Ruby on Rails

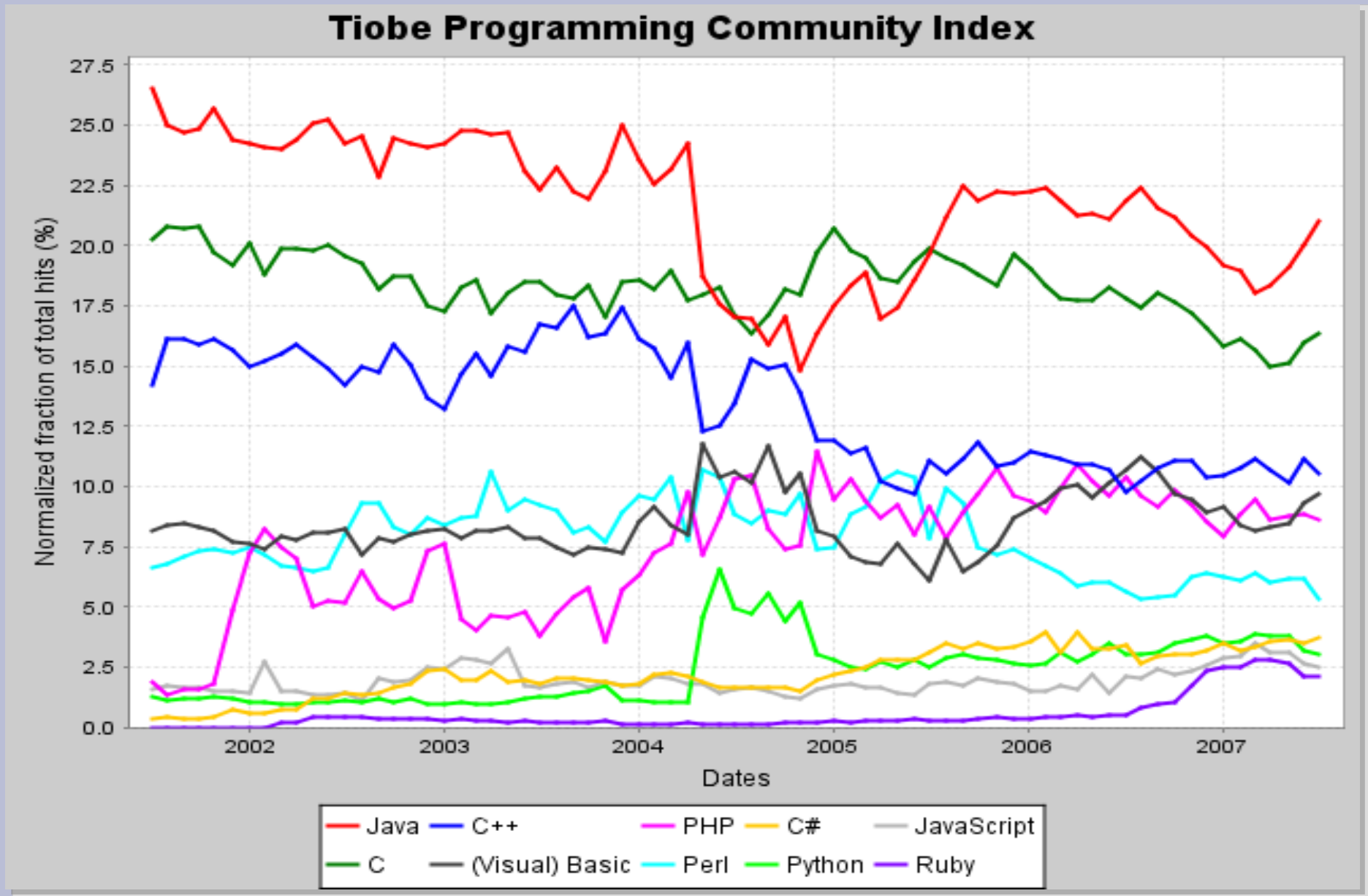
- Geschichte
- Ruby
- Rails
- Live – Demo

Geschichte


Ruby's architect,
Yukihiro Matsumoto

- Ruby
  - "Vater": Yukihiro “matz” Matsumoto
  - 1993: Beginn der Arbeiten an Rubys
  - 1995: 1. public Release von Ruby
  - 1996: erlangt "Akzeptanz der Massen" (aktive Usergruppen, gefüllte Konferenzen)
  - aktuell: 1.8.6
- Rails
  - Gründer: David Heinemeier Hansson
  - 2003: Beginn der Entwicklung (aus Basecamp abgeleitet)
  - 2004: 1. Release
  - aktuell: 1.2.3

