

Rails Spleunking

by Wilhelm Chung

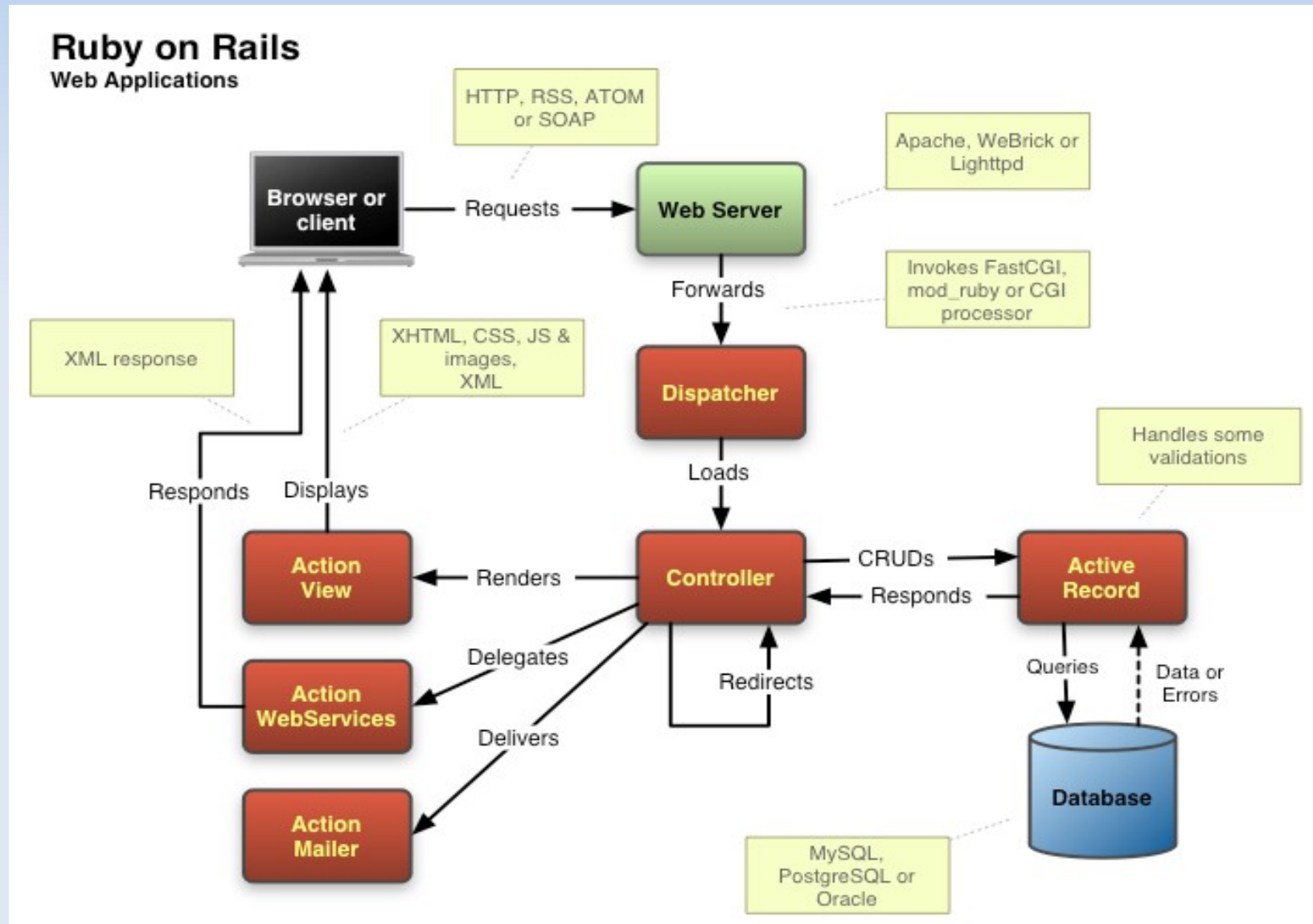
Background

- EE communication systems,
CS machine learning
- C++, Assembly, Lisp/Scheme, Matlab,
Embedded C, Java
- Ruby, Javascript, Erlang

Overview

- Rails Architecture Overview
- Brief dip in each MVC part
- **Rails metaprogramming**
 - **Dynamic finders**
 - **Method extension at runtime**
- Enlightenment

Rails Architecture



ActiveRecord

- Deals with the Database
- Provides validation methods to help keep code clean
- Provides association methods
- Provides find() for SQL generation

