Stefan Tilkov • innoQ Deutschland GmbH • stefan.tilkov@innoq.com

Metaprogramming with Ruby

How to write code that writes code – and why this is a good idea

innoQ

The essence of modern software engineering
OOP 2008
Software meets Business

Contents

Motivation

1. Languages are not Equal
2. Mindless Repetition is Productivity's Natural Enemy
3. Language Influences Thought
4. Languages Should Support Growth

1. Languages are not Equal

Machine Code

Assembler

C

C++

Java

Python

Ruby

Scheme/Lisp

All Turing-complete: every task doable in all of them

Big differences in runtime behavior (speed, efficiency)

Even bigger differences in development support

You can program C in OO style … but why would you?

“Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of CommonLisp.”

Philip Greenspun's Tenth Rule of Programming
http://philip.greenspun.com/research/

2. Repetition is Productivity's Natural Enemy

Repetition equals redundancy

Manual pattern execution introduces errors ...

... and spoils the fun

Changes become harder, quality decreases

3. Language Influences Thought

You only apply patterns and concepts that you know of

A programming language's capabilities influence the way you express a solution

Anything out of the ordinary seems "weird"

"We cut nature up, organize it into concepts, and ascribe significances as we do, largely because we are parties to an agreement to organize it in this way — an agreement that holds throughout our speech community and is codified in the patterns of our language."

Whorf, Benjamin (John Carroll, Editor) (1956). Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf. MIT Press.

"Sapir-Whorf Hypothesis" (note: now disputed);
see also http://en.wikipedia.org/wiki/Sapir-Whorf_hypothesis

Blub falls right in the middle of the abstractness continuum... As long as our hypothetical Blub programmer is looking down the power continuum, he knows he's looking down. Languages less powerful than Blub are obviously less powerful, because they're missing some feature he's used to. But when our hypothetical Blub programmer looks in the other direction, up the power continuum, he doesn't realize he's looking up. What he sees are merely weird languages... Blub is good enough for him, because he thinks in Blub.

Paul Graham, “Beating the Averages”
http://www.paulgraham.com/avg.html

4. Languages Should Support Growth

General purpose programming languages can cover general cases

Abstractions match *every* domain

Key idea of DSLs: A language suitable to the *specific* problem domain

A growable language enables definition of new constructs that look and feel *as if they were part of the language*

Ruby Intro

Statements & Control Flow

```ruby
puts "Hello World"

num = 5

if num > 4 then
  puts "num > 4"
elsif num <= 4 then
  puts "num <= 4"
else
  puts "WTF?"
end

puts "num is 5" unless num != 5
```

Loops

```ruby
for i in (1..10) do
  puts i
end

i = 0
while i < 10 do
  puts i
  i += 1
end
```

Comments

```
# One line comment
=begin
A comment spanning multiple lines
=end
```

Iteration & Blocks

```
# don't do this
array = ["alpha", "beta", "gamma"]
for i in 0..2 do
  puts array[i]
end

# much better
array.each { | elem | puts elem }
```

Iteration & Blocks (2)

```ruby
1.upto(10) { | x | puts x }
1.upto(10) { | x | puts "Count: #{x}" }
1.upto(10) do | x |
  puts "Count: #{x}"
end
```

Hashes

```
hash = { "one" => '1', "two" => '2', "three" => '3'}
puts hash["one"]

table = { "p1" => { "last" => "Schulze", "first" =>
"Hans"},
          "p2" => { "last" => "Meier", "first" => "Klaus"}
        }
puts table["p1"]
puts table["p1"]["first"]

require 'pp'
pp table
pp table["p1"]
```

Methods

```ruby
def mymethod(a, b, c)
  puts "a = #{a}, b = #{b}, c=#{c}"
end

mymethod(1, 2, 3)
mymethod 1, 2, 3
```

Classes

```ruby
class Person
  @@people_count = 0

  def initialize(first, last)
    @first = first
    @last = last
    @id = @@people_count
    @@people_count += 1
  end

  def to_s
    "#{@last}, #{@first}"
  end

end

p = Person.new("John", "Doe")
puts p
```

Inheritance

```
class Friend < Person
  def initialize(first, last, nick)
    super(first, last)
    @nick = nick
  end

  def drink
    puts "Cheers from #{@nick}"
  end

  def to_s
    "#{super.to_s}, a.k.a. #{@nick}"
  end
end

f = Friend.new("Jack", "Daniels", "Buddy")
puts f
f.drink
```

Modules

```ruby
module M1
  def self.module_method(s)
    puts "Module method: #{s}"
  end

  def mixin
    puts "Value of a: #{@a}"
  end
end

M1.module_method("hello")

class X
  include M1
  def initialize
    @a = 4711
  end
end

x = X.new
x.mixin
```

“Getters” and “Setters”

```
class AttributeHolder
  def name=(n)
    @name = n
  end

  def name
    @name
  end
end

ah = AttributeHolder.new
ah.name = "AH Test"
puts ah.name
```

