Learn how to become more productive developing Web applications, and how to get back the fun of programming in a powerful & dynamic language

Who am I?

Guillaume Laforge

- **Groovy Project Manager** at Codehaus
  JSR-241 Spec Lead
  Initiator of the Grails project

- Software Architect at **OCTO Technology**,
  a French consulting company focusing on
  Software architecture and Agile methodologies

http://www.octo.com

Agenda

- What's **Grails**?
- Getting started
- MVC
  - **Model**: transparent Hibernate persistence
  - **View**: GSP, layout with SiteMesh, dynamic tag libs
  - **Controller**
  - And: Services, Jobs, AJAX…
- Sweet spot: enterprise-readiness
- Roadmap
- Further reading

What's Grails?

A modern MVC Web Framework

- Grails is an MVC Web framework
  - Initially inspired by **Ruby on Rails**
  - Built upon solid bricks & best of breed components
    - **Spring**: IoC, DI, Spring MVC, transactional support, experimental Spring WebFlow…
    - **Hibernate**: ORM, querying mechanism…
    - **Groovy**: for focusing on everything that matters
    - And: SiteMesh, Quarz, AJAX frameworks…
  - « **Convention over configuration** »:
    Focus **not** on wiring and configuration!

Cabinet d'Architectes en Systèmes d'Information

Getting started

Downloading & Installing Grails

- Go to http://grails.org
- Download the latest release or nightly
- Extract the archive
- Set the GRAILS_HOME environment variable
- Add $GRAILS_HOME/bin to your path
- *(You don't need to install Groovy, but it won't hurt)*

- You can type « grails » on the command-line, and you're ready to go!

Let's build something together?

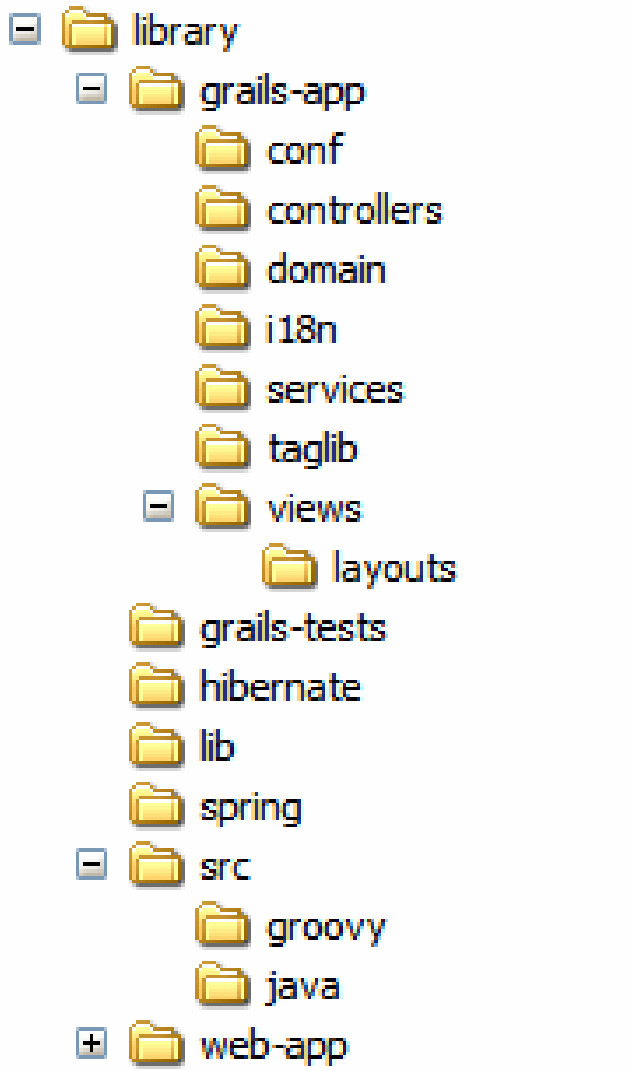
- Instead of hundreds of slides…
  let's create a small app together!
- We'll learn both Groovy and Grails
  as we build the app

