

# AOP System Development: Accomplishing More with Less

Presented by:

**Andrew Oswald**  
**Chariot Solutions**

[aoswald@chariotsolutions.com](mailto:aoswald@chariotsolutions.com)



# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- High level system requirements.
- Can Object-Oriented systems be improved upon?
- How do we know what has been done and/or is happening in our system?
- Does it work? Is it extensible, maintainable, etc.?
- What's left?
- Wrap up – meet me for a beer (or two) at the reception...



- **What is Aspect-Oriented Programming?**
- High level system requirements.
- Can Object-Oriented systems be improved upon?
- How do we know what has been done and/or is happening in our system?
- Does it work? Is it extensible, maintainable, etc.?
- What's left?
- Wrap up – meet me for a beer (or two) at the reception...



# What is Aspect-Oriented Programming?

- “In software engineering, the programming paradigms of aspect-oriented programming (AOP), and aspect-oriented software development (AOSD) attempt to aid programmers in the **separation of concerns**, specifically cross-cutting concerns, as an advance in modularization. AOP does so using primarily language changes, while AOSD uses a combination of language, environment, and method.” -- wikipedia.org



# What is a cross-cutting concern?

- “In computer science, cross-cutting concerns, or crosscutting concerns, are aspects of a program which affect (crosscut) other concerns. **These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and result in either scattering or tangling of the program, or both.**” -- wikipedia.org



# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- **High level system requirements.**
- Can Object-Oriented systems be improved upon?
- How do we know what has been done and/or is happening in our system?
- Does it work? Is it extensible, maintainable, etc.?
- What's left?
- Wrap up – meet me for a beer (or two) at the reception...



# System Requirements

- Technology conference
  - Conference has participants
    - Name
    - Email
  - Conference has speakers
    - Summary
  - Speakers have sessions
    - Title
    - Abstract
- Store information for subsequent use
- Some kind of User Interface



- Java will be used to develop the domain model.
  - Business Logic
  - Security
  - Logging
  - Transactions
  - Persistence
  - Performance
- A database will be used to store the information.
- A browser will be used for the User Interface.

Yawn... “you're telling me this is *emerging*?”





# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- High level system requirements.
- **Can Object-Oriented systems be improved upon?**
- How do we know what has been done and/or is happening in our system?
- Does it work? Is it extensible, maintainable, etc.?
- What's left?
- Wrap up – meet me for a beer (or two) at the reception...



# Can Object-Oriented systems be improved?

- After analyzing the system's requirements, we've got several cross-cutting concerns.
- “Aspectual decomposition” to the rescue.
- Architecture – *emerging*:
  - In order to address inherent OO req't. to implementation redundancy (tangling/scattering), **AspectJ** will *also* be used to develop the domain model.
  - Objects and Relational databases require mapping... We need to get this app into production “yesterday”. In order to expedite delivery (with a nice side effect of better performance), the relational DB and ORM will be replaced with an Object DataBase (DB4O).
  - Continuing with our lazy, er, I mean *emerging* approach, Stripes will be used as the MVC framework.



# System Aspectual Decomposition

- Can AOP wire together an entire application so that its components retain their (clearly) visible/understandable role(s)?
  - In an MVC, procuring and providing the “M” is an aspect. Can this be done in an elegant manner?
  - In a system with persistence capability, persistence of the domain without the objects' knowledge (aka, transparent) is an aspect. Can this be done in an elegant, non-intrusive, timely manner?



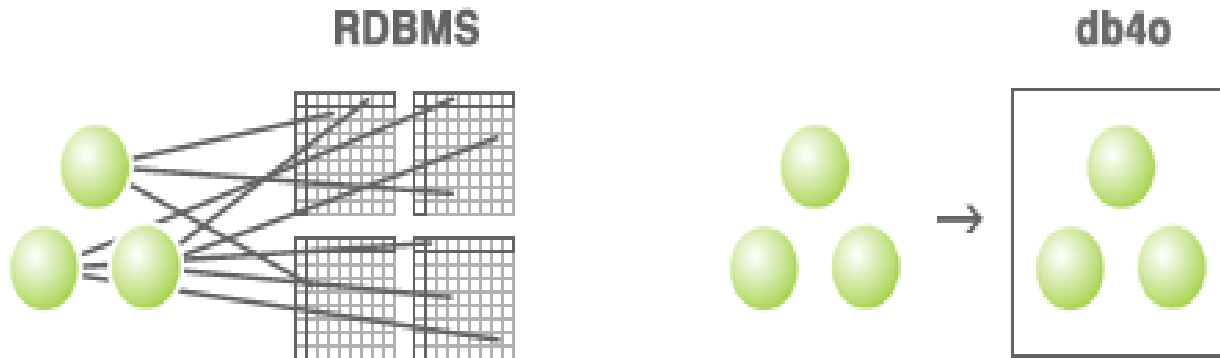
# Stripes – a lazy, er, *emerging* web framework.

