St. John’s Linux Users’ Group

Ruby on Rails

Donald Craig

March 23, 2006

Introduction

- Ruby on Rails is an open source framework for developing web applications.
- The framework was extracted from an existing project management application called Basecamp, written by 37signals.
- Uses a Model-View-Controller Architecture (MVC).
- Provides an Object-Relational Mapper (ORM).
- Version 1.0 released December 13, 2005.
- Disclaimer...

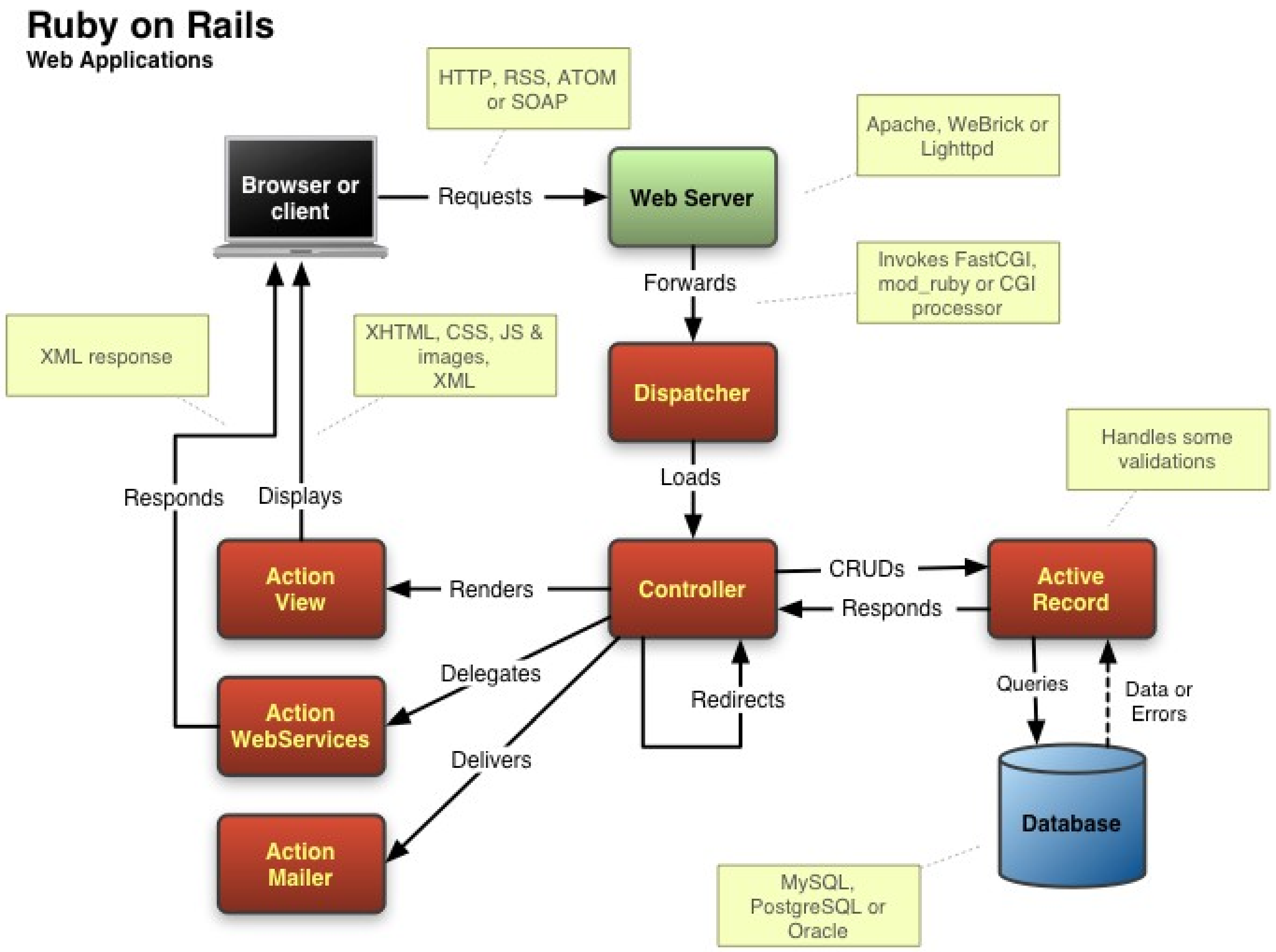
Philosophies

- Don’t Repeat Yourself (DRY)
  - Keep duplication at a minimum.
  - Makes modifications to the system less tedious and error prone.
- Convention over configuration
  - Use sensible (and natural) defaults for the location of files, names of database tables, class names etc.
- Encourage testing during development.
  - Supports unit tests (for models) and functional tests (for controllers).

Model-View-Controller

- From Wikipedia:
  - "Model-view-controller (MVC) is a software architecture that separates an application's data model, user interface, and control logic into three distinct components so that modifications to one component can be made with minimal impact to the others."
- **Model**: typically sits on top of a database.
- **View**: What the user actually sees (web page).
- **Controller**: Takes requests; acts as a link between the model and the view.
- The controller and the view are closely related (together they make up the *presentation* layer).

The Rails Cycle



http://wiki.rubyonrails.com/rails/pages/UnderstandingRailsMVC

Active Record (AR)

- Implements the “model” part of MVC.
- All database accesses go through the AR.
- Conventions:
  - Database tables are plural with underscores
    - *e.g.* `stock_holdings`
  - AR class names are singular with camel-case
    - *e.g.* `StockHolding`
  - Primary keys are always named `id`