- We're going to create a simple library application:

  books, authors, publishers…

Create the application skeleton

- > grails create-app



Let's run the app

- What? We haven't written a single line of code?
  > grails run-app *(**Jetty** under the hood)*

Cabinet d'Architectes en Systèmes d'Information

Domain Modeling with GORM

MVC: M like Model

- Now that the skeleton is there, focus on the model
- Model backed by **GORM**
- The model is just a set **POGO**s
  (Plain Old Groovy Objects)

- First class of our domain: the books

```
class Book {
    String title
    String author
    String publisher
}
```

Automatic injection of methods

- Automatically, Grails add dynamic instance & static methods to all your domain classes:

Static methods:

- Book.get(1)
- Book.find()
- Book.findAll()

Instance methods:

- book.save()
- book.validate()
- book.update()
- book.delete()

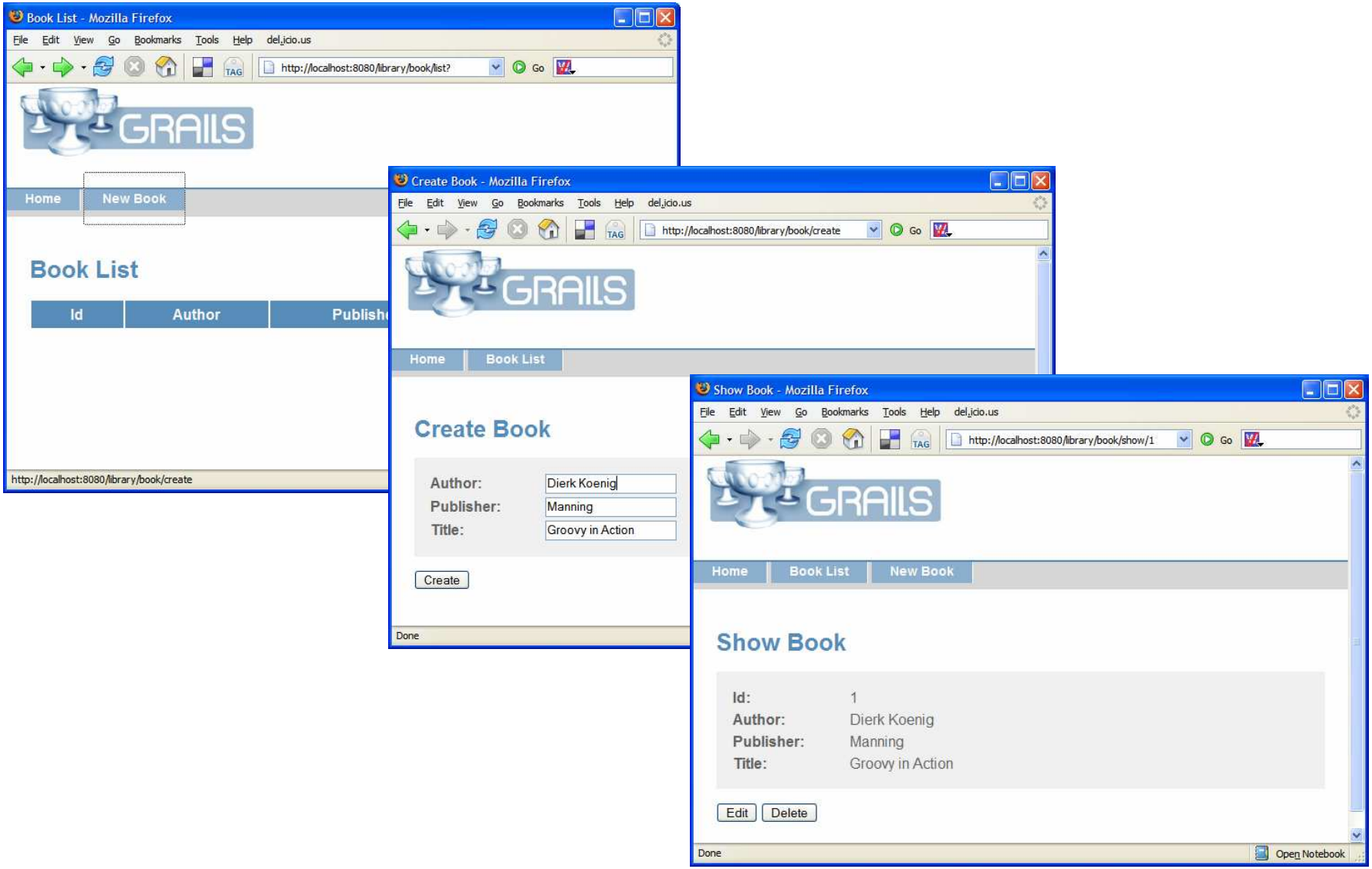
Static scaffolding

- Once the Book class is created,
  let's benefit from static scaffolding

  > grails generate-all
  > grails run-app

- A BookController is created, as well as four GSP:
  - create.gsp
  - edit.gsp
  - list.gsp
  - show.gsp

CRUD controllers & views



Back to the Model: GORM

- Hibernate is the de facto ORM solution
- Domain classes are automatically and **transparently mapped** with Hibernate
- 1:1, 1:n & m:n relationships supported

- Various database environments (dev, test, prod)
- Default HSQLDB in-memory config

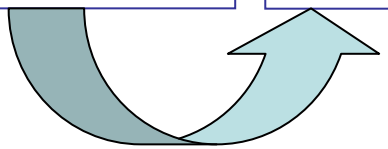
- You can even provide **your own Hibernate mapping files for legacy schemas**
- You can also **reuse your EJB3**!

After books: authors & publishers

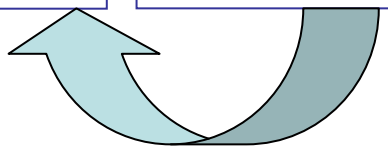
```
class Author {
  String firstName
  String lastName

  def hasMany =
    [books: Book]

  String toString(){
    "$firstName
     $lastName"
  }
}
```

```
class Book {
  String title
  Author author
  Publisher publisher

  def belongsTo =
    [Publisher, Author]

  String toString(){
    title
  }
}
```

```
class Publisher {
  String name

