INTERFACE21

ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

The Vocabulary of Enterprise Applications

Adrian Colyer, CTO, Interface21

adrian.colyer@interface21.com

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Agenda

- The Spring Approach
- A common vocabulary
- Talk the talk... walk the walk
- AOP in Spring 2.0

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The Spring approach

- Program simply using objects
  - aka "POJOs"
  - non-invasive
  - object-oriented
- Retain architectural choice
  - no environmental assumptions or dependencies in application objects
- Facilitate test-driven development
- Simple, Powerful, Proven

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The Spring approach

Simple does not mean

simplistic

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

The Spring approach

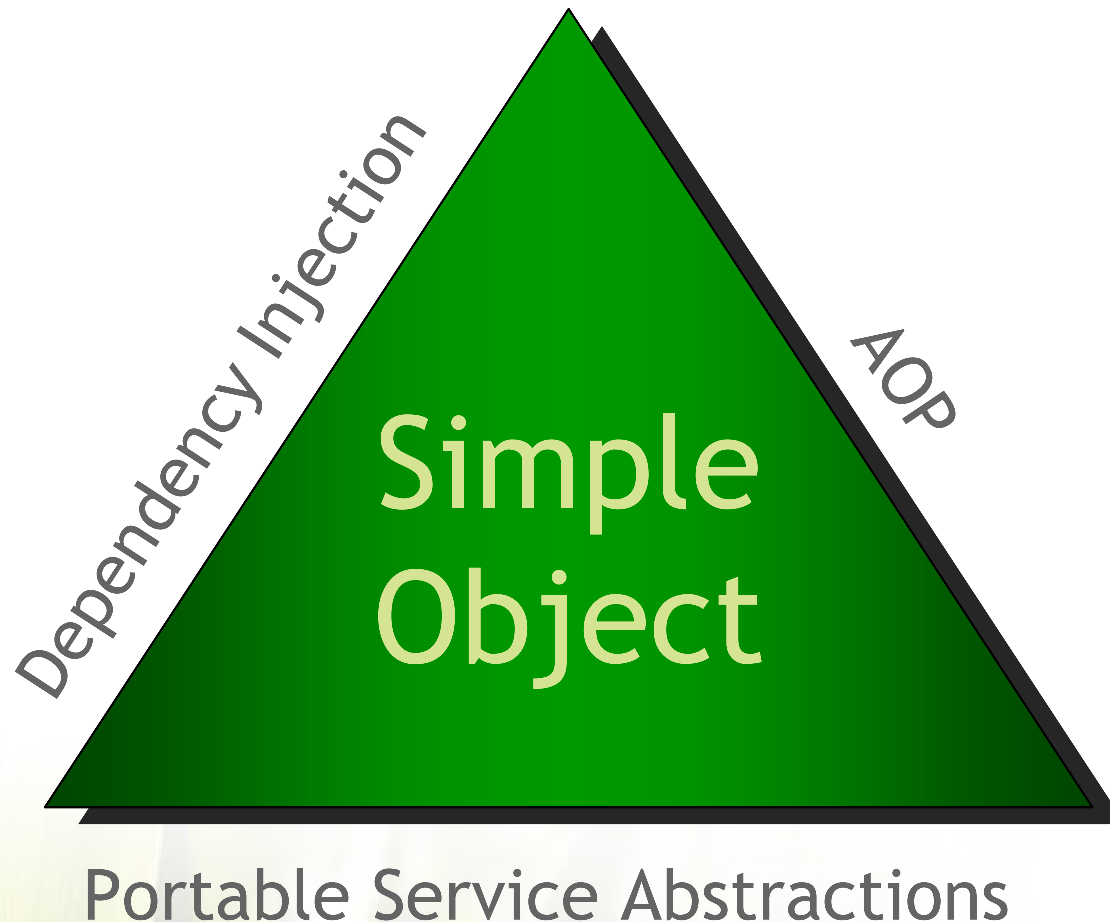
Simpler can be more

powerful

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring

Enabling Technologies



Portable Service Abstractions

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

A Common Vocabulary

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Enterprise application vocabulary

the **vocabulary** of enterprise applications

business service

service layer

repository

dao

controller

web layer

data access layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed *in terms of this vocabulary*
  - the service layer should be transactional
  - when a Hibernate dao operation fails the exception should be translated
  - service layer objects should not call the web layer
  - a business service that fails with a concurrency related failure can be retried

