Naked Objects and Groovy Grails

Domain driven web applications made simple

IIMS Seminar March 21st 2012

Associate Professor David Parsons

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Naked Objects

- The ‘Naked Objects’ pattern was originally described by Richard Pawson in his PhD thesis
  - Based on earlier work on ‘Expressive Systems’
- 3 principles:
  1. Business logic should be encapsulated in domain objects
  2. The user interface should be a direct representation of the domain objects
  3. The user interface should be automatically generated from the definition of the domain objects

Benefits

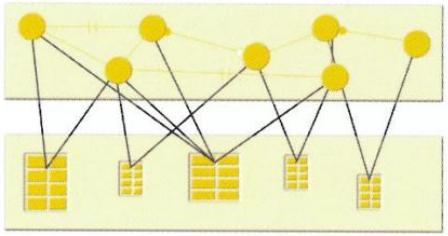
- A faster development cycle
  - There are fewer layers of code to develop
- Greater agility
  - Easier to accommodate future changes in business requirements
- A more empowering style of user interface
  - Direct interaction with the domain
- Easier requirements analysis
  - Common domain understanding

- “an expressive system has a ‘make it so’ button”
  - – Richard Pawson

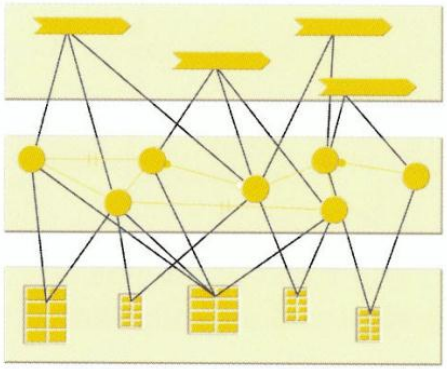
Same old claims?

- As 1960s high level languages?
- As 1970s rapid application development?
- As 1980s 4GLs?
- As 1990s visual programming?
- As 2000s web frameworks?

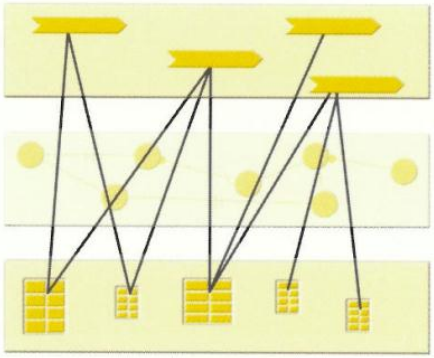
Architecture Layers



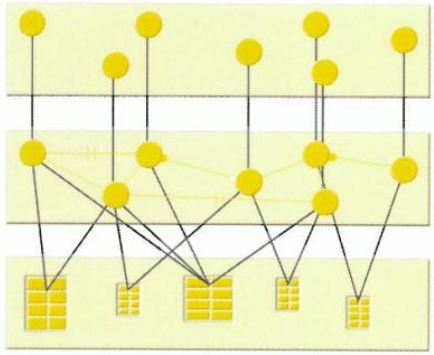
In many object-based systems, the object model must be translated into a relational database representation for storage.



The object model is also translated into a series of user tasks or business processes, implemented using a client-side development tool.