  def hasMany =
   [books: Book]

  String toString(){
    name
  }
}
```



Author **has many** Books



Publisher **has many** Books

Adding constraints to your model

Add an email to Author

```
class Author {
    String email
    // ...
    static constraints = {
        email(email: true)
    }
}
```

Add an ISBN to Book

```
class Book {
    String isbn
    // ...
    static constraints = {
        isbn(matches:
        "[0-9]{9}[0-9X]")
    }
}
```

Constraints available

- Many constraints available:

  blank, creditcard, email, **inList**, length, min, minLength, minSize, **matches**, max, maxLengths, maxSize, notEqual, **nullable**, **range**, size, **unique**, **url**, validator

- And you can create your own closure validator:

```
even( validator: { it % 2 == 0 } )
```

Querying your model

- It’s time to sentence DAO to death! ☺
- Grails provides various querying mechanisms:
  - Dynamic finder methods
  - Query by example
  - Criteria builders
  - Full-blown HQL queries

Dynamic finder methods

- `Book.findByTitle("The Stand")`

```
Book.findByTitleLike("Harry Pot%")
Book.findByReleaseDateBetween(start, end)
Book.findByTitleLikeOrReleaseDateLessThan(
          "%Grails%", someDate)
```

- Find by relationship

```
Book.findAllByAuthor( Author.get(1) )
```

- Affect sorting

```
Book.findAllByAuthor(me, [sort:'title',order:'asc'])
```

Query by example

- Use the find() method and pass it an example instance:

```
Book.find( new Book(title:'The Shining') )
```

Criteria builder

- A builder is an arbitrary tree structure used for DSL

```
def c = Account.createCriteria()
def results = c {
    like("holderFirstName", "Fred%")
    and {
        between("balance", 500, 1000)
        eq("branch", "London")
    }
    maxResults(10)
    order("holderLastName", "desc")
}.list()
```

HQL queries

- When dynamic finders, query by example or criteria builders don’t cut it

```
Book.find(
   "from Book as b where b.title like 'Lord of%'")

Book.find("from Book as b where b.title like ?",
                    ["The Shi%"])
```

