



# Agile Enterprise Development with Groovy & Grails



Björn Beskow

[bjorn.beskow@callistaenterprise.se](mailto:bjorn.beskow@callistaenterprise.se)

[www.callistaenterprise.se](http://www.callistaenterprise.se)



# Who I am

---



- **Name:** Björn Beskow
- **Assignment:** JavaEE Coach
- **Experience:** Alecta, SCA, Extenda, Papyrus, Volvo, ..
- **Sectors:** Automotive, eCommerce, Retail, Finance, ..

# Agenda

---

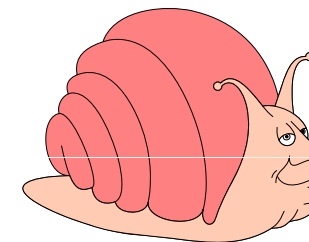
- Why yet another application stack?
- Why Groovy and Grails?
- Groovy overview
- Grails overview
- Grails demo
- Integration Scenarios
- Conclusions
- Questions



# Java EE Success Story?

---

- The JavaEE platform has been a tremendous success story in terms of
  - Stability
  - Scalability
  - Robustness
  - Interoperability
  - Flexibility
  
- ... but not in terms of simplicity and productivity



# Essential vs. Accidental Complexity

---

- If most (or at least a major part) of the Complexity involved with Software Development comes from the problem we are trying to solve, we might accept some accidental complexity to get the job done.
- But what if most of the complexity is Accidental?





# Accidental Complexity in JavaEE 5, Hibernate & Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hib
<!DOCTYPE struts-config PUBLIC
w.w3.org/2001/XMLSchema-instance"
work.org/schema/beans"
framework.org/schema/util"
v.springframework.org/schema/beans
org/schema/beans/spring-beans-2.5.xsd
org/schema/util
org/schema/util/spring-util-2.5.xsd">
8"?>
Sun Microsystems, Inc.//DTD JavaServer
a.sun.com/dtd/web-facesconfig_1_0.dtd">
from-view-id>
ome>secondpage</from-outcome>
d>/secondpage.jsp</to-view-id>
ue(strategy=GenerationType.AUTO)
d() {
public class EmployeeDao {
    private HibernateTemplate hibernateTemplate;
    public void setHibernateTemplate(HibernateTemplate hibernateTemplate){
        this.hibernateTemplate = hibernateTemplate;
    }
    public HibernateTemplate getHibernateTemplate(){
        return hibernateTemplate;
    }
    public Employee getEmployee(final String id){
        HibernateCallback callback = new HibernateCallback() {
            public Object doInHibernate(Session session)
                throws HibernateException,SQLException {
                return session.load(Employee.class, id);
            }
        };
        return (Employee)hibernateTemplate.execute(callback);
    }
    public void saveOrUpdate(final Employee employee){
        HibernateCallback callback = new HibernateCallback() {
            public Object doInHibernate(Session session)
                throws HibernateException,SQLException {
                session.saveOrUpdate(employee);
            }
        };
    }
}
```



# Dynamic Frameworks challenges JavaEE

---

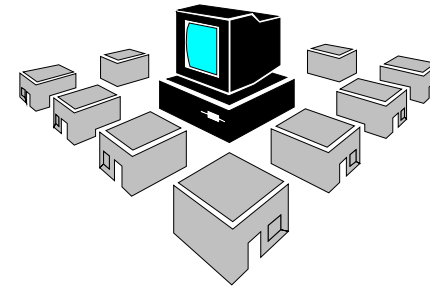
- Frameworks like Ruby on Rails, Python w. Django and Scala w. Lift claims magnitude of increased developer productivity, by
  - Relying heavily on conventions and proven patterns
  - Utilizing the tremendous power of dynamic languages



- So what is the problem with RoR, Django or Scala/Lift?
  - They don't protect and leverage your current investment in the JavaEE platform

# Protect investment in the JavaEE platform

- Existing applications
- Existing APIs
- Existing 3rd party components
- Existing competence



JMS



JDBC



JUnit



JNDI



# What makes Groovy and Grails different?

---

- Builds on the existing Java EE platform
  - Groovy is Java, only so much better!
  - Grails is just a coherent packaging of a solid, working foundation: Hibernate, Spring, SiteMesh, Just so much simpler!
- A natural part of the JavaEE Eco System
  - Integrates seamlessly with Spring, EJB, JPA, Quartz, Log4J, XFire, Axis, Ant, Maven, ...



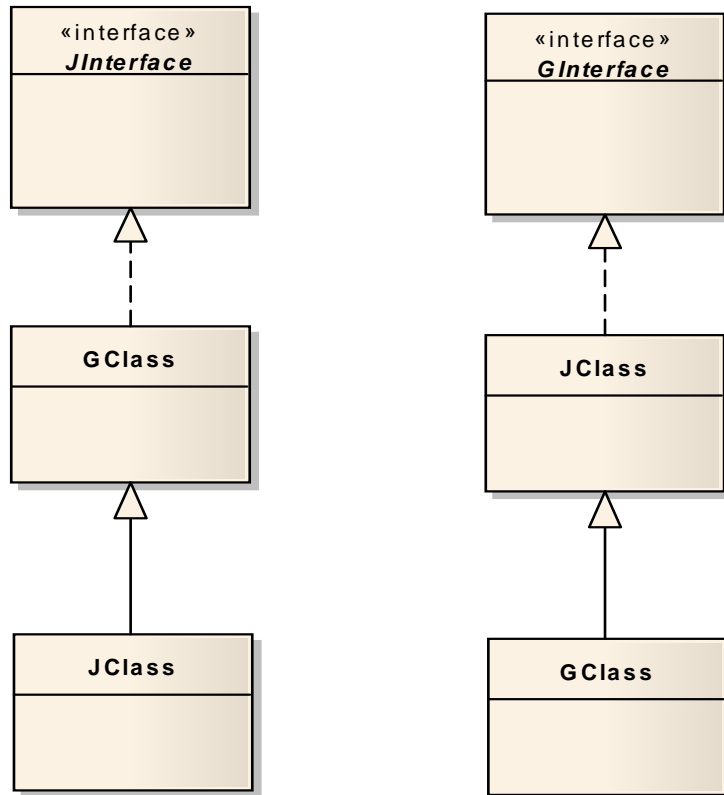
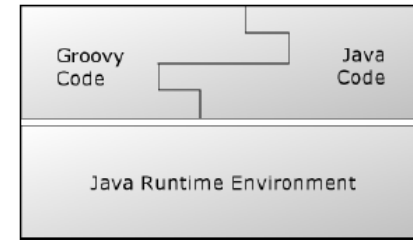
# Groovy Basics

---

- Groovy is a **dynamic language** on and for the JVM
  - Compiles directly down to bytecode
- Inspired by other dynamic OO languages: Smalltalk, Ruby, Python
- Goal is to **greatly simplify the life for developers**
- Totally object-oriented
- Dynamically typed, with optional static typing
- Why is Groovy more Java than JRuby or Scala?
  - JRuby and Scala are languages in their own right
  - They have their own type systems and frameworks, but gain interoperability with Java classes through the JVM



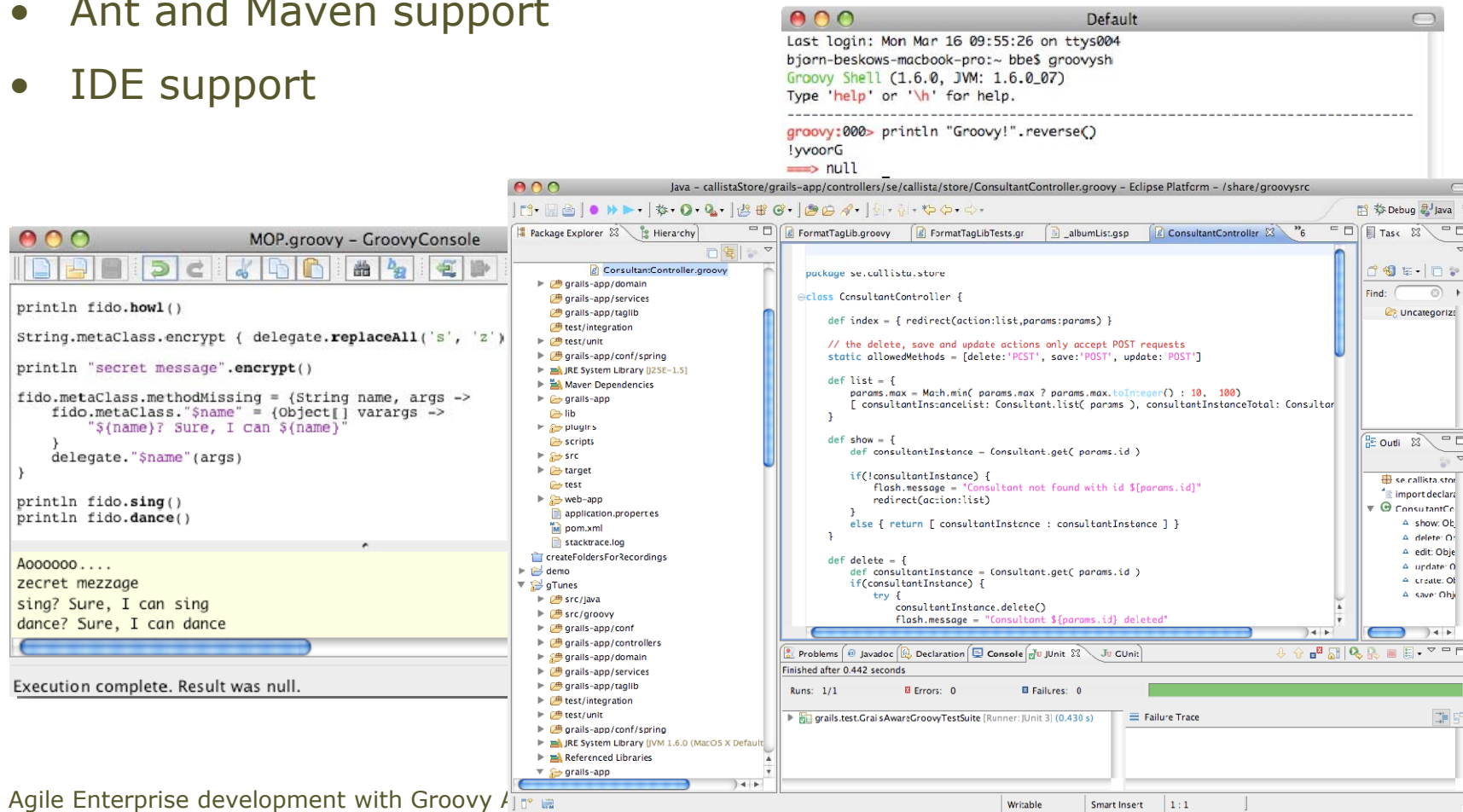
# Groovy / Java Integration



- Same Type System
  - Same Strings, same RegExp, ...
- Same APIs
  - JDK
  - Collections
  - ...
- Same security model
- Same threading model
- ...
- Dynamically compiled or pre-compiled into regular .class files

# Groovy Tools

- Rapid prototyping feedback via groovysh and groovyConsole
- Ant and Maven support
- IDE support



# Syntax basics: A Java program

---

