Ruby on Rails
An Overview

J. Dew

Department of Computer Science and Engineering
University of South Carolina

April 11, 2007
Outline

1. Ruby: The Foundation
   - Language Basics
     - Completely Object Oriented
     - Methods, Classes and Modules

2. Rails: The Framework
   - Model-View-Controller Architecture
   - Database-centric Programming
   - Convention over Configuration

3. Conclusion
Ruby is . . .

“A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.”

– http://ruby-lang.org/

Examples

- `puts "Hello World"
- `name = gets
- `puts "Hello #{name}"
- `hash = { :id => 42 }
- `hash.has_key? :id
Ruby is . . .

“A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.”

– http://ruby-lang.org/

Examples

- puts "Hello World"
- name = gets
  puts "Hello #{name}"
- hash = {:id => 42}
  hash.has_key? :id
Ruby is . . .

“A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.”

– http://ruby-lang.org/

Examples

- puts "Hello World"
- name = gets
  puts "Hello #{name}"
- hash = {:id => 42}
  hash.has_key? :id
Ruby is . . .

“A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.”

– http://ruby-lang.org/

Examples

- `puts "Hello World"
- `name = gets
  `puts "Hello #{name}"`
- `hash = { :id => 42 }
  `hash.has_key? :id"`
Examples

```ruby
3.downto(0) do |count|
  unless count == 0 then
    print "#{count}.."
  else
    puts "Blastoff!"
  end
end
```

### Conditionals and Looping
Conditionals and Looping

Examples

3.downto(0) do |count|
  unless count == 0 then
    print "#{count}..
  else
    puts "Blastoff!"
  end
end

Produces 3..2..1..Blastoff!
Outline

1. Ruby: The Foundation
   - Language Basics
   - Completely Object Oriented
   - Methods, Classes and Modules

2. Rails: The Framework
   - Model-View-Controller Architecture
   - Database-centric Programming
   - Convention over Configuration

3. Conclusion
Everything’s an Object

Examples

42.methods.sort

returns

['%', '&', '*', '**', '+', '+@', '-', '-@', '/', '<', '<<<', '<=', '<=>', '==', '===', '=~', '>', '>=', '>>>', '[]', '^', '___id___', '___send___', 'abs', 'between?', 'ceil', 'chr', 'class', 'clone', 'coerce', 'display', 'div', 'divmod', 'downto', 'dup', 'eql?', 'equal?', ...
'type', 'untaint', 'upto', 'zero?', '|', '~']
Outline

1. Ruby: The Foundation
   - Language Basics
   - Completely Object Oriented
   - Methods, Classes and Modules

2. Rails: The Framework
   - Model-View-Controller Architecture
   - Database-centric Programming
   - Convention over Configuration

3. Conclusion
Methods

Example

```ruby
def factorial(n)
    n == 1 ? 1 : n * factorial(n-1)
end
```
**Example**

class Client

    def Client.most_lucrative(clients)
        # class method
        end

    def paid_in_full!
        # instance method
        end

end
Example from Rails

module ActiveRecord

  class Base

    def save
      # create or update a record
    end

  end

end
Outline

1. Ruby: The Foundation
   - Language Basics
   - Completely Object Oriented
   - Methods, Classes and Modules

2. Rails: The Framework
   - Model-View-Controller Architecture
   - Database-centric Programming
   - Convention over Configuration

3. Conclusion
The Model

Concept

- Interacts directly with the RDBMS
- Maintains the state of the application
- Enforces any rules or validations related to the data
- Encapsulated by ActiveRecord in Rails
The Model

Concept

- Interacts directly with the RDBMS
- Maintains the state of the application
- Enforces any rules or validations related to the data
- Encapsulated by ActiveRecord in Rails
The Model

Concept

- Interacts directly with the RDBMS
- Maintains the state of the application
- Enforces any rules or validations related to the data
- Encapsulated by ActiveRecord in Rails
The Model

Concept

- Interacts directly with the RDBMS
- Maintains the state of the application
- Enforces any rules or validations related to the data
- Encapsulated by ActiveRecord in Rails
The View

**Concept**

- Responsible for generating the UI
- Where the HTML, JS, CSS, etc resides
- Mostly printing variables and simple loops
- ActionView in Rails
The View

Concept

- Responsible for generating the UI
- Where the HTML, JS, CSS, etc resides
- Mostly printing variables and simple loops
- ActionView in Rails
The View

