Modularizing Grid Middleware Concerns with AspectJ

Thomas Cottenier
Dragos Morar
Tzilla Elrad

May 2006
Modularizing Grid Middleware Concerns with AspectJ

Outline

• Quick Demo 1 : Tracing container execution in 1 step
• Aspect-Oriented Software Development
• Aspect-Oriented Programming – AspectJ
5. Demo 2: Globus Toolkit Aspects in Action
6. Conclusions
Modularizing Grid Middleware Concerns with AspectJ

Quick Demo 1: Tracing Container execution in 3 steps


2. Deploy Woven Library:

3. Run:
**Modularizing Grid Middleware Concerns with AspectJ**

**Aspect-Oriented Software Development**

**AOSD is about Modularity**

**Evolution of Software Design: Monolithic → Modular**

**Abstraction**
- Focus only on relevant properties

**Decomposition**
- Divide software into separately named and addressable modules

**Encapsulation**
- Group related things together.

**Information Hiding**
- Hide implementation details from the outside

**Separation of Concerns**
- Ensure that each module only deals with one concern
- **Low Coupling** among the modules
- **High Cohesion** within one module
Socket creation in Tomcat
- fits nicely into one package (3 classes)
AOSD - Introduction

Good Modularity

Class Loading in Tomcat
- mostly in one package (9 classes)
Logging in Tomcat
- scattered across the packages and classes
- error handling, security, business rules, …
AOSD – Introduction

Crosscutting Concerns

Scattering
In a given implementation, the code for a concern is spread out. Single concern affects multiple modules.

Tangling
Multiple concerns are interleaved in a single module

Crosscutting Concern
The inherent structure of the concern cannot be modularized in the decomposition mechanism of the language (Objects or procedures), because it follows different composition rules.
Crosscutting Concerns: Scattering

All service operations need to retrieve the resource managed by the service.

```java
package org.globus.examples.services.core.singleton.impl;

import java.rmi.RemoteException;
import org.globus.examples.services.core.singleton.impl.MathResource;
import org.globus.wsrf.ResourceContext;
import org.globus.examples.stubs.MathService instance>AddResponse;
import org.globus.examples.stubs.MathService_instance.SubtractResponse;
import org.globus.examples.stubs.MathService_instance.GetValuer;

public class MathService {

    /*
     * Private method that gets a reference to the resource specified in the
     * endpoint reference.
     */
    private MathResource getResource() throws RemoteException {
        Object resource = null;
        try {
            resource = ResourceContext.getResourceContext().getResource();
        } catch (Exception e) {
            throw new RemoteException("Unable to access resource.", e);
        }
        MathResource mathResource = (MathResource) resource;
        return mathResource;
    }

    /* Implementation of add, subtract, and getValue operations */
    public AddResponse add(int a) throws RemoteException {
        MathResource mathResource = getResource();
        mathResource.setValue(mathResource.getValue() + a);
        mathResource.setLastOp("ADDITION");
        return new AddResponse();
    }

    public SubtractResponse subtract(int a) throws RemoteException {
        MathResource mathResource = getResource();
        mathResource.setValue(mathResource.getValue() - a);
        mathResource.setLastOp("SUBTRACTION");
        return new SubtractResponse();
    }

    public int getValueRP (GetValueRP params) throws RemoteException {
        MathResource mathResource = getResource();
        return mathResource.getValue();
    }
}
```
Crosscutting Concerns: Tangling