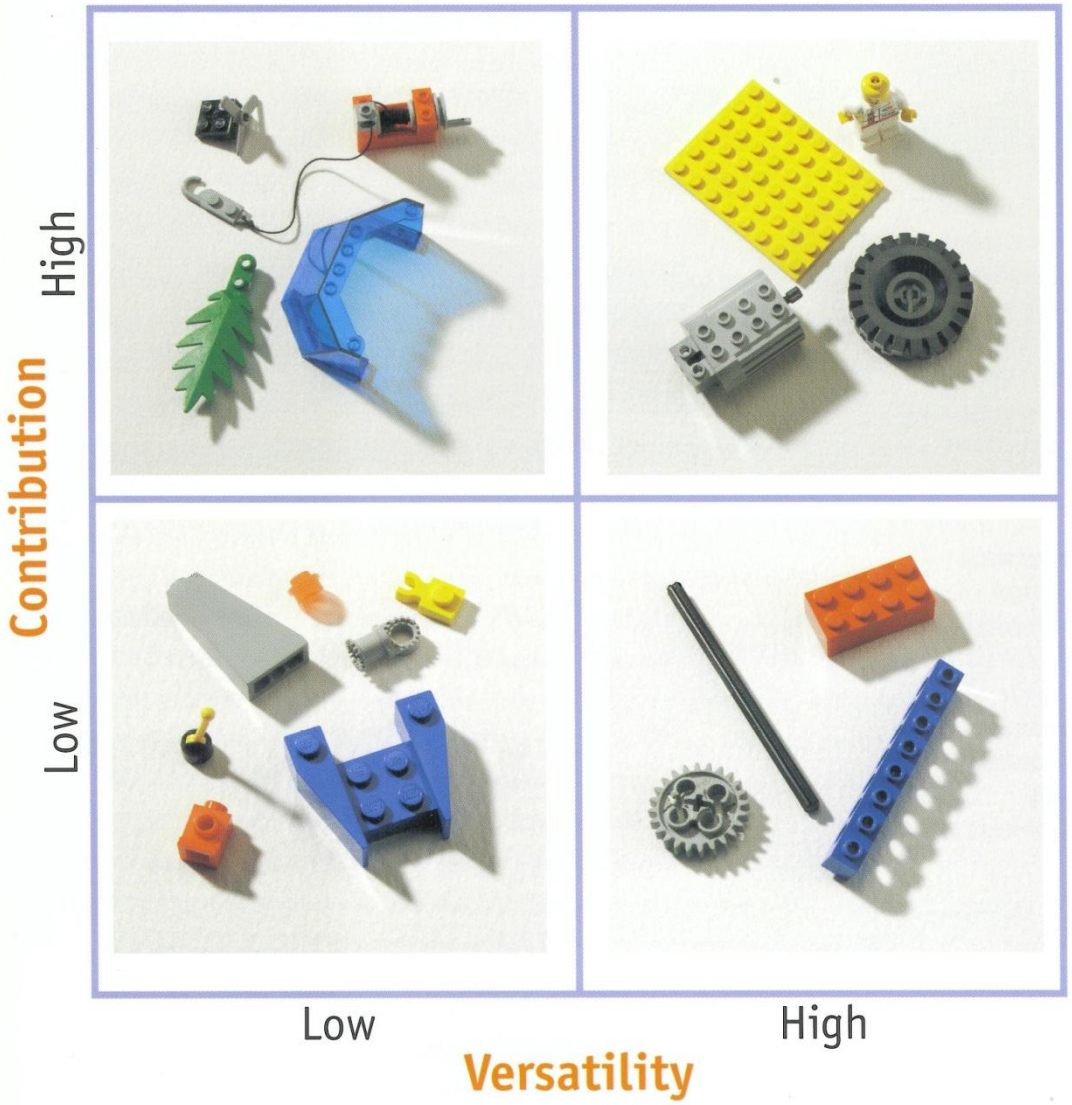
Programmers are then frequently tempted to connect the user interface scripts directly to the relational database, bypassing the business object model.



Keeping a one-to-one mapping between the user constructs and the core business objects reduces the risk of bypassing, and maintains the value of the objects.

Lego

- Isn't what it used to be…



Lego pieces can be classified according to their versatility and average contribution to a model. The same is true of software components or objects. In expressive system design, the aim is to identify the handful of objects in the top right-hand corner and expose these directly to the user.

CRUD Apps

- A whole bunch of web apps are basically just create/read/update/delete (CRUD)
  - Facebook
  - Twitter
  - On-line banking
  - Google Docs
  - YouTube
  - Flickr
  - …
- I could go on, you get the point…

Repetition in CRUD Web Apps

- A simple CRUD application involves four views for each domain concept
  - Creating
  - Editing (update/delete)
  - Listing many items (read)
  - Showing single item details (read)
- n entities = 4*n pages
- These views are all very much the same apart from the fields being exposed
- Useful if a framework can build these views automatically

So, anyway

- Over the last decade or so, the naked objects pattern has appeared in a number of tools
- Grails (a Groovy web framework) is one of these….

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Groovy

- A dynamic language, compiled to Java bytecode, to run on a Java virtual machine
- Uses a Java-like syntax
- Interoperates with other Java code and libraries
- Most Java code is also syntactically valid Groovy
- Groovy implicitly generates data access operations on domain objects

Grails

- Grails (formerly Groovy on Rails) is an open source web framework built on Spring using the Groovy language
  - Spring is a Java web framework that uses other frameworks
- Grails takes its architectural style from Ruby on Rails
- You can also find this style in other tools such as Scala Lift

Full Stack

- Grails includes everything you need
- Integrates several common libraries and frameworks
- Developers can focus on business logic rather than integration
- You don't have to manually glue all the different components together

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Grails and Spring

- Grails is based on Spring in order to reuse some core services
- Spring uses dependency Injection, a specific type of *Inversion of Control*
- The framework can inject capabilities into objects that follow certain rules of coding
- Think of it as being like different lightbulbs that have the same fitting

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

DRY: Don't Repeat Yourself

- Execution of repetitive tasks by the framework
- Use of scaffolding
- The framework generates artifacts related to repetitive tasks
  - views and controllers
- Repetitive tasks are gone
- Developer customizes the artifacts

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Zero Configuration

- You don't need configuration files if everything is in place
- Grails stipulates conventions that make configuration files unnecessary
  - e.g. every controller is stored in a specific directory
- Probably the only configuration file you'll need will be the one which is for database access

