

A Foundation for Multi-Level Modelling

Tony Clark¹
Cesar Gonzalez-Perez²
Brian Henderson-Sellers³

¹Middlesex University, UK

²Institute of Heritage Sciences Santiago de Compostela, Spain

³University of Technology, Sydney, Australia

September 28, 2014

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

- System Modelling is used in many industries.
- One-size modelling technology does not fit all.
- Move towards domain-specificity.
- No consensus on meta-technology.
- Require a foundation for *specifying* languages.
- Strict meta-modelling is too restrictive.
- Proposals for power-types and clabjects.

Technology Requirements

- **Meta-circularity:** it is a self-describing language, semantics is part of a language definition, arbitrary levels - *fractal*.
- **Uniformity:** tools run over multiple levels, different languages can be integrated (where meaningful).
- **Extensibility:** modular language definitions.
- **Views:** syntax can be domain specific.

Technology Requirements

- Meta-circularity: it is a self-describing language, semantics is part of a language definition, arbitrary levels - *fractal*.
- Uniformity: tools run over multiple levels, different languages can be integrated (where meaningful).
- Extensibility: modular language definitions.
- Views: syntax can be domain specific.

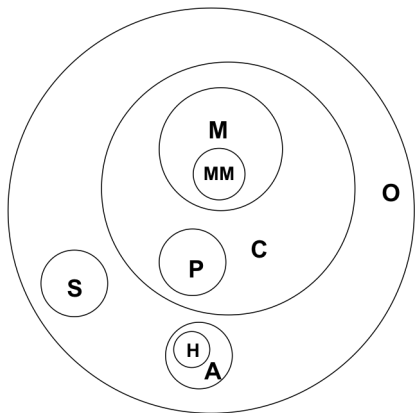
Technology Requirements

- Meta-circularity: it is a self-describing language, semantics is part of a language definition, arbitrary levels - *fractal*.
- Uniformity: tools run over multiple levels, different languages can be integrated (where meaningful).
- Extensibility: modular language definitions.
- Views: syntax can be domain specific.

Technology Requirements

- Meta-circularity: it is a self-describing language, semantics is part of a language definition, arbitrary levels - *fractal*.
- Uniformity: tools run over multiple levels, different languages can be integrated (where meaningful).
- Extensibility: modular language definitions.
- Views: syntax can be domain specific.

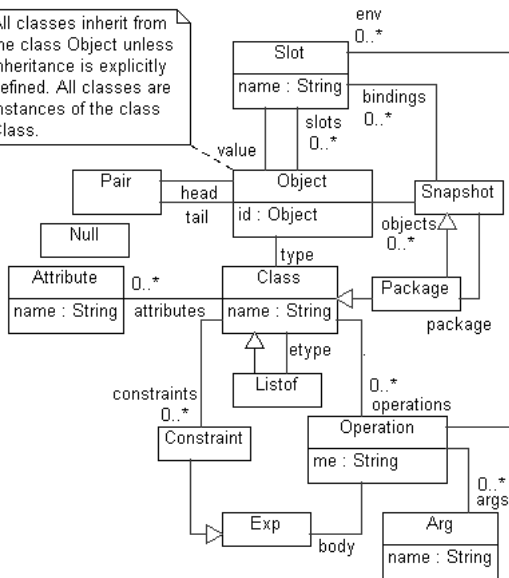
Claim: Everything is an Object



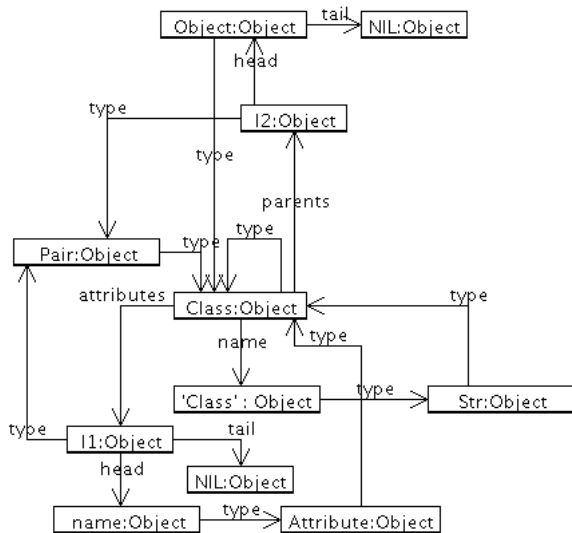
- O = Objects
- C = Classes
- M = Meta-classes
- MM = Meta-Meta-classes
- P = Packages
- S = Snapshots
- A = Animals
- H = Herbivores

A Meta-Modelling Kernel

All classes inherit from the class Object unless inheritance is explicitly defined. All classes are instances of the class Class.