  - Foreign keys use the singular name of the foreign table with `_id` appended

Active Record (AR)

- Data validations are handled by AR.
  - *e.g.* `validate_presence_of`, `validate_uniqueness_of`, `validate_format_of`
- Relationships between tables are handled by AR.
  - *e.g.* `has_one`, `has_many`, `belongs_to`, `has_and_belongs_to_many`
- AR handles the Object/Relational Mapping layer.
  - Makes database access/modification easier.

Active Record (example)

Consider the following two database tables:

```
CREATE TABLE patients (
  id          int unsigned  NOT NULL auto_increment,
  given_names varchar(64)   NOT NULL,
  last_name   varchar(64)   NOT NULL,
  mcp_number  varchar(12)   NOT NULL,

  PRIMARY KEY(id)
);


CREATE TABLE charts (
  id         int unsigned NOT NULL auto_increment,
  patient_id int unsigned NOT NULL,
  weight     int(4)       NOT NULL,
  height     int(4)       NOT NULL,
  comment    varchar(255) NOT NULL,
  date       datetime     NOT NULL,

  PRIMARY KEY(id)
);
```

Active Record (example)

Here are the corresponding Active Records:

```
class Patient < ActiveRecord::Base
  validates_presence_of   :given_names, :last_name
  validates_uniqueness_of :mcp_number
  validates_format_of     :mcp_number,
                          :with => /^\d{12}$/,
                          :message => "must be 12 digits"
  has_many :charts
end
```

```
class Chart < ActiveRecord::Base
  validates_presence_of :date, :comment

  belongs_to :patient
end
```

Action Pack (Controller/Views)

- The controller class contains a collection of methods which represent actions to be performed on the database.
- Each action typically consults the database performing necessary reads and/or writes.
- The controller then delegates control to the corresponding view which is simply an `.rhtml` file (HTML + snippets of Ruby code).
- The controller also contains methods that are called when forms are submitted.

