XMF: A Language for Language Oriented Programming

Tony Clark[1]

[1]School of Engineering and Information Sciences
University of Middlesex

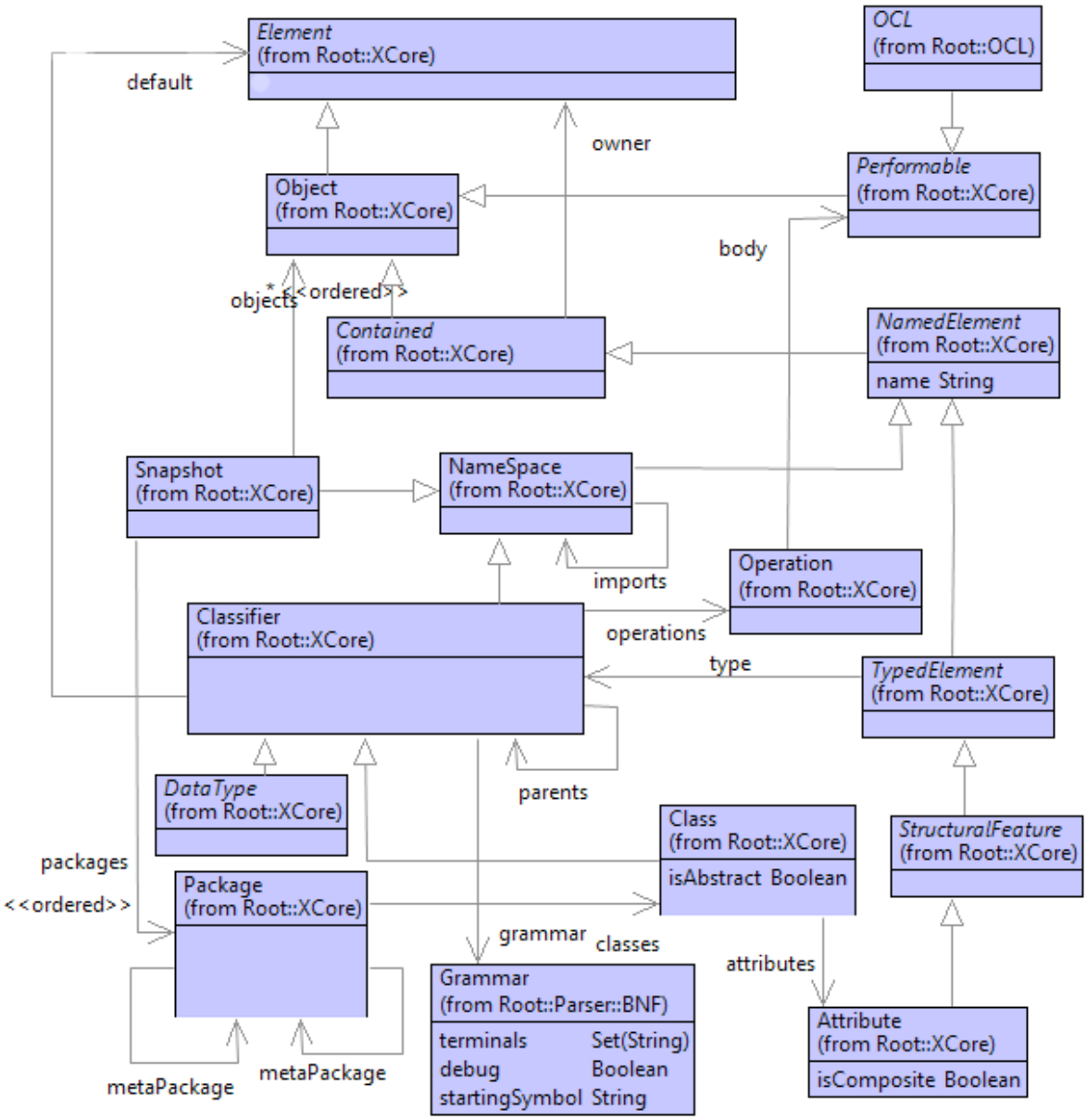
July 25, 2011

Background

- UML 2.0 started around 2000.
- The 2U Submission: UML as family of languages.
- Aim: to merge modelling and programming.
- Tools: MMT, XMT, XMF.
- Programming language based on FP and OCL.
- Important features: meta-; reflection; OO.
- Xactium 2003–2008.

