Ruby:

productivity or penance?

Andrew Hume
AT&T - Research

My motivation

- efficiently write distributed control software
  - I hate sockets and TCP/IP
- looking for a replacement for ksh/awk
  - have tried perl; the answer is **no**!
- learn something new and useful
  - Python without the ugly
  - groundwork for Ruby on Rails

Opportunity

- 9 nodes each managing 8 tuners
- 1 admin node
- how to coordinate tuners recording:
  - write standard sockets and TCP/IP C goo
  - do (anything) else
- performance not an issue; ease was
- after eliminating *perl* and *python*, consider *ruby*!
  - but had to be done purely with free online stuff

Looking around

- wow! a real book

http://www.ruby-doc.org/docs/ProgrammingRuby/

- library documentation? wow!

http://www.ruby-doc.org/core/

- lots of examples

(examples from Ruby Cookbook, sadly not available any more)

The acid test

- how to do distributed communication?

```
qthane = DRbObject.new_with_uri(
    "druby://#{thane_ip}:#{::THANE_PORT}")
qchurl = Queue.new
churl_addr = "druby://#{my_ip}:#{::CHURL_PORT}"
DRb.start_service(churl_addr, qchurl)

while job = qchurl.deq
    …
end
```

- how whizzy is that?

Ruby real fast

- like AWK, but a real language with structures, objects and regexes
- many syntactic weirdos to make Perlites feel at home (can be safely ignored)
- full support for threads
- garbage collected memory
- iterators
- google for intros and reference guides

{new | cool | odd} things (1)

- much more on-the-fly constructions

```
a = [2, 3, 4]
h = {‘abc’ => ‘2234’, ‘def’ => [1, 2, 3]}

qthane.enq(‘op’ => ‘status’, 'name' => my_ip)
job = qthane.deq
puts job[‘op’]
```

- evaluated strings

```
“a=#{a} at time #{Time.now.ctime} #{`date`}”
```

{new | cool | odd} things (2)

- new styles for file I/O

```
f = File.new(‘testfile’)
puts “line 1 is #{f.readlines[0]}”

File.new(‘testfile’).each_line{ |b|
    puts “read line #{b}
}

puts “we just read #{f.lineno}”
```

{new | cool | odd} things (3)

- objects like simple classes

```
class Churl
    def initialize(name, state=0)
            @name = name
            @s = state
            @t_op = 0
    end
    def name
            @name
    end
    def to_s
            “churl#{@name} state=#{@s} op=#{@t_op}”
    end
end
```

{new | cool | odd} things (4)

- threads

```
threads << Thread.new(name){ |myname|
    # code here
}

threads.each{ |t| t.join }
```

- as always, use mutexes to synchronise

```
mutex = Mutex.new
. . .
mutex.synchronise do
    . . .
end
```

{new | cool | odd} things (5)

- use if and unless modifiers

```
print t unless t == nil
puts “howdy!” if type == ‘Friend’
```

- case statement

```
case inputline
when ‘exit’
    exit(0)
when /print (\w+)/
    print_var($1)
else
    puts “what the heck? >#{inputline}<“
end
```

{new | cool | odd} things (6)

- exceptions

```
f = File.new(‘testfile’, ‘w’)
begin
    while data = socket.read(512)
            f.write(data)
    end

rescue SystemCallError
    $stderr.print “I/O failed: ” + $!
    f.close
end
```

{new | cool | odd} things (7)

- good libraries
  - cgi
  - kernel
  - Drb
  - Tk
  - many Gems

- good packaging as Gems

http://rubygems.org

Careful now, C guy

- globals are trickier than they should be
- avoid and or (use && ||)
- compound statements (while, if) have odd rules for delimiters
- be aware of exceptions
- 0 argument procedure calls can omit ()
- numbers require care, esp floating point

Epilog

- Ruby has been effective, entertaining and only a little frustrating
- Language of choice for latest project (recursive descent compiler and geometric modeller)
- Many good books (your call)
- Give it a go!