Action Pack (example)

```
class ClinicController < ApplicationController
  def index
    list
    render :action => 'list'
  end

  def list
    @patient_pages, @patients =
      paginate :patients, :per_page => 10
  end

  def show
    @patient = Patient.find(params[:id])
  end

  def new
    @patient = Patient.new
  end

  def create
    ...
  end

  def edit
    @patient = Patient.find(params[:id])
  end

  def update
    ...
  end

  def destroy
    ...
  end
end
```

list.rhtml

```
<h1>Listing patients</h1>

<table>
  <tr>
  <% for column in Patient.content_columns %>
    <th><%= column.human_name %></th>
  <% end %>
  </tr>

<% for patient in @patients %>
  <tr> ...
```

show.rhtml

```
<% for column in Patient.content_columns %>
<p>
  <b><%= column.human_name %>:</b>
  <%=h @patient.send(column.name) %>
</p>
<% end %>

<%= link_to 'Edit',
            :action => 'edit', :id => @patient %> |
<%= link_to 'Back', :action => 'list' %>
```

new.rhtml

```
<h1>New patient</h1>
<%= start_form_tag :action => 'create' %>
  <%= render :partial => 'form' %>
  <%= submit_tag "Create" %>
<%= end_form_tag %>
<%= link_to 'Back', :action => 'list' %>
```

Object Relational Mapping (ORM)

- A class represents a database table, an object represents a row in the table and an object attribute represents a column of the row.
- Allows one to access/modify a database using constructs native to an object-oriented language instead of having to use SQL.
- Relationships between tables can also be created and navigated using native object-oriented constructs.

ORM (examples)

```
pat = Patient.new
pat.given_names = "Homer J."
pat.last_name = "Simpson"
pat.mcp_number = "123456789012"
pat.save

Patient.create(:given_names => "Marge",
               :last_name   => "Simpson",
               :mcp_number  => "210987654321")
require 'pp'

pp Patient.find(1)
pp Patient.find_by_last_name("Simpson")
pp Patient.find_all_by_last_name("Simpson")

pat = Patient.find_by_given_names("Homer J.")
pat.given_names = "Homer Jay"
pat.save
pp Patient.find(1)

Patient.find(params[:id]).charts.create(params[:chart])
```

Scaffolding

- Allows one to generate a Controller (with its associated views) and Model very quickly.
- Intended to get a functional web application up and running quickly.
- The web interface is very primitive. It’s not really intended for production web sites.
- You can use scaffolding to help learn some of the fundamentals of Rails.
- As you further develop your application, the scaffolding disappears.