- Takes advantage of naming conventions, annotations, and various reflection mechanisms.
- Out-of-box wiring of urls to views (in an ActionBean):
  - package naming: web, stripes, www, and action
  - ActionBean impls: removes “Action” or “Bean” or “ActionBean”
  - converts above info into a path and appends “.action”
  - @UriBinding annotation may be used
- Very powerful binding and validation capability!
- Very nice out-of-box JSP tags.
- @HandlesEvent and reflection handling discovery



# Db4o – can this be real? An *object* database?

- “Embed db4o's native Java and .NET open source object database engine into your product and store even the most complex structures with only **one** line of code.” -- db4o.com
- Open source and free under the GPL



# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- High level system requirements.
- Can Object-Oriented Programming be improved upon?
- **How do we know what has been done and/or is happening in our system?**
- Does it work? Is it extensible, maintainable, etc.?
- What's left?
- Wrap up – meet me for a beer (or two) at the reception...



# Code scattering and tangling.

- Stripes requires ActionBeans to implement ActionBean... (at least it's not extension)
  - get/setContext
  - hold onto ActionBeanContext for subsequent reuse
- For URL mapping, it's either via annotation or naming, or the fact that the class implements ActionBean
  - does your package structure fall under one of the naming conventions?
  - is your package structure representative of a RESTful url?



# Person POJO screenshot:

```
person.jsp  person-update.jsp  *Person.java  log4j.properties  MyActio

package domain;

import java.util.List;

public class Person {
    private String name;
    private List<Pet> pets;
    private Long id;
    private transient Long[] deleteIds;

    public void setDeleteIds(Long[] deleteIds){this.deleteIds = deleteIds;}
    public Long[] getDeleteIds(){return deleteIds;}

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

    public void setId(Long id){this.id = id;}
    public Long getId(){return id;}

    public void setPets(List<Pet> pets){this.pets = pets;}
    public List<Pet> getPets(){return pets;}
}
```





# PersonActionBean screenshot:

```
person.jsp | person-update.jsp | log4j.properties | MyActionBean.java | *Pers
package domain,

import java.util.List;

@UrlBinding("/Person.action")
public class Person implements ActionBean {
    private String name;
    private List<Pet> pets;
    private Long id;
    private transient Long[] deleteIds;
    private transient ActionBeanContext context;

    public ActionBeanContext getContext() {
        return context;
    }
    public void setContext(ActionBeanContext context) {
        this.context = context;
    }

    @DefaultHandler
    public Resolution view(){
        return new ForwardResolution("/WEB-INF/jsp/person.jsp");
    }

    public void setDeleteIds(Long[] deleteIds){this.deleteIds = deleteIds;}
    public Long[] getDeleteIds(){return deleteIds;}

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

    public void setId(Long id){this.id = id;}
    public Long getId(){return id;}

    public void setPets(List<Pet> pets){this.pets = pets;}
    public List<Pet> getPets(){return pets;}
}
```



# Aspectual Decomposition: POJO to ActionBean adaptation

- Implement “ActionBean”
- Keep the passed in ActionBeanContext for subsequent re-use.
- Provide “event” methods so the class actually performs something.
- In light of our naming convention not falling under Stripes' out-of-box qualifiers, provide a @UrlBinding annotation.



# “Introducing” “static crosscutting”

- Artist formerly known as “Introduction” now known as “Inter-type” (although “introduction” was still freely tossed around at AOSD '07)
- Provides static enhancements to a class or classes:
  - implement an interface (in this case, ActionBean)
  - provide methods (get/setActionBeanContext(...))
  - provide member(s) (private ActionBeanContext abc;)
  - provide annotation(s)(!!!!) (@UrlBinding(“/Person.action”))
- Presenter's opinion: “introduction” is still more easily digestible to newcomers...



# First aspect: StripesActionBeannes

```
package aop;

import net.sourceforge.stripes.action.ActionBean;

public interface AspectActionBean extends ActionBean {
    //convenience interface to assign Inter-type members and methods
}
```

```
package aop;

import net.sourceforge.stripes.action.ActionBeanContext;

public aspect StripesActionBeanness {
    private ActionBeanContext AspectActionBean.abc;

    public ActionBeanContext AspectActionBean.getContext(){
        return this.abc;
    }

    public void AspectActionBean.setContext(ActionBeanContext abc){
        this.abc = abc;
    }
}
```



