The Ruby Programming Language...

it's really fun and it feels good!

John Pagonis
Developer Services

symbian

Menu

- Why?
- What brought us here
- Ruby in twenty minutes (or so)
- Why we should all have a look at it?

Before we start… a word from the wise

A reminder from Fred. P. Brooks ‘’ No Silver Bullet - essence and accidents of software engineering’’,1986

There is NO silver bullet!

Language disclaimer

- I am NOT a language bigot!
- I am a C++ programmer (actually a Symbian OS C++ programmer) and I like C++ for getting to the metal.
- I have programmed enough Java.
- I also teach ‘C’ because it is so basic (and blunt:-) and because it helps explain to students how things work.
- I am also very much interested in Objective-C, Smalltalk and occasionally Python.
- ..even ISO C0x C++ sometimes :-)

Lately in my life

- There is a a lot of stuff I need to automate
- There are a lot of stuff I want to develop
- There are a lot of platforms I need to be using
- I am running out of time….
- I need to be more efficient when coding.
- I have realised that my time and memory is more expensive than my CPUs' time and RAM.
- Everywhere I look, I see code, ugly code!
- I haven't been getting any younger
- I haven't been getting much wiser :-)

Consequently

Life is too short, to not have fun…

I have to cheat!

There must be a better way to program… there must be!!

There must be a better way to program… there must be!!

..to clarify that

There must be a much better way than C,C++ (and maybe Objective-C) to code software that

- … doesn't need to talk to the metal directly
- … doesn't need to be as efficient at runtime
- … lets me change my mind and accommodate change
- … is fun
- … is not ugly, it doesn't get in the way and feels good
- … has a lot of frameworks to get the job done
- … I can use in many domains and on many platforms
- … supports open-world system reuse

Lately in my Developer Services life

- I see code, a lot of code, ugly code
- I see developers struggling to write Symbian OS C++ apps (among other things)
- I see people spending a lot of time developing code that will not work on mobiles correctly
- I see enterprises wanting apps yesterday
- I see people not having enough time to code
- I see enthusiasts and students not learning or wanting to learn C++
- I see re-use going down the drain

I felt so weak

So I went out to find a silver bullet!!!

Then I remembered that there isn't one and that F.P. Brooks said

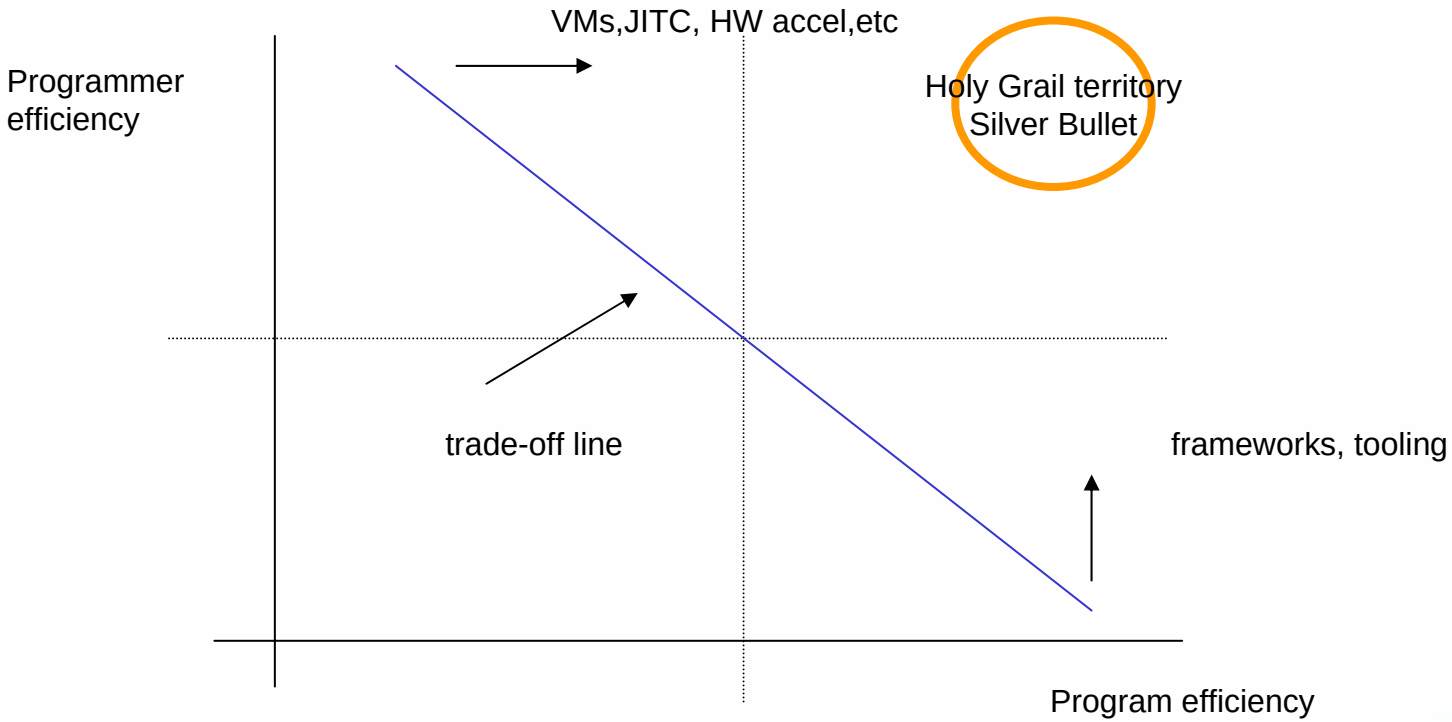
…that compilers are three times as hard to write as application programs and systems programs are three times as hard to write as compilers (may not be true for C++:-)