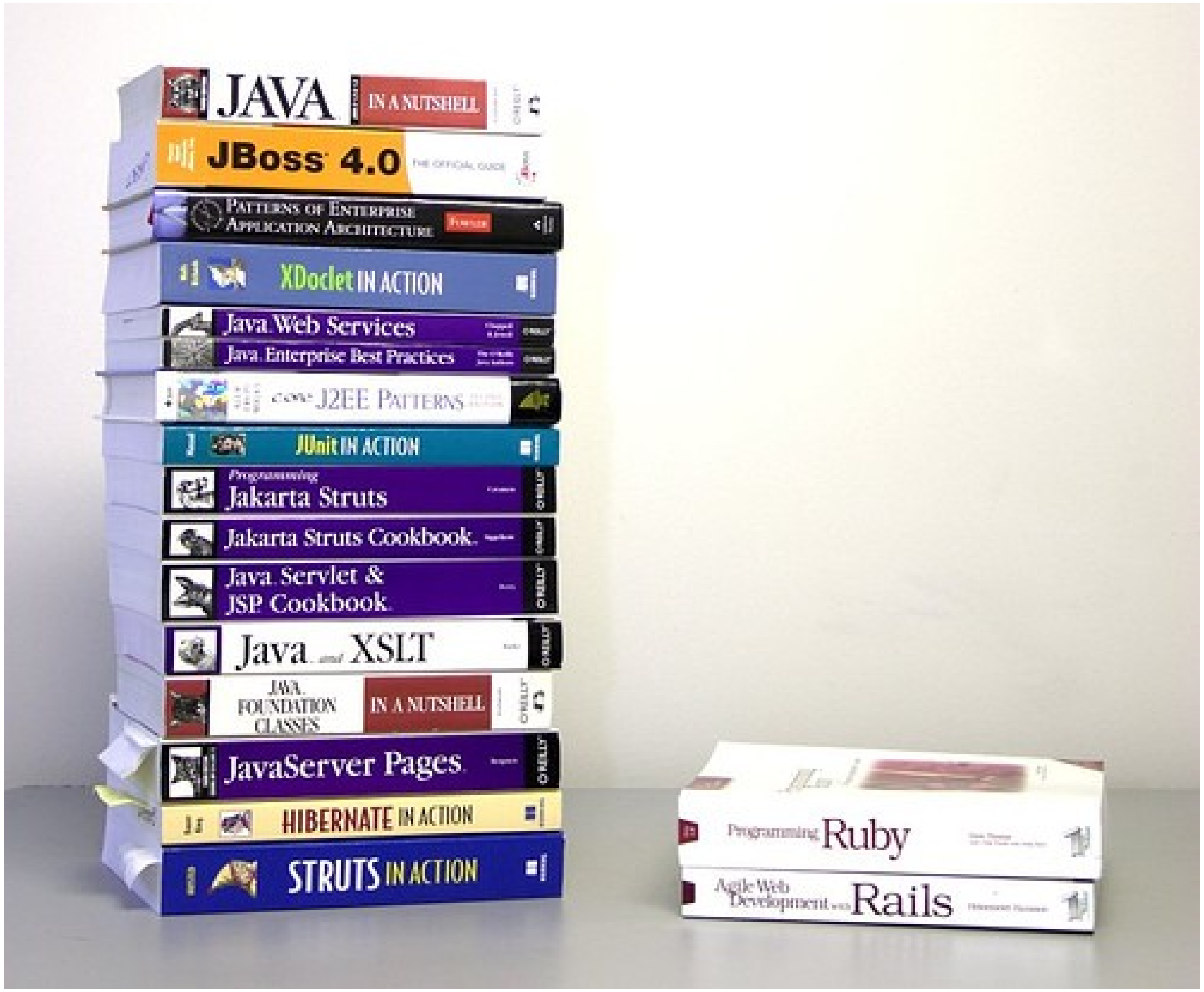
Software Compatibility

- Ruby on Rails is not tied to any one specific web server, database or even operating system.
  - Web servers known to work with Rails include: Apache, WEBrick, LightTPD, ...
  - Databases that work with Rails include: MySQL, PostgreSQL, SQL Server, DB2, Oracle, ...
  - Rails friendly Operating Systems include: Linux, Mac OS X, Windows, ... (many of the core developers work on Mac OS X).

Miscellaneous

- Downloading and installing the Rails framework is straight forward, but a bit tedious:
  - See http://www.rubyonrails.org/down for details.
  - Alternatively, Mac OS X users can try: http://locomotive.sourceforge.net/
  - Windows users can try: http://instantrails.rubyforge.org/
- RadRails, an IDE built on the Eclipse “Rich Client Platform” (RCP), is available at: http://www.radrails.org/

Comparing the Java/J2EE stack with the Ruby on Rails stack...


JAVA IN A NUTSHELL
JBoss 4.0
Patterns of Enterprise Application Architecture
XDoclet IN ACTION
Java Web Services
Java Enterprise Best Practices
core J2EE Patterns
JUnit IN ACTION
Programming Jakarta Struts
Jakarta Struts Cookbook
Java Servlet & JSP Cookbook
Java and XSLT
JAVA FOUNDATION CLASSES IN A NUTSHELL
JavaServer Pages
HIBERNATE IN ACTION
STRUTS IN ACTION
Programming Ruby
Agile Web Development with Rails

http://jutopia.tirsen.com/articles/2005/10/28/why-ruby-on-rails

References

- Helpful Links:
  - http://www.rubyonrails.org
  - http://wiki.rubyonrails.org/rails
  - http://developer.apple.com/tools/rubyonrails.html
  - http://www.rubyonrails.org/docs
- Useful Books:
  - Agile Web Development with Rails
  - Programming Ruby (2nd edition)
  - More books on the way...

Demo...

http://www.cs.mun.ca/~donald/slug/2006-03-23/demo.php