# Introduction to Ruby on Rails

Chris Panayiotou

## Ruby on Rails

- Ruby on Rails is a web application framework written in Ruby, a dynamically typed programming language
- The amazing productivity claims of Rails is the current buzz in the web development community

## At First Sight

- Can only be used for web-based, specifically HTML-based, applications
- Designed for small to medium CRUD-based applications
- Cross-platform
- Can use same tools and middleware on Windows, Linux and OS X
- Easy-install packages for Eclipse (with RADRails and Ruby editor plugins), MySQL, Apache, and other Eclipse plugins, eg Subversion

# What is Ruby?

- Ruby is a pure object-oriented programming language with a super clean syntax that makes programming elegant and fun.
    - In Ruby, everything is an object
- Ruby is an interpreted scripting language, just like Perl, Python and PHP
- Ruby successfully combines Smalltalk's conceptual elegance, Python's ease of use and learning and Perl's pragmatism
- Ruby originated in Japan in 1993 by Yukihiro “matz” Matsumoto, and has started to become popular worldwide in the past few years as more English language books and documentation have become available
- Ruby is a metaprogramming language
    - Metaprogramming is a means of writing software programs that write or manipulate other programs thereby making coding faster and more reliable

## What is Rails? Ruby on Rails or just Rails (RoR)

- Rails is an open source Ruby framework for developing database-backed web applications
- Created by David Heinemeier Hansson – DHH Partner, derived from the "37Signals" project
- The Rails framework was extracted from real-world web applications
  - The result is an easy to use and cohesive framework that's rich in functionality, and at the same time it does its best to stay out of your way
- All layers in Rails are built to work together so you Don't Repeat Yourself and can use a single language from top to bottom
- Everything in Rails (templates to control flow to business logic) is written in Ruby
  - Except for configuration files – YAML

## Rails Strengths

- Metaprogramming techniques use programs to write programs
  - Other frameworks use extensive code generation, which gives users a one-time productivity boost but little else, and customization scripts let the user add customization code in only a small number of carefully selected points
  - Metaprogramming replaces these two techniques and eliminates their disadvantages
  - Ruby is one of the best languages for metaprogramming, and Rails uses this capability well
- Scaffolding
  - You often create temporary code in the early stages of development to help get an application up quickly and see how major components work together
  - Rails automatically creates much of the scaffolding you'll need