“Getters” and “Setters” (2)

```ruby
class AttributeHolder2
  def name=(n)
    @name = n
  end

  def name
    @name
  end

  def first_name=(n)
    @first_name = n
  end

  def first_name
    @first_name
  end
end

ah = AttributeHolder2.new
ah.name = "AH Test"
ah.first_name = "AH First"
puts ah.name, ah.first_name
```

Attribute Accessor

```ruby
class AttributeHolder3
  attr_accessor :name, :first_name
end

ah = AttributeHolder3.new
ah.name = "AH Test"
ah.first_name = "AH First"
puts ah.name, ah.first_name
```

Ruby Metaprogramming

Metaprogramming

Programs that write (or modify) programs

Including, but not limited to, code generation

Blurring boundaries between development time & run time

Linked to *reflective* capabilities

Structures

```
Person = Struct.new "Person", :first_name, :last_name
p1 = Person.new
p1.last_name = "Doe"
p1.first_name = "John"
p1 # => #<struct Struct::Person first_name="John", last_name="Doe">

p2 = Person.new("Jane", "Doe")
p2 # => #<struct Struct::Person first_name="Jane", last_name="Doe">
```

Creating Objects and Classes by Name

```ruby
s = Kernel.const_get('String').new "Teststring" # => "Teststring"
s.class # => String

Test = Class.new # => Test
Test.class_eval do
  def test1
    "test1"
  end
end
Test.new.test1 # => "test1"

Test.class_eval do
  define_method "test2" do
    "test2"
  end
end
Test.new.test2 # => "test2"
```

Individual Object Methods

```
t1 = Test.new
t2 = Test.new
t1.standard_method # => "standard_method; self: #<Test:0x16ee0>"
t2.standard_method # => "standard_method; self: #<Test:0x16e04>"

class << t1
  def object_method
    "object_method; self: #{self}"
  end
end

t1.object_method # => "object_method; self: #<Test:0x16ee0>"
t2.object_method # => NoMethodError: undefined method
                      'object_method' for #<Test:0x16e04>
```

Classes & Constants

```
cls = Class.new
cls.class_eval do
  define_method :test_method do
    "test_method"
  end
end

cls.new.test_method # => "test_method"
cls # => #<Class:0x1b2b0>
SomeArbitraryConstant = cls
cls # => SomeArbitraryConstant
```

‘eval’ Methods

eval	evaluates a string as Ruby code, receives binding
instance_eval	evaluates block in context of receiver
class_eval (a.k.a module_eval)	evaluates block in context of class or module, usually used to add methods

Runtime Definitions

```ruby
class TestClass
  puts "before definition, self: #{self}"

  def my_instance_method
    puts "my_instance_method, self: #{self}"
  end

  puts "after definition, self: #{self}"
end

# >> before definition, self: TestClass
# >> after definition, self: TestClass
# >> my_instance_method, self: #<TestClass:0x19f00>
```

Runtime Definitions (2)

```ruby
TestClass.new.my_instance_method

class TestClass
  def self.my_class_method
    puts "my_class_method, self: #{self}"
  end

  my_class_method
end

# >> my_class_method, self: TestClass
```

Methods Adding Methods

```ruby
class Meta
  def initialize(value)
    @value = value
  end

  def self.add_multiplier(factor)
    define_method "times#{factor}" do
      @value * factor
    end
  end

  add_multiplier 5
end

Meta.new(3).times5 # => 15
```

Methods Adding Methods (2)

```ruby
module Multiplication

  module ClassMethods
    def new_class_m
      puts "new_class_m - self: #{self}"
    end

    def add_multiplier(factor)
      define_method "times#{factor}" do
        @value * factor
      end
    end

  end

  def self.included(clazz)
    clazz.extend(ClassMethods)
  end

end
```

```ruby
class MultiplyTest
  include Multiplication

  def initialize(value)
    @value = value
  end

  add_multiplier 3
end
MultiplyTest.new(3).times3 # => 15
```

(Re-)Opening Classes

```ruby
def to_label(s)
  (s.split '_' ).map {|c| c.capitalize}.join ' '
end

to_label("LONG_UNREADBLE_CONSTANT") # => "Long Unreadble Constant"
to_label("unwieldy_name") # => "Unwieldy Name"
```

```ruby
class String
  def to_label
    (self.split '_' ).map {|c| c.capitalize}.join ' '
  end
end

"LONG_UNREADBLE_CONSTANT".to_label # => "Long Unreadble Constant"
"unwieldy_name".to_label # => "Unwieldy Name"
```

