Aspect-Oriented Programming for Web Services

Ron Bodkin
rbodkin@newaspects.com
New Aspects of Software
This talk is about...

- Aspect-Oriented Programming
  - Concepts
  - Tools and integration
- Applications of AOP
  - 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
  - DRY ... inconsistency
  - incomprehensibility
  - complexity
The AOP Solution

- Crosscutting is natural
  - can’t decompose requirements in one-dimension
- The problem is a lack of support
- Aspects provide *modular support* for crosscutting
- Evolutionary step for software development:
  - structured → objects → aspects
AspectJ Demo
Example: Online Music Service

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

Inspired by the “Implementing Observer in .NET” example at MSDN and Figures from the original AspectJ tutorial
New Requirement: Metering User Activity

- When using titles
  - Individual songs... including lyrics
  - Playing play lists
- Should track usage to allow charging user account
- Exact billing may vary on a number of factors
Join Points

- Client
  - play
  - Playlist.play()
    - execution
    - Song.play()
      - call
      - return
  - return

- Playlist
  - Playlist.play()
  - return

- Song
  - play
  - return
  - Song.play()
    - execution
Pointcuts: Queries over Join Points

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

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

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

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

after() returning : useTitle()
{
    //code to run after using a title
}
```
An Aspect

```java
public aspect MeteringPolicy {

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

    after() returning : useTitle() {
        MeteringService.trackUse();
    }
}
```

- 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 MeteringPolicy {

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

    after() returning : useTitle(){
        MeteringService.trackUse();
    }
}

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

public aspect MeteringPolicy {

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

    after(Playable playable) returning :
        useTitle(playable){
            MeteringService.trackUse(playable);
        }
}

- This version exposes the currently executing object at each join point (*i.e.* the Playlist or Song) using this()
Java Implementation

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

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

class Song{
    private String name;

    public void play() {
        // play song
        MeteringService.trackUse();
    }

    public void showLyrics(){
        // show lyrics
        MeteringService.trackUse();
    }
}
Java Implementation

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

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

class Song{
    private String name;

    public void play() {
        // play song
        MeteringService.trackUse();
    }

    public void showLyrics(){
        // show lyrics
        MeteringService.trackUse();
    }
}
```
Java Implementation

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

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

class Song{
    private String name;

    public void play() {
        // play song
        MeteringService.trackUse(this);
    }

    public void showLyrics(){
        // show lyrics
        MeteringService.trackUse(this);
    }
}
```

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

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

class Song{
    private String name;

    public void play() {
        // play song
    }

    public void showLyrics(){
        // show lyrics
    }
}
### AspectJ Implementation

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

    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** MeteringPolicy {
    pointcut useTitle() :
        execution(public void Song.play()) || execution(public void Song.showLyrics());

    after() returning : useTitle(){
        MeteringService.trackUse();
    }
}
### AspectJ Implementation

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

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

    after() returning : useTitle() {
        MeteringService.trackUse();
    }
}
```
AspectJ Implementation

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

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

    after(Playable playable) returning :
        useTitle(playable) {
        MeteringService.trackUse(playable);
    }
}

Billing code centralized in MeteringPolicy

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()

MeteringPolicy

Playlist

+ play()

* * 

Song

+ play()
+ showLyrics()```
What Is 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
  - 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
- Can package as class files, jars, or in memory bytecodes
Double Billing

- Don’t want to bill twice for songs played within the context of playing a Playlist
- Can accomplish using control flow pointcuts
Control Flow Pointcuts

All these join points are in the control flow of the original play call

These join points are in the control flow below the original play call
Using the cflowbelow() Pointcut

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

  public pointcut topLevelUse(Playable playable) :
    useTitle(playable) &&
    cflowbelow(useTitle(Playable));

  after(Playable playable) returning :
    topLevelUse(playable) {
      MeteringService.trackUse(playable);
    }
}

- cflowbelow() selects only join points within the control flow of another join point.
Configuring Metering

- Thus far we’ve assumed a single metering service
- But metering can depend on
  - Rate plans for user’s account
    - all you can eat, daily pass, pay per use
  - Outsourced metering partner(s)
  - Publisher or title
- How can we accommodate this?
**Abstract Metering:**