```
public class HelloWorld {  
  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String sayHello() {  
        return "Hello, " + name + "!";  
    }  
  
    public static void main(String[] args) {  
        HelloWorld helloWorld = new HelloWorld();  
        helloWorld.setName("Groovy");  
        System.out.println(helloWorld.sayHello());  
    }  
}
```

# Corresponding Groovy program

---

```
public class HelloWorld {  
  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String sayHello() {  
        return "Hello, " + name + "!";  
    }  
  
    public static void main(String[] args) {  
        HelloWorld helloWorld = new HelloWorld();  
        helloWorld.setName("Groovy");  
        System.out.println(helloWorld.sayHello());  
    }  
}
```


# A more Groovy program

---

```
class HelloWorld {
```

```
String name
```

Getters and setters are generated for properties



```
String sayHello() {  
    "Hello, ${name}!"  
}
```

Groovy Strings performs automatic variable interpolation



```
static void main(args) {  
    def helloWorld = new HelloWorld(name: "Groovy")  
    println helloWorld.sayHello()  
}
```

Named parameters to generated constructor




# An even more Groovy program

---

```
class HelloWorld {  
    def name  
    def sayHello() {  
        "Hello, ${name}!"  
    }  
}
```

“Duck typing” for name property:  
Any object that responds to toString()  
will do



```
println new HelloWorld(name: "Groovy").sayHello()
```

Free groovy “script”



# Efficient native syntax constructs

---

- GStrings

```
"Hello, ${name}!"  
'''This is a  
multi-line string'''
```

- Lists

```
def colors = ["blue", "green", "red"]
```

- Maps

```
def frameworks = [groovy: "Grails", ruby: "Rails"]
```

- Regular expressions

```
def whitespace = ~/\s+?/
```

# Closures

---

- A reusable / assignable block of code delimited by curly braces
  - Can take parameters
  - Can be assigned to a variable
  - Can be passed as parameters to methods, or inlined

```
def printGreeting = { name -> println "Hello ${name}" }
```

```
public Employee getEmployee(final String id){  
    HibernateCallback callback = new HibernateCallback() {  
        public Object doInHibernate(Session session)  
            throws HibernateException, SQLException {  
            return session.load(Employee.class, id);  
        }  
    };  
    return (Employee)hibernateTemplate.execute(callback);  
}
```

# Closures in action

---

- The IO-related API:s in Java have highly coupled methods:

1. create/open resource
2. use resource
3. close resource

- Capture prototypical usage, and provide the essential work as a closure:

```
def sql = Sql.newInstance(url, usr, pwd, driver)
def list = sql.eachRow("select * from USER") { println it.name }
```

- Iterating over collections, performing essential work for each item:

```
List<Orders> orders = ...
orders.each { order -> if (order.valid()) order.dispatch() }
```

# Meta-programming: Meta Object Protocol

---

- Meta-programming:  $\approx$  "programming the program"
  - enables extensions to a program at runtime
- Groovy gives every class or instance a **metaClass** property, which allows you to
  - Add methods and properties
  - Intercept missing methods
- Extremely powerful mechanism, which provides the foundation for DSLs, Builders and for most of the magic in Grails



Demo!

# Groovy Development Kit

---

- Cool! I'd like to have the same expressive power in the standard Java APIs.
- But we can't extend `java.lang.String` or `java.io.File`, can we?
  - Yes we can, using the metaClass of the MOP!
- The Groovy Development Kit (GDK) is a pimped JDK!
  - <http://groovy.codehaus.org/groovy-jdk/>



Demo!

# Groovy unit-testing

---

- When giving up static typing, aggressive testing becomes (even more) crucial
- Unit test framework support built in
  - `GroovyTestCase` (JUnit 3.8.2)
  - JUnit 4
- Superb support for *Mocking*
  - ducktyping and meta-programming give super-powers!
  - Built-in DSLs for creating Mock Objects

**Writing tests won't hurt  
- but skipping it will!**

# What about Groovy performance?

---

Nice dynamic features, but won't that be very slow in runtime... ?

- Yes, its slower. (Be aware: these are micro-benchmarks):  
*<http://shootout.alioth.debian.org/>*
  - Groovy (1.6-beta-1)  $\approx$  JRuby (1.1)  $\approx$  Ruby
  - Java 6  $\approx$  5-100 times faster than Groovy
- That's one reason why we need seamless integration with Java!
- JSR-292 (invokedynamic) for Java SE 7
  - expected to bring performance improvements

# Grails overview

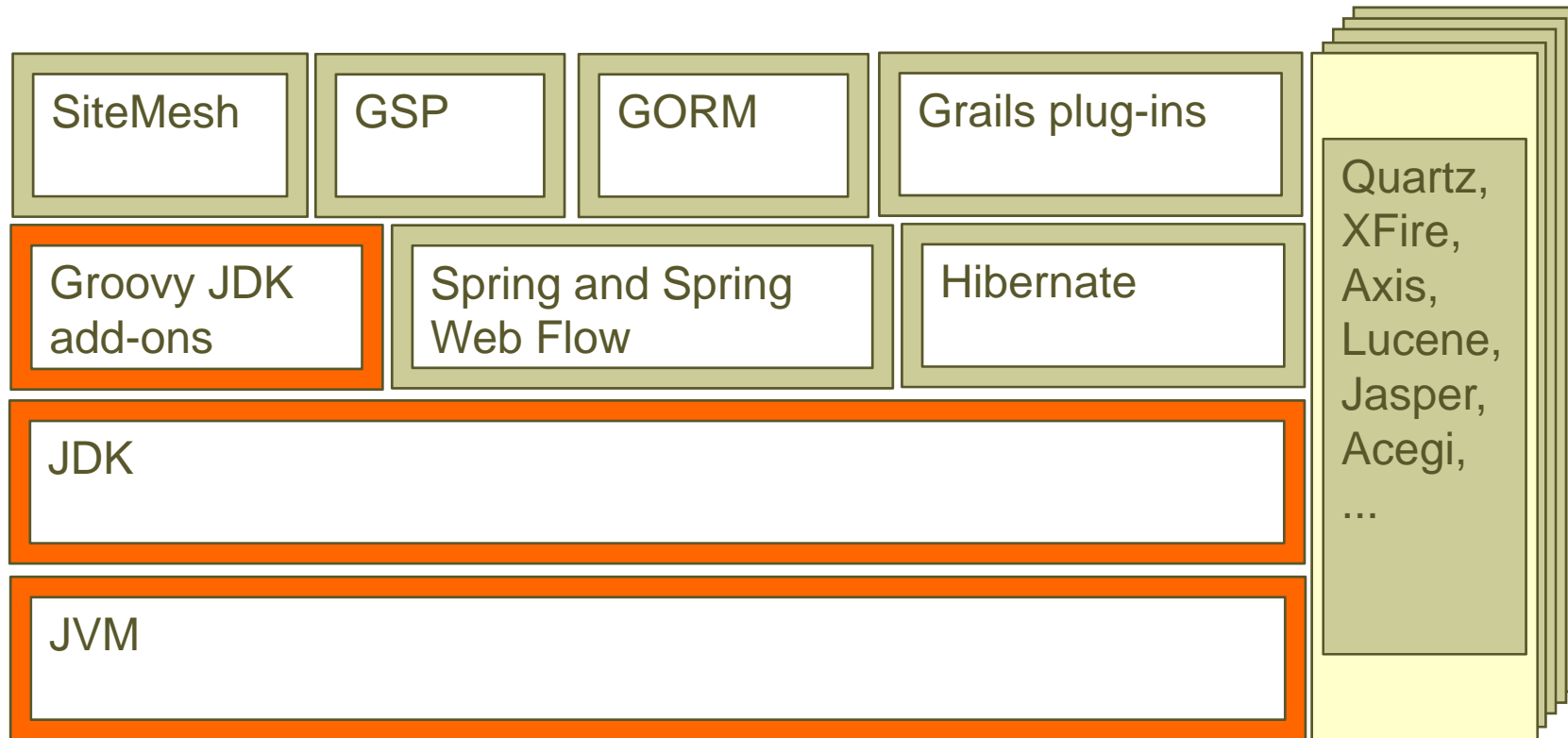


- Full Stack MVC Framework for the Java platform
  - heavily *inspired* by Ruby-on-Rails
- Built on a solid, well-known foundation
  - Java, Spring, Hibernate, SiteMesh
- Convention over configuration
  - Conventions and defaults *everywhere*
  - meta-programming instead of code generation
- Don't Repeat Yourself (DRY) *taken seriously*
- Release 1.1, march 2009 (*started mid 2005*)
- Backed by G2One/SpringSource

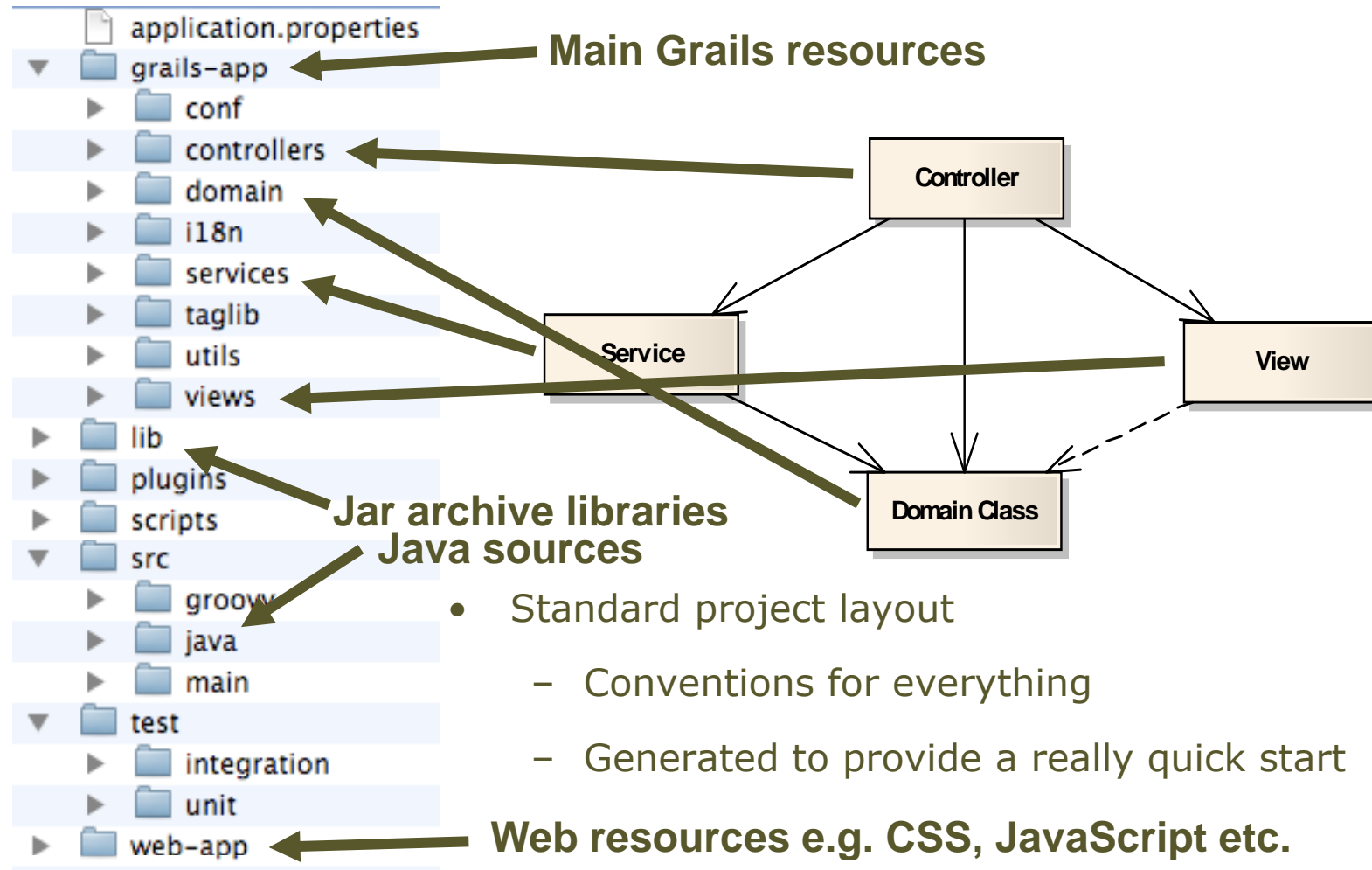
Major diff to Rails:  
*domain-centric* instead  
of *database-centric*

# Major Grails building blocks

---



# MVC framework with additional Service Layer



# Scaffolding

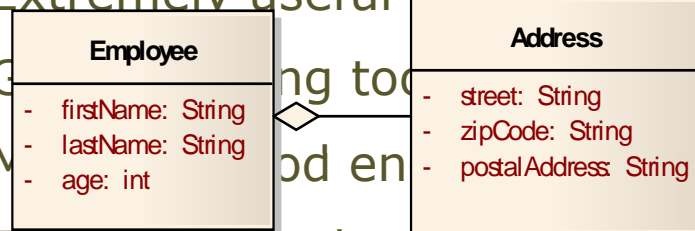
- Straight-forward CRUD controllers and views can be generated, to provide a jump start
  - Dynamic scaffolding, using Groovy Meta-programming magic
  - Static scaffolding
- Don't misinterpret this as a 4GL approach:

– Extremely useful to get quickly up to speed

– Good enough to

– Most of the time

– Typically, used to get a jump-start, then tweaked to fit individual needs



Demo!

– For example, CRUD admin interfaces

# GORM: Object Relational Mapping made simple

- Full Hibernate/JPA power in a concise mapping DSL defaults and conventions
  - id, version added dynamically to domain objects
  - dateCreated and lastUpdated properties, if present, will be automatically timestamped
- Constraints can be stated declaratively
- Dynamic lifecycle and finder methods injected at runtime