TIOBE Index



www.tiobe.com

Ruby

- "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."
  (www.ruby-lang.org)

Ruby

- objektorientiert (*alles* ist ein Objekt)
- interpretiert (20KB executable in C geschrieben)
- Open Source – "Ruby License" oder GPL
- "Object" als Wurzel
- Single Inheritance (+ "mixin"s)
- Garbage Collection
- vollständig Kommandozeilen-orientiert
- wird mit Standard-Library geliefert

Ruby - Code-Beispiel

```ruby
class Song
  attr_accessor :name # accessor methods
  @@plays = 0 # static field

  def initialize(name)# constructor
    @name = name # instance field
  end

  def play # instance method
    @@plays += 1 # "return" omitted
  end
end

song = Song.new('my Song')
puts song.name # => my Song
puts song.play # => 1
```

Ruby – etwas genauer (1)

- streng und dynamisch typisiert
- es gibt "public", "protected", "private"
  - *protected*: Aufruf aus der definierten Klasse und deren Subklassen
  - *private*: nur innerhalb eines Objekts (mit implizitem "this" als Empfänger)
- Operatoren können überladen werden
- Definitionen (Klassen, Methoden...) werden mit "end" abgeschlossen (kein { })
- "()" bei Methoden-Aufrufen fehlen in der Regel
- Variablen-Scope über Naming-Convention

Ruby – etwas genauer (2)

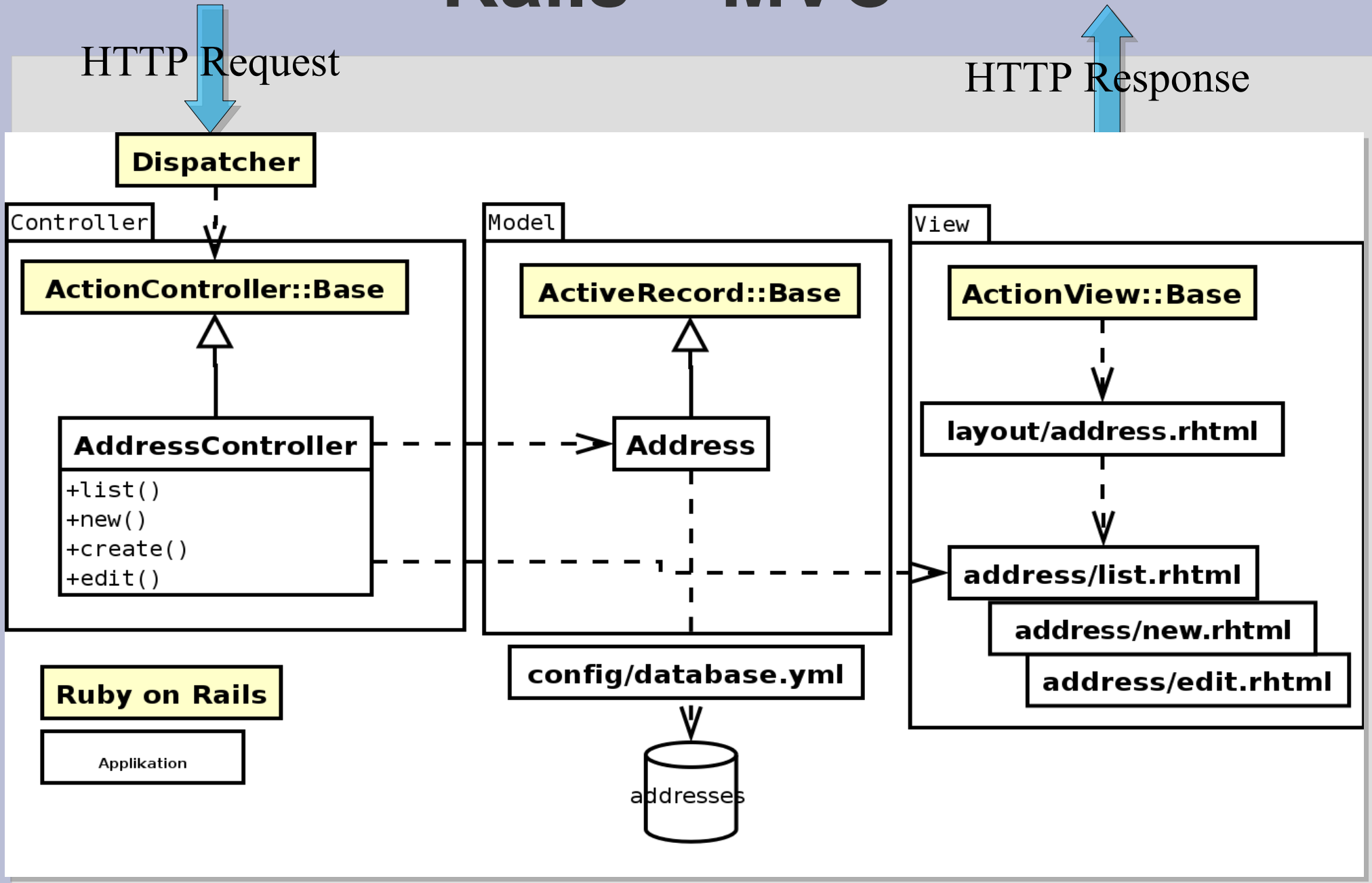
- alle Klassen sind "offen"
  - jede Klasse (auch Object) kann jederzeit erweitert werden
- Closures
  - Code-Blöcke mit Zugriff auf alle umliegenden Variablen (ähnlich wie anonyme Klassen in Java)
- Iteration mit Blöcken
  - ```
    %w[dies sind Elemente einer Liste].each { |element|
      puts element
    }
    ```
- Mixin – einbinden von Modulen
  - ```
    include Enumerable
    ```
  - ähnliche wie Interface in Java
- Variable Konstanten
  - gibt zur Laufzeit eine Warnung...