XMF

- Meta-Circular Language (like MOF and ECore).
- XCore Based on ObjvLisp.
- File based or world-state.
- Features for:
  - Packages of models/programs.
  - Higher-order operations.
  - OCL.
  - Meta-Object Prototcol (MOP).
  - Language Engineering (grammars, syntax processing).
  - Daemons (object listeners).
  - Pattern matching.
  - Code generation templates.
  - Threads.
  - XML processing (parsing).
  - Java integration.

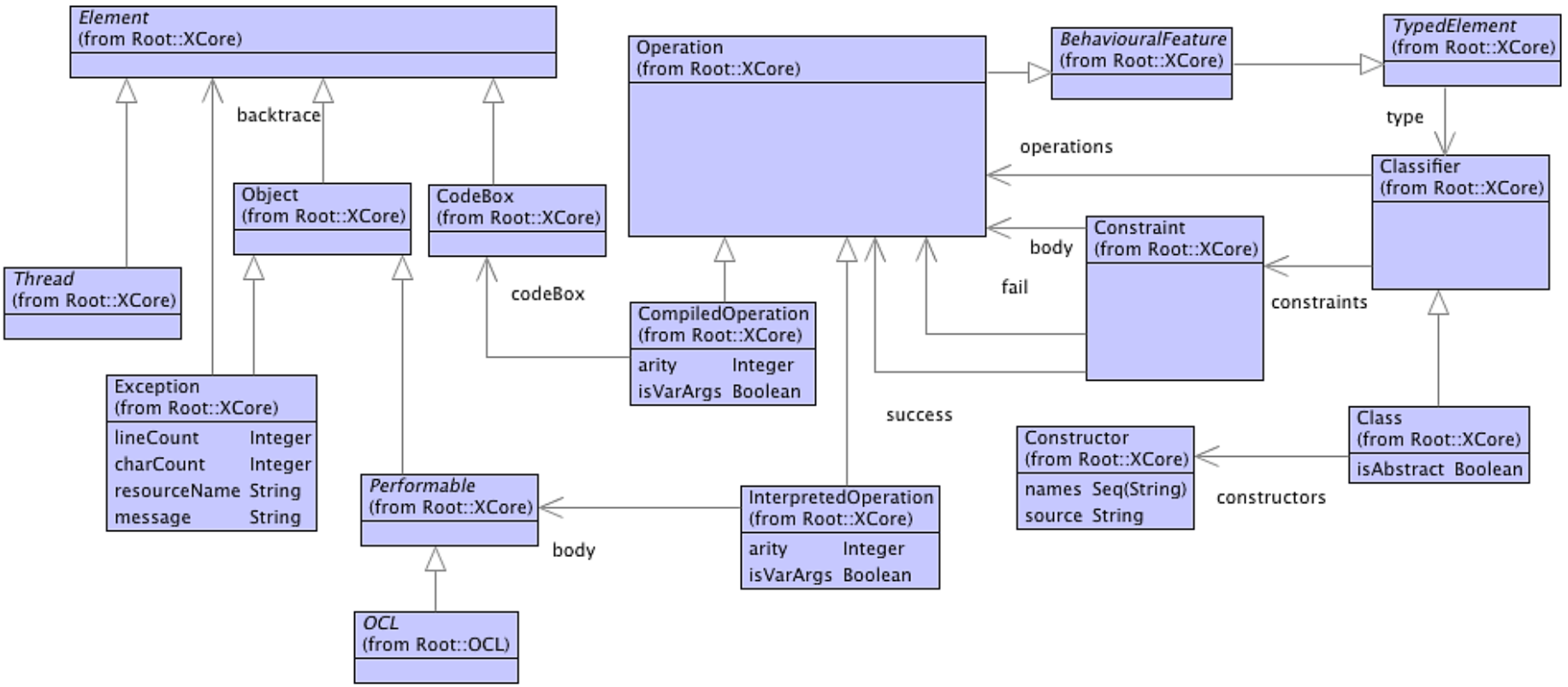


Meta-language

Models

```
1  context Root
2    @Class Request
3      @Attribute id : String end
4      @Constructor(id) ! end
5    end
6
7  context Root
8    @Class RBuffer
9      @Attribute requests : Seq(Request) end
10     @Constructor(requests) ! end
11   end
```

Behaviour


Element
(from Root::XCore)
backtrace
Object
(from Root::XCore)
CodeBox
(from Root::XCore)
Thread
(from Root::XCore)
Exception
(from Root::XCore)
lineCount Integer
charCount Integer
resourceName String