# So we've got some funny new stuff in the gutter...

The screenshot shows an IDE window with a 'Cross Refer...' view on the left and a code editor on the right. The 'Cross Refer...' view shows a tree structure for 'StripesActionBeanness' with 'AspectActionBean.abc' selected. The code editor shows the source code for 'StripesActionBeanness' with a gutter icon on the line containing the private field declaration.

```
package aop;

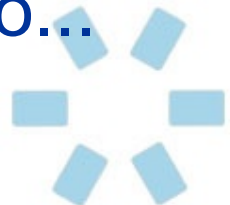
import net.sourceforge.stripes.action.ActionBeanContext;

public aspect StripesActionBeanness {
    private ActionBeanContext AspectActionBean.abc;

    public ActionBeanContext AspectActionBean.getActionBeanContext(){
        return this.abc;
    }

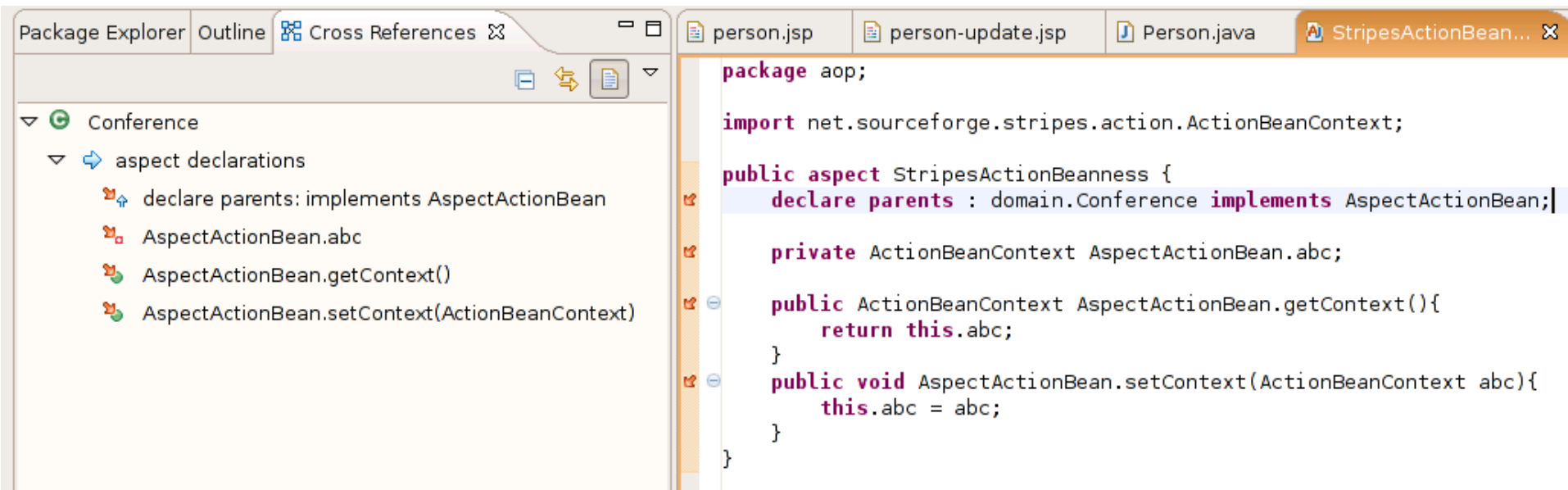
    public void AspectActionBean.setActionBeanContext(ActionBeanContext abc){
        this.abc = abc;
    }
}
```

- Gutter decorations and AJDT's “Cross References” view!
- We can see that our inter-type declarations have been applied to the “AspectActionBean”. Sooo...



# StripesActionBeanness (continued)

- ...what about declaring that a domain object implements “AspectActionBean”?



The screenshot shows an IDE with the Package Explorer on the left and the editor on the right. The Package Explorer shows the following structure:

- Conference
  - aspect declarations
    - declare parents: implements AspectActionBean
    - AspectActionBean.abc
    - AspectActionBean.getContext()
    - AspectActionBean.setContext(ActionBeanContext)

The editor shows the following Java code for `StripesActionBeanness`:

```
package aop;

import net.sourceforge.stripes.action.ActionBeanContext;

public aspect StripesActionBeanness {
    declare parents : domain.Conference implements AspectActionBean;

    private ActionBeanContext AspectActionBean.abc;

    public ActionBeanContext AspectActionBean.getContext(){
        return this.abc;
    }

    public void AspectActionBean.setContext(ActionBeanContext abc){
        this.abc = abc;
    }
}
```



# StripesActionBeanness (continued)

- ...and what about that `@UrlBinding` type annotation?

```
public aspect StripesActionBeanness {  
    declare @type : domain.Conference : @UrlBinding("/Conference.action");  
}
```

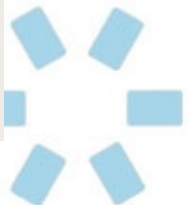
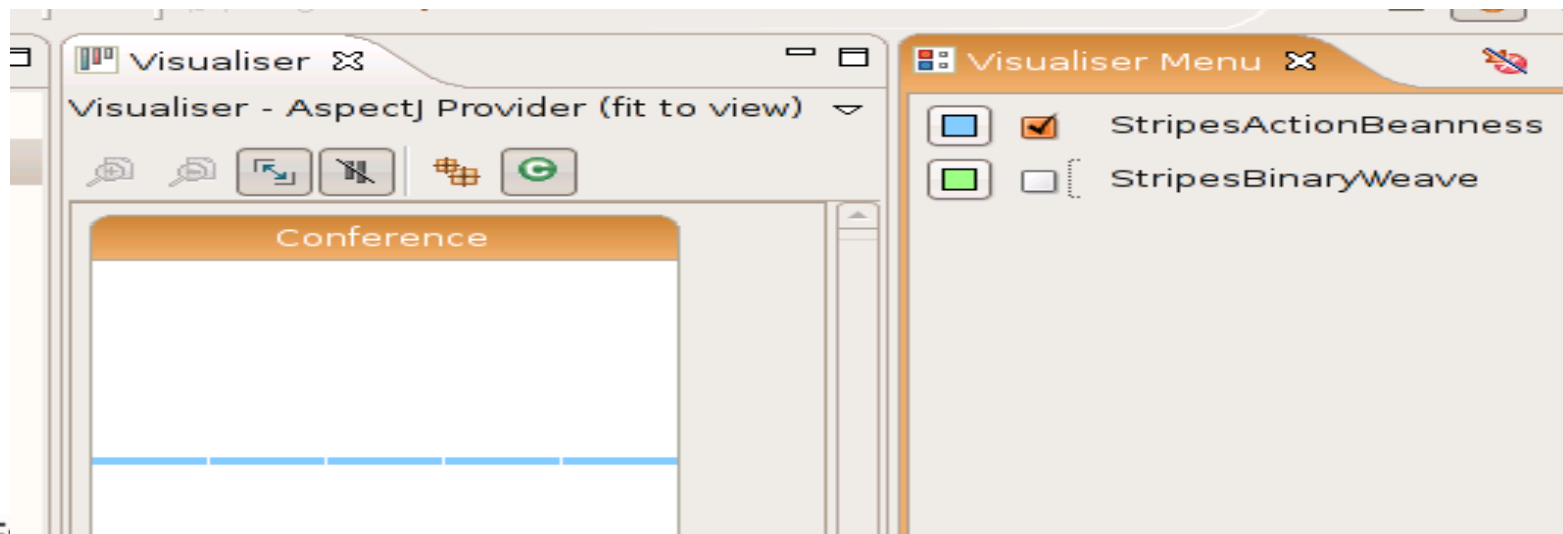
Conference