Base Resource Functionality

Initialiation of the Resource Properties

Factory Implementation: Unique Resource Identifier

Resource Lifecycle Management

Topic Lists for Notifications
Crosscutting Concerns: Tangling

Base Resource Functionality

Initialization of the Resource Properties

Factory Implementation: Unique Resource Identifier

Resource Lifecycle Management

Topic Lists for Notifications

```java
/* Get/Setter for the RPs */
public int getValue() {
    return value;
}

public void setValue(int value) {
    this.value = value;
}

public String getLastOp() {
    return lastOp;
}

public void setLastOp(String lastOp) {
    this.lastOp = lastOp;
}

/* Required by interface ResourceProperties */
public ResourcePropertySet getResourcePropertySet() {
    return this.propSet;
}

/* Required by interface ResourceIdentifier */
public Object getID() {
    return this.key;
}

/* Required by interface ResourceLifetime */
public Calendar getCurrentTime() {
    return Calendar.getInstance();
}

public Calendar getTerminationTime() {
    return this.terminationTime;
}

public void setTerminationTime(Calendar terminationTime) {
    this.terminationTime = terminationTime;
}

/* Required by interface TopicListAccessor */
public TopicList getTopicList() {
    return topicList;
}
```
Crosscutting Concerns: Tangling

- Base Resource Functionality
- Initialization of the Resource Properties
- Factory Implementation: Unique Resource Identifier
- Resource Lifecycle Management
- Topic Lists for Notifications
Crosscutting Concerns

- **Auxiliary concerns are scattered and tangled**
  Synchronization, Real-time constraints, Error-checking, Object interaction constraints, Memory management, Persistency, Security, Caching, Logging, Monitoring, Testing, Domain specific optimization

- **80% of problems come from this 20% of code**
  - **Inflexibility**: hard to change/maintain
    have to find all the code involved and be sure to change it consistently
  - **Redundancy**: same fragment of code in many places
    Leads to inconsistency
  - **Incomprehensibility**: can’t see the forest for the trees
The AOSD idea

- crosscutting is inherent in complex systems
- crosscutting concerns
  - have a clear purpose
  - have a natural structure
- let’s capture the structure of crosscutting concerns explicitly
  - in a modular way
  - with linguistic and tool support
AOSD – Introduction

Aspects

Goal of AOSD
Enable the modular (not scattered) implementation of crosscutting concerns.

Aspect
Explicit abstraction for representing crosscutting concerns

Aspect-Oriented Programming
Language that provides mechanisms to capture aspects

Aspect Weaver
Composition mechanism that coordinates aspects with the rest of the code.
Aspect-Oriented Programming

Aspect-Oriented Language Elements

- **join point (JP) model**
  certain principled points in program execution such as method calls, field accesses, and object construction

- **means of identifying JPs**
  picking out join points of interest (predicate)
  **pointcut**: set of join points

- **means of specifying behavior at JPs**
  what happens
  **advice** declarations

```java
pointcut traced():
call(* Line.*) ||
call(* Point.*);

before(): traced() {
    println("Entering:" + thisjoinpoint);
}
```
Aspect-Oriented Programming

Aspect-Oriented Language Elements

Joinpoints

certain principled points in program execution such as method calls, field accesses, and object construction

- method execution
- method call
- field get & set
- exception handler execution
- static & dynamic initialization
- …
Pointcuts: Queries over Joinpoints

- filters picking out join points
- primitive and user-definable
- composable with &&, ||, !

```java
pointcut moves():
call(void Line.setP1(Point)) ||
call(void Line.setP2(Point)) ||
call(void Point.setX(int)) ||
call(void Point.setY(int)) ||
call(void FigureElement.incrXY());
```

```java
pointcut traced():
call(* Line.*) || call(* Point.*);
```
**AOSD – Aspect-Oriented Programming**

**Aspect-Oriented Language Elements**

**Advice**
- code body attached to a pointcut
- code runs at every join point picked out by a pointcut
- after advice runs after the computation of a join point
  (both returning normally and throwing an exception)

```java
after(): moves() {
    Display.update();
}

before(): (sets(int Point.x)) &&
    !withincode(void Point.setX())) ||
    (sets(int Point.y)) &&
    !withincode(void Point.setY())) {
    throw new AccessDeniedException();
}
```
An Aspect is a type that crosscuts other types

```java
aspect MoveCounting {
    int count = 0;

    pointcut moves():
        ...as before ...

    after(): moves() && !cflowbelow(moves()) {
        count++;
    }
}
```
AOSD – AOSD for Middleware