Rails

- "Web development that doesn't hurt"

- Model View Controller
  - ActiveRecord
  - ActionView
  - ActionController
- Convention over Configuration
- Don't repeat yourself
- Unterstützung für DB-Migrationen / Unittesting
- ...

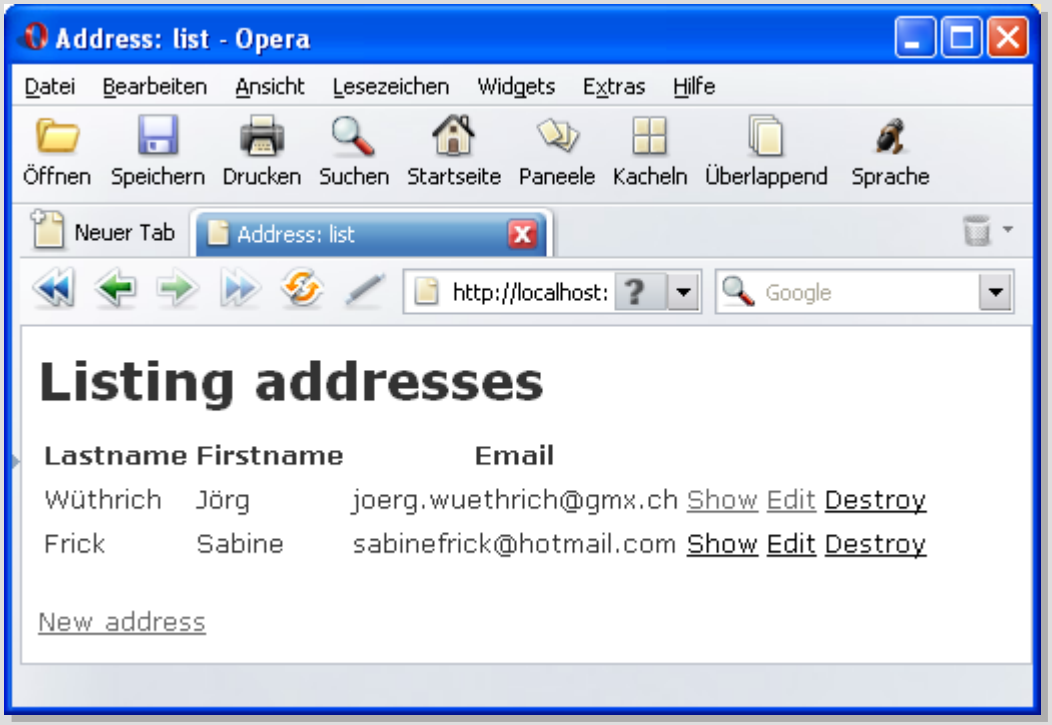
Rails – MVC


HTTP Request
HTTP Response
Dispatcher
Controller
ActionController::Base
AddressController
+list()
+new()
+create()
+edit()
Model
ActiveRecord::Base
Address
config/database.yml
addresses
View
ActionView::Base
layout/address.rhtml
address/list.rhtml
address/new.rhtml
address/edit.rhtml
Ruby on Rails
Applikation

