AspectJ for Debugging

CSU670 – Spring 2004
What is AspectJ?

- General-purpose AOP language
- Extension to Java
- Helps modularize crosscutting concerns
Brief History

- From Xerox PARC
  - Reflection and MOP at Xerox PARC
  - Open-implementation (white-box abstraction)

- From Northeastern
  - Demeter traversals abstracted
  - DSLs for each concern
    - D for distributed computing
    - COOL for synchronization
    - RIDL for communication

- Crista Lopes went to PARC and AspectJ was born in 1997
What is AspectJ?

• What is crosscutting?
  – A local concern in one view is non-local in another view [UBC]
  – Two decompositions don’t fit neatly together [Kiczales]
What is AOP?

• **Quantification and Obliviousness** [Filman,Friedman]
  – Quantification: Separate unitary statements can affect multiple places in the code
  – Obliviousness: These places are unaware of these quantifications

• **Piecing together decomposed concerns** [Hyper/J]
What is AOP?

• For AspectJ… modularization concerns through *aspects* that place *advice* on *join points*

• Joinpoints
  – Points in the execution of a program

• Advice
  – Code to execute at these joinpoints – i.e. join points trigger advice

• Aspects
  – Later
Language elements

• Join point model
  – Dynamic
  – Static
• Means of identifying join points
  – Pointcuts
• Behavior at join points
  – Advice
## Join point model

<table>
<thead>
<tr>
<th>What</th>
<th>Dynamic</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join points</td>
<td>Points in call graph</td>
<td>Class members</td>
</tr>
<tr>
<td>Identifying join points</td>
<td>Pointcuts</td>
<td>Type patterns</td>
</tr>
<tr>
<td>Semantics of join points</td>
<td>Advice</td>
<td>Defining members</td>
</tr>
</tbody>
</table>
Dynamic join points

- Method & constructor call
- Method & constructor execution
- Field get & set
- Exception handler execution
- Static & dynamic initialization
Pointcuts

• Means of identifying join points
• Ex: Capture all calls to methods \texttt{f} & \texttt{g}, and label these pointcuts \texttt{pf} & \texttt{pg}:
  
  \begin{verbatim}
  pointcut pf(): calls(void f());
  pointcut pg(): calls(void g());
  \end{verbatim}

• Compose \texttt{pf} & \texttt{pg} into \texttt{pall}:
  
  \begin{verbatim}
  pointcut pall(): pf() || pg();
  \end{verbatim}
Advice

• Says *what to do* at a join point
• Ex: Before all points matching `pall`, print a message:

```python
before() : pall() {
    print("a message");
}
```
## Language elements

<table>
<thead>
<tr>
<th>Join Points</th>
<th>Point cuts</th>
<th>Advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>method call</td>
<td>calls</td>
<td>before</td>
</tr>
<tr>
<td>object creation (call to new)</td>
<td>gets</td>
<td>after</td>
</tr>
<tr>
<td>field get/set</td>
<td>sets</td>
<td>after throwing</td>
</tr>
<tr>
<td>exception handler execution</td>
<td>instanceof</td>
<td>after finally</td>
</tr>
<tr>
<td>method execution</td>
<td>cflow, cflowbelow</td>
<td>around</td>
</tr>
<tr>
<td>class-specific initialization</td>
<td>user-defined executions</td>
<td></td>
</tr>
<tr>
<td>static initializer execution</td>
<td>handlers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>initializations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>staticinitializations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>within</td>
<td></td>
</tr>
<tr>
<td></td>
<td>withincode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>callsto</td>
<td></td>
</tr>
</tbody>
</table>
Aspects

• Aspects combine the three ideas into a functional unit:

```java
aspect A {
    pointcut pf(): calls(void f());
    pointcut pg(): calls(void g());
    pointcut pall(): pf() || pg();
    before() : pall() {
        print("a message");
    }
}
```
cflow (control flow)

pointcut JP(): ...
pointcut inflow(): cflow(JP());

all join points on this slide are within the control flow of this join point JP
cflowbelow (control flow 2)

pointcut JP(): ...;
pointcut inflow(): cflowbelow(JP());
Examples