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Meaningful abstractions

It would be simpler…

and more *powerful*

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Meaningful abstractions

if we could use these

~~terms~~ abstractions

directly in the

implementation

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Talk the talk… Walk the walk

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

System Architecture

```
@Aspect
public class SystemArchitecture {

  @Pointcut("within(a.b.c.service..*")
  public void inServiceLayer() {}

  @Pointcut("within(a.b.c.dao..*")
  public void inDataAccessLayer() {}

  @Pointcut("execution(* a.b.c.service.*.*(..))")
  public void businessService() {}


  …
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed in terms of this vocabulary
  - **the service layer should be transactional**
  - when a Hibernate dao operation fails the exception should be translated
  - a business service that fails with a concurrency related failure can be retried
  - service layer objects should not call the web layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Transactions

```
<aop:config>
  <aop:advisor
      pointcut="SystemArchitecture.businessService()"
      advice-ref="tx-demarcation"/>
</aop:config>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21

Transactions

```
<tx:advice id="tx-demarcation">
  <method name="*"
    propagation="REQUIRED"
    isolation="DEFAULT"/>
</tx:advice>
```

- Gives us TX-REQUIRED semantics for the service layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Requirements

- Many requirements are expressed in terms of this vocabulary
  - ✓ the service layer should be transactional
  - ❑ **when a Hibernate dao operation fails the exception should be translated**
  - – a business service that fails with a concurrency related failure can be retried
  - – service layer objects should not call the web layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Scenario...

- You have your own data access layer written using Hibernate 3
  - not using the Spring HibernateTemplate
- In the service layer, you want to insulate yourself from Hibernate exceptions, and take advantage of Spring's fine-grained DataAccessException hierarchy
- After throwing a hibernate exception from a data access operation, convert it into a DataAccessException…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Step 1: Define the abstraction

```
@Aspect
public class SystemArchitecture {

  …

  @Pointcut("execution(* a.b.c.dao.*.*(..))")
  public void dataAccessOperation() {}

  …
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Step 2: Use the abstraction

- How do we make use of this dataAccessOperation abstraction?
- Advice!
- Advice is associated with a pointcut expression
- Executes every time a join point matched by the expression occurs

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Which advice kind?

- "After throwing a hibernate exception from a data access operation, convert it into a DataAccessException…"

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

After throwing

```
@AfterThrowing(
  throwing="hibernateEx",
  pointcut="SystemArchitecture.dataAccessOperation()"
)
public void rethrowAsDataAccessException(
  HibernateException hibernateEx) {
  // convert exception and rethrow…
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Where does advice live?

- Advice is declared in an aspect
- Aspect are like classes
  - instances
  - state (fields)
  - behaviour (methods)
- Aspects can also have
  - pointcuts
  - advice
  - and a few other things…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Aspect

```
@Aspect
public class HibernateExceptionTranslator {
    // …

    @AfterThrowing(
      throwing="hibernateEx",
      pointcut="SystemArchitecture.dataAccessOperation()"
    )
    public void rethrowAsDataAccessException(
      HibernateException hibernateEx) {
      // convert exception and rethrow…
    }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Step 3: Configuration

```
<aop:aspectj-autoproxy/>

<bean id="hibernateExceptionTranslator"
    class="HibernateExceptionTranslator">
      <property name="hibernateTemplate"
                    ref="hibernateTemplate"/>
</bean>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Schema alternative

- For JDK 1.4 and below
  - and annophobes ;)
- The exact same aspect can be declared in Spring XML, backed by a plain Java class

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Schema-based configuration

```
<aop:config>

  <aop:aspect ref="hibernateExceptionTranslator">
    <aop:after-throwing
        throwing="hibernateEx"
        pointcut="SystemArchitecture.dataAccessOperation()"
        method="rethrowAsDataAccessException"/>
  </aop:aspect>

</aop:config>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Bean Implementation