Rails – don't repeat yourself

- ActiveRecord
  - Methoden aus DB-Metadaten generiert
- ActionView
  - Layout pro Model
  - Template pro Methode
  - Referenzen zwischen Templates
- Scaffolding
  - generiert Gerüst einer Applikation
  - macht viele "Fingerübungen" überflüssig
    - neuer Controller
    - neues Model-Element
    - neue View
    - neuer Testfall

Live - Demo


Address: list - Opera
Datei Bearbeiten Ansicht Lesezeichen Widgets Extras Hilfe
Öffnen Speichern Drucken Suchen Startseite Paneele Kacheln Überlappend Sprache
Neuer Tab
Address: list
http://localhost:
Google
Listing addresses
Lastname Firstname Email
Wüthrich Jörg joerg.wuethrich@gmx.ch Show Edit Destroy
Frick Sabine sabinefrick@hotmail.com Show Edit Destroy
New address

Rails - ActionController

- Zentrale Drehscheibe für Webrequests
- bestehen aus Actions, die als Request oder als Redirect aufgerufen werden können
  - AddressController.list
  - AddressController.create...

```
def create
  @address = Address.new(params[:address])
  if @address.save
    flash[:notice] = 'Address was successfully created.'
    redirect_to :action => 'list'
  else
    render :action => 'new'
  end
end
```

- rendern ein Template aus app/views

Rails - ActionView

- Template-Renderer für Output
- 3 Varianten im Standard enthalten
  - .rhtml – Mischung aus ERb ("eRuby") und Html
  - .rxml – programmatische Alternative zu .rhtml
  - .rjs – verwendet den JavaScriptGenerator

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

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

- enthält diverse "Helpers" (vergleichbar mit Tag Libraries)

Rails – ActiveRecord (1)

- Repräsentieren eine DB-Tabelle
- führen selbst keine Attribute, sondern leiten diese aus der Tabellen-Definition ab
- Änderungen werden nicht am ActivRecord-Objekt gemacht, sondern immer direkt auf der Tabelle
- Standard-Verhalten kann übersteuert werden
- Unterstützt optimistisches / pessimistisches Locking

```
def edit
  @address = Address.find(params[:id], :lock => true)
end
```

Rails – ActiveRecord (2)

DB-Migrationen

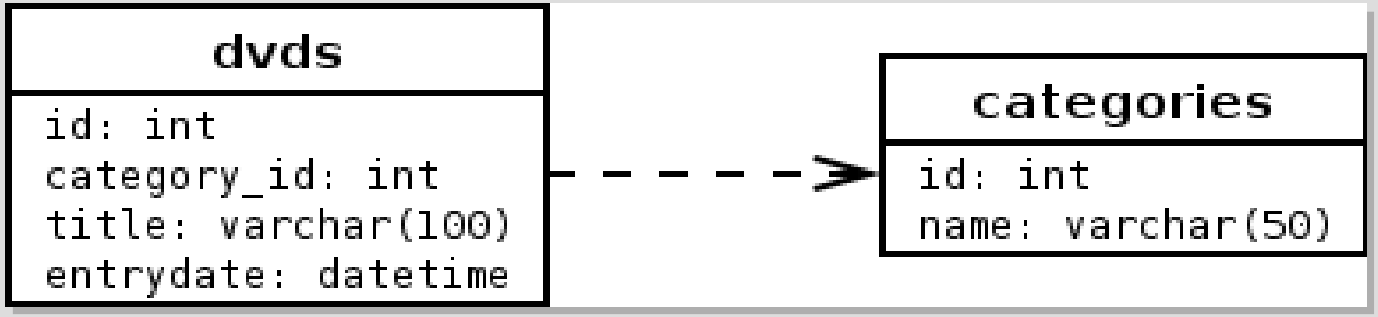
- Verwaltung des Lifecycles von Tabellen
  - Neue Tabelle
  - Spalten hinzufügen / entfernen
  - Index setzen ...
- Ruby Notation in Standard-Fällen

```
class InitAdresslist < ActiveRecord::Migration
  def self.up
    create_table :addresses do |table|
      table.column :lastname, :string, :limit => 40
      table.column :firstname, :string, :limit => 40
      table.column :email, :string, :limit => 100
    end
  end
  ...
end
```

- SQL-Notation, falls benötigt

Rails – ActiveRecord (3)

Assoziationen



ActiveRecord

```
class Dvd < ActiveRecord::Base
  belongs_to :category
end
class Category < ActiveRecord::Base
  has_many :dvd
end
```