**Reusable Base Aspect**

```java
public abstract aspect MeteringPolicy {
    public pointcut useTitle(Playable playable) :
        <as before>

    public pointcut topLevelUse(Playable playable) :
        <as before>

    public abstract MeteringService getMeteringService();

    after(Playable playable) returning : topLevelUse(playable) {
        getMeteringService().generateChargeFor(playable);
    }
}
```
Per User Metering

```java
public aspect UserMeteringPolicy extends MeteringPolicy percflow(
    ServletPointcuts.requestExec(*, *, *)) {
    private MeteringService meteringService;
    private AccountManager accountManager;
    private MeteringManager meteringManager;

    before(HttpServletRequest request) :
        ServletPointcuts.requestExec(*, request, *) {
            meteringService = getService(request);
        }

    public MeteringService getMeteringService() {
        return meteringService;
    }

    public void setMeteringManager(MeteringManager mgr) { … }
    public void setAccountManager(AccountManager mgr) { … }
    public MeterManager getMeterManager() { … }
    public AccountManager getAccountManager() { … }
```

- Creates one instance per control flow
- Reuse library definition
Service Initialization

```java
private MeteringService getService(HttpServletRequest request) {
    HttpSession session = request.getSession();
    MeteringService service = (MeteringService)session.getAttribute(METERING_SERVICE);
    if (service == null) {
        service = findService(request.getRemoteUser());
        session.setAttribute(METERING_SERVICE, service);
    }
    return service;
}

private MeteringService findService(String user) {
    Account account = accountManager.getForUser(user);
    return meteringManager.getForAccount(account);
}
```
Configuring the Aspect with Spring IoC

```java
public aspect MeteringPolicyConfig {
    declare @type: *MeteringPolicy: @Bean;
}
```

**Beans.xml:**

```xml
<beans>
    <bean name="UserMeteringPolicy">
        <property name="accountManager">
            <ref bean="accountManager"/>
        </property>
        <property name="meteringManager">
            <ref bean="meteringManager"/>
        </property>
    </bean>
    <bean name="meteringManager"
        class="music.service.metering.MeteringManagerImpl">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>
</beans>
```

- Uses Colyer's Spring configuration aspects
- Adds an annotation to all metering policy aspects so they will be auto-configured upon initialization
public aspect PerTitleMeteringPolicy extends MeteringPolicy

   perthis(topLevelUse(*)) {

   private MeteringService meteringService;

   before(Playable playable) : topLevelUse(playable) {
       meteringService = getService(playable);
   }

   public MeteringService getMeteringService() {
       return meteringService;
   }

   ...
More on Metering...

- Tracking user per request can be refactored into a separate aspect
- Consuming externally defined services can be hard because of variations
- Aspects are a great way to integrate variations, e.g., metering can
  - interact at different points: check credit before use, after completing play, when connecting
  - require different context: check recent use or recent purchase, have ads been viewed, regulatory variations
  - support affiliate assertions
- Adding to mainline code gets tangled very quickly
Part II:
Development Time Aspects

- Architectural enforcement
- Testing
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, Don’t Print

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

```java
public aspect EnforceLogging {

    pointcut scope():
        within(com.example..*) && !within(TestCase+);

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

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

}
```
Architectural Layering

- Persistence (Layer 4)
- Model (Layer 3)
- View (Layer 2)
- Controller (Layer 1)
Mapping Packages to Layers...

```java
aspect Architecture {

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

  ...
}
```
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..*(..)) ;

...
Compiler 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";

}
Testing: Database Errors

... public void testDatabaseFailure() {
    try {
        from.transfer(2.25, to);
        fail("no exception");
    } catch (ModelException e) {
        assertEquals(SQLException.class, e.getCause().getClass());
        // success
    }
}
...

