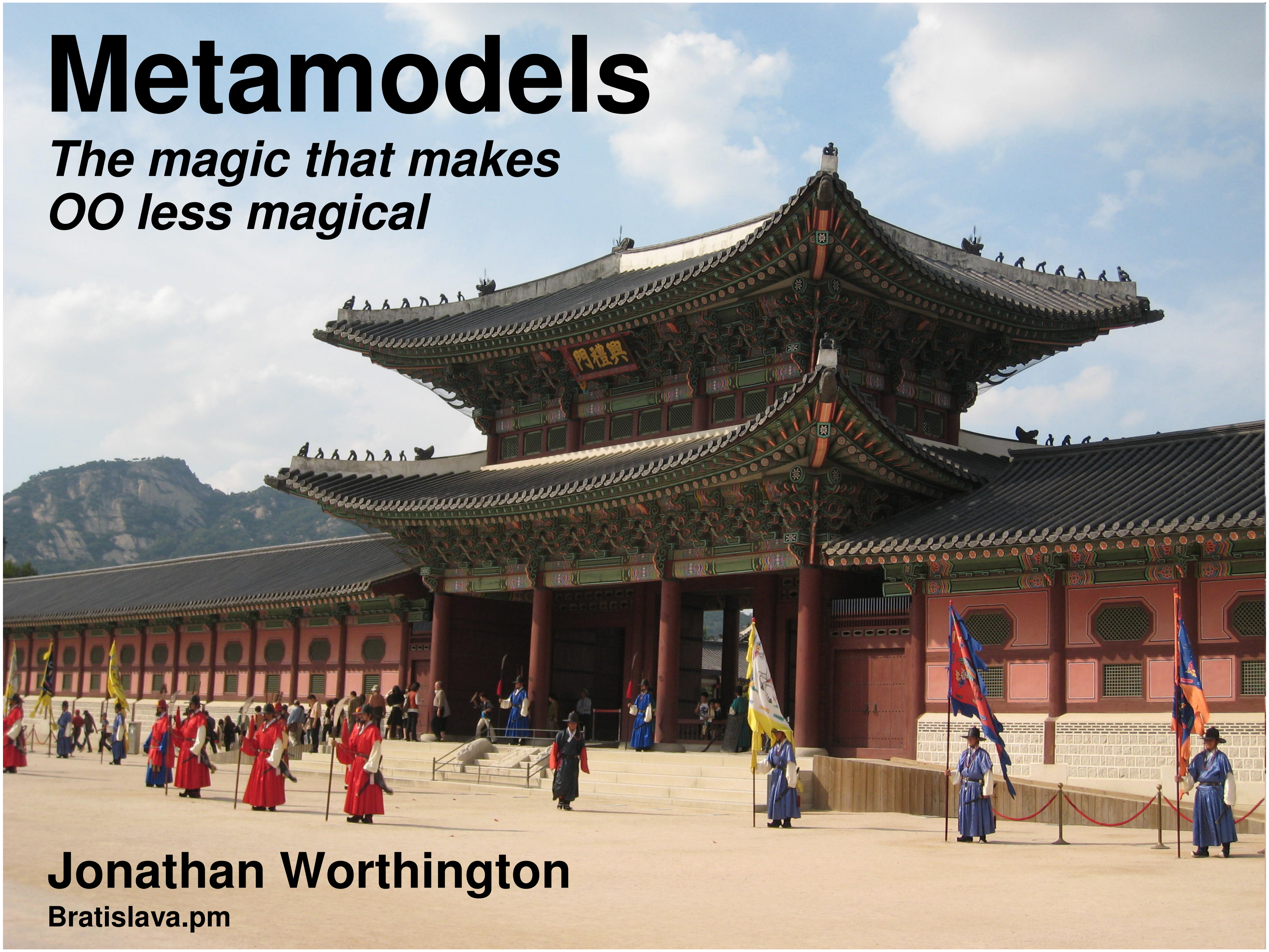

Metamodels

The magic that makes OO less magical

Jonathan Worthington

Bratislava.pm

OH HAI

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

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomale);
        }
    }
}
```

…and things were OK.

I thought my work was done and I could go to the bar, but…

…then my class started asking me some hard questions.

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

How was I created?

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

What does it mean to have methods and attributes?

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

What does it mean to have a parent?

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

Do all classes behave the same way I do?

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

What does it mean to be a class anyway?

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

What about languages that do OO without classes?

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        while outside() {
            self.sniff;
            say("HAU HAU HAU!");
            self.move(:pomaly);
        }
    }
}
```

…and can a class drink beer too?