message String
codeBox
Operation
(from Root::XCore)
BehaviouralFeature
(from Root::XCore)
TypedElement
(from Root::XCore)
type
operations
Classifier
(from Root::XCore)
body
fail
Constraint
(from Root::XCore)
constraints
CompiledOperation
(from Root::XCore)
arity Integer
isVarArgs Boolean
success
Class
(from Root::XCore)
isAbstract Boolean
Constructor
(from Root::XCore)
names Seq(String)
source String
constructors
Performable
(from Root::XCore)
body
InterpretedOperation
(from Root::XCore)
arity Integer
isVarArgs Boolean
OCL
(from Root::OCL)

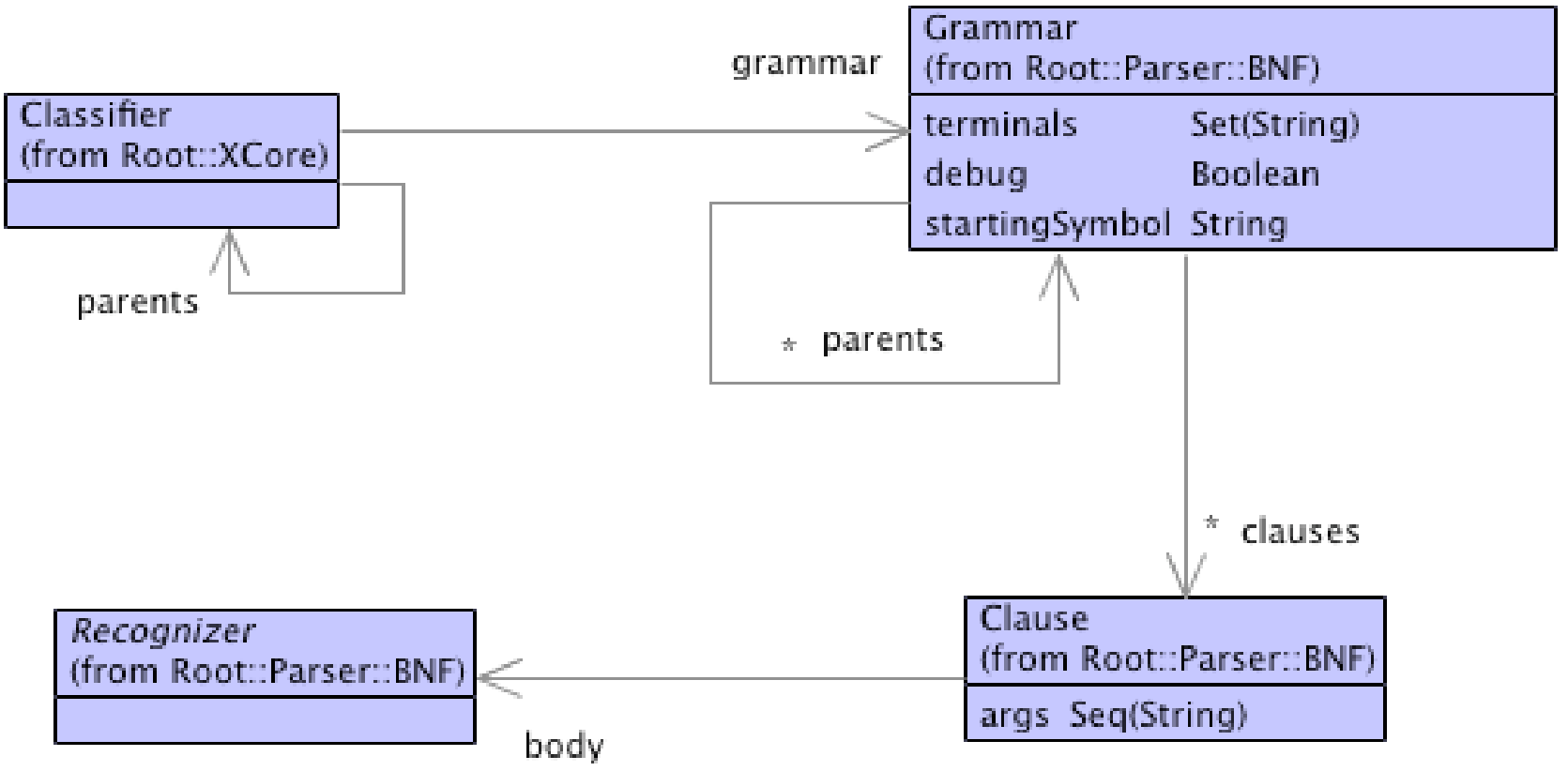
Programs: XOCL

```
1  context RBuffer
2    @Operation add(id:String)
3      self.requests := requests + Seq{Request(id)}
4    end
5
6  context RBuffer
7    @Operation handle(id:String)
8      @Match(requests) {
9        R1 + Seq{Request(${id})} + R2 ->
10         self.requests := R1 + R2,
11       else self.error("No request with id " + id)
12     }
13   end
```