- Account should convert a SQLException to a ModelException
- How to generate a SQLException?
Using Simple Virtual Mocks

```java
public aspect AccountTestFailureInjection {
    pointcut inFailureTest():
        cflow(execution(* TestAccount.*Failure(..)));

    before() throws SQLException:
        call(* save()) && target(DAO) && inFailureTest() {
            throw new SQLException("can't reach database");
        }
}
```
Testing: *Web Service* Errors
with ajMock

```java
...

private VirtualMock mockCreditCheck;
public void setUp() {
    mockCreditCheck = virtualMock("call(* " +
    CreditService.class.getName()+".getCreditScore(..))");
}

public void testCreditCheckFailure() {
    mockCreditCheck.expects(atLeastOnce()).will(
        throwException(new RemoteException("can't connect")));
    try {
        account.applyForLoan(1000.);
        fail("no exception");
    } catch (ModelException e) {
        assertEquals(RemoteException.class,
                    e.getCause().getClass());
        // success
    }
}
```

Testing: *Disconnected from Service*

```
... public void testOkCredit() {
    final int GOOD_CREDIT = 500;
    mockCreditCheck.expects(atLeastOnce()).will(
        returnValue(GOOD_CREDIT);
    account.applyForLoan(1000.);
    // assert conditions for successful application
}
...```

- Allows testing without generating stubs, and without wiring up dummies
- Does *not* replace end-to-end integration tests
- But is useful for faster testing in isolation
- Similarly for testing service implementation outside of container
Part III: Infrastructure Aspects

- Monitoring
- Error Handling
- Transaction Management
public aspect MonitorRequests {
  Object around(Remote recipient) throws RemoteException:
    jaxRpcCall(recipient) && if(enabled) {
      PerfStats stats = getPerfStats(recipient);
      int start = getTime();
      try {
        Object obj = proceed(recipient);
        stats.recordExecution(getTime(), start);
        return obj;
      } catch (RemoteException e) {
        stats.recordFailure(getTime(), start);
        throw e;
      }
    }
  }

public pointcut jaxRpcCall(Remote recipient) :
  call(public * Remote+.*(..) throws RemoteException) &&
  target(recipient);

More at Performance Management for Web Services with AOP
Error Handling Challenges

- Duplicate logic
- Keeping consistent
  - Appropriate exception types
  - Not swallowing
  - Logging exactly once... with enough context to fix
- Added as an afterthought
- Hard to change
- Presenting error information to users
- User support (what really went wrong?)
- Contain Auxiliary Errors
aspect ModelErrorConversion {
    pointcut modelExec() : execution(* model..*(..));

    after() throwing (SQLException e): modelExec() {
       (convertException(e, thisJoinPoint);
    }

    after() throwing (NamingException e): modelExec() {
        convertException(e, thisJoinPoint);
    }

    declare soft: SQLException: modelExec();
    declare soft: NamingException: modelExec();

    private void convertException(Exception e, JoinPoint jp) {
        ModelException me = new ModelException(e);
        me.setObject(jp.getThis());
        me.setArgs(jp.getArgs());
        throw me; // unchecked exception
    }
}
Error Handling: Calls to Model

```java
aspect ModelErrorHandling {

  pointcut modelEntry() :
    call(* model..*(..)) && !within(model..*);

  after(Object caller) throwing (ModelException e) :
    modelEntry() && this(caller) {
      handleModelError(e, caller);
    }

  after(Object caller) throwing (ModelException e) :
    modelEntry() && !this(*) {
      handleModelError(e, null);
    }

  private void handleModelError(ModelException e, Object caller) {
    // force transaction rollback
    // log caller, executing object, and args
  }
}
```
UI Error Handling

```java
aspect UIErrorHandling {
    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 (Continued)

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

ActionForward handleError(Exception e, ActionMapping mapping) {
    session.setAttribute(Constants.MESSAGE_TEXT, translate(e.getMessage()));
    session.setAttribute(Constants.EXCEPTION, e);
    return mapping.findForward(Constants.ERROR_PAGE);
}
```
Error *Isolation* – for Before & After Advice

```java
aspect AspectErrorHandling {
    public pointcut handlingScope():
        scope() && adviceexecution();

    declare soft: Exception: handlingScope();

    void around() : handlingScope() {
        try {
            proceed();
        } catch (Throwable e) {
            // log but don't rethrow – to contain errors
            handleError(e, thisJoinPointStaticPart);
        }
    }

    public synchronized void handleError(Throwable t) {
        // log 1st and then every 1000 times
    }
}
```
Simple Transaction Management

```java
public aspect TransactionManagement {
    protected pointcut transactionPoint() :
        execution(public * org.sample.model..*(..));
    // could also use annotation to define pointcut…
    public pointcut topLevelTransaction():
        transactionPoint() && !cflowbelow(transactionPoint());