- aspect declarations
  - declare parents: implements AspectActionBean
  - AspectActionBean.abc
  - AspectActionBean.getContext()
  - AspectActionBean.setContext(ActionBeanContext)
- annotated by
  - StripesActionBeanness.declare @type: domain.Conference : @UrlBinding("/Conference.action")



# More Advisement/Declaration Visualization

- AJDT offers the “Aspect Visualization” perspective.
- Presenter's opinion: this perspective can give you a good “big picture” view, but beware that aspect “precedence” isn't necessarily shown as it executes at runtime!





# Yet even more Aspect Identification...

- **AJDoc: the AspectJ documentation tool**
  - available via command line tool
  - available from within AJDT
  - Presenter's opinion: this tool acts flaky on my Linux machine – YMMV..
- **AJBrowser: gui for compiling and navigating crosscutting structure**
- **Presenter's opinion: I stick w/ AJDT for basically everything.**



# Back to the system....

- So via inter-type declarations, we've made the Conference class into an ActionBean. This seems like overkill, doesn't it. Why yes, requiring every domain object have its own inter-type AspectActionBean parent is ugly. Fortunately, AspectJ offers *wildcarding* for cases like these...
- In our system, all classes in the “domain” package should be ActionBeans (if you're thinking this sounds like an “exposed domain model”, you're correct). Let's see the enhanced declaration:



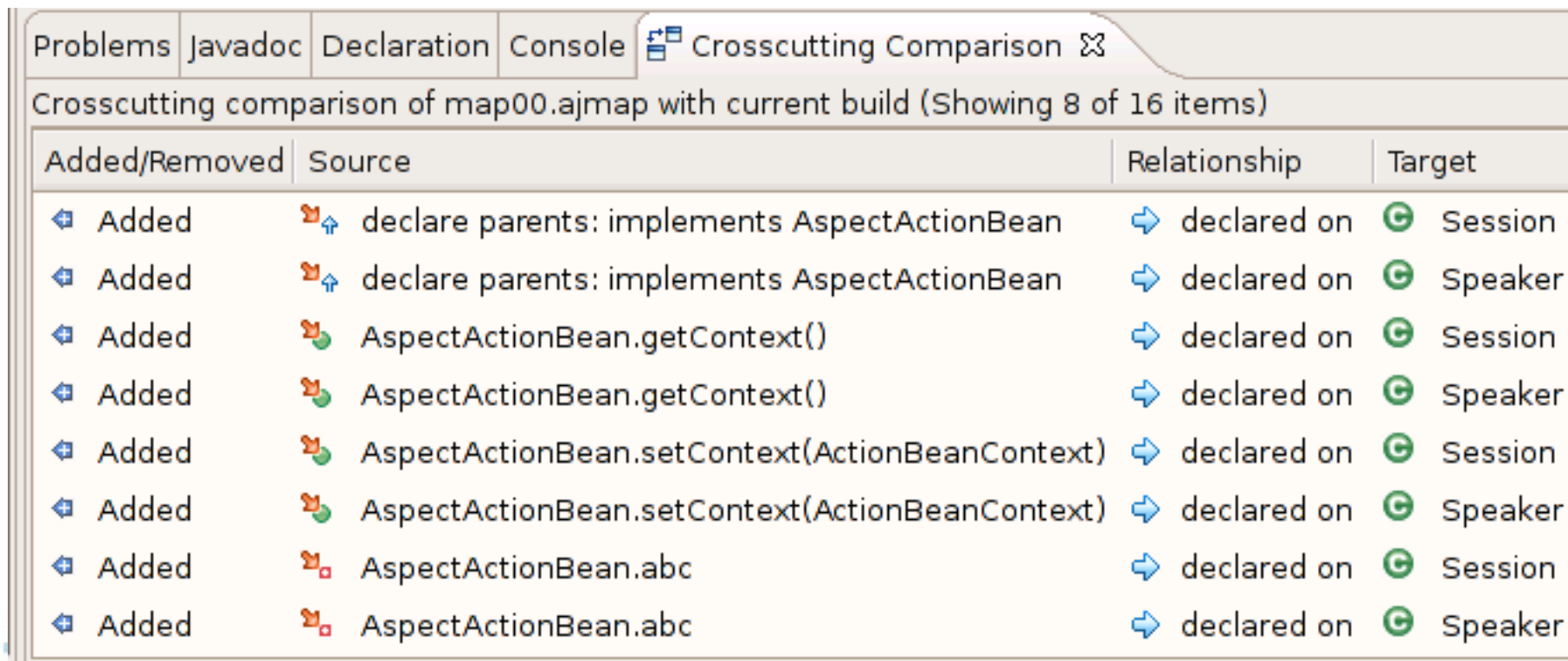
# Accomplishing More with Less: AspectJ's wildcards