ActionView

- Generates file based on templates and parameters from ActionController
 - .rhtml – Erb and HTML
 - .rxml – Builder::XmlMarkup
 - .rjs – ActionView::Helpers::PrototypeHelper::JavascriptGenerator
- One can generate their own custom templates

ActionController

- Business logic resides here
- before, after, and around filters to help keep code clean
- One can alter the HTTP responses to fit your specific needs

Metaprogramming Trickery

- Dynamic Finders
 - such as "Account.find_by_username(...)"
- Method extensions at Runtime
 - such as "render_without_layout()"

Find by an attribute

```
Account.find(:first,  
:conditions => ["username = ?", "david"])
```

Dynamic Finder Instead!

```
Account.find_by_username("david")
```

Where is this method? I don't see it anywhere in the docs or source!

If can't find, create

```
account = Account.find_by_username("david")  
if account.nil?  
  Account.create(:username => "david")  
end
```

or

```
account = Account.find_by_username("david") ||  
  Account.create(:username => "david")
```

Dynamic Finder Instead!

```
account = Account.find_or_create_by_username("david")
```

Can't find this method in the docs either. :(

Dynamic Finders use `method_missing()`

`method_missing()` is called when an object gets a message for a method it can't find in its call hierarchy.

ActiveRecord's method_missing()

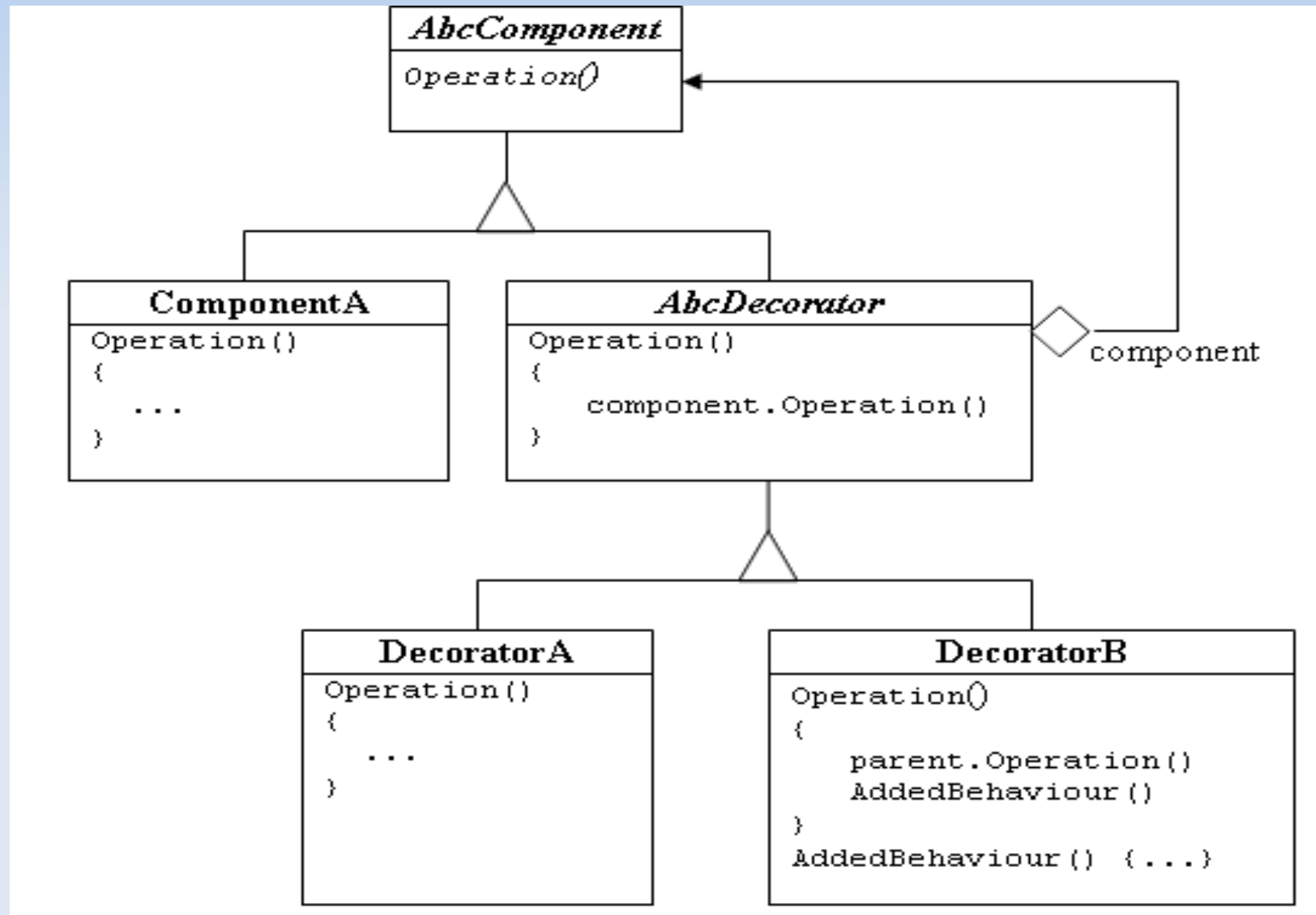
```
def method_missing(method_id, *arguments)
  if match = /^find_(all_by|by)_([_a-zA-Z]\w*)$/>.match(method_id.to_s)
    # figure out which finder to use
    # extract options and attributes
    # call the finder!
  elsif match = /^find_or_(initialize|create)_by_([_a-zA-Z]\w*)$/>.match(method_id.to_s)
    # do more find stuff
  else
    super
  end
end
```

