Metaprogramming Ruby - What the hell's a DSL?

- murphee (Werner Schuster)
- Blog @ http://jroller.com/page/murphee

Meta?

Programs that write Programs

Compiler

DSLs

Regexes, anyone?

Always good to start with quotes by...

Alan Kay

The general attitude seems to be that people should wear square shoes, because squares are easier to design and manufacture than foot shaped shoes.

...

If the shoe industry has gone the way of the computer industry it would now be running a $200-a-day course on how to walk, run and jump in square shoes."

Alan Kay

DSLs

- Internal
  - use host language
- External
  - Yacc, ANTLR, ...

Executable Spec

Let's start with... Dylan

- http://www.networknightvision.com/
- Dylan
- declarative protocol spec

Ethernet spec in Dylan

```
define protocol ethernet-frame ( header-frame )
 summary "ETH %= -> %=/%s " ,
 source-address , destination-address , compose ( summary , payload ) ;
 field destination-address : : <mac-address >;
 field source-address : : <mac-address >;
 field type-code : : <2byte-big-endian-unsigned-integer >;
 var iably-typed-field payload ,
 type-function:
 select ( frame . type-code )
 #x800 => <ipv4-frame >;
 #x806 => <arp-frame >;
 otherwise => <raw-frame >;
 end ;
 end ;
```

Ethernet spec in Dylan

```
define protocol ethernet-frame ( header-frame )
 summary "ETH %= -> %=/%s " ,
 source-address , destination-address , compose ( summary , payload ) ;
 field destination-address : : <mac-address >;
 field source-address : : <mac-address >;
 field type-code : : <2byte-big-endian-unsigned-integer >;
 var iably-typed-field payload ,
 type-function:
 select ( frame . type-code )
 #x800 => <ipv4-frame >;
 #x806 => <arp-frame >;
 otherwise => <raw-frame >;
 end ;
 end ;
```

Ethernet spec in Dylan

```
define protocol ethernet-frame ( header-frame )
 summary "ETH %= -> %=/%s " ,
 source-address , destination-address , compose ( summary , payload ) ;
 field destination-address : : <mac-address >;
 field source-address : : <mac-address >;
 field type-code : : <2byte-big-endian-unsigned-integer >;
 var iably-typed-field payload ,
 type-function:
 select ( frame . type-code )
 #x800 => <ipv4-frame >;
 #x806 => <arp-frame >;
 otherwise => <raw-frame >;
 end ;
 end ;
```

Design Process

Incremental does it

- Sketching
- Make it work
- ?
- Profit!

Example time

- `http://www.infoq.com/articles/properties-metaprogramming`

Properties in Ruby/1

- C#-like Properties in Ruby

- Why?
  - to piss of Java zealots

Properties in Ruby/2 - Sketching

```
class Foo
  property name
end
```

Properties in Ruby/2 - Sketching

```
class Foo
  property name
end
```

```
NameError:
undefined local variable or method `name'
 for main:Object
```

Properties in Ruby/3 - Fix

```
class Foo
  property :name
end
```

Properties in Ruby/3 - Fix

```
class Foo
  property :name
end
```

```
NameError:
undefined method `property' for main:Object
```

Properties in Ruby/3 - Fix

```
def property(sym)
  ...
end
```

```
class Foo
  property :name
end
```

Properties in Ruby/3 - Fix

```
def property(sym)
  define_method("#{sym}=") do |value|
    instance_variable_set("@#{sym}", value)
  end
end


class Foo
  property :name
end
```

Properties in Ruby/3 - Fix

```
def property(sym)
  define_method("#{sym}=") do |value|
    instance_variable_set("@#{sym}", value)
  end
end


class Foo
  property :name
end
```

name=

Properties in Ruby/3 - Fix

```
def property(sym)
  define_method("#{sym}=") do |value|
    instance_variable_set("@#{sym}", value)
  end
end
```

@name

```
class Foo
  property :name
end
```

Properties in Ruby/4 - Expand

```
def property(*sym, &bl)
  # Code: Homework
  define_method("#{sym}=") do |value|
    # More Homework
    instance_variable_set("@#{sym}", value)
  end
end


class Tower
  property(:width, :height) {|val| val > 200}
end
```

What we know by now

- Class definitions are executed
- Symbols are nice
- Blocks too

Another one

Pattern Matching

Parseweasel

Ruby ParseTree

```
foo.hello(1)
```

Ruby ParseTree

```
foo.hello(1)

->

[:vcall,
    :hello,
    :foo,
    [:args,
      [:lit, 1]
    ]
]
```

Ruby ParseTree

```
foo.hello(1)

->

[:vcall,
    :hello,
    :foo,
    [:args,
      [:lit, 1]
    ]
]
```

AST as:
s-expr
symbolic expression

ParseWeasel pattern

```
[:vcall, :name_, :recv_, :args_]
```

ParseWeasel pattern

```
[:vcall, :name_,:recv_, :args_]
```

```
someone.something
foo.bar
murphee.speak
```

ParseWeasel pattern/2

```
[:vcall, :hello,:recv_, :args_]
```

ParseWeasel pattern/2

```
[:vcall, :hello,:recv_, :args_]

foo.bar
foo.hello
huey().lewey().hello()
huey().speak
```

ParseWeasel handlers

```
handler([:vcall, :hello,:recv_, :args_]){|s|
  puts “Found a hello #{s[:recv]}”
}
```

ParseWeasel use case: optimizer

```
replace([:call, :op_, :*, [:lit, 0] ]){|s|
  [:lit, 0]
}
```

ParseWeasel use case: optimizer

```
replace([:call, :op_, :*, [:lit, 0] ]){|s|
  [:lit, 0]
}
```

```
x = foo * 0
```

Ruby DSLs

- Keep it simple
- Stick to basics
- If it limps like a hack
  - it probably is one

Let's push it further

Let's mess up the syntax

Hello LISP

```
(defun factorial (x)
    (if (zerop x) 1
        (* x (factorial (- x 1)))
    )
)


(factorial 42)
```

Macros

C Preprocessor

~~C Preprocessor~~

LISP Macro example: Infix

```
(* a 42)
```

I want my infix!

LISP Macro example: Infix

```
(infix (a * 42) )
```

what I write

LISP Macro example: Infix

```
(defmacro infix (one op two  &rest body)
  `(,op ,one ,two )
)
```

```
(infix (a * 42) )
```

what I write

LISP Macro example: Infix

```
(defmacro infix (one op two  &rest body)
  `(,op ,one ,two )
)
```

```
(infix (a * 42) )
```

what I write

```
(* a 42)
```

what gets executed

LISP Macro: real use cases

- Practical Lisp
  - MP3 binary parser
  - executable spec
- Many LISP control structures

Macros in the real world

- Mathematica
  - D[x ^ 2]
  - Integrate[x ^ 3]
  - Expand[x ^ 42]
  -
- Macro-like
- “FullForm” ~ s-expr
  - 3x -> Times[3, x]

Let's wrap up

- Practical Lisp
  - http://www.gigamonkeys.com/book/
- http://ola-bini.blogspot.com/2006/09/ruby-metaprogramming-techniques.html
- http://www.infoq.com/articles/properties-metaprogramming