- Declaring all classes in the domain package to implement AspectActionBean couldn't be easier – just replace the “domain.**Conference**” with “domain.\*”!
- This raises the question: “how do I keep track of advisement deltas?”
- Again, AJDT to the rescue.. let's take a look.



# Monitoring holistic “advisement deltas”

- As you can imagine, the change from `domain.Conference` to `domain.*` is very significant! The “Crosscutting Comparison” view can show us what's been changed:



Added/Removed	Source	Relationship	Target
Added	declare parents: implements AspectActionBean	declared on	Session
Added	declare parents: implements AspectActionBean	declared on	Speaker
Added	AspectActionBean.getContext()	declared on	Session
Added	AspectActionBean.getContext()	declared on	Speaker
Added	AspectActionBean.setContext(ActionBeanContext)	declared on	Session
Added	AspectActionBean.setContext(ActionBeanContext)	declared on	Speaker
Added	AspectActionBean.abc	declared on	Session
Added	AspectActionBean.abc	declared on	Speaker

# Holistic “advisement deltas” (continued)

- AJDT also offers the capability to take Crosscutting Map “snapshots” of the current codebase in order to do comparisons. As far as I'm aware, this is only available through AJDT (no ANT or Maven). Nevertheless, it's very good info!
- To save a Crosscutting Map, simply right click on an AspectJ project and follow AspectJ Tools > Save Crosscutting Map As...



# Back to Stripes..

- During system startup, Stripes introspects ActionBeans for methods that:
  - return a “Resolution”
  - have a @DefaultHandler annotation
  - have a @HandlesEvent annotation
- At least at this point, our system's “event” requirements are quite meager – create, read, and update the domain.
- If you're thinking this looks like another job for inter-type declarations, you're right! Let's take a look..



# ActionBean event handlers via inter-type.

```
@DefaultHandler
@HandlesEvent("view")
public Resolution AspectActionBean.view(){
    return null;
}
@HandlesEvent("update")
public Resolution AspectActionBean.update(){
    return null;
}
@HandlesEvent("save")
public Resolution AspectActionBean.save(){
    return null;
}

@pointcut aspectActionBeanViewExecution(AspectActionBean bean) :
    execution(* AspectActionBean+.view())
    && this(bean);

@pointcut aspectActionBeanUpdateExecution(AspectActionBean bean) :
    execution(* AspectActionBean+.update())
    && this(bean);

@pointcut aspectActionBeanSaveExecution(AspectActionBean bean) :
    execution(* AspectActionBean+.save())
    && this(bean);
```



# Introducing... the “pointcut”

- “A pointcut is used to select “join points”. It acts as a filter, matching join point that meet its specification, and blocking all others.” -- [Colyer – eclipse AspectJ]
- “A join point is any identifiable execution point in a system.” -- [Laddad – AspectJ in Action]
- So in the case of the pointcuts in the StripesActionBeanEventHandling aspect, the view(), update(), and save() method *executions* are the join points. (yes, even methods declared via inter-type can be used in a pointcut!)





# So now that we've filtered our events via pointcuts...

- “Pointcuts match join points, but **advice** is the means by which we specify what to do at those join points.” -- [Colyer – eclipse AspectJ]
- AspectJ has three kinds of advice:
  - before – works before the specified pointcut's join point(s)
  - after – works after the specified pointcut's join point(s)
  - around – works before and after and sometimes *in place of* the specified pointcut's join point(s) --- more on this to come!
- Let's take a look...



# StripesActionBeanEventHandling aspect's advice.

```
Resolution around(AspectActionBean bean) : aspectActionBeanViewExecution(bean) {
    String shortName = getShortName(bean.getClass().getName());
    String jspFile = new StringBuffer(JSP_DIR).append(shortName).append(".jsp").toString();
    return new ForwardResolution(jspFile);
}

Resolution around(AspectActionBean bean) : aspectActionBeanUpdateExecution(bean) {
    String shortName = getShortName(bean.getClass().getName());
    String jspFile = new StringBuffer(JSP_DIR).append(shortName).append("-update.jsp").toString();
    return new ForwardResolution(jspFile);
}

Resolution around(AspectActionBean bean) : aspectActionBeanSaveExecution(bean) {
    datastore.save(bean);
    String shortName = getShortName(bean.getClass().getName());
    String urlBindingAndId = new StringBuffer("/").append(shortName).append(".action?id=").append(b
    return new RedirectResolution(urlBindingAndId);
}
```