Runtime filter chains

- Add functionality to existing methods
 - ie. Add Validation to Save
- before and after filters

What happens when we need to add functionality during run time?

Decorator Pattern



<http://commons.wikimedia.org/wiki/Image:Decorator.GIF>

http://en.wikipedia.org/wiki/Decorator_pattern

Rails doesn't use the Decorator Pattern

How does Rails add validation to a save method?

alias_method

alias_method(new_name, old_name) => self

```
module Mod
  alias_method :copy_of_exit, :exit
  def exit(code=0)
    puts "Exiting with code #{code}"
    copy_of_exit(code)
  end
end
```

```
include Mod
exit(99)
```

produces:

Exiting with code 99

alias_method_chain

```
def alias_method_chain(target, feature)
  aliased_target, punctuation = target.to_s.sub(/([?!=])$/, ""), $1

  alias_method "#{aliased_target}_without_#{feature}", target
  alias_method target, "#{aliased_target}_with_#{feature}"
end
```

```
copy "save" as "save_without_validation"
copy "save_with_validation" as "save"
```

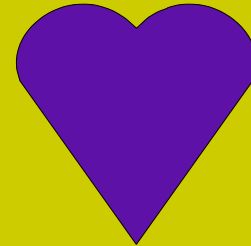
Start

ActiveRecord



save

Validation



save_with_validation

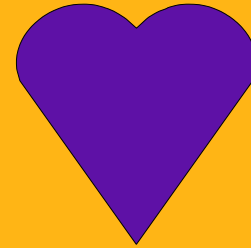
include Validation

ActiveRecord



save

Validation



save_with_validation

alias_method_chain, Part I

ActiveRecord

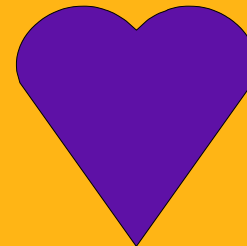


save_without_validation



save

Validation



save_with_validation

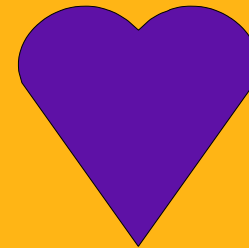
alias_method_chain, Part II

ActiveRecord



save_without
_validation

Validation



save

alias_method_chain

Before:

```
class ActiveRecord
  def save
    # do saving stuff
  end
end

module Validation
  def save_with_validation
    # do validation
    save_without_validation
  end
end

class ActiveRecord
  include Validation
  alias_method_chain
    :save, :validation
end
```

After:

```
class ActiveRecord
  def save_without_validation
    # do saving stuff
  end

  def save
    # do validation
    save_without_validation
  end
end
```

The Common Rails Pattern

```
module ActionController #:nodoc:
  module SessionManagement #:nodoc:
    def self.included(base)
      base.extend(ClassMethods)
      base.send :alias_method_chain, :process,
                :session_management_support
      base.send :alias_method_chain, :process_cleanup,
                :session_management_support
    end

    module ClassMethods
      # Class methods go here
    end

    # Instance methods go here
    def process_with_session_management_support(...)
      set_session_options(request)
      process_without_session_management_support(...)
    end
  end
end
```

The Common Rails Pattern

```
require 'action_controller/session_management'
```

```
ActionController::Base.class_eval do  
  include ActionController::SessionManagement  
end
```

Resources and Tools

- Rails API docs - <http://api.rubyonrails.org/>
- Searchable API (including protected and private methods) - <http://railsmanual.net/>
- Rails Trac - <http://dev.rubyonrails.org/>