes
• Capturing context
• User support (what really went wrong?)
Sample Error Handling Policy

• **Handle exceptions**
  – correctly and consistently
  – at well-defined points

• **Log exceptions**
  – once, when first resolved
  – with appropriate context information

• **Convert exception types**
  – soften (most) checked exceptions
  – at defined layers (tiers)
  – distinguish fatal from possible to retry?
Data Access Error Handling

```java
aspect DataAccessErrorHandling {
    pointcut dataAccessCall() :
        within(persistence..*) &&
        call(* *(..) throws SQLException);

    declare soft: SQLException: dataAccessCall();

    after(DAO obj) throwing (SQLException e) :
        dataAccessCall() && this(obj) {
            RuntimeException pe = null;
            if (e.getErrorCode() == ..) {
                pe = new NoRemoveInUseException(e);
            } else {
                pe = new PersistenceException(e);
            }
            pe.setCurrent(obj);
            throw pe;
        }
    ...
}
```
Model Error Handling

```java
aspect ModelErrorHandling {
    pointcut layerEntry() : execution(public * model..*(..));

    pointcut layerBoundary() :
        layerEntry() && !cflowbelow(layerEntry());

    after() throwing (PersistenceException e):
        layerBoundary() {
            logger.error("database error for "+e.getContext(), e);
            // code to force transaction rollback
            throw new ModelException(e); // does exception conversion
        }

    after(Object cause) throwing (PersistenceException e):
        execution(* model..*(..)) && this(cause) {
            // capture context from calling code
            if (e.getCaller() == null) {
                e.setCaller(cause);
                e.setArgs(thisJoinPoint.getArgs());
                logger.debug("caller :");
                // iterate over args and log
            }
        }
}
```
aspect UI Error Handling {

    pointcut actionMethod(ActionMapping mapping) :
        execution(ActionForward Action+.execute(..)) &&
        args(mapping, ..);

    ActionForward around(ActionMapping mapping) :
        actionMethod(mapping) {
            try {
                return proceed(mapping);
            } catch (NoRemoveInUseException e) {
                errors.add(ActionErrors.GLOBAL_ERROR,
                        new ActionError("error.noremoveinuse",
                            ((BusinessObj)e.getContext()).getName(),
                            ((BusinessObj)e.getContext()).getKey()));
                return mapping.getInputForward();
            } catch (ModelException e) {
                return handleError(e, mapping);
            }
        }

    …
}
UI Error Handling (cont.)

```java
...
    catch (InvocationTargetException e) {
        logger.error("populating form ", e);
        return handle(e, mapping);
    }
    catch (Throwable t) {
        logger.error("unknown throwable ", t);
        return handle(t, mapping);
    }
}

ActionForward handle(Exception e, ActionMapping mapping) {
    session.setAttribute( Constants.MESSAGE_TEXT,
        translate(e.getMessage()));
    session.setAttribute( Constants.EXCEPTION, e);
    return mapping.findForward( Constants.ERROR_PAGE);
}
```
Simple Transaction Management

```java
public aspect TxnManagement {
    protected pointcut txnalPoint() :
        execution(public * *(..)) && this(org.sample.model..*);
    // could also use annotation to define pointcut...
    private pointcut topLevelTxn():
        txnalPoint() && !cflowbelow(txnalPoint());

    Object around() : topLevel Txn() {
        Object returnValue = null;
        try {
            returnValue = proceed();
            txn.commit();
        } catch (ModelException ex) {
            txn.rollback();
            throw ex;
        }
        return returnValue;
    }
}
```
Part V: Business Logic Aspects

- Account Suspension
- Other Business Aspects
Account Suspension

• Example business aspect

• Rule: no business transaction may be performed on any suspended account
  – Many different types of accounts
  – Attempts should fail and give an error
  – The UI should not offer unavailable operations… just as with security rules
Telecom Example

Account
+ bill()
+ cancel()
...

Service
+ placeCall()
+ addFeature()

ConsumerAcct
+ doNotCall()
...

SME_Acct
+ analyze()
...

Mobile
+ setVoicemail()
...