# A look at our DataStore interface.

- The save() event's execution's advice simply calls `dataStore.save(bean)`. Let's take a look at the interface:

```
package aop;

public interface DataStore {
    public <T> T getById(long ID);
    public void attach(Object object);
    public void save(Object object);
    public Long getId(Object object);
}
```



# DataStore (continued)

- And the *Db4oDataStore*:

```
public class Db4oDataStore implements DataStore {
    private static Db4oDataStore instance;
    static {
        Db4o.configure().objectClass(domain.Conference.class).cascadeOnActivate(true);
        Db4o.configure().objectClass(domain.Conference.class).cascadeOnUpdate(true);
        Db4o.configure().objectClass(domain.Speaker.class).cascadeOnActivate(true);
        Db4o.configure().objectClass(domain.Speaker.class).cascadeOnUpdate(true);
    }
    private static ExtObjectContainer DB = Db4o.openFile("db4o").ext();

    private Db4oDataStore(){}

    public static DataStore instance(){
        if(instance == null){
            instance = new Db4oDataStore();
        }
        return instance;
    }

    public void save(Object object) {
        DB.set(object);
    }
}
```



# But how do the respective aspects get a DataStore?

- To generically “announce” that an aspect (or class) needs a DataStore, it simply needs to implement the “NeedsDataStore” interface:

```
package aop;  
  
public interface NeedsDataStore {  
    public abstract void setDataStore(DataStore dataStore);  
}
```



# DataStore procurement (continued)

- Based on that interface, if you were thinking the DataStore gets supplied via dependency injection, you'd be correct.
- To supply DataStore implementations, we simply advise “NeedsDataStore” impl (the “+”) construction:

```
package aop;

public aspect DataStoreProvider {
    pointcut needsDataStoreConstruction(NeedsDataStore nds) :
        execution(NeedsDataStore+.new())
        && this(nds);

    after(NeedsDataStore nds) : needsDataStoreConstruction(nds){
        nds.setDataStore(Db4oDataStore.instance());
    }
}
```



# And the actual implementation (w/ IDE cues)..

```
public aspect DataStoreProvider {  
    pointcut needsDataStoreConstruction(NeedsDataStore nds) :  
        execution(NeedsDataStore+.new())  
        && this(nds);  
  
    after(NeedsDataStore nds) : needsDataStoreConstruction(nds){  
        nds.setDataStore(Db4oDataStore.instance());  
    }  
}
```

```
public aspect StripesActionBeanness implements NeedsDataStore {  
    private DataStore dataStore;  
  
    public void setDataStore(DataStore dataStore){  
        this.dataStore = dataStore;  
    }  
}
```

Package Explorer Outline Cross References X

StripesActionBeanness

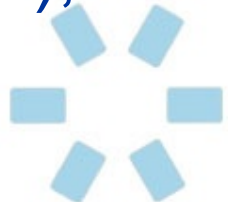
advised by

DataStoreProvider.after(NeedsDataStore): needsDataStoreConstruction..

declare parents

# (Very) Brief introduction to Db4o

- Easy to install – just put the respective (Java 5 or otherwise) .jar in your classpath.
- A couple configurations should get you started:
  - specify what classes should “cascade on activation”
  - specify what classes should “cascade on delete”
- *ObjectContainer* and its more powerful version *ExtObjectContainer* are the interfaces you'll use over 99% of the time.
- Db4o cache is by weak reference.
- Db4o stores an ID for every object stored – obtainable via `ExtObjectContainer.getID(object)`;





## (Very) Brief introduction to Db4o (continued)

- Objects can be retrieved via their Db4o id – this is an **extremely fast** way to query. However, objects retrieved in this manner aren't necessarily guaranteed to be fully hydrated:  
`ExtObjectContainer.getByID(long)`.
- To hydrate objects retrieved via `getByID`, simply call `ExtObjectContainer.activate(object)`.
- As you may have guessed, *transient* members are not persisted.



# Back to Stripes..

- So now we can transparently get hydrated objects, what advantage does this pose?
- We can bind directly into our “exposed domain model”!
- Stripes examples on the web talk of using hooks and the framework itself offers a very handy Interceptor architecture.
- The concept for both, in this case, is the same: if the request, specifically, the *RequestContext*, contains an ID, attempt to retrieve the object, otherwise, it's an insert (or error).