E.g.: J2EE without EJB – Lightweight Containers

• **Lightweight Container**:
  - Manage application code, but does not impose dependencies on that code (non-invasiveness)
  - Quick to start up
  - Does not require special deployment steps for objects
  - Light footprint and minimal API dependencies
  - Low deployment effort and performance overhead
  - Possible to deploy fine-grained objects

• **Inversion of Control**: avoid having application code depending on a container

Aspects can encapsulate the dependencies on the Container
Modularizing Grid Middleware Concerns with AspectJ

Globus Toolkit Aspects in Action

```java
package org.globus.examples.services.core.aspectfactory.impl;

import java.rmi.RemoteException;
import org.globus.examples.stubs.MathService_instance.AddResponse;
import org.globus.examples.stubs.MathService_instance.GetValueRP;
import org.globus.examples.stubs.MathService_instance.SubtractResponse;

public class MathService {
    public MathResource mathResource = new MathResource();

    public AddResponse add(int a) throws RemoteException {
        mathResource.setValue(mathResource.getValue() + a);
        mathResource.setLastOp("ADDITION");
        return new AddResponse();
    }

    public SubtractResponse subtract(int a) throws RemoteException {
        mathResource.setValue(mathResource.getValue() - a);
        mathResource.setLastOp("SUBTRACTION");
        return new SubtractResponse();
    }

    public int getValueRP(GetValueRP params) throws RemoteException {
        return mathResource.getValue();
    }
}
```
Modularizing Grid Middleware Concerns with AspectJ

Globus Toolkit Aspects in Action

```java
package org.globus.examples.services.core.aspectresource.impl;

public class MathResource {

    private int value;
    private String lastOp;

    public void initialize() throws Exception {
        setValue(0);
        setLastOp("NONE");
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public String getLastOp() {
        return lastOp;
    }

    public void setLastOp(String lastOp) {
        this.lastOp = lastOp;
    }
}
```
Generic Resource Management Aspect for Singleton Resources:
Reusable as-is, without any modification
Modularizing Grid Middleware Concerns with AspectJ

Globus Toolkit Aspects in Action

Generic Resource Management
Aspect for Singleton Resources:
Instantiation for a particular service and resource:

```java
package org.globus.examples.services.core.aspectresource.impl;

public aspect ResourceAspectImpl
    extends ResourceGenericAspect
    <org.globus.examples.services.core.aspectresource.impl.MathResource,
     org.globus.examples.services.core.aspectresource.impl.MathQNames,
     org.globus.examples.services.core.aspectresource.impl.MathService>
{
    before(): ResourceInitialization()
    {
        try{
            Class.forName("org.globus.examples.services.core.aspectresource.impl.MathQNames");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```
Generic Resource Management Aspect for Factory Resources:
Reusable as-is, without any modification
Modularizing Grid Middleware Concerns with AspectJ

Globus Toolkit Aspects in Action

Generic Resource Management
Aspect for Factory Resources:
Instantiation for a particular service and resource:

```java
package org.globus.examples.services.core.aspectfactory.impl;

public aspect FactoryAspectImpl extends FactoryGenericAspect

<org.globus.examples.services.core.aspectfactory.impl.MathResource,
 org.globus.examples.services.core.aspectfactory.impl.MathQNames,
 org.globus.examples.services.core.aspectfactory.impl.MathService>

{  
    before(): ResourceInitialization()
    try{
        Class.forName("org.globus.examples.services.core.aspectfactory.impl.MathQNames");
    }catch(Exception e){
        e.printStackTrace();
    }
}
```
How to use AspectJ with the Programmer’s tutorial:

2. In the Build.xml file, comment out the ‘compile’ task. We will compile files manually.
4. Compile the service files with the AspectJ compiler and copy the class files to the build directory.

```
$ ajc -1.5 -classpath "build/stubs/classes;${CLASSPATH};." org/globus/examples/services/core/aspectfactory/impl/*.java org/globus/examples/services/core/aspectfactory/impl/*.aj
```

```
$ cp org/globus/examples/services/core/aspectfactory/impl/*.class build/classes
```

9. Build the gar archive by running globus-build-service.sh again.
10. Deploy the service and run.
Thank You

Questions?