```
class Customer {  
    String name  
    String address  
  
    Timestamp lastUpdated  
  
    static hasMany = [bookings:Booking]  
  
    static constraints = {  
        name blank:false  
        address size:1..30  
    }  
}
```

```
class Booking {  
    ...  
}
```

# GORM: Using JPA annotations

---

- JPA Annotations can be used for Domain Objects instead, if preferred

```
@Entity
class Customer {

    String name
    String address

    Timestamp lastUpdated

    @OneToMany(mappedBy="customer")
    Set<Booking> bookings

}
```

Demo!

# Understanding Controllers - conventions at work

http://localhost:8080/callistaStore/**consultant/show/1**

URLs are mapped to controllers, actions and parameters

Actions by default render a model map

Actions are closure properties

Views are closure properties

```
class ConsultantController {  
    def show = {  
        def consultantInstance = Consultant.get( params.id )  
  
        if(!consultantInstance) {  
            flash.message = "Consultant not found with id ${params.id}"  
            redirect(action:list)  
        }  
        else { return [ consultantInstance : consultantInstance ] }  
    }  
}
```

# More conventions: Custom URL Mappings

---

```
class UrlMappings {
    static mappings = {
        "/$controller/$id?" {
            action = [GET: 'show', PUT: 'save',
                    POST: 'update', DELETE: 'delete']
        }
        ...
    }
}
```

# GSP: JSP done right

---

- A view technique similar to JSP, but utilizing the awesome power of Groovy
  - Without all JSP annoyances, GSP is *much* more expressive and concise
- Encourages a clean decoupling between presentation and application logic
- Eliminates redundancy and repetition using Layouts, Templates and Tags

```
<html>
  <body id="body">
    <h2>Album: ${album.title}</h2>
    <g:render template="album" model="[album:album, artist:album.artist]"></g:render>
    <div style="margin-top:10px">
      <g:link controller="store" action="shop">Back to Store</g:link>
    </div>
  </body>
</html>
```

**Embedded GString, referencing the model returned by the controller**

**Split composite page into parts using templates**

# GSP: Layouts

- Layout support through seamless integration with SiteMesh
- Effectively separate out the components of a page
- Sensible conventions for layouts

layouts/main.gsp:

```
<html>
<h
```

view/consultant/main.gsp:

```
<html>
<head>
  <meta http-equiv="Content-type" content="text/html; charset=UTF-8" />
  <meta name="layout" content="main" />
</head>
<body>
  <div class="nav">
    ...
  </div>
  ...
</body>
```

Apply layout by convention, or explicitly

Definition of layed out elements (placeholders)

Fill layout elements

# GSP Templates: Remove Repetition!

---

- Templates (a.k.a. "partials") are small, reusable blocks of mark-up
- Brings structural decomposition to views, allowing you to **Don't Repeat Yourself**
- Can be rendered directly from controllers
  - `render(template:"assignments")`
- Or from GSP pages
  - `<g:render template="assignments" />`

view/consultant/\_assignments.gsp:

```
<ul>
  <g:each in="{assignments?}" var="assignment">
    <li>${assignment?.name}</li>
  </g:each>
</ul>
```

# GSP Tags: Taglibs done right

- Tags provide an excellent mechanism for separation logic and presentation
- Tags may be invoked as methods, making them extremely convenient, and trivial to test
- Grails comes with a long list of highly useful tags:
  - Logical, iterative, links, forms, Ajax, Validation, UI Widgets, ...
- Trivial to create Custom Tags

**taglibs/FormatTagLib.groovy:**