… that the use of a suitable high-level language may dramatically improve programmer productivity

So lets keep C++ for operating system coding..

So what I need is:

At least, a fun language for non system level stuff, which is mainly programmer efficient as opposed to program efficient, which is also general purpose (that excludes Fortran I guess:-).



So the search is on:

- Objective-C is very clever, simple, cute and powerful, but not ubiquitous, although it is excellent for open-world compiled components. Symbian OS frameworks almost got written in it.
- Java is Objective-C in bondage due to static typing and gets in my way, (for many it has also become the asylum of the incompetent C++ coder). Plus I can deal with memory management myself. But it has a lot of frameworks and tooling.
- Perl is powerful, but ugly as hell.
- Pyhton, is interesting and powerful with good VM support, has a lot of frameworks and tool support, but it carries some baggage, will get better though in Python 3000.
- Smalltalk is my favourite, but is really secluded :-(
- Lisp is assembler for the brain and quite difficult and niche for me

Then I came across Ruby

Actually it was at a Python seminar at ACCU 2006 !!!

- Ruby was first released to the public in 1995 by Matz (Yukihiro Matzimoto)
- Ruby was more popular than Python in Japan already by 2000 and allegedly almost as popular as Perl by 2004
- Ruby is a play on the word Perl :-)
- Ruby was designed to be beautiful, fun and ‘’stay out of the way’’
- It comes with a standard documentation system ‘RDoc’, a standard packaging system ‘RubyGems’, an interactive shell ‘ird’, the Ruby VM and a hell of a lot of frameworks and a standard library.

Ruby in 20 minutes (mostly ripped of the Ruby website by a novice Ruby coder)

- Ruby is a genuine object-oriented language
- The result of every expression is an object
- Objects are garbage collected
- Like with Smalltalk and Objective-C, objects respond to messages
- Such messages contain a method's name together with the parameters that the method may need
- Ruby is a single inheritance language
- Classes can include the functionality of any number of 'mixins'
- Ruby is a dynamic (late-binding) language
- There is access control in Ruby
- You can use curly braces {} if you want to :-) (because it is important:-)

Hello Ruby

```
irb(main):001:0> puts ‘Hello Ruby world’
Hello Ruby world
=>nil
```

This is more interesting though:

```
10.times {better}
```

The message ‘times’ is sent to the object ’10’, with the function ‘better’ as payload, packaged in a block denoted by ‘{}’

Or

```
“Hello Ruby world”.length        -→ 16
```

Duck typing

If it walks like a duck, talks like a duck and lucks like a duck…well it must be a duck

Defining a method

```
def say_hi
   puts “hi”
end
```

Is called by:

```
   say_hi   Or   say_hi()
```

```
...
def say_hi(name)
   puts “hi #{name}”
end
...
def say_hi(name = “there”)
   puts “hi #{name}”
end
```

Defining a class

```
def Greeter
    def initialize(name = “world”)
        @name = name
    end
    def say_hi
        puts “Hi #{name.capitalize}”
    end
end

g = Greeter.new(“john”)
```

Note the coding naming convention there!

Under the object's skin

```
irb(main):010:0> g.@name
SyntaxError: compile error
(irb):52: syntax error
        from (irb):52
```

Let's see what is inside the object:

```
irb(main):039:0> Greeter.instance_methods
=> ["method", "send", "object_id", "singleton_methods",
    "__send__", "equal?", "taint", "frozen?",
    "instance_variable_get", "kind_of?", "to_a",
    "instance_eval", "type", "protected_methods", "extend",
    "eql?", "display", "instance_variable_set", "hash",
    "is_a?", "to_s", "class", "tainted?", "private_methods",
    "untaint", "say_hi", "id", "inspect", "==", "===",
    "clone", "public_methods", "respond_to?", "freeze",
    "__id__", "=~", "methods", "nil?", "dup",
    "instance_variables", "instance_of?"]
```

That was… a lot of methods!