Metamodels – *the magic that makes OO less magical*

Huh?!

But actually, they are good questions.

Metamodels – *the magic that makes OO less magical*

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>
}
…
```

Metamodels – *the magic that makes OO less magical*

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>
}
…
```

Roles and classes have many things in common (methods, attributes, ...)

grammar

=

class + inherit from Grammar by default

My first cut: just hardwire the differences

Metamodels – *the magic that makes OO less magical*

My first cut: just hardwire the differences.

FAIL

Make the easy things easy and the hard things <u>possible</u>.

Declaring a class in Perl 6

=

Easy! ☺

Adding a new package type in the future

=

Should be possible ☹

Metamodels to the rescue!

What is "meta"?

Something that <u>describes</u> something else.

Natural languages can be used as meta-languages. The have words to describe language.

Word

Sentence

Verb

Adjective

Case

Meta-class

=

something that describes a class

Meta-objects

=

objects that we use to describe our object model

Metamodels – *the magic that makes OO less magical*

```
class Dog is Animal {
    has $.name;
    has $!brain;
    method go_for_walk() {
        …
    }
}
```

```
my $meta := ClassHOW.new;
ClassHOW.add_attribute($meta, Attribute.new(
    name => '$!name',
    accessor => True
));
ClassHOW.add_attribute($meta, Attribute.new(
    name => '$!brain',
    accessor => False
));
ClassHOW.add_method($meta, 'go_for_walk', method () { … });
ClassHOW.add_parent($meta, Animal);
my $type_object := ClassHOW.compose($meta);
```

MOP

=

Meta-object Protocol

=

API our meta-objects should implement

Metamodels – *the magic that makes OO less magical*

ClassHOW is just a class implementing a bunch of methods related to building up a class declaration, according to a standard API (e.g. our Meta-object Protocol)

```
my $meta := ClassHOW.new;
ClassHOW.add_attribute($meta, Attribute.new(
    name => '$!name',
    accessor => True
));
ClassHOW.add_attribute($meta, Attribute.new(
    name => '$!brain',
    accessor => False
));
ClassHOW.add_method($meta, 'go_for_walk', method () { … });
ClassHOW.add_parent($meta, Animal);
$type_object := ClassHOW.compose($meta);
```

Metamodels – *the magic that makes OO less magical*

Had I declared a role instead, this only changes in one place. The differences between classes and roles are encapsulated in the meta-object.

```
my $meta := RoleHOW.new;
ClassHOW.add_attribute($meta, Attribute.new(
    name => '$!name',
    accessor => True
));
ClassHOW.add_attribute($meta, Attribute.new(
    name => '$!brain',
    accessor => False
));
ClassHOW.add_method($meta, 'go_for_walk', method () { … });
ClassHOW.add_parent($meta, Animal);
my $type_object := ClassHOW.compose($meta);
```

GrammarHOW

=

just a subclass of ClassHOW that sets Grammar as the default parent ☺

The parser has a hash of the mappings from package declarators to meta-classes.

```
my %*HOW;
%*HOW<class>    := 'ClassHOW';
%*HOW<grammar> := 'GrammarHOW';
%*HOW<role>     := 'RoleHOW';
```

You temporize and modify the hash when declaring a sub-language...and you're done.

The meta-class API also includes methods for introspection.

```
for Dog.^attributes -> $attr {
    say "Class has attribute " ~ $attr.name;
}
for Dog.^methods(:local) -> $meth {
    say "Class has method " ~ $meth.name;
}
```

```
Class has attribute $!name
Class has attribute $!brain
Class has method name
Class has method go_for_walk
```

We can also have sub-protocols for defining other bits of our object model…

Attribute Sub-protocol

Defines how accessor generation is done, and allows for attribute introspection.

Composition Sub-protocol

Defines how role composition takes place and how conflicts are resolved.

Harder Problems

Meta-circularity

(Solvable, just a little mind-bending ☺)

The metaclass should just be a normal object that is also described by a metaclass. All metaclasses are "first class", as such

Interoperability

(Difficult problem; topic of ongoing research)

What happens when I inherit from something with a different meta-class?

Can we get incompatibilities?

Keeping It Sane

Want to try and avoid limiting what's possible in the future…

…without creating an excessively complex object meta-model.

Questions?

Ďakujem ☺