Demo

- The Boating Lake Management System
  - Captain Bob runs a business hiring out rowing boats on boating lakes
  - He wants a system to help him manage his thriving business
  - Two of the key concepts in the system are Rowing Boat and Lake
  - There is a one to many relationship between them (one lake can have many rowing boats)
- Demo uses the SpringSource Toolsuite for Grails development

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Create the RowingBoat

- This is a domain class
- We add a few properties and a relationship (to one other object)

RowingBoat.groovy

```
package com.water

class RowingBoat {

    static constraints = {
    }
    int seats
    String colour
    int length
    static belongsTo = [aLake:Lake]
}
```

Create the Lake

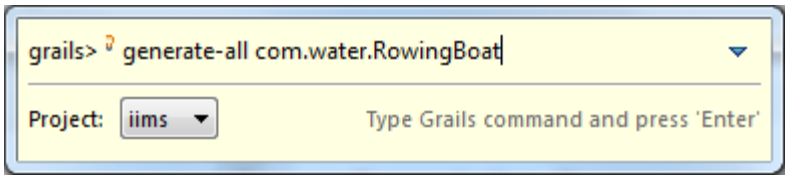
- This is also a domain class
- Again we add a few properties and a relationship (to many other objects)

RowingBoat.groovy | Lake.groovy

```
package com.water

class Lake {

    static constraints = {
    }
    String name
    String location
    static hasMany=[boat:RowingBoat]
}
```

Generate code

- e.g. for the Rowing Boat



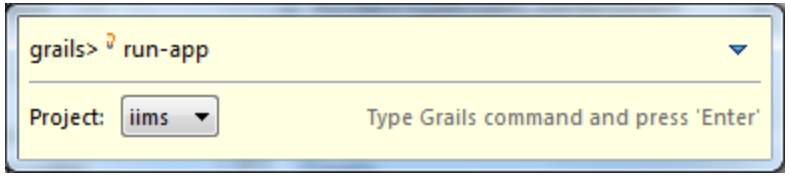
- To make the controller and views dynamic, use a scaffold in the controllers

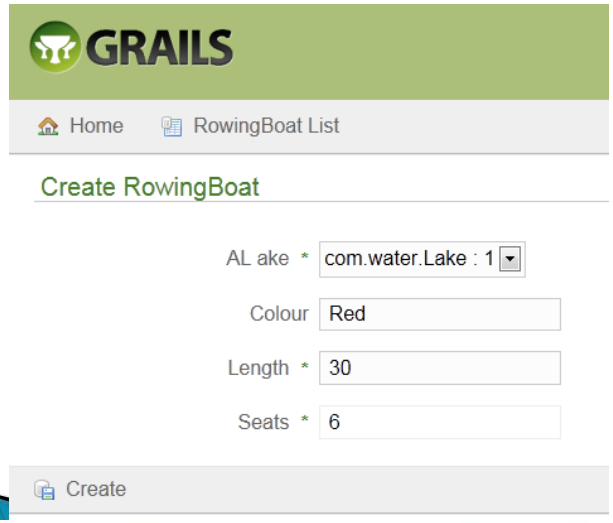
Lake.groovy | RowingBoatController | LakeController.groov

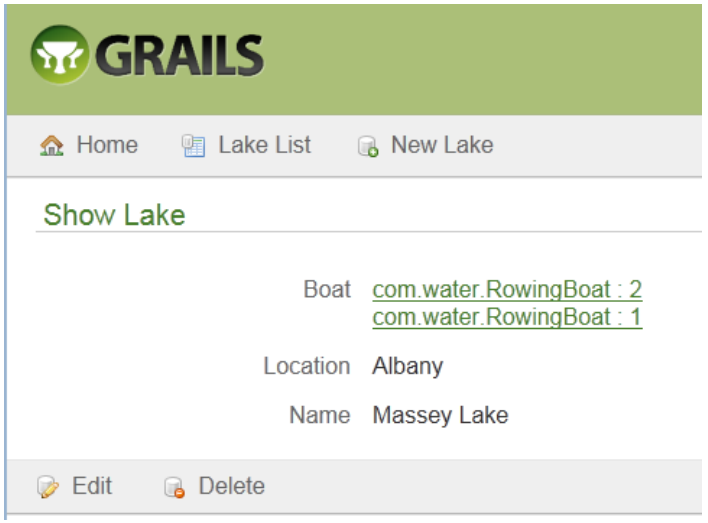
```
package com.water

import org.springframework.dao.DataIntegrityViolationException

class RowingBoatController {
    def scaffold=true
}
```

Make it so!

- Web app to go
  - (‘prod run-app’ would also persist the data in the database)







And so on…

- Easy to add validation, field ordering and visibility, formatting, style sheets etc.
- Most tasks driven from within the domain objects
- For further info see http://grails.org/