- `Greeter`'s ancestor is `Object`, where it got all those methods

So let's get only the ones we defined then:

```
irb(main):040:0> Greeter.instance_methods(false)
=>["say_hi"]
```

And even check them out

```
irb(main):041:0> g.respond_to?("name")
=> false
irb(main):042:0> g.respond_to?("say_hi")
=> true
irb(main):043:0> g.respond_to?("to_s")
=> true
```

Altering Classes—It's Never Too Late

- In Ruby, you can open a class up again and modify it.
- The changes will be present in any new objects that you create and even in existing objects of that class.

Now consider how powerful this is in the context of BC and over the air upgrades or of a 24/7 system that you can debug and alter when it is live !!

Opening up a class and adding an accessor

```
class Greeter
    attr_accessor :name
End


irb(main):048:0> g.respond_to?("name")
=> true
irb(main):049:0> g.respond_to?("name=")
=> true
```

Using the `attr_accessor` defined two new methods (as opposed to doing it manually) for the existing `Greeter` class which became available to the instantiated g object! The new methods know how to deal with the instance `@name` variable.

Let us see some more tricks

..doing more

```
def say_hi
    if @names.nil?
      puts "..."
    elsif @names.respond_to?("each")
      # @names is a list of some kind, iterate!
      @names.each do |name|
        puts "Hello #{name}!"
      end
    else
      puts "Hello #{@names}!"
    end
  end
```

Iteration, using iterators

```
...
   @names.each do |name|
      puts "Hello #{name}!"
    end
```

If the object pointed by @names responds to the message ‘each’ then you can iterate over its elements… it must be some kind of list!

‘each’ is a method that accepts a ‘block’ of code then runs that ‘block’ of code for every element in a list, and the bit between ‘do’ and ‘end’ is just such a ‘block’. A ‘block’ is like an anonymous function or ‘lambda’ (for Lispians). The variable between the pipe characters is the parameter for this ‘block’

More iteration

```
for i in 1..100
    print “Now at #{i}. Restart? “
    retry if gets =~ /^y/i
end

Now at 1. Restart? n
Now at 2. Restart? y
Now at 1. Restart? n
...
0.upto(10) do |x|
    print x, “ “
end
0.step(10,2) {|x| print x, “ ”}
```

…and there is much more

Blocks

- Blocks are chunks of code (whoa that's new …not)
- Blocks play well with iterators
- Iterators are methods that invoke a block of code repeatedly
- A block may appear only in the source adjacent to a method call
- The code in the block is not executed at the time it is encountered
- Ruby remembers the context in which the block appears and then enters the method

Blocks, where the magic starts…

- Within a method the block may be invoked using the `yield` statement (Python 3000 will also have one)
- Whenever a `yield` is executed it invokes the block
- You can pass parameters to them and receive values from them

e.g.,

```
def two_times
    yield
    yield
End
two_times {puts “Hello”}
```

Conditionals

```
...
kind = case year
       when 1850..1889 then “Blues”
       when 1890..1909 then “Ragtime”
       when 1910..1929 then “New Orleans Jazz”
       when 1930..1939 then “Swing”
       when 1940..1950 then “Bebop”
       else “Jazz”
       end
```

Exceptions handling

- There is an `Exception` class and its children
- Every `Exception`, has a stack backtrace and a string
- Exception handling is enclosed in a `begin`/`end` block
- `rescue` tells Ruby which exceptions to handle
- The `ensure` clause contains code that will always be executed
- There is an `else` clause as well
- And there is way to `retry` !!!

Handling exceptions

```
f = File.open(“thefile”)
begin
    # ... Process
rescue IOError => exc
    # ... Some handling and change of state for retry
    retry
rescue SecurityError
    # ...
else
    puts “No errors,well done!”
ensure
    f.close unless f.nil?
end
```

So `retry` takes us back at the beginning of the block!!

Raising exceptions

```
raise
raise “message from the deep”
raise SomeException, “Message attached”, caller
```

`Kernel.raise` is the actual call and `Kernel.caller` is the method to get the stack trace

And much much more

- Modules and mixins
- `catch` and `throw`
- Parallel assignment
- Lambda
- Hash tables, lists, ranges, arrays, slicing etc
- C bindings
- DRb
- Regular expressions
- Closures
- More conditionals
- More iteration
- Message interception
- Dynamic evaluation of code at runtime
- Profiler
- And much more

