

# A Little Reflection On Metamodels

A scenic photograph of a church with a tall, pointed spire, likely a Gothic or Romanesque style, situated behind a line of trees. The church and the surrounding landscape are reflected in a calm body of water in the foreground. The sky is filled with soft, golden light from a setting or rising sun, with scattered clouds. In the background, to the left, there are several tall, thin chimneys or towers, possibly part of an industrial or historical site. The overall mood is peaceful and contemplative.

**Jonathan Worthington**

Netherlands Perl Workshop 2010

**OH HAI!**

**Metamodels  
sound a little  
scary... 😐**

**...but don't  
worry, I'm just  
going to tell  
you a story. 😊**

# Chapter 1

## The anthropomorphic class

**Once upon a time, I  
wrote a class.**

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

# A Little Reflection On Metamodels

**I thought  
my work  
was done,  
and I could  
go for a  
beer.**



**But then  
my class  
started  
asking me  
questions...**





# A Little Reflection On Metamodels

How was  
I created?

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

# A Little Reflection On Metamodels

What does it mean to  
have methods?

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

## What does it mean to inherit?

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

# A Little Reflection On Metamodels

Do other classes all  
behave like me?

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

## What about prototype OO?

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

**But I didn't know  
how to answer.**



# Chapter 2

**jnthn tries to  
implement  
Perl 6 OO**

**Rakudo  
development is  
generally  
breadth-first.**



# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

---

# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**

**Implementation  
Awesomeness**

**Hey nice!**

**Better than  
stroopwafels**



# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**

**Implementation  
Awesomeness**

**Hey nice!**

**Better than  
stropwafels**



**Before we  
called it Rakudo**

# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**

**Implementation  
Awesomeness**

**Hey nice!**

**Better than  
stropwafels**



**Hack**

# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**



**Hack hack**

**Hey nice!**

**Better than  
stropwafels**



**Implementation  
Awesomeness**

# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**

**Implementation  
Awesomeness**

**Hey nice!**

**Better than  
stropwafels**



**Hack hack hack**

# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**

**Implementation  
Awesomeness**

**Hey nice!**

**Better than  
stropwafels**



**OMG new grammar  
engine!**

# A Little Reflection On Metamodels

**Feature**

**OO**

**Regexes**

**Built-ins**

**So Crap**

**Meh**

**Not so bad**

**Implementation  
Awesomeness**

**Hey nice!**

**Better than  
stropwafels**



**Mmm....beer  
and hacking!**



**Means you get a  
compiler with  
some coverage of  
many features...**

**...meaning  
that people can  
start to write  
programs...**

**...and then  
complete and  
improve features  
over time.**

# A Little Reflection On Metamodels

STD.pm

```
...
token package_declarator:class {
    :my $*PKGDECL := 'class';
    <sym> <package_def>
}
token package_declarator:grammar {
    :my $*PKGDECL := 'grammar';
    <sym> <package_def>
}
token package_declarator:role {
    :my $*PKGDECL := 'role';
    <sym> <package_def>
}
...
```

# A Little Reflection On Metamodels

STD.pm

```
...
token package_declarator: class {
    :my $*PKGDECL := 'class';
    <sym> <package_def>
}
token package_declarator: grammar {
    :my $*PKGDECL := 'grammar';
    <sym> <package_def>
}
token package_declarator: role {
    :my $*PKGDECL := 'role';
    <sym> <package_def>
}
...
```

**First cut(s):  
needed something  
that works, so  
fairly hard coded.**

# A Little Reflection On Metamodels



**But...**

**Having the details  
all hard-coded  
bloats the  
compiler**



**But...**

**Not extensible,  
so no way to add  
more package  
types in future**

# Chapter 3

**Metamodels  
to the  
rescue!**

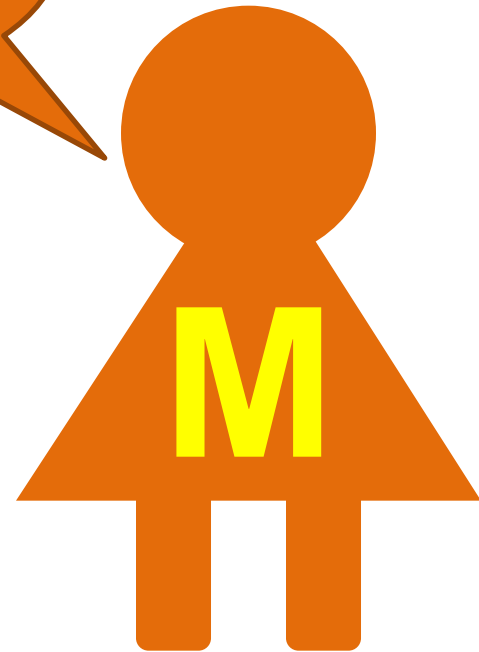
**Just when I thought I'd  
never work it out...**



**...along came  
Metamodel Man!**



**OH HAI!**



**He gave me knowledge of  
the wonder of metamodels.**



**So what is a  
metamodel?**

**Can understand  
a word from  
its parts.**

# Politics



# Politics

**Poli**

**tics**

# Politics

**Poli**

**tics**



**Latin**

# Politics

**Poli**

**tics**



**Latin**

**Many**

**Blood  
sucking  
creatures**

# Metamodel

# Metamodel

**Meta                      model**

# Metamodel

**Meta**                      **model**

Because I said so

# Metamodel

**Meta model**

**Because I said so**

**Things  
that  
describe...**

**...objects  
in our  
system.**

**Each package type  
maps to some “meta-  
package” type**

**class => ClassHOW**

**role => RoleHOW**



# Compile a class definition...

```
class Stroopwafel is Cake {  
  has $!area;  
  has $.filling;  
  method eat() {  
    for 1..$area {  
      say "om nom nom nom nom";  
    }  
  }  
}
```

**...to calls on a meta-class instance.**

```
my $temp = ClassHOW.new('Stroopwafel');
trait_mod:<is>($temp, Cake);
$temp.^add_attribute(Attribute.new(
    name => '$!filling', has_accessor => True
));
$temp.^add_attribute(Attribute.new(
    name => '$!area`
));
$temp.^add_method('eat', method () {
    ...
});
my $type-object = $temp.^compose();
```

**The semantics of a class are whatever the ClassHOW metaclass decides that they are.**

**Differences between package types can be encapsulated in the meta-packages.**

**Means that  
programmers are able  
to create their own  
types of package  
cleanly.**

**...and they  
all lived  
hackily  
ever after.**

**The End**