```
class FormatTagLibTests extends TagLibUnitTestCase {  
    class {  
        void testDateFormat() {  
            def date = new GregorianCalendar(1999, 0, 1).getTime()  
            tagLib.dateFormat(format: "yyyy/MM/dd", date: date)  
            assertEquals '1999/01/01', tagLib.out.toString()  
        }  
    }  
}
```

**Call the tag as a regular method**

# Grails Services & Spring Dependency Injection

- Services are typically used for orchestration logic
  - Transactional by default, using Spring's PlatformTransactionManager
- Can be automatically injected into controllers (and into other services)

```
class BookingService {  
    def transactional = true  
  
    def createBooking(bookingDetails) {  
        ...  
    }  
}
```

**Automatic dependency injection**

```
class BookingController {  
    def bookingService  
  
    def create = {  
        bookingService.createBooking(  
            params.booking)  
        }  
    }  
}
```

# Spring Configuration DSL

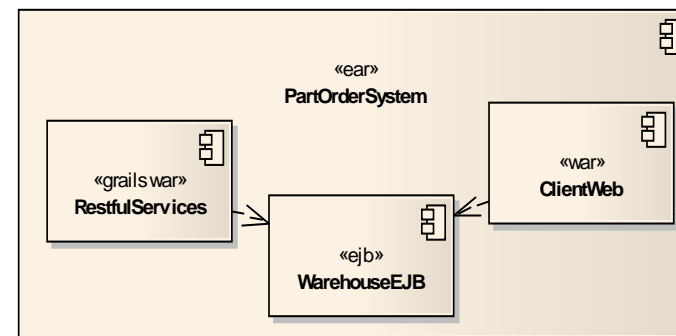
---

- Dependency Injection is available for all Grails components, both as **sources** and **targets** of injection
  - Allows for seamless and transparent integration
- Grails provides a Domain-Specific Language for Spring configuration, to simplify integration further
  - `conf/spring/resources.groovy`