Concept

- Responsible for generating the UI
- Where the HTML, JS, CSS, etc resides
- Mostly printing variables and simple loops
- ActionView in Rails
The View

Concept

- Responsible for generating the UI
- Where the HTML, JS, CSS, etc resides
- Mostly printing variables and simple loops
- ActionView in Rails
The Controller

Concept

- Coordinates between the model(s) and the view(s)
- Responds to user input and routes accordingly
- Most of the application logic goes here
- ActionController in the Rails framework
The Controller

Concept

- Coordinates between the model(s) and the view(s)
- Responds to user input and routes accordingly
- Most of the application logic goes here
- ActionController in the Rails framework
The Controller

Concept

- Coordinates between the model(s) and the view(s)
- Responds to user input and routes accordingly
- Most of the application logic goes here
- ActionController in the Rails framework
The Controller

Concept
- Coordinates between the model(s) and the view(s)
- Responds to user input and routes accordingly
- Most of the application logic goes here
- ActionController in the Rails framework
Outline

1. Ruby: The Foundation
   - Language Basics
   - Completely Object Oriented
   - Methods, Classes and Modules

2. Rails: The Framework
   - Model-View-Controller Architecture
   - Database-centric Programming
   - Convention over Configuration

3. Conclusion
Assumptions

Rails assumes that:
- you have a RDBMS backend
- you don’t mind following certain naming conventions
- you want to be database agnostic
- you think SQL embedded in your code is ugly
Assumptions

Rails assumes that

- you have a RDBMS backend
- you don’t mind following certain naming conventions
- you want to be database agnostic
- you think SQL embedded in your code is ugly
Assumptions

Rails assumes that

- you have a RDBMS backend
- you don’t mind following certain naming conventions
- you want to be database agnostic
- you think SQL embedded in your code is ugly
Assumptions

Rails assumes that

- you have a RDBMS backend
- you don’t mind following certain naming conventions
- you want to be database agnostic
- you think SQL embedded in your code is ugly
The PHP Way

PHP Code

```php
$dbh = mysql_connect("host","user","pwd");
mysql_select_db('my_project');

$id = $_GET['id'];
$sql = "SELECT * FROM users WHERE id=$id";
$result = mysql_query($sql);
$row = mysql_fetch_assoc($result);

echo "Welcome, " . $row['first_name'] . ".";

mysql_free_result($result);
mysql_close($link);
```
The Rails Way

Rails Database Config

```ruby
development:
  adapter: mysql
  host: host
  username: user
  password: pwd
  database: my_project
```

This would normally appear in `config/database.yml`. 
The Rails Way (continued)

Rails Controller Code

```ruby
@user = User.find(@params[:id])
```

Rails View Code

```html
Welcome, <%= @user.first_name %>
```

In this case, the controller code would be found in `app/controllers/user_controller.rb` and the model code would be found in `app/models/user.rb`. 
What is Rails doing behind the scenes here?

**Concept**

- Maps database tables to classes
- Maps rows to objects
- Maps columns to attributes
Object/Relational Mapping

What is Rails doing behind the scenes here?

**Concept**
- Maps database tables to classes
- Maps rows to objects
- Maps columns to attributes
Object/Relational Mapping

What is Rails doing behind the scenes here?

Concept

- Maps database tables to classes
- Maps rows to objects
- Maps columns to attributes
Example

Assume you have a table with all of your employees stored in a table called `employees`.

```ruby
Employee.find(:all) do |employee|
  if employee.last_name == "Dew"
    employee.salary = employee.salary * 2
    employee.save
  end
end
```
Outline

1. Ruby: The Foundation
   - Language Basics
   - Completely Object Oriented
   - Methods, Classes and Modules

2. Rails: The Framework
   - Model-View-Controller Architecture
   - Database-centric Programming
   - Convention over Configuration

3. Conclusion
Model Naming

Concept

- Database names should be plural
- Class names should be singular in mixed case
- Filenames should be singular with underscores

Example

If you want a model for book orders, the database name should be `book_orders`, the class should be named `BookOrder`, and stored in the file `book_order.rb`. 
Model Naming

Concept

- Database names should be plural
- Class names should be singular in mixed case
- Filenames should be singular with underscores

Example

If you want a model for book orders, the database name should be `book_orders`, the class should be named `BookOrder`, and stored in the file `book_order.rb`.