Cabinet d'Architectes en Systèmes d'Information

Controllers, Services & Jobs

BookController, an excerpt

```
class BookController {
  def index = { redirect(action:list,params:params) }

  def list = { [ bookList: Book.list( params ) ] }

  def show = { [ book : Book.get( params.id ) ] }

  def edit = {
    def book = Book.get( params.id )
    if(!book) {
      flash.message = "Book ${params.id} not found"
        redirect(action:list)
    } else return [ book : book ]
  }
}
```

Controllers

- URL mapping convention: controller/action/id
  http://localhost:8080/library/**book**/**show**/**1**

- Scaffolding can be
  - dynamic (def scaffold = true)
  - static (code generation)
- Controllers pass data to the view through maps
- Direct access to parameters
- Easy redirect and forward
- Can define allowed methods for each action

Unit & Functional Testing

- Empty test cases are created for each controller
- Functional tests with **Canoo Web Test**

- You can run your tests on the command-line:

  > grail test-app

  > grails run-webtest

  > grails generate-webtest (for a new Canoo web test)

Services

- Services are Groovy classes that should contain your **business logic**

- **Automatic injection of services** in controllers & services simply by declaring a field:

```
class BookController {
      MySuperService mySuperService
}
```

Scheduling Jobs

- You can create recuring events with Quartz under the hood, configured by Spring
- Again a convention on name and directory
- Regular intervals, or cron definitions

```
class MyJob {
    def cronExpression = "0 0 24 * * ?"
    def execute() {
        print "Job run!"
    }
}
```

Cabinet d'Architectes en Systèmes d'Information

Views: Groovy Server Pages, Dynamic Taglibs and AJAX

Views

- **Spring MVC** under the hood
- Support for flash scope between requests
- **GSP**: Groovy alternative to JSP
- **Dynamic taglib** development: no TLD, no configuration, just conventions
- **Adaptive AJAX tags** (Yahoo, Dojo, Prototype)
- Customizable **layout with SiteMesh**
- Page fragments through reusable templates
- Views under grails-app/views

GSP: Groovy Server Pages

```
<html>
    <head>
        <meta name="layout" content="main" />
        <title>Book List</title>
    </head>
    <body>
        <a href="${createLinkTo(dir:'')}">Home</a>
        <g:link action="create">New Book</g:link>
        <g:if test="${flash.message}">
            ${flash.message}
        </g:if>
        <g:each in="${bookList}">${it.title}</g:each>
    </body>
</html>
```

Rich set of Dynamic Taglibs

- **Logical**: if, else, elseif
- **Iterative**: while, each, collect, findAll…
- **Linking**: link, createLink, createLinkTo
- **Ajax**: remoteFunction, remoteLink, formRemote, submitToRemote…
- **Form**: form, select, currencySelect, localeSelect, datePicker, checkBox…
- **Rendering**: render*, layout*, paginate…
- **Validation**: eachError, hasError, message
- **UI**: richTextEditor…

Write your own taglib!

- Yet another Grails convention:

```
class MyTagLib {
    def isAdmin = { attrs, body ->
        def user = attrs['user']
        if(user != null && checkUserPrivs(user))
            body()
    }
}
```

- Use it in your GSP:

```
<g:isAdmin user="${myUser}">
     some restricted content
</g:isAdmin>
```

Cabinet d'Architectes en Systèmes d'Information

Enterprise readiness

Protect your investment!

- **Reuse**
  - Existing Java libraries
  - **Employee skills & knowledge**
  - **Spring** configured beans
  - **Hibernate** mappings for legacy schemas (but still benefit from dynamic finders)
  - **EJB3** annotated mapped beans
  - **JSPs**, taglibs for the view
- Deploy on your pricey Java app-server & database
- **Grails will fit in your JEE enterprise architecture!**

Cabinet d'Architectes en Systèmes d'Information

Roadmap

Still more to come!