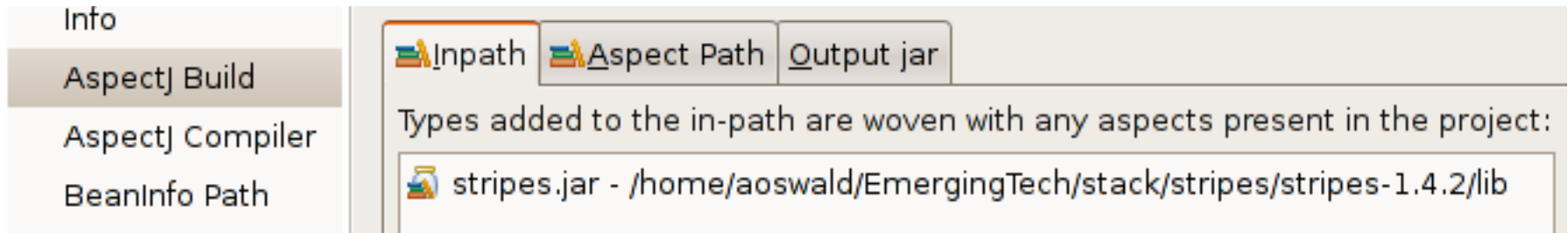
# Stripes Binding (continued)

- In the Stripes execution lifecycle, binding *always* occurs before event execution.
- Can I replace “hook” or Interceptor implementation with an aspect? Absolutely, but this will entail a new concept - “binary weaving”.
- Binary weaving involves creating pointcuts that qualify for code in previously existing jars.
- At compile time, the third party jar(s) contents are examined and where applicable, in essence, the pointcuts qualify, appropriate advice is “woven”.



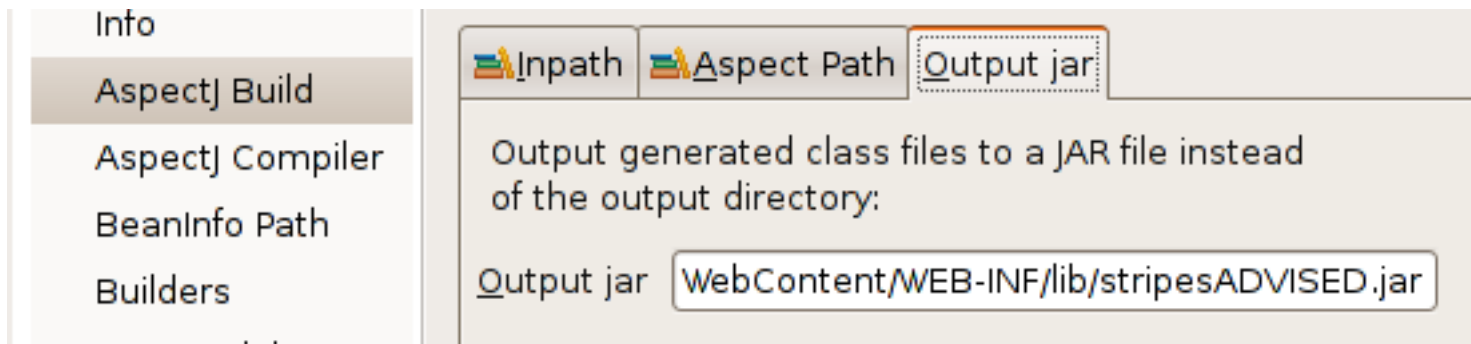
# Binary weaving in AJDT

- On a project that has the AspectJ nature associated, right click on the project and select AspectJ Tools > Configure AspectJ Build Path...
- For this system, we need to weave advise into the Stripes jar – enter its contents into the “Inpath” tab:



# AJDT binary weaving of the Stripes jar (continued)

- Next, we need to specify where the advised jar should be written to.
- Presenter's note: I like to make it obvious that a .jar has been advised by appending ADVISED to its formal name.
- Specify an applicable location and name in the “Output jar” tab:



# Advice to procure (hydrated) exposed domain objects.

- Easiest way to figure out what needs to be advised is to get a “Hello World” app running and simply set breakpoints.
- By doing so, we discover that the method call “makeNewActionBean(...)” of Stripes' AnnotatedClassActionResolver class needs an around advice – remember – if we can't handle the request (no id/bad id/etc.), we simply delegate back to Stripes. Let's take a look...



# StripesBinaryAdvisement (ActionBean creation)

```
/**
 * This pointcut allows us to advise ActionBean creation. The ability of Stripes' binding
 * power is quite impressive. Stripes is able to bind into as many layers deep of an object
 * graph as you may need. Traditionally this is done to domain objects that are encapsulated
 * in a would-be "ActionBean".
 */
pointcut makeNewActionBeanExecution(Class clazz, ActionBeanContext context) :
    execution(ActionBean makeNewActionBean(Class<? extends ActionBean>, ActionBeanContext))
    && within(AnnotatedClassActionResolver)
    && args(clazz, context);
/**
 * This advice determines if an "id" is located in the context and if so, attempts
 * to return a previously stored, fully hydrated ActionBean object. This functionality
 * enables an "exposed domain model".
 */
ActionBean around(Class clazz, ActionBeanContext context) : makeNewActionBeanExecution(clazz, context) {
    ActionBean actionBean = null;
    String stringId = context.getRequest().getParameter("id");
    try{
        Long id = Long.valueOf(stringId);
        if(id != 0){
            actionBean = datastore.getById(id); //call to datastore
            if(clazz.isAssignableFrom(actionBean.getClass())){
                datastore.attach(actionBean); //activate the actionBean
            }else{
                throw new RuntimeException(clazz.getName()+" is not assignable from the retrieved object");
            }
        }
    }catch(NumberFormatException nfe){
        actionBean = proceed(clazz, context);
    }
    return actionBean != null ? actionBean : proceed(clazz, context);
}
```

# So how can we tell if binary weaving did anything?

- Running the app is an option, but there's got to be something better, right?
- AJDT to the rescue (again).. Window > Preferences > AspectJ Compiler -> Information:

## Information

Unresolved member in type

Shadow not in structure:

Calculating serial version UID:

Output weaving info messages to problems view

When binary weaving, this option allows you to see detailed weaving information.