(Re-)Opening Classes

```ruby
def array_shuffle!(array)
  0.upto(array.length-1) do |i|
    r = (rand * array.length).to_i
    array[i], array[r] = array[r], array[i]
  end
  array
end

array = %w(7 8 9 10 B D K A)
array_shuffle!(array)
# => ["A", "D", "9", "7", "10", "8", "K", "B"]
```

```ruby
class Array
  def shuffle!
    0.upto(length-1) do |i|
      r = (rand * length).to_i
      self[i], self[r] = self[r], self[i]
    end
    self
  end
end

array = %w(7 8 9 10 B D K A)
array.shuffle!
# => ["9", "B", "K", "A", "8",
      "10", "7", "D"]
```

method_missing

```ruby
class Recorder
  def method_missing(name, *args)
    @calls ||= []
    @calls << { :name => name, :args => args}
  end

  def print_calls
    @calls.each do |call|
      puts "#{call[:name]}(#{call[:args].join(', ')})"
    end
  end
end

r = Recorder.new
r.first_call 1, 2, 3
r.second_call "Hello"
r.third_call :bumm
r.print_calls
# =>
# >> first_call(1, 2, 3)
# >> second_call(Hello)
# >> third_call(bumm)
```

Examples

Rails ActiveRecord

```ruby
class Person < ActiveRecord::Base
  has_many :adresses
  has_one :home_address, :class_name => "Address"
  belongs_to :region
end

class Region
  has_many :people
end

class Address
  belongs_to :person
end
```

Generated Methods

```
class Project < ActiveRecord::Base
```

belongs_to :portfolio	Project.portfolio Project.portfolio=(*portfolio*) Project.portfolio.nil?
has_one :project_manager	Project.project_manager, Project.project_manager=(*project_manager*) Project.project_manager.nil?
has_many :milestones	Project.milestones.empty? Project.milestones.size Project.milestones Project.milestones<<(*milestone*) Project.milestones.delete(*milestone*) Project.milestones.find(*milestone_id*) Project.milestones.find(:all, *options*) Project.milestones.build, Project.milestones.create
has_and_belongs_to_many :categories	Project.categories.empty? Project.categories.size Project.categories Project.categories<<(*category1*) Project.categories.delete(*category1*)

```
end
```

acts_as_state_machine

```
class Cat < ActiveRecord::Base
   acts_as_state_machine :initial => :sheltered, :column => 'status'

   state :sheltered #Initial state - Cat is at the shelter being cared for
   state :incare # Cat is with a shelter appointed carer (nursing the cat to health)
   state :returned # Owner located and cat returned
   state :housed # New owner is found for cat

   event :shelter do
     transitions :to => :sheltered, :from => :incare
   end

   event :care do
     transitions :to => :incare, :from => :sheltered
   end

   event :return do
     transitions :to => :returned, :from => :sheltered
     transitions :to => :returned, :from => :incare # Cat can be given straight from care
   end

   event :house do
     transitions :to => :housed, :from => :sheltered
     transitions :to => :housed, :from => :incare
   end
end
```

Atom with XML Builder

```
xml.instruct! 'xml-stylesheet', :href=>'/stylesheets/atom.css', :type=>'text/css'

xml.feed :xmlns=>'http://www.w3.org/2005/Atom' do
  xml.div :xmlns=>'http://www.w3.org/1999/xhtml', :class=>'info' do
   xml << <<-EOF
     This is an Atom formatted XML site feed.
     It is intended to be viewed in a Newsreader or syndicated to another site.
     Please visit <a href="http://www.atomenabled.org/">atomenabled.org</a> for more info.
   EOF
  end

  xml.title   'Sam Ruby'
  xml.link    :rel=>'self',
    :href=>url_for(:only_path=>false, :action=>'posts', :path=>['index.atom'])
  xml.link    :href=>url_for(:action=>'posts', :path=>nil)
  xml.id      :href=>url_for(:only_path=>false, :action=>'posts', :path=>nil)
  xml.updated Time.now.iso8601
  xml.author  { xml.name 'Sam Ruby' }

  @entries.unshift @parent if @parent
  @entries.each do |entry|
    xml.entry do
      xml.title   entry.title
      xml.link    :href=>url_for(entry.by_date)
      xml.id      entry.atomid
      xml.updated entry.updated.iso8601
      xml.author  { xml.name entry.author.name } if entry.author
      xml.summary do
        xml.div :xmlns=>'http://www.w3.org/1999/xhtml' do
          xml << entry.summary
        end
      end if entry.summary
      xml.content do
        xml.div :xmlns=>'http://www.w3.org/1999/xhtml' do
          xml << entry.content
        end
      end
    end
  end

end
```

see: http://intertwingly.net/stories/2005/09/21/app/views/blog/atom.rxml

Summary

Ruby has a rich set of metaprogramming features - tied into its object model

Metaprogramming enables another level of abstraction

Metaprogramming is fun!

More information and resources:

http://www.innoq.com/resources/ruby-metaprogramming

Stefan Tilkov

http://www.innoq.com/blog/st/

Architecture Consulting

SOA WS-* REST

MDA MDSD MDE

J(2)EE RoR .NET

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone +49 2102 77162-100
info@innoq.com · www.innoq.com

innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Phone +41 41 743 01 11

http://www.innoq.com