Local
+ conference()
Simple Implementation

```java
public aspect AccountSuspension {
    private boolean Account.suspended = false;
    public void Account.setSuspended() { ... }
    public boolean Account.isSuspended() { ... }

    pointcut suspendedAccountBusinessTxn(Account account):
        this(account) && if(account.suspended) &&
        execution(* *(..)) && BPM.inBusinessTransaction();

    before(Account account) :
        suspendedAccountBusinessTxn(account) {
            throw new SuspendedAccountException("...");
        }
    // exceptions caught by UI/interaction tier
}
```
public aspect ServiceSuspension {
    pointcut suspendedServiceBusinessTxn(Service service):
        if (service.getAccount().isSuspended) &&
          this(service) && execution(* *(..)) &&
          BPM.inBusinessTransaction();

    before(Service service) :
        suspendedServiceBusinessTxn(service) {
          // record attempted access to service.getAccount()
          throw new SuspendedAccountException("...");
        }
}
Interesting Questions

- Defining business transactional operations: use of annotations or stable properties?
- Restore (or close) suspended account: is this a business transaction? Pointcut exception?
- Allowing non-transactional operations on suspended: not common in practice?
- Proactive checking for suspension in UI
  - Challenge: avoiding redundant rules while decoupling
  - Some UI designs/frameworks can simplify
  - Use of annotations may facilitate (possibly with declare annotation)
  - Topic for research in expressive pointcuts…
Business Aspects

• Many other possibilities
  – Account limits
  – Account authorization
  – Metering (extending our online music store)
  – Publishing business events
  – Detecting business exceptions (real-time monitoring)
  – Feature management
    • for specific partners/customers etc.
  – Object history/versioning

• As many business aspects as business objects
Part VI: Conclusion …

• The state of AOP
• Standards and Libraries
• Design With Aspects
• Adoption
The State of Play

What is the state of enterprise AOP?

• Lots of players:
  – AspectJ 5
    • Power & tools integration of AspectJ
    • Load-time weaving & easy adoption of AspectWerkz
    • IBM & BEA support
    • also abc: optimizing AspectJ compiler
  – JBossAOP: “Transparent middleware”. Aspects bundled for you (TX, caching, etc)
  – Spring: Out of the box aspects (TX, persistence, security, etc), AspectJ integration
  – Other projects: AspectC++, AOPHP, Dynaop, Nanning, etc.

Cricket: a sport that lasts for 5 days, and you stop for tea!

Slide concept courtesy Dion Almaer
Standards

• **What, when, and how to standardize: tough questions**
  – AOP Alliance: Standard interfaces for interceptors
  – Groovy precedent for alternative languages on JVM
  – Sun is *conservative* (e.g. generics was JSR-14)

• **The What**
  – Can we agree on a pointcut language?
    • Implementations are converging… and sharing the same notions
  – Can we standardize on aspect .class file formats?
    • Would allow sharing pointcut definitions
    • Could even share aspect libraries among AO implementations
  – Standard extensions?

*Slide concept courtesy Dion Almaer*
Reusable Aspect Libraries

• Aspen (from aTrack sample AspectJ project)
  – Persistence & Transactions
  – Session management
  – Controller logic
• AspectWerkz libraries
• Spring AOP libraries
• JBoss AOP libraries
• AspectJ 5 library base
• Eclipse AspectJ sample app
• Contract4J
• Can we combine?
Early Aspects

• Analysis
  – Adjectives & adverbs not nouns (Colyer)

• Build on OO methods that work
  – Refactoring aspects is natural (Laddad, Monteiro, …)
  – Good OO design helps AO
  – TDD (aUnit tool in progress)

• UML extensions
  – Naïve/Stereotypes, Theme/UML, Jacobson, …
Design Aspects

- Focus on ARC: aspect, responsibilities, collaborators
- Pointcuts are key: describe stable properties (e.g., whenever state changes), not using today’s AOP pointcut languages nor using enumeration
- Advice: @name to describe what not how
- ITD’s: design like state/methods for class
Smooth Adoption Curve

- benefit
  - 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
  - AO architecture

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

Made available under the EPL v1.0.
March 15, 2005
Conclusion

• AOP is rapidly gaining adoption in the enterprise
• An incremental adoption strategy is available
• These slides at http://www.newaspects.com/presentations
• Training, consulting, and support available
Thank You

Ron Bodkin  rbodkin@newaspects.com
New Aspects of Software, AspectMentor

Ramnivas Laddad  ramnivas@aspectivity.com
Aspectivity, AspectMentor

Thanks to our fellow AspectMentors for concepts in and feedback on this presentation, and to Ron’s former colleagues on the AspectJ team at PARC for teaching him some effective ways to teach AOP.