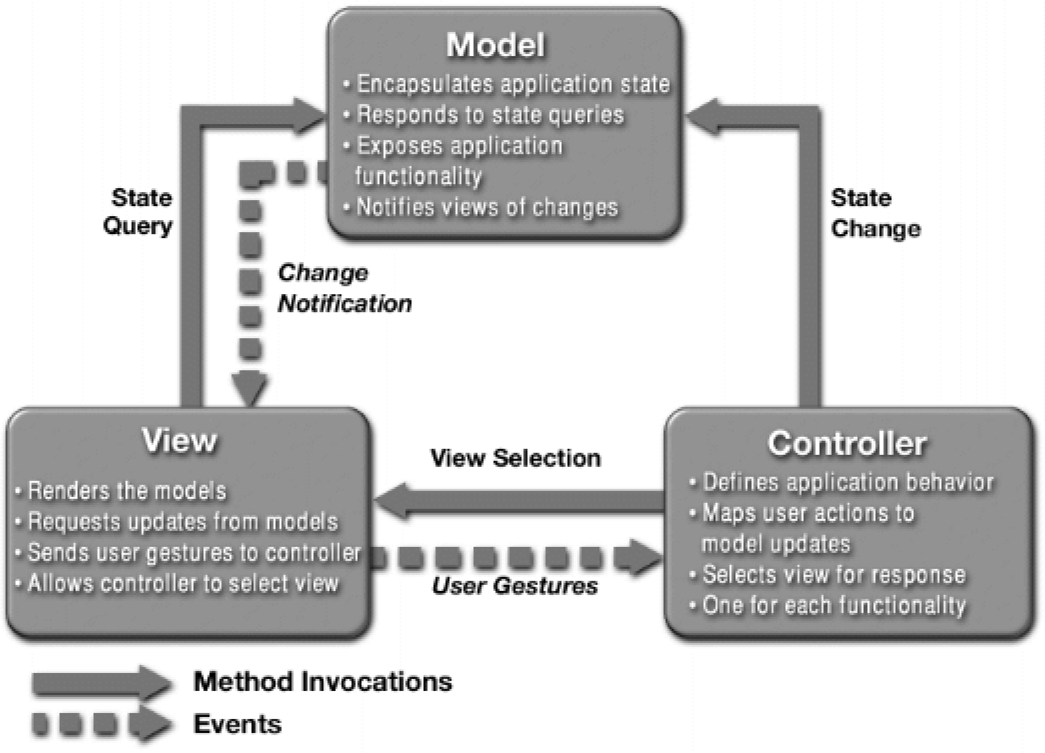
## Rails Strengths

- Convention over configuration
  - Most Web development frameworks for .NET or Java force you to write pages of configuration code
  - If you follow suggested naming conventions, Rails doesn't need much configuration. In fact, you can often cut your total configuration code by a factor of five or more over similar Java frameworks just by following common conventions
    - Naming your data model class with the same name as the corresponding database table
    - ‘id’ as the primary key name
- Rails introduces the Active Record framework, which saves objects to the database
  - The Rails version of Active Record discovers the columns in a database schema and automatically attaches them to your domain objects using metaprogramming
  - This approach to wrapping database tables is simple, elegant, and powerful

# Rails Strengths

- Rails implements the model-view-controller (MVC) architecture.  The MVC design pattern separates the component parts of an application

- Model encapsulates data that the application manipulates, plus domain-specific logic
- View is a rendering of the model into the user interface
- Controller responds to events from the interface and causes actions to be performed on the model.
- *MVC pattern allows rapid change and evolution of the user interface and controller separate from the data model*