```
public class HibernateExceptionTranslator {
  private HibernateTemplate hibernateTemplate;

  public void setHibernateTemplate(
    HibernateTemplate aTemplate){
    this.hibernateTemplate = aTemplate;
  }

  public void rethrowAsDataAccessException(
        HibernateException hibernateEx) {
    throw this.hibernateTemplate
         .convertHibernateAccessException(hibernateEx);
  }
}
```

parameter bound in pcut expression

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed in terms of this vocabulary
  - ✓ the service layer should be transactional
  - ✓ when a Hibernate dao operation fails the exception should be translated
  - ❑ **a business service that fails with a concurrency related failure can be retried**
  - – service layer objects should not call the web layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Retry

- One subtype of DataAccessException is…
  - ConcurrencyFailureException
- Concurrency failures are potentially recoverable
  - If the operation is idempotent, we can retry it*
- We need a ConcurrentOperationExecutor…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Concurrent Operation Recovery

```
public class ConcurrentOperationExecutor {
  private static final int DEFAULT_MAX_RETRIES = 2;
  private int maxRetries = DEFAULT_MAX_RETRIES;

  /** configurable via dependency injection */
  public void setMaxRetries(int numTimesToRetry) {
    this.maxRetries = numTimesToRetry;
  }

 …
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Concurrent Operation Recovery

```
public Object doConcurrentOperation(ProceedingJoinPoint pjp)
  throws Throwable {
   int numAttempts = 0;
   ConcurrencyFailureException failureException;
   do {
     numAttempts++;
     try { return pjp.proceed(); }
     catch(ConcurrencyFailureException ex) {
       failureException = ex;
     }
   }
   while(numAttempts <= this.maxRetries);
   throw failureException;
  }

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Concurrent Operation Recovery

```
<aop:config>

  <aop:aspect ref="concurrentOperationExecutor">
    <aop:pointcut id="idempotentOperation"
        expression= "SystemArchitecture.businessService()"/>

    <aop:around
          pointcut-ref="idempotentOperation"
          method="doConcurrentOperation"/>
  </aop:aspect>

</aop:config>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21

Concurrent Operation Recovery

```
<bean id="concurrentOperationExecutor"
   class="ConcurrentOperationExecutor">
   <property name="maxRetries" value="3"/>
</bean>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Recap:

- Created an abstraction: idempotentOperation
- Used around advice to retry failing idempotentOperations
- Packaged in a ConcurrentOperationExecutor aspect
- Configured using Spring

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed in terms of this vocabulary
  - ✓ the service layer should be transactional
  - ✓ when a Hibernate dao operation fails the exception should be translated
  - ✓ a business service that fails with a concurrency related failure can be retried
  - ❑ ***service layer objects should not call the web layer***

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Permitted component interactions

```
/** … */
public aspect SystemArchitecture {
 …
 /*
  * no other module should depend on the
  * web tier
  */
 declare warning
   : callToWebTier() && !inWebTier()
   : "no external dependencies on web tier";


  …
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

AOP in Spring 2.0

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring 2.0 AOP

- Aspects are defined in Spring configuration file
  - supports both XML based definition
  - and @AspectJ aspects!
- XML defined aspects are backed by regular classes

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Spring 2.0 and AspectJ

- Spring and AspectJ are still distinct projects
- Spring just uses the AspectJ pointcut parsing and matching APIs
  - using AspectJ as a library, not as a weaving engine
- Gives the same syntax and semantics across Spring AOP and AspectJ
  - perfect if you are going to use both
  - or start out with Spring AOP, and then want to introduce AspectJ at some point

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Adoption Roadmap

1. Use the out-of-the-box Spring aspects
   - transactions etc.
2. Introduce your own Spring-AOP based aspects as needed
3. Add enforcement and exploration aspects using AspectJ
4. Add infrastructure aspects using AspectJ
5. Add domain aspects using AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Summary

- We want to implement enterprise requirements in as simple and straightforward a manner as possible
  - use the appropriate implementation "vocabulary"
- AOP provides the necessary abstractions
- AspectJ and Spring AOP are the leading AOP implementations
  - tightly integrated
  - Can use together or independently
- Adoption can proceed in phases
  - start simple, grow over time as needed

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Questions...?

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.