```
beans = {  
    warehouseGateway(org.springframework.LocalStatelessSessionProxyFactoryBean) {  
        jndiName="ejb/WarehouseGateway"  
        businessInterface="se.callista.store.WarehouseGateway"  
    }  
}
```

# Deploying Grails applications

- Grails provides a built-in Jetty server for use in development mode.
- A Grails project is in essence a Web app, and can be packaged into a standard WAR file using the `grails war` command
- As such, it can
  - be deployed standalone to any Servlet container (Tomcat, Jetty, ...) or J2EE/JavaEE server, or
  - be bundled with other Web and/or Ejb modules into an EAR application and deployed to any J2EE or JavaEE server



# But wait! There is more!

---

- What we haven't covered:
  - Grails Web Flow
  - Grails Ajax Support
  - Grails Plugins
  - SOAP support
  - Internationalization
  - GANT and GMaven
  - Grails Bootstrapping
  - Grails Security
  - ...
- Turn to [www.grails.org](http://www.grails.org) for more details



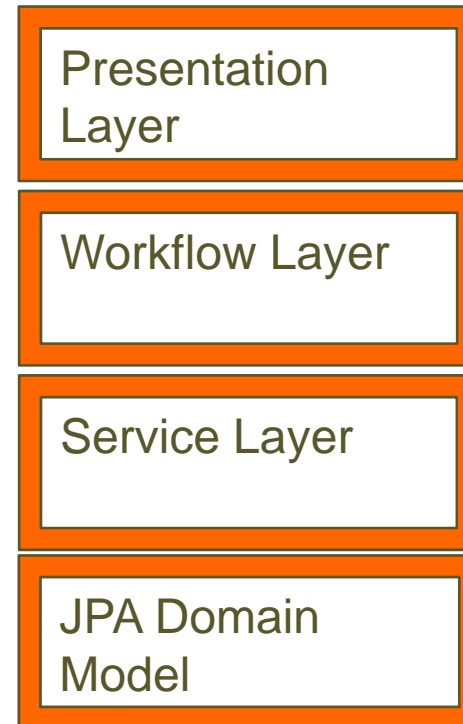
# Integration Scenarios & Migration Paths

---

- ✓ Turbo-charge existing unit and integration tests
- ✓ Grails CRUD application on top of existing JPA domain model
- ✓ Grails CRUD application on top of existing Service Layer
- ✓ Existing application on top of (partial) Grails Service Layer