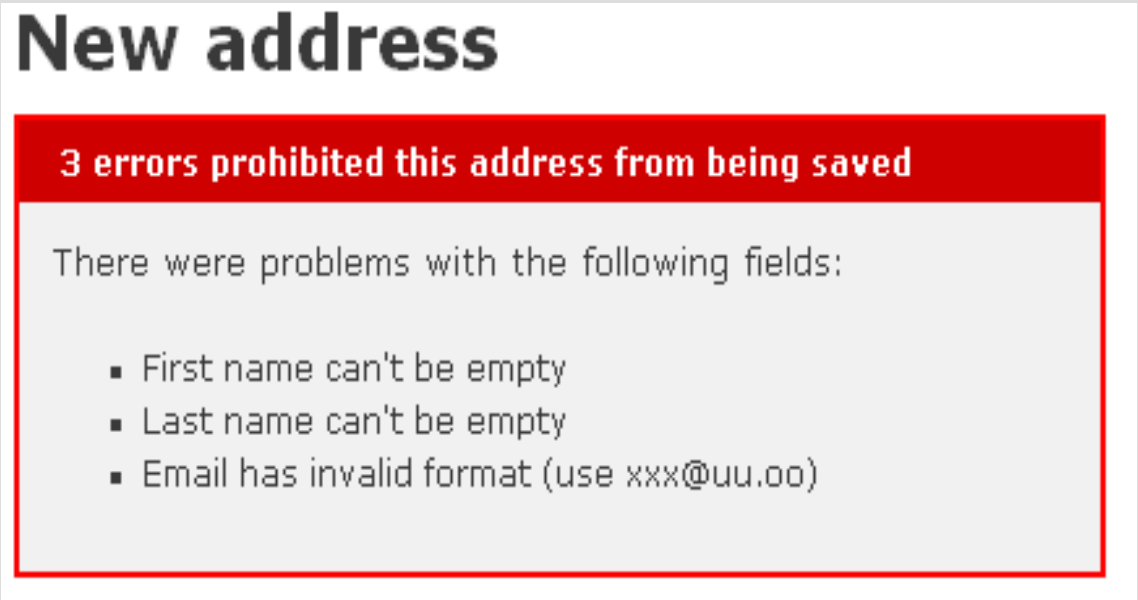
- Zugriff

```
dvdsWithCategories = Dvd.find(:all, :include => :category)
puts dvdsWithCategories.title # => "Bourne Identity"
puts dvdsWithCategories.category.name # => "Action"
```

Rails – ActiveRecord (4)

- Validation

```
class Address < ActiveRecord::Base
  protected
  def validate
    errors.add_on_empty %w(first_name last_name)
    errors.add("email", "has invalid format (use xxx@uu.oo)")
      unless email =~ /[a-z]*@[a-z]*\.[a-z]*/
  end
end
```



Warum Ruby on Rails?

- Wenig Konfiguration notwendig
- Kaum Wiederholungen, relativ sauberer Code
- voll objektorientiert
- DB-Anbindung simpel
- Web 2.0 "ready"
- Integriertes Unittesting
- Integriertes Staging
- Open Source mit aktiver Community

- relativ jung; Erfahrungen mit wirklich grossen Projekten fehlen
- langsamer als PHP oder ASP

Referenzen

- www.ruby-lang.org - die Ruby Webseite
- http://rubyonrails.org/ – die Ruby on Rails Webseite
- http://wiki.rubyonrails.org/rails - Beantwortet viele Fragen rund um Ruby on Rails
- http://homepage2.nifty.com/sakazuki/rde_en/index.html - RDE (Ruby Development Environment)
- http://www.aptana.com/ - Eclipse basierte Ruby on Rails Entwicklungs-Umgebung
- http://www.martinfowler.com/eaaCatalog/activeRecord.html - das ActiveRecord Pattern
- http://www.meshplex.org/wiki/Ruby/Ruby_on_Rails_programming_tutorials - guter Überblick über die Möglichkeiten von Ruby on Rails

backup

Ruby Tools

- ruby.exe – Interpreter
- rake.bat – ruby make
- rdoc.bat – Ruby Doc
  - analog zu javadoc -> output als html, chm, ri, xml)
- ri.bat – Ruby Information
  - ähnlich "man" auf Unix
- irb.bat – interactive ruby
- gem.bat – package manager

Rails - Testing

- Unit Tests
  - Test für einzelnes Modul (ActiveRecord)
- Functional Tests
  - Test der Actions eines Controllers separat
- Integration Tests
  - Test des Zusammenspiels von mehreren Controllern und Actions