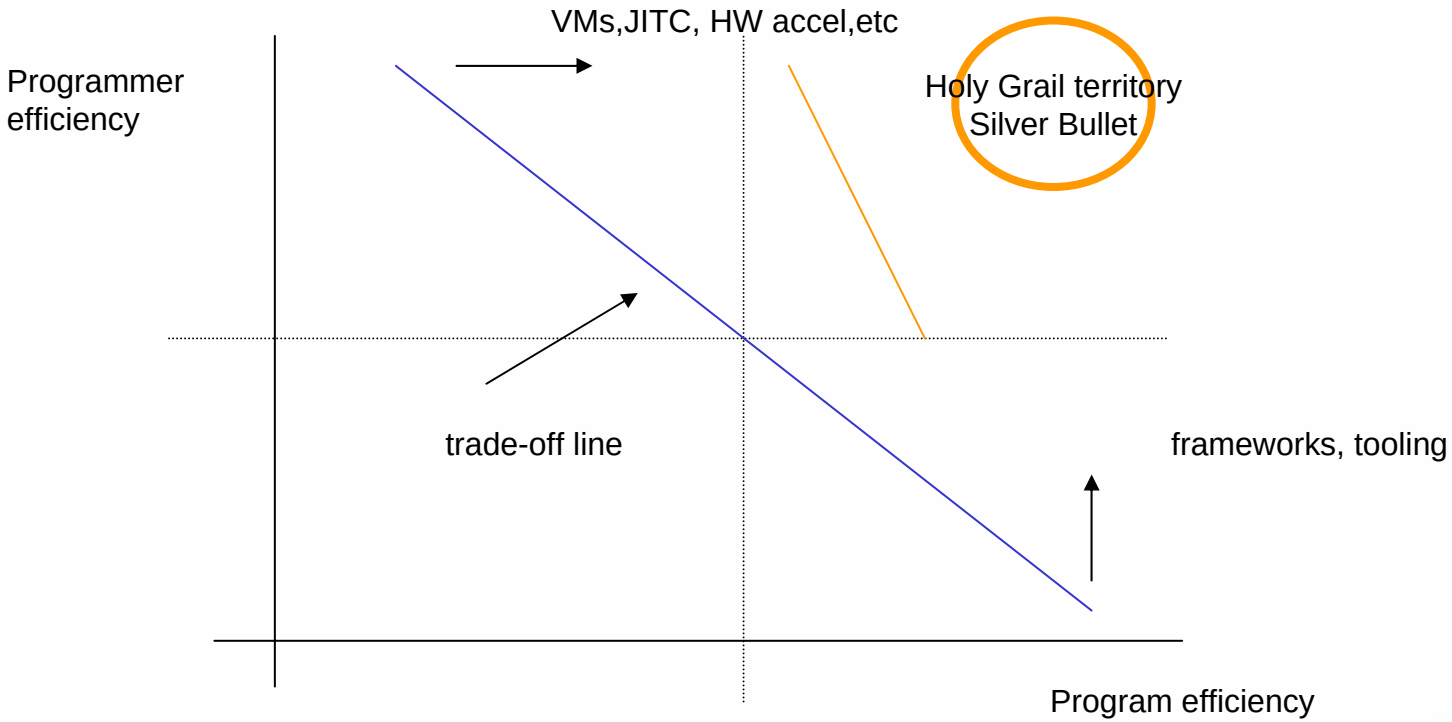
Enough…

So what's next ?

- Ruby is here, it is useful, friendly and fun
- We can take advantage of it both for mobiles and for our operations

So what we need is to move over the trade-off line

In many cases absolute program efficiency becomes a moot point, thanks to Moore's law, while programmer efficiency becomes an acute problem.



What is happening out there

- Ruby is the most discussed and hyped new programming ‘’thing’’
- Sun/IBM are adding JSR 292 to the JVM
- Sun hired the JRuby team
- The (Perl) Parrot VM is designed to accommodate Python and Ruby
- Python has proven that dynamic languages are production ready and popular and powerful (it took us some30+ years but anyway)
- Python and now Ruby is becoming popular at universities
- Java is getting somehow tired (but is here to stay)
- Python on mobiles is happening (a new book coming soon)
- People are struggling to code apps for Symbian OS
- You don’t need permission to hack and extend the Ruby VM

If I was king

- I would include an open VM such as Ruby or Parrot in the platform and make it first class citizen.
- I would investigate whether the above can be achieved with a JSR292 enabled JVM.
- I would contribute to the Ruby or Parrot VM effort to add optimisations necessary for mobiles and Symbian OS, such as a JIT Compiler , signing, and pre-compilation.
- I would build an application framework on Ruby for mobile _applications_
- I would spec and design a universal Ruby GUI for small devices
- I would add instrumentation and tool support for live and interactive on-target development and debugging

To probe further

- www.ruby-lang.org
- www.ruby-doc.org
- “Programming Ruby” by Dave Thomas (also online)