- ✓ Grails RESTful web service on top of existing domain model or existing service layer
- ✓ Quartz integration with existing application
- ✓ Grails Web 2.0 functionality as part of existing application

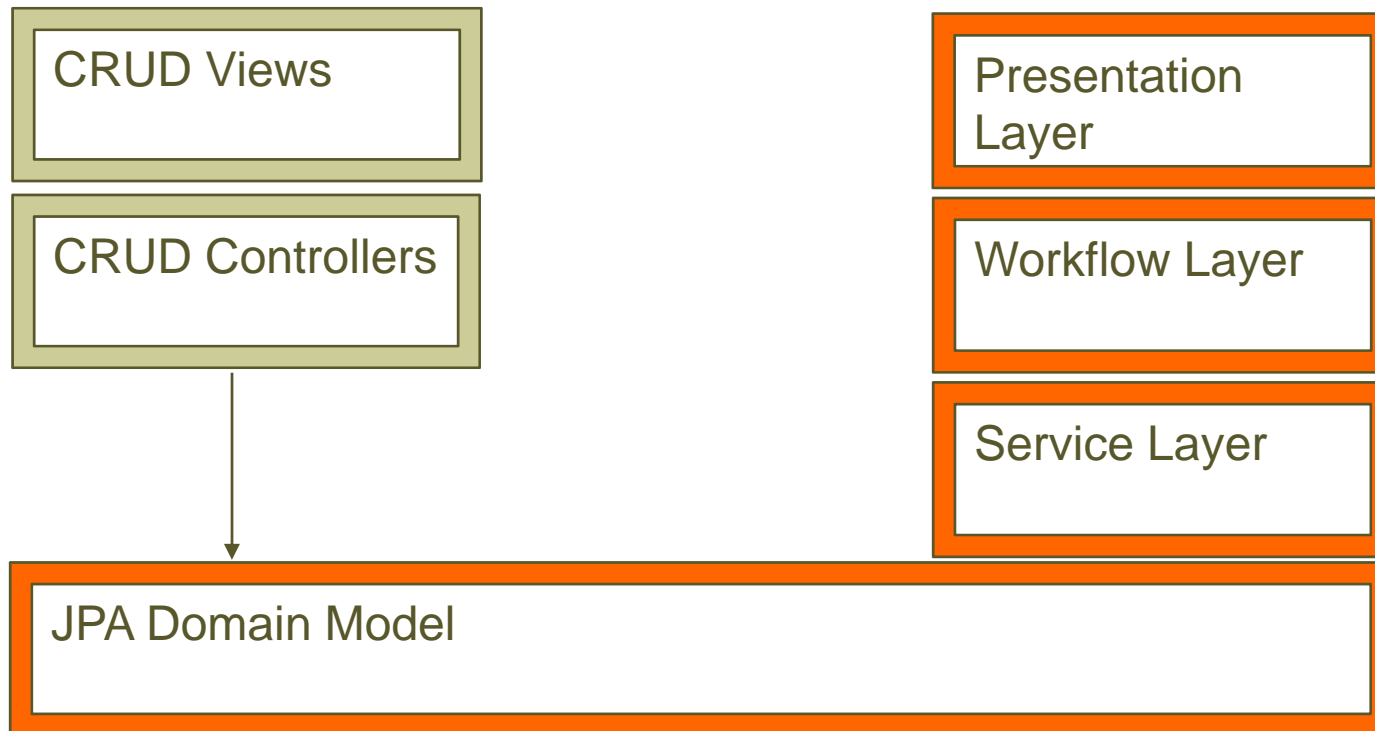
# Grails CRUD application on top of existing JPA domain model

---



# Grails CRUD application on top of existing JPA domain model (contd)

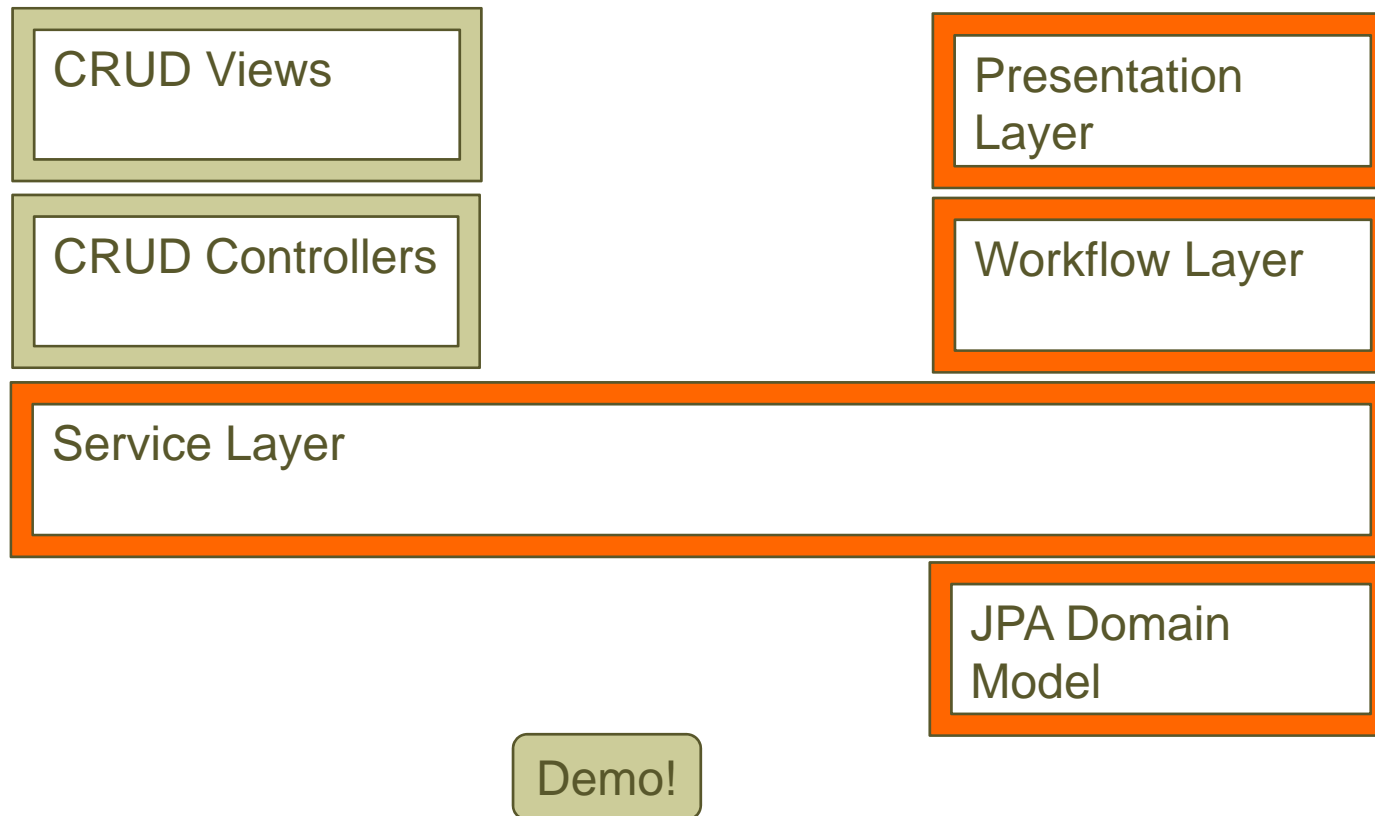
---



Demo!

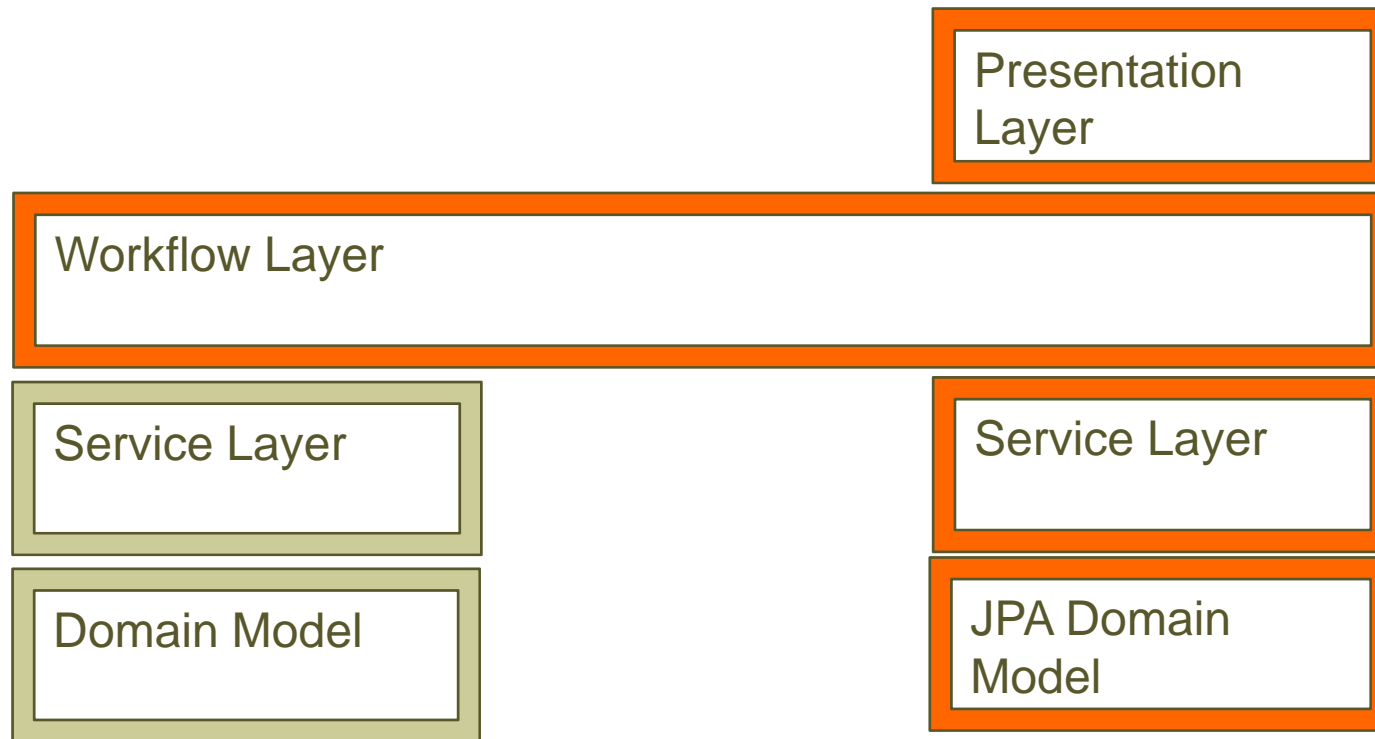
# Grails CRUD application on top of existing Service Layer

---



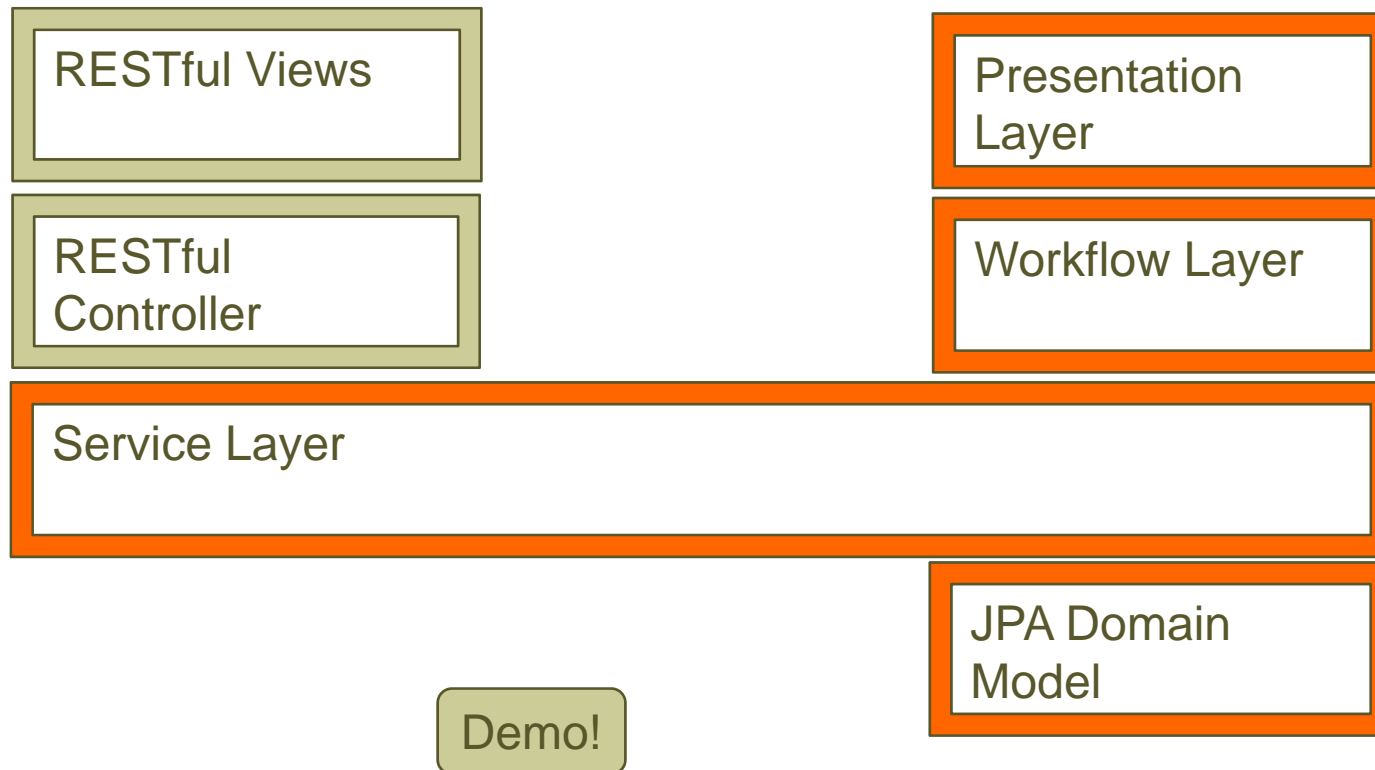
# Existing application on top of (partial) Grails Service Layer

---



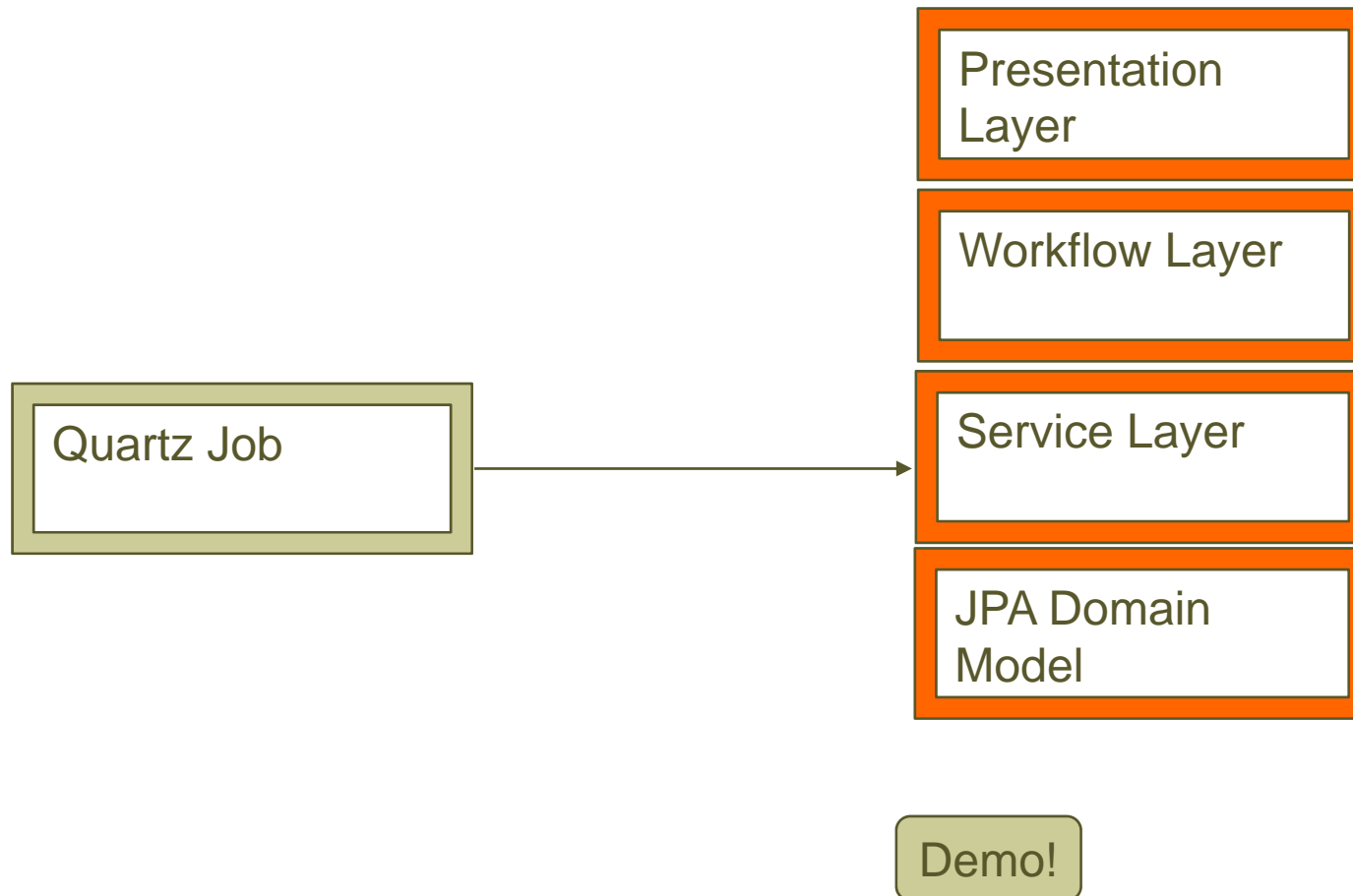
# Grails RESTful web service on top of existing domain model or existing service layer

---



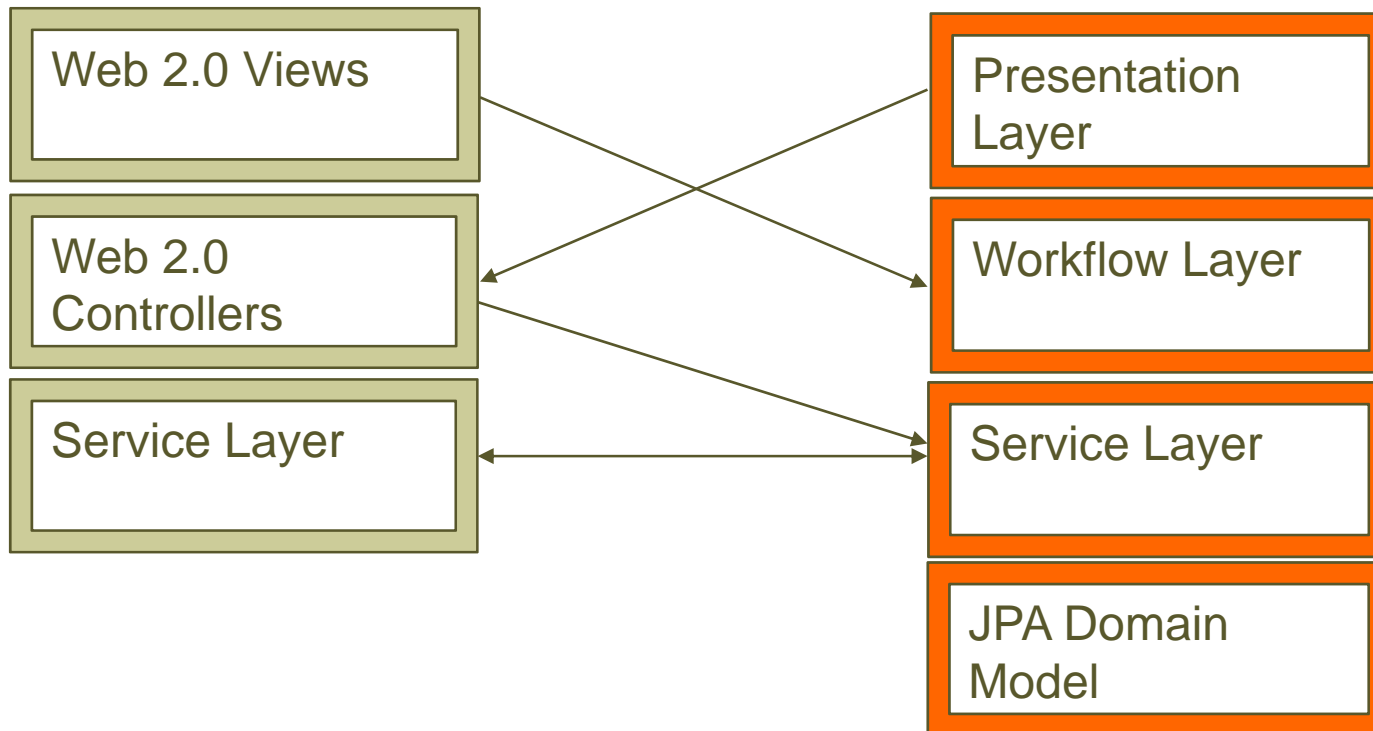
# Quartz integration with existing application

---



# Grails Web 2.0 functionality as part of existing application

---



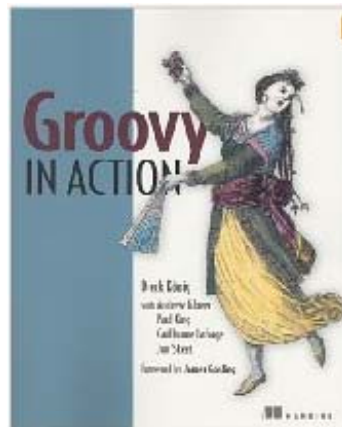
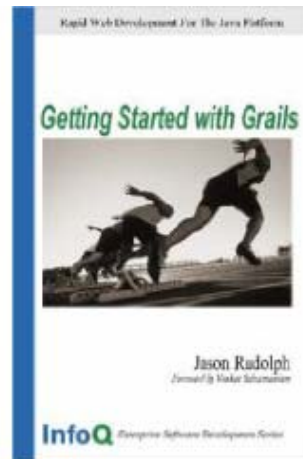
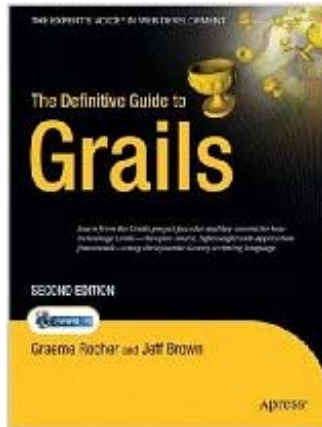
# Conclusions

---

- Agility is key, and current Java frameworks have problems catching up
  - Dynamic frameworks like Grails, Ruby on Rails and Django provides a **dramatic productivity gain**
- Several of these frameworks (besides Grails) may however compromise your current investment in JavaEE