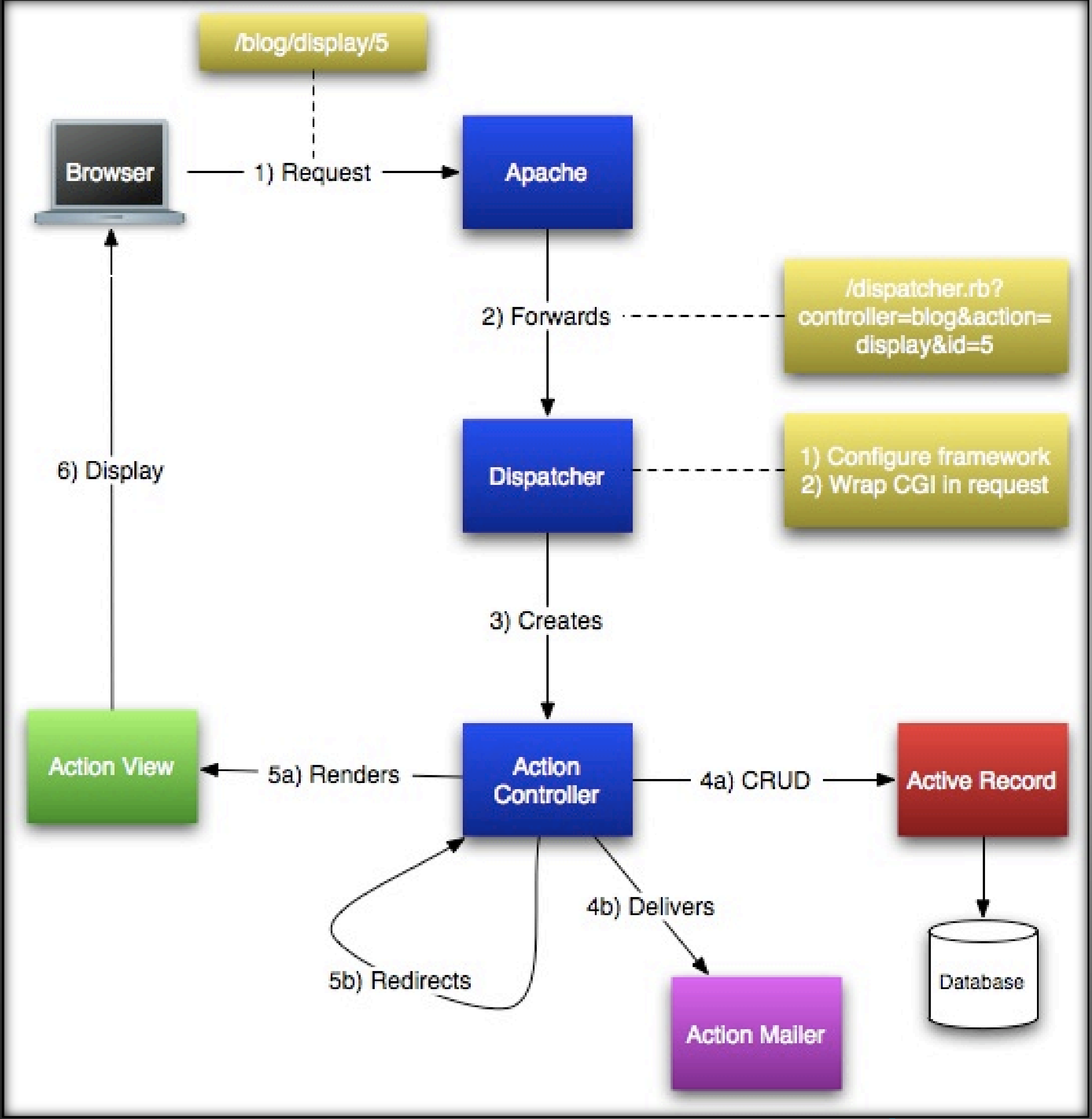

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

## Rails Strengths

- Rails embraces test-driven development.
  - Unit testing: testing individual pieces of code
  - Functional testing: testing how individual pieces of code interact
  - Integration testing: testing the whole system
- Three environments: development, testing, and production
- Database Support: Oracle, DB2, SQL Server, MySQL, PostgreSQL, SQLite

## Disadvantages

- No big corporate backer
- Very few expert Ruby programmers
- Runs slowly (Java ~ 5 times faster but Ruby may be improved by new VM – YARV)
- Poor editor support and very slow debugger
- No clustering, failover
- No two-phase commit
- Does not support compound primary keys
- Internationalization support is weak
- No off-the-shelf reporting tool



# Ruby on Rails Request Flow

## Configuration

- Uses Convention over Configuration, and Reflection
- Therefore very little configuration compared to other frameworks
- ActiveRecord configuration can use SQL
- Uses YAML (easy to read) rather than XML

```
development:
      adapter: oci
      host: 192.168.0.50/examplesid
      username: exampleuser
      password: examplepass
```

# Rails Environment and Installing the Software

- Rails will run on many different Web servers
  - Most of your development will be done using WEBrick, but you'll probably want to run production code on one of the alternative servers
    - Apache, Lighttpd (Lighty),Mongrel
- Development Environment
  - Windows, Linux and OS X
  - No IDE needed although there a few available like Eclipse, RadRails
- Installing Ruby for Windows
  - Download the “One-Click” Ruby Installer from http://rubyinstaller.rubyforge.org
- Installing Ruby for Mac
  - It’s already there!
- RubyGems is a package manager that provides a standard format for distributing Ruby programs and libraries
- Installing Rails
  - >gem install rails –include-dependencies

# MVC the Rails way

- Controller (DispatchServlet) orchestrates the application:
  - Extracts parameters and interacts with the Model through an ActionController subclass:
    - Example: http://localhost/item/delete/100 directs to ItemController (defined in item_controller.rb) and passes to the method delete the value 100
  - Invokes the View providing the rendering of the Model
  - ActionController and ActionView form the Action Pack: core requests processing and responses generation