    Object around() : topLevelTransaction() {
        Object returnValue = null;
        try {
            returnValue = proceed();
            transaction.commit();
        } catch (ModelException ex) {
            transaction.rollback();
            throw ex;
        }
        return returnValue;
    }
}
```
Part IV: Business Logic Aspects

- Security
- Account Suspension
AOP for Security Promises...

- Correct implementation
- Separation of policy from implementation
- Defense in depth
- Auditability
- Fine-grained control
- Integration
- Pluggability (across services)
Example Business Model

**Employee**
- + address
- + salary

**Manager**
- + bonus

**EmpRegulation**
- + calcTax()

**US_Regulation**
- + ssn
- + state
- + calcTax()

**CanadaRegulation**
- + sin
- + province
- + calcTax()
Role-Based Authorization

```java
... before() : sensitiveOperations() {
    Permission permission =
        getPermission(thisJoinPointStaticPart);

    AccessController.checkPermission(permission);
}

private Permission getPermission(StaticPart jsp) {
    // config or database lookup
}
```
Data-Driven Authorization
Data Authorization Aspect

```java
public aspect SensitiveDataAuthorization {
    pointcut readSensitiveEmployeeData(Employee employee):
        this(employee) && (execution(* getSalary()) || execution(* getAddress()) ||
        execution(* getBonus()));

    pointcut securityContext(Subject subject, Action action):
        cflow(execution(* Subject.doAs*(Subject, Action, ..)) &&
        args(subject, action));

    before(Subject subject, Employee employee) :
        readSensitiveEmpIData(employee) &&
        securityContext(subject, *) {
            checkAccess(subject, employee);
        }

    ...
}
Data Authorization Access Check

```java
... void checkAccess(Employee employee, Worker worker) {
    Employee caller =
        Employee.getEmployee(worker.getSubject());
    if (caller==null || !employee.reportsTo(caller))) {
        // record attempted security violation
        throw new AuthorizationException("...");
    }
    // and log data access to audit trail
}
...```
Data Authorization Aspect: Helper

```java
...

pointcut sensitiveRegOp(Employee_Regulation reg):
    this(reg) && (
        execution(* get*) || execution(* calcTax());

before(Subject subject, Employee_Regulation reg, Employee employee):
    sensitiveRegOp(reg) && securityContext(subject, *) &&
    cflow(execution(* Employee.*(..)) && this(employee)) {
        if (!reg.equals(employee.getRegulation())) {
            employee = RegMgr.lookupEmployee(reg);
        }
        checkAccess(subject, employee);
    }
```
Refactoring Auditing Out

public aspect SecurityAuditing {
    pointcut securityCheck() :
        // better: @annotation
        within(ajee.security..Authorization*) &&
        advice execution();

    after() returning:
        securityCheck() && authenticated(worker) {
            auditor.recordAccess(worker.getSubject(),
                getAdvisedJp());
        }

    after() throwing (SecurityException se):
        securityCheck() && authenticated(worker) {
            auditor.recordViolation(worker.getSubject(),
                getAdvisedJp(), se);
        }

    ...
}
public aspect P3PDataAuthorization {
  pointcut readP3pData(P3P prefs) :
    get((* @P3P) *.* ) && @annotation(prefs);

  pointcut writeP3pData(P3P prefs) :
    set((* @P3P) *.* ) && @annotation(prefs);

  before(P3P prefs) : readP3pData(prefs) {
    AccessController.checkPermission(new P3P_Permission(prefs, Operation.READ));
  }

  before(P3P prefs) : writeP3pData(prefs) {
    AccessController.checkPermission(new P3P_Permission(prefs, Operation.MODIFY));
  }
}
public aspect ToplinkQuerySecurityFilter {
    pointcut readingObject(Class clazz, Expression expression) :
        (call(* Session.readObject(..)) ||
        call(* Session.readAllObjects(..))) &&
    args(clazz, expression);

    Object around(Class aClass, Expression expression, 
        AbstractJaasAuthentication.Worker worker) :
        readingObject(aClass, expression) &&
        AbstractJaasAuthentication.authenticated(worker) {
            if (aClass == Employee.class) {
                Subject subject = worker.getSubject();
                Manager mgr = ManagerDao.findManager(subject);
                Vector employees = mgr.getEmployees();
                expression = expression.and(new
                    ExpressionBuilder().get("id").in(employees));
            } 
            return proceed(aClass, expression, worker);
        }
}
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:
Account

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

    pointcut suspendedAccountBusinessTrans(Account account):
        this(account) && if(account.suspended) &&
        execution(* *(..)) && BPM.inBusinessTrans();

    before(Account account) :
        suspendedAccountBusinessTrans(account) {
            // record attempted access
            throw new SuspendedAccountException("...");
        }
        // exceptions caught by UI/interaction tier
    }
```