  - For *some parts* of *large-scale, critical* enterprise applications, static typing and IDE support will still be crucial
- Groovy and Grails, through its full and seamless integration with the JavaEE platform finds the sweet spot in a *blended approach*:
  - **Use and build upon what you already know and have, just do it at bit faster and better!**



# Further reading



- The Definite Guide to Grails, Grahame Rocher & Jeff Brown, ISBN 1590599950
- Getting Started with Grails, Jason Rudolph, ISBN 143030782X, free downloadable version at InfoQ
- Groovy in Action, Dierk Koenig *et.al.*, ISBN 1932394842
- [www.grails.org](http://www.grails.org)

# Callista blir Sverige-partner för G2One

---

## Callista blir Sverige-partner för G2One

[Skriv ut](#) 

Som ett led i vår satsning på agil utveckling för Java-plattformen, har Callista ingått samarbetsavtal med [G2One](#). Vi är "G2One training and consulting partner in Sweden".



Förutom att erbjuda kompetens inom agil utveckling med Groovy och Grails, arrangerar vi kurser i Göteborg, Stockholm och Malmö. Kurserna hålls företrädesvis av lärare från G2One - företaget bakom Groovy och Grails.

Information och anmälningsskylt för våra öppna Grailskurser hittar du [här](#).

<http://www.callistaenterprise.se/nyheter/nyhetsarkiv/callistablirsverigepartnerforg2one>

# Questions

---