## MVC the Rails way

- View: a combination of templates, partials and layouts using Ruby code tag libraries (similar to Java tag libraries):
  - Allows data display and input, but never handles incoming data
  - ActionView module provides templates rendering (HTML or XML):
    - .rxml templates render XML pages, while
    - .rhtml templates render HTML pages
  - writing a view = writing a template (i.e. HTML fragments are interwoven with Ruby code statements)
  - Controller instance variables and public methods accessible from templates (actions communicate data to templates)

## MVC the Rails way:

- Model: **ActiveRecord wrapping framework**
  - ActiveRecord subclass wraps a row in a database table/view, encapsulates access/domain logic, and acts as the gatekeeper
  - Wraps class hierarchies to relational tables through single table inheritance (a string column 'TYPE' is added to every table)
  - Wraps classic table relationships (**one-to-one**, **one-to-many**, **many-to-many**) through declarations in the model (**belongs-to**, **has-one**, **has-many**, **has-and-belongs-to-many**), and also supports: **acts_as_tree**, **acts_as_list**
  - Differs from other ORM (Object Relational Mapping) implementations through its "convention over configuration" principle which implies a sensible set of defaults

## Rails Relationships

- Model Relations
  - Has_one => One to One relationship
  - Belongs_to => Many to One relationship (Many)
  - Has_many => Many to One relationship (One)
  - Has_and_belongs_to_many =>Many to Many relationships

## Rails Tutorial

- Create the Rails Application
  - Execute the script that creates a new Web application project
  - >Rails projectname
  - This command executes an already provided Rails script that creates the entire Web application directory structure and necessary configuration files

## Rails Application Directory Structure

- App> contains the core of the application
  - /models> Contains the models, which encapsulate application business logic
  - /views/layouts> Contains master templates for each controller
  - /views/controllername> Contains templates for controller actions
  - /helpers> Contains helpers, which you can write to provide more functionality to templates
- Config> contains application configuration, plus per-environment configurability - contains the database.yml file which provides details of the database to be used with the application
- Db> contains a snapshot of the database schema and migrations
- Log> application specific logs, contains a log for each environment
- Public> contains all static files, such as images, javascripts, and style sheets
- Script> contains Rails utility commands
- Test> contains test fixtures and code
- Vendor> contains add-in modules.

## Hello Rails!

- Need a controller and a view
  >ruby script/generate controller Greeting
- Edit app/controllers/greeting_controller.rb
- Add an index method to your controller class

```
class GreetingController < ApplicationController
  def index
    render :text => "<h1>Hello Rails World!</h1>"
  end
end
```

  - Renders the content that will be returned to the browser as the response body
  - If you use blank for the name of the action it maps to the index action
- Start the WEBrick server
  >ruby script/server
  - http://localhost:3000

## Hello Rails!

- Add another method to the controller

```
def hello
end
```

- Add a template app/views/greeting>hello.rhtml

```
<html>
  <head>
    <title>Hello Rails World!</title>
  </head>
  <body>
    <h1>Hello from the Rails View!</h1>
  </body>
</html>
```

## Hello Rails!

- ERb – Embedded Ruby: Embedding the Ruby programming language into HTML document
  - An erb file ends with .rhtml file extension.
  - Similar to ASP, JSP and PHP, requires an interpreter to execute and replace it with designated HTML code and content
- Making it Dynamic

```
<p>Date/Time: <%= Time.now %></p>
```

- Making it Better by using an instance variable to the controller

```
@time = Time.now.to_s
```

  - Reference it in .rhtml <%= @time %>
- Linking Pages using the helper method link_to()

```
<p>Time to say <%= link_to "Goodbye!", :action =>
"goodbye" %>
```

## Rails Resources

- Books
  - Agile Web Development with Rails
  - Programming Ruby
- Web sites
  - Ruby Language
    - http://www.ruby-lang.org/en/
  - Ruby on Rails
    - http://www.rubyonrails.org/
  - Rails API
    - Start the Gem Documentation Server
      - Gem_server http://localhost:8808
  - MVC architectural paradigm
    - http://en.wikipedia.org/wiki/Model-view-controller
    - http://java.sun.com/blueprints/patterns/MVC-detailed.html