# The Ruby advantage - metaprogramming and DSLs

Niclas Nilsson

http://niclasnilsson.se

http://twitter.com/niclasnilsson

http://factor10.com

# Metaprogramming?

factor10

# DSLs?

factor10

# Why both in the same talk?

factor10

# Problem:
# Code does not reveal intention

(or is just plain ugly)

factor10

# Written any code like this?

```
..., 3 * 24 * 60 * 60, ...
```

or

```
..., 3 * SECS_PER_DAY, ...
```

or

```
..., 3 * HOURS_PER_DAY * MIN_PER_HOUR *
      SECS_PER_HOUR, ...
```

factor10

```ruby
def compound_interest(amount, time, interest)
  # ...
end

compound_interest 1000, 6 * 12 * 30, 0.035
```

factor10

```
SEK = "SEK"

def amount(n, currency)
  # ...
end

def years(y); y * 12; end

def percent(n); n.to_f / 100; end


compound_interest(amount(1000, SEK), years(12),
                  percent(3.5))
```

factor10

```
compound_interest :amount => 1000,
                  :months => 6 * 12,
                  :percent => 3.5
```

factor10

```ruby
 class Numeric
  def SEK
    self    # Or some currency conversion
  end

  def years; self / 12.0; end

  def percent; self / 100.0; end
end


compound_interest 1000.SEK, 6.5.years, 3.5.percent
```

factor10

```ruby
class Numeric
  def compound_interest(months, interest_rate)
    # ...
  end
end

1000.50.SEK.compound_interest(6.years, 3.5.percent)
```

factor10

```ruby
class Integer
  def hours
    self * 60 * 60
  end

  def from_now
    Time.now + self
  end
end


puts 2.hours.from_now
```

factor10

# Dangerous?

Hell, yes!

factor10

# Useful?

Hell, yes!

factor10

# Problem:
# Needed metadata on attributes

(for serialization, schemas, validations, conversions, ...)

factor10

```
class Address

  property :street,  String
  property :city,    String
  # ...
end

class Customer
  property :entity_id,   Integer
  property :first_name,  String
  property :last_name,   String
  property :birthdate,   DateTime

  property :addresses, Address, :collection => true
end
```

factor10

```ruby
# Usage example:

person = Person.new
person.first_name = "Niclas"
puts person.first_name      # prints "Niclas"

Person.new.first_name = 1   # Blows up
```

factor10

# Eval style

```
   def property name, kind
  # Do diff things depending on kind
  gen_accessors_for_simple_property name, kind
end

def gen_accessors_for_simple_property name, kind
  eval "
    def #{name}
      @#{name}
    end"

  code = "
    def #{name}=(v)
      raise 'Type error...' unless v.kind_of?(#{kind})
      @#{name} = v
    end
  "
  eval code    # Debug by 'puts code'
end
```

factor10

# define_method style

```
 def property name, kind
# Create a new property object and store it
@__properties__[name] =
    Properties::Property.new(name, kind)

# ...

self.send :define_method, "#{name}=" do |value|
  # ...

  property = self.class.__properties__[name]
  value = Properties::Value.new(value, property)
  @__property_values__[name] = value
end

# ...
end
```

factor10

```
class Customer
  property :entity_id,    Integer
  property :first_name,   String
  property :last_name,    String
  property :birthdate,    DateTime

  property :addresses, Address, :collection => true
end
```

factor10

```
# A bit nicer?

class Customer
  property :entity_id,    Integer
  property :first_name,   String
  property :last_name,    String
  property :birthdate,    DateTime

  collection :addresses, Address
end
```

factor10

```ruby
def collection name, kind, options = {}
  options[:collection] = true
  property name, kind, options
end
```

factor10

# Similar problem for another client

Just started writing a gem for it

```
gem install properties
```

factor10

```ruby
# Constraints from JSON schema spec

class Customer
  # ...
  property :first_name, String, :min_length => 1
end

class Customer
  # ...
  property :first_name, String, :pattern => /[A-Z][a-z]*/
end
```

factor10

```
  # or plain Ruby code

class Customer
  # ...
  property :first_name,  String, :validation =>
    "not value.blank? and value.start_with?('A')"
end
```

...which is simply evaluated inside by

```
eval validation
```

factor10

Problem:
All entities needs certain properties

factor10

```
  # Don't want to repeat mandatory things for
  entities

class Entity
  property :entity_id,    Integer
  # and some other things
end

class Customer < Entity
  property :entity_id,    Integer
  property :first_name,   String
  property :last_name,    String
  property :birthdate,    DateTime

  collection :addresses, Address
end
```

factor10

“is a Person less than an Entity?”