A Meta-Modelling Kernel Without the Goggles



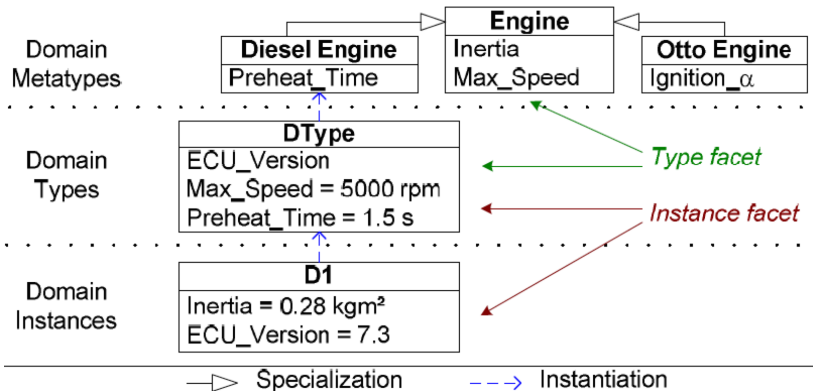
Self Description

```
class Class {
  name      : Str;
  supers    : [Class];
  attributes : [Attribute];
  operations : [Binding];
  constraints : [Constraint]
  operations {
    supers() = [self]+[c | p ← supers; c ← p.supers()].remDups()
    ↑(c) = supers().∃(c)
    atts() = [a | c ← supers(), a ← c.attributes]
    ops() = [b | c ← supers(), b ← c.operations]
    cond() = [a | c ← supers(), a ← c.constraints]
    ::(n, d) = atts().←(λ(n' ↦ a) n' = n, λ(n ↦ a) a,
                    ops().←(λ(n' ↦ o) n' = n, λ(n ↦ o) o, d))
    ?(o) = o.type.↑(self) and
          atts().∀(λ(a) o.slots.∃(λ(s) s.name = a.name and
                                a.type.?(s.value))) and
          cond().∀(λ(c) c.eval([self ↦ o] +
                               [s.name ↦ s.value | s ← o.slots]))
  }
}
```


Self Description

```
class Object {  
  id      : Object;  
  type    : Class;  
  slots   : [Slot]  
  constraints { type.?(self) }  
  operations {  
    dot(n) = slots.⋈(λ(n' ↦ _) n=n', λ(_ ↦ v) v, error)  
    send(n, args) =  
      type.ops().⋈(λ(n' ↦ (Operation) [args ↦ args'])  
        n=n' and #args = #args',  
        λ(_ ↦ f) f.invoke(self, args),  
        error)  
  }  
}
```

Validation: Language Definition and Use



From: Thomas Aschauer, Gerd Dauenhauer, and Wolfgang Pree. Representation and traversal of large clabject models. In Model Driven Engineering Languages and Systems, pages 17–31. Springer, 2009.

Language Definition

```
package CKernel:Kernel extends Kernel {  
  class CAtt extends Attribute {  
    level:Integer;  
  }  
  class CClass extends Class {  
    operations {  
      atts() = catts(1,self)  
      catts(n,c=(_,c)[]) = []  
      catts(n,c) =  
        [a | a ← c.atts(),  
          ?a.type=CAtt,a.level=n] +  
        catts(n+1,c.type)  
    }  
    constraints {atts.∀(λ(a)a.type=CAtt)}  
  }  
}  
  
snapshot DomainInstances:DomainTypes {  
  (DType) [inertia ↦ 0.28;ECU_version ↦ 7.3]  
}
```

Language Use

```
package DomainMetaTypes:CKernel {  
  class Engine:CClass extends CClass {  
    inertia[2]:Float;  
    max_speed[1]:Integer  
  }  
  class DieselEngine:CClass extends Engine {  
    preheat_time[1]:Float  
  }  
  class OttoEngine:CClass extends Engine {  
    ignition_alpha[1]:Float  
  }  
}  
package DomainTypes:DomainMetaTypes {  
  class DType:DieselEngine {  
    ECU_version[1]:Float;  
    max_speed=5000;  
    preheat_time=1.5  
  }  
}
```

Validation: Practicality

The screenshot displays the GUI2-XMF2 IDE interface. The main editor shows XMF code for a class diagram. The left sidebar shows a project tree with a 'Kernel' folder containing various XMF files. The bottom-left pane shows the 'Property:Set(Element)' editor, and the bottom-right pane shows the 'Console' window.

```
131
132 context SetOfElement
133 // Produce the intersection of two sets.
134 @Operation intersection(set:Set(Element)):Set(Element)
135 @Doc
136 Returns the intersection of two sets.
137 end
138 if self->isEmpty
139 then self
140 else let value = self->sel
141 in if set->includes(value)
142 then self->excluding(value)->intersection(set)->including(value)
143 else self->excluding(value)->intersection(set)
144 end
145 end
146 end
147 end
148
149 context SetOfElement
150 @Operation isEmpty():Boolean
```

Property:Set(Element)

self	Set(Element)
of	Set
isFinal	<input checked="" type="checkbox"/>
default	Set{}
elementType	Element
grammar	null
name	Set(Element)
owner	XCore

Console

```
BST 2014)
Version
Type ?h for top level help.
[1] XMF> [Load Init]
0:0:-24:-337 ms ]
Clients.browse();
ModelBrowserCommandInterpreter(null)
[1] XMF> 0
262244
XCore.browse();
ModelBrowserCommandInterpreter(null)
[1] XMF> 0
```

Conclusion

- Proposal is for a self-describing, fractal, specification language for modelling languages.
- Does not suffer from limitations of strict modelling.
- Can specify other proposals such as clabjects, power-types.
- Can form the basis of a practical tool (with extensions).

Conclusion

- Proposal is for a self-describing, fractal, specification language for modelling languages.
- Does not suffer from limitations of strict modelling.
- Can specify other proposals such as clabjects, power-types.
- Can form the basis of a practical tool (with extensions).

Conclusion

- Proposal is for a self-describing, fractal, specification language for modelling languages.
- Does not suffer from limitations of strict modelling.
- Can specify other proposals such as clabjects, power-types.
- Can form the basis of a practical tool (with extensions).

Conclusion

- Proposal is for a self-describing, fractal, specification language for modelling languages.
- Does not suffer from limitations of strict modelling.
- Can specify other proposals such as clabjects, power-types.
- Can form the basis of a practical tool (with extensions).