Failure

```
1  context SeqOfElement
2    @Operation removeDups()
3      @Match(self) {
4        S1 + Seq{x} + S2 + Seq{y} + S3 ->
5          if x = y
6          then (S1+S2+S3)->removeDups
7          else fail()
8          end,
9        else self
10     }
11   end
```

Syntax Classes



Quasi-Quotes

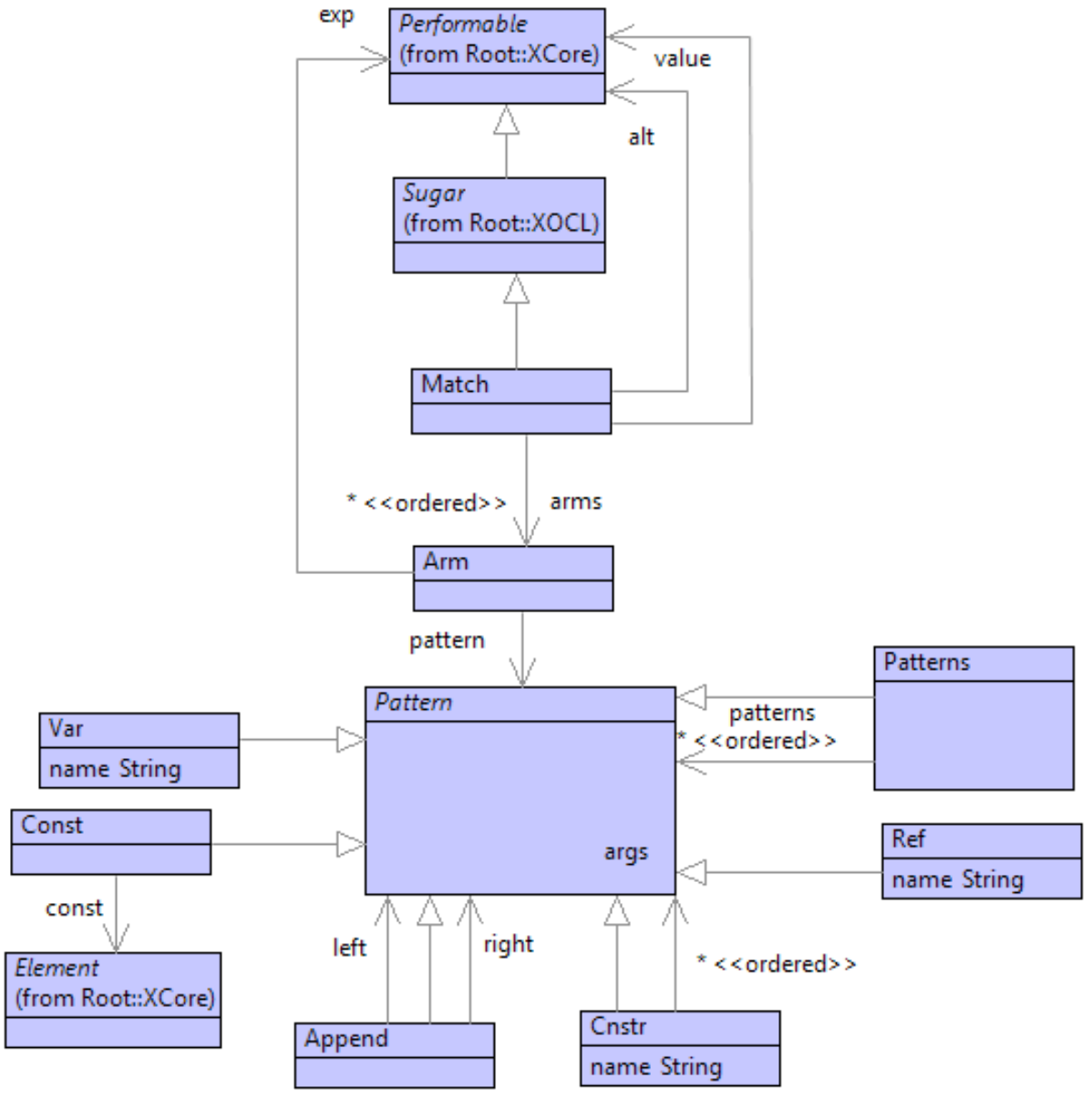
```
1  context Root
2    @Operation add1_exp(exp:Performable):Performable
3      [| 1 + <exp> |]
4    end
5
6  context Root
7    @Operation seq_exp(exps:Seq(Performable)):Performable
8      exps->iterate(e x = [| Seq{} |] |
9        [| <x>->including(<e>) |])
10   end
```

Grammars

```
1  parserImport Parser::BNF;
2  parserImport XOCL;
3
4  Root::g :=
5    @Grammar
6      Start ::= i=Int o=Op j=Int {
7        @Case o of
8          "+" do i + j end
9          "*" do i * j end
10       end
11     }.
12     Op ::= '+' { "+" } | '*' { "*" }.
13   end;
```

```
[1] XMF> g.parseString("1 + 2","Start",Seq{});
3
[1] XMF>
```

Match Language Construct



Match Syntax Class

```
1  context Root
2   @Class Match extends XOCL::Sugar
3     @Attribute value : Performable end
4     @Attribute arms : Seq(Arm) end
5     @Attribute alt : Performable end
6     @Constructor(value,arms,alt) ! end
7     @Grammar extends OCL::OCL.grammar
8       Match ::= e=Exp '{' as=Arm* 'else' a=Exp '}' {
9         Match(e,as,a)
10      }.
11      Arm  ::= p=Pat '->' e=Exp ',' {Arm(p,e)}.
12      Pat  ::= l=Atom ('+' r=Pat {Append(l,r)} | {l}).
13      Pats ::= p=Pat ps=(',' Pat)* {Seq{p|ps}}.
14      Atom ::= i=Int {Const(i)}
15      | '$' '{' n=Name '}' {Ref(n)}
16      | v=Name ('(' ps=Pats ')' {Cnstr(v,ps)} | {Var(v)})
17      | 'Seq{' ps=Pats '}' {Patterns(ps)}.
18    end
19  end
```