- No, it’s inheritance. A Person gets everything an Entity has.
- Ok...

factor10

```ruby
# Nicer?

entity :Customer do
  property :first_name,  String
  property :last_name,   String
  property :birthdate,   DateTime
end


...

def entity name, &block
  eval "#{name} = Class.new(&block)"
end
```

factor10

Problem:
There is something missing...

factor10

```
   persons = ArrayWithFinders.new
persons << Person.new("John Smith", 75)
persons << Person.new("Andy Taylor", 50)
persons << Person.new("John Smith", 45)

puts persons.find_by_name("John Smith")
puts persons.find_by_age(50)
puts persons.find_by_occupation("programmer")

# Output
John Smith, 75
John Smith, 45

Andy Taylor, 50
```

factor10

```ruby
class ArrayWithFinders < Array
  def method_missing(name, *args, &block)
    name = name.to_s
    if name =~ /find_by_.*/
      property = name.sub(/find_by_/, "")
    end

    select do |person|
      person.respond_to?(property) &&
        person.send(property) == args[0]
    end
  end
end
```

factor10

```
# method_missing can also turn this...

entity :Customer do
  property :first_name,  String
  property :last_name,   String
  property :birthdate,   DateTime
end

# ...into this if you prefer that?

entity :Customer do
  first_name String
  last_name  String
  birthdate  DateTime
end
```

factor10

```ruby
require 'sinatra'

get '/' do
  'Hello world!'
end

# What happens here?
```

factor10

# Problem: Interception

"If I only could hook when this method was called..."

factor10

```ruby
 class Foo
  def do_something
    puts 'A Foo object is doing something'
  end
end

class LogProxy
  def initialize(obj)
    @obj = obj
  end

  def method_missing(method, *args, &block)
    puts "Entering #{@obj.class}.#{method}"
    @obj.send(method)
    puts "Exiting #{@obj.class}.method #{method}"
  end
end
```

factor10

```
   # Usage:

obj = Foo.new
obj = LogProxy.new(obj)
obj.do_something

# Output:

Entering Foo.do_something
A Foo object is doing something
Exiting Foo.method do_something


# Only for the objects you created this way
```

factor10

```ruby
 class Class
  alias_method :original_new,  :new
  def new(*args)
    obj = original_new(*args)
    unless obj.instance_of?(LogProxy) do
      obj = LogProxy.new(obj)
    end
    obj
  end
end

# Same output
obj = Foo.new
obj.do_something

# Tip: Can also hook when Object
# is inherited
```

factor10

Problem:
Something is computed often

factor10

```ruby
  def compound_interest_at_month(month, amount,
                                 interest_rate)
   return 0 if month == 0
  interest = amount * interest_rate / 12
  interest + compound_interest_at_month(
    month - 1, amount, interest_rate)
end

# Stack overflow eventually
memoize(:compound_interest_at_month)
```

factor10

```
def memoize(method_name)
  alias_append method_name, "_unmemoized"
  define_method method_name do |*args|
    $memoized[ [method_name, args] ] ||=
      send("#{method_name}_unmemoized", *args)
  end
end
```

factor10

Problem:
But there is too many dots and stuff...

factor10

# External DSL

```
# In an external textfile

  find all persons
    older than 21
    with a wage between 10_000 and 20_000
```

factor10

# In a textfile:

```
# 1. Read the file

# 2. Do some parsing, search/replace, regex
#    or whatever until you got something like:

persons.select do |o|
  o.age > 21 && o.wage >= 10000 && o.wage <= 20000
end

# 3. and eval in the right context!
```

factor10

```
 CleanRoom.new.instance_eval do
  # But I want an internal DSL

  find all persons with age > 21 and
    wage > 7 and wage2 < 100_000

  # method_missing catches most,
  # but not all (pun intened!)

  # So lets use another operator

  find all persons with age > 21 &
    wage > 7 & wage2 < 100_000

  # Wow! Can now produce an exact string like that!
  # But now what?
end
```

factor10

```
  CleanRoom.new.instance_eval do
  # This looks even nicer

  find all persons older than 21 with a wage between 10_000 & 20_000

  # ...but it's not legal Ruby :-(

  # syntax error, unexpected tIDENTIFIER, expecting kEND
  #
  # find all persons older than 21 with a wage between 10_000 & 20_000
  #                                    ^
end
```

factor10

```
 CleanRoom.new.instance_eval do
# But this is...

find all persons
  older than 21
  with a wage between 10_000 & 20_000

# Is this now an internal DSL?

# Is this really a good thing?

end
```

factor10

# In a the code:

```
# ~~1. Read the file~~

# 1. Run the code in a CleanRoom with method_missing
#    and replaced operators

# 2. Do some parsing, search/replace, regex
#    or whatever until you got something like:

persons.select do |o|
  o.age > 21 && o.wage >= 10000 && o.wage <= 20000
end

# 3. and eval in the right context!
```

factor10

```
 find all persons
older than 21
with a wage between 10_000 & 20_000

# Is this now an internal DSL?
```

factor10

```
find all persons
  older than 21
  with a wage between 10_000 & 20_000

# Is this now an internal DSL?

# Is this really a good thing?
```

factor10

# Questions?

Niclas Nilsson

http://niclasnilsson.se
http://twitter.com/niclasnilsson
http://factor10.com