J. Dew

Ruby on Rails
Model Naming

Concept
- Database names should be plural
- Class names should be singular in mixed case
- Filenames should be singular with underscores

Example
If you want a model for book orders, the database name should be `book_orders`, the class should be named `BookOrder`, and stored in the file `book_order.rb`.
Model Naming

Concept
- Database names should be plural
- Class names should be singular in mixed case
- Filenames should be singular with underscores

Example
If you want a model for book orders, the database name should be `book_orders`, the class should be named `BookOrder`, and stored in the file `book_order.rb`. 
Why use Rails?

Benefits

- Easy division of labor between programmers and designers
- Less time spent writing configuration files
- Programmers and/or designers new to a project know where to find all assets in the project
- Doing AJAX requests are just as easy as not
- Very nice and comprehensive community support and documentation
- Allows you to create complete web applications in days, not months
Why use Rails?

Benefits

- Easy division of labor between programmers and designers
- Less time spent writing configuration files
  - Programmers and/or designers new to a project know where to find all assets in the project
  - Doing AJAX requests are just as easy as not
- Very nice and comprehensive community support and documentation
- Allows you to create complete web applications in days, not months
Why use Rails?

Benefits

- Easy division of labor between programmers and designers
- Less time spent writing configuration files
- Programmers and/or designers new to a project know where to find all assets in the project
- Doing AJAX requests are just as easy as not
- Very nice and comprehensive community support and documentation
- Allows you to create complete web applications in days, not months
Why use Rails?

Benefits

- Easy division of labor between programmers and designers
- Less time spent writing configuration files
- Programmers and/or designers new to a project know where to find all assets in the project
- Doing AJAX requests are just as easy as not
  - Very nice and comprehensive community support and documentation
  - Allows you to create complete web applications in days, not months
Why use Rails?

Benefits

- Easy division of labor between programmers and designers
- Less time spent writing configuration files
- Programmers and/or designers new to a project know where to find all assets in the project
- Doing AJAX requests are just as easy as not
- Very nice and comprehensive community support and documentation
- Allows you to create complete web applications in days, not months
Why use Rails?

Benefits

- Easy division of labor between programmers and designers
- Less time spent writing configuration files
- Programmers and/or designers new to a project know where to find all assets in the project
- Doing AJAX requests are just as easy as not
- Very nice and comprehensive community support and documentation
- Allows you to create complete web applications in days, not months
Why use Rails? (continued)

Benefits

- Rails (and Ruby) are open-source software
- Since Ruby is an interpreted language, it can be moved from platform to platform with very minimal changes
- Rapid prototyping allows you to show your customer a working demo instead of static mockups at the design meetings
- Proven to be reliably scaleable
Why use Rails? (continued)

Benefits

- Rails (and Ruby) are open-source software
- Since Ruby is an interpreted language, it can be moved from platform to platform with very minimal changes
- Rapid prototyping allows you to show your customer a working demo instead of static mockups at the design meetings
- Proven to be reliably scaleable
Why use Rails? (continued)

Benefits

- Rails (and Ruby) are open-source software
- Since Ruby is an interpreted language, it can be moved from platform to platform with very minimal changes
- Rapid prototyping allows you to show your customer a working demo instead of static mockups at the design meetings
- Proven to be reliably scaleable
Why use Rails? (continued)

Benefits

- Rails (and Ruby) are open-source software
- Since Ruby is an interpreted language, it can be moved from platform to platform with very minimal changes
- Rapid prototyping allows you to show your customer a working demo instead of static mockups at the design meetings
- Proven to be reliably scaleable
Final Remarks

Apologies

- I only began to scratch the surface of Ruby
  More information about Ruby can be found at http://www.ruby-lang.org/
- I only began to scratch the surface of Rails
  More information about Rails can be found at http://rubyonrails.org/
Final Remarks

Apologies

- I only began to scratch the surface of Ruby
- More information about Ruby can be found at http://www.ruby-lang.org/
- I only began to scratch the surface of Rails
- More information about Rails can be found at http://rubyonrails.org/
Final Remarks

Apologies

- I only began to scratch the surface of Ruby
- More information about Ruby can be found at http://www.ruby-lang.org/
- I only began to scratch the surface of Rails
- More information about Rails can be found at http://rubyonrails.org/
Final Remarks

Apologies

- I only began to scratch the surface of Ruby
- More information about Ruby can be found at http://www.ruby-lang.org/
- I only began to scratch the surface of Rails
- More information about Rails can be found at http://rubyonrails.org/
Questions?