Simple Implementation:
Account
public aspect ServiceSuspension {
    pointcut suspendedServiceBusinessTrans(Service service):
        if(service.getAccount().isSuspended()) &&
        this(service) && execution(* *(..)) &&
        BPM.inBusinessTrans();

    before(Service service) :
        suspendedServiceBusinessTrans(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)
Business Aspects

- Many other possibilities
  - Object ownership
  - 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 V: Conclusion ...

- The state of AOP
- Use throughout development lifecycle
- Adoption strategy
The State of AOP

- **AspectJ 5**: the leader
  - Java 1.1 through 5.0 support
  - Power & tools integration of AspectJ
  - Load-time weaving of AspectWerkz
  - Libraries emerging
  - IBM & BEA leading development

- **Spring, JBoss**
  - Integrated AOP with aspect libraries

- Emerging for .NET, C++, PHP, Ruby, ...
Aspect-Oriented Software Development

- Build on OO methods
- Good OO practices help apply AO
  - And *vice-versa*: good AO encourages and rewards good OO
- Treat Aspects as modules (analog to classes)
- UML extensions
  - Naïve/Stereotypes, Theme/UML, Jacobson, ...
Requirements and Analysis

- Identify crosscutting requirements separately
  - Not scattered and tangled throughout use cases
- Identifying Aspects (Colyer)
  - From adjectives and adverbs: “sensitive data”...
  - Compare to using nouns and verbs for objects
  - Temporal statements: “before calling service, check cache”
  - Policy statements: “whenever a trade exceeds $1 million”
- Analysis Aspects
  - Represent key concepts
  - Have high-level responsibilities (key advice, operations, state)
Design Aspects

- Focus on ARC: aspect, responsibilities, collaborators
- Sequence diagrams natural for interactions
- **Pointcuts** are key: describe **stable properties** (e.g., whenever state changes), not using today’s AOP pointcut languages nor using enumeration
  - Many patterns: template pointcuts, public pointcut describing properties, private pointcuts for implementation needs), ...
- Advice: [@name to describe *what* not *how*
- **Inter-Type Declarations**: design like state/methods for class
Detailed Design and Development

- Refactoring aspects is natural
- Test-Driven Development is useful to flesh out detailed requirements
  - Build on standard tools like JUnit, jMock
  - *e.g.*, virtual mocks with jMock, aUnit for aspects
- Aspects should be unit tested
  - Test drivers mock collaborators (generate join points)
  - Testing crosscutting concerns in isolation is new
- Aspects are also tested in integration tests
A Phased Adoption Strategy

reward

time & confidence
Don't Run before You Can Walk

Risky Space

reward

time & confidence
Phases of Adoption

AO Analysis, Design, Strategic Implications

exploration
enforcement

auxiliary / infrastructure

core business

reward

time & confidence
Within the Phases

- local → spread
- individual → team
- project → library
- team → company
- product → platform
Conclusion

- AOP is rapidly gaining adoption in the enterprise
  - AspectJ is the clear leader: 20,000 downloads/mo.

- Analysis and design approaches emerging

- Incremental adoption works best

- These slides at http://www.newaspects.com/presentations

- Training, consulting, and support available
Thank You

Ron Bodkin  rbodkin@newaspects.com
New Aspects of Software

Thanks to my former colleagues on the AspectJ team at PARC for pioneering how to teach AOP.
Some slides in this presentation are copyright Gregor Kiczales, IBM, Nicholas Lesiecki, Ramnivas Laddad, or Dion Almaer, and are used with permission.