Desugar

```
1 context Match
2   @Operation desugar()
3     [| let value = <value>
4        in <arms->iterate(arm exp=[| @Operation()
5                                      <alt>
6                                    end |]
7                           | arm.desugar(exp))>
8        end |]
9   end
```

Arms

```
1 context Arm
2   @Operation desugar(fail:Performable)
3     [| let fail = <fail>
4        in <pattern.match(exp)>
5        end |]
6   end
```

Constants

```
1 context Const
2   @Operation match(succ)
3     [| if value = <const.lift()>
4        then <succ>
5        else fail()
6        end |]
7     end
```

Variables and Refs

```
1  context Var
2    @Operation match(succ)
3      [| let <name> = value in <succ> end |]
4    end
5
6  context Ref
7    @Operation match(succ)
8      [| if value = <OCL::Var(name)>
9         then <succ>
10        else fail()
11        end |]
12   end
```

Splits

```
1  context SeqOfElement
2    @Operation split()
3      (0.to(self->size))->iterate(i pairs=Seq{} |
4         pairs->including(Seq{self->take(i),
5                              self->drop(i)}))
6    end
7
8  context SeqOfElement
9    @Operation select(succ,fail)
10     if self->isEmpty
11     then fail()
12     else succ(self->head,@Operation()
13                            self->tail.select(succ,fail)
14                          end)
15     end
16   end
```

Append

```
1  context Append
2    @Operation match(succ)
3      [| if value.isKindOf(Seq(Element))
4         then
5           value->split.select(
6             @Operation(pair,fail)
7               let value = pair->at(0)
8               in <left.match([| let value = pair->at(1)
9                                 in <right.match(succ)>
10                                end |])>
11              end
12            end,fail)
13        else fail()
14        end |]
15   end
```

Patterns

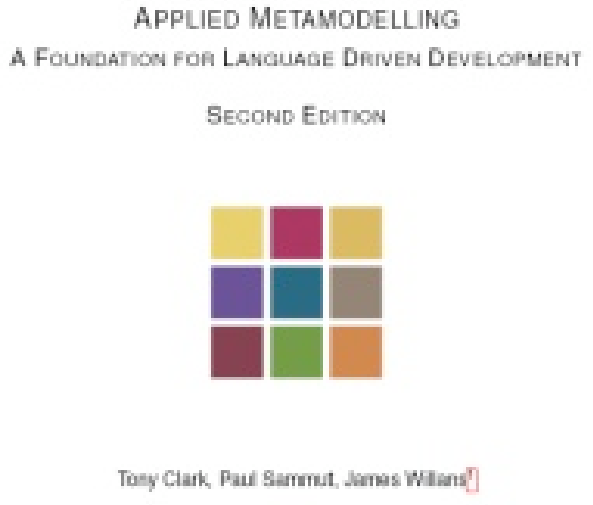
```
1  context Patterns
2   @Operation match(succ)
3     [| if value.isKindOf(Seq(Element)) andthen value->
          size = <patterns->size.lift()>
4        then <(0.to(patterns->size-1))->iterate(i s=succ |
5                [| let value = value->at(<i.lift()>)
6                   in <patterns->at(i).match(succ)>
7                   end |])>
8        else fail()
9        end |]
10  end
```

Constructors

```
1  context Cnstr
2   @Operation match(succ)
3     [| if value.of() = Root.getElement(<name.lift()>)
4        then <let c = Root.getElement(name)
5                           .constructors
6                           ->select(c |
7                             c.names->size =
8                             args->size)
9                           ->asSeq->head
10             in (0.to(args->size-1))->iterate(i exp=succ|
11                  [| let value = value.<c.names->at(i)>
12                     in <args->at(i).match(succ)>
13                     end |])
14             end>
15       else fail()
16       end |]
17    end
18   end
```

Availability, Documentation, Research

http://www.eis.mdx.ac.uk/staffpages/tonyclark/



XMF

The Extensible Programming Language

XPL