- DSL for scripting Spring
- Plugin architecture for extending Grails
- Upgrade to Spring 2.0
- Flow / conversation concept with Spring WebFlow
- JPA Support
- Web Services convention
- Customized URL mapping
- JSP custom taglib support
- Finer-grained DB schema control

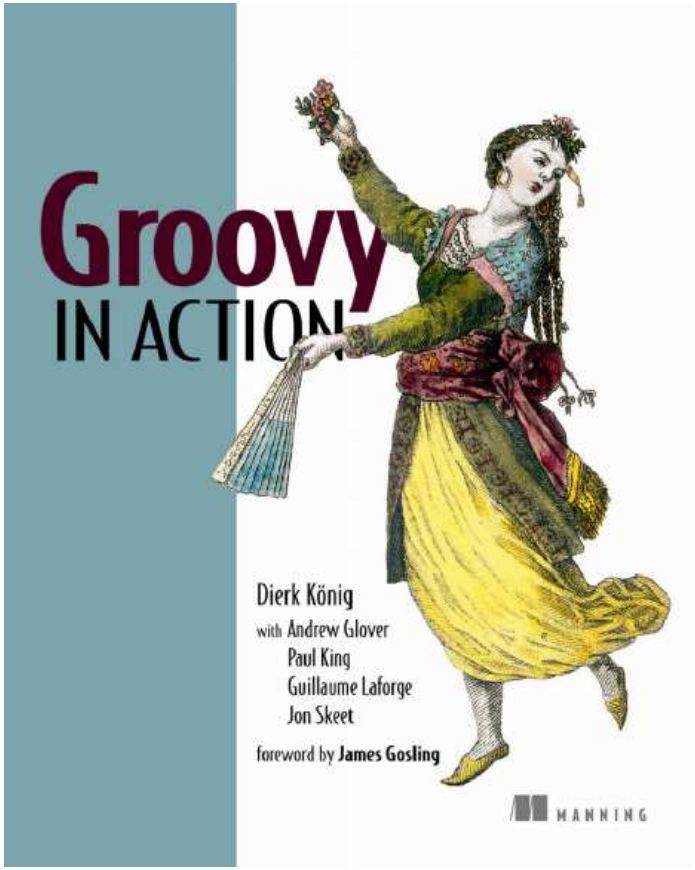
- **It’s up to you to vote for and prioritize new features ☺**

Further reading

Books on Groovy & Grails

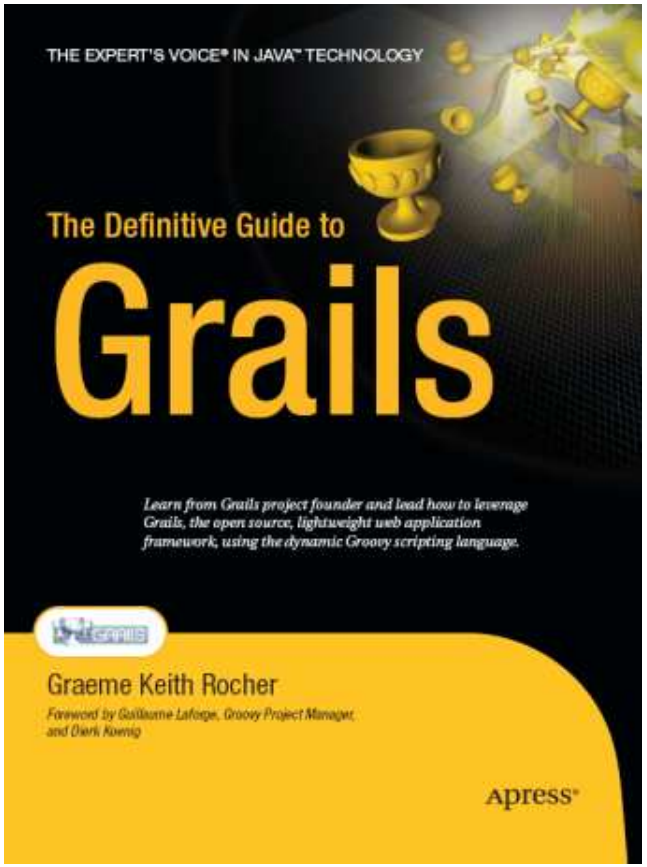
Groovy in Action

Manning



The Definitive Guide to Grails

Apress



Cabinet d'Architectes en Systèmes d'Information

Q & A