# Did our binary weave work?

- **Yes! The message:** Join point 'method-execution(net.sourceforge.stripes.action.ActionBean net.sourceforge.stripes.controller.AnnotatedClassActionResolver.makeNewActionBean(java.lang.Class, net.sourceforge.stripes.action.ActionBeanContext))' in Type 'net.sourceforge.stripes.controller.**AnnotatedClassActionResolver**' (AnnotatedClassActionResolver.java:361) **advised by around advice** from 'aop.StripesBinaryAdvice' (StripesBinaryAdvice.aj:65) **Appears in the “Problems” view:**

Join point 'method-execution(net.sourceforge.stripes.action.ActionBean net.sourceforge.stripes.controller.AnnotatedClassActionResolver.makeNewActionBean(java.lang.Class, net.sourceforge.stripes.action.ActionBeanContext))' in Type 'net.sourceforge.stripes.controller.**AnnotatedClassActionResolver**' (AnnotatedClassActionResolver.java:361) **advised by around advice** from 'aop.StripesBinaryAdvice' (StripesBinaryAdvice.aj:65) **Appears in the “Problems” view:**



# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- High level system requirements.
- Can Object-Oriented systems be improved upon?
- How do we know what has been done and/or is happening in our system?
- **Does it work? Is it extensible, maintainable, etc.?**
- What's left?
- Wrap up – meet me for a beer (or two) at the reception...



# System lifecycle sequence.

- Project is compiled (domain classes, aspects, binary weaving).
- Project is deployed (or (re)published).
- Tomcat startup:
  - Stripes analyzes classpath for ActionBeans and maps them when found - taking note of UriBindings
    - **\*\*domain.\*** classes inter-typed as AspectActionBeans.
    - **\*\*Binary weaving for dynamic @UrlBinding values.**
  - Stripes analyzes mapped ActionBeans for “event handlers”
    - **\*\*Inter-typed @DefaultHandler and @HandlesEvent methods are mapped.**
- System ready to service..
- Client makes a request
- Stripes' front controller intercepts the request...



# System lifecycle sequence (continued).

- Stripes determines the request is mapped and if so, creates a new instance of the ActionBean.
  - **\*\*Binary weave steps in and determines whether the request includes an id. If an id is present, the aspect attempts to return a hydrated object from Db4o. Otherwise, the aspect delegates back to Stripes.**
- The returned ActionBean is first bound with any request parameters.
- The ActionBean's event is then invoked.
  - **\*\*If applicable, the actionBean is saved to the object database.**
  - **\*\*Around advice dynamically provide the Response type.**
- Stripes analyzes the Response type and dispatches accordingly.
- **Live Demonstration with Debugging.**



# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- High level system requirements.
- Can Object-Oriented systems be improved upon?
- How do we know what has been done and/or is happening in our system?
- Does it work? Is it extensible, maintainable, etc.?
- **What's left?**
- Wrap up – meet me for a beer (or two) at the reception...



# What's left?

- **Aspect library to dynamically:**
  - configure multiple types of persistence
  - configure an adaptation to a completely different MVC framework
- **Deleting objects:**
  - time didn't allow for this to be fully investigated
- **Pretty up the UI**
  - Sitemesh comes to mind
- **Your imagination is the limit!**



# In Summary

- After gathering requirements, take a step back and exercise *Aspectual Decomposition*.
- In order to determine where binary weaving may need to take place, get a HelloWorld application running and simply set breakpoints.
- If you run into problems with the compiler, particularly when working with annotations in AspectJ 5, there is almost always a different way to “skin the cat”. (But don't forget to file a bug report!)



# Relevance/Practicality?

- This demo showed that POJOs can be adapted to environments not necessarily envisioned at their time of creation.
- This approach also enables a delay in architectural decision.
  - Domain may be developed while **concurrent** research is being done on other architectural components.
  - Once the components have been identified, it only takes a couple aspects to adapt the model to them.
- This approach, enables easy migration from “legacy” frameworks to “modern”
  - For example, Spring to Guice (simply inter-type new annotations).





# AOP System Development -- Agenda

- What is Aspect-Oriented Programming?
- High level system requirements.
- Can Object-Oriented systems be improved upon?
- How do we know what has been done and/or is happening in our system?
- Does it work? Is it extensible, maintainable, etc.?
- What's left?
- **Wrap up – meet me for a beer (or two) at the reception...**



# Any Questions?

- Please feel free to stop up and check out the code for yourself!
- The code and presentation will be made available.
- All code and slides were written using Open-Source Software:
  - Ubuntu Linux 6.10: [ubuntu.com](http://ubuntu.com)
  - Eclipse/AJDT: [eclipse.org](http://eclipse.org)
  - OpenOffice: [openoffice.org](http://openoffice.org)
  - Stripes: [stripes.mc4j.org](http://stripes.mc4j.org)
  - Db4o: [db4o.com](http://db4o.com)
- Ladies and Gentlemen, Thank-you!

