Tutorial of the ATL transformation language
http://github.com/jesusc/atl-tutorial

Part V

HIGHER-ORDER TRANSFORMATIONS

jesus.sanchez.cuadrado@gmail.com
@sanchezcuadrado
http://sanchezcuadrado.es

Definition

- *A **higher-order transformation** is a model transformation such that its input and/or output models are themselves transformation models*

- Pre-requisite
  - The transformation program must be expressed as a model, which means:
  - The ATL abstract syntax is defined as a meta-model

Tisi, M., Jouault, F., Fraternali, P., Ceri, S., & Bézivin, J. (2009, June). On the use of higher-order model transformations. ECMFA’09.

HOT and ATL

- Many people have used HOTs
  - Perhaps the most relevant feature of ATL
- Examples
  - Co-evolution
  - Genericity
  - Modularity
  - Model integration

Categories

- Synthesis
  - Input model: any model, but not a transformation
  - Output model: a transformation
  - Example: generate a copier
- Analysis
  - Input model: a transformation
  - Output model: any model, but not a transformation
  - Example: metrics, type checking

Tisi, M., Jouault, F., Fraternali, P., Ceri, S., & Bézivin, J. (2009, June). On the use of higher-order model transformations. ECMFA’09.

Categories

- De(composition)
  - Input model: at least one transformation
  - Output model: at least one transformation
  - Between input and output the #total of transformation is three or greater
  - Example: a superimposer
- Modification
  - Input model: a transformation
  - Output model: a transformation (refactored, changed)
  - Example: add behaviour to record explicit trace links

ATL abstract syntax

- To write a HOT :
  - You need to understand the ATL abstract syntax
- Where is the meta-model?
  - Look for ATL.ecore
    - Plug-in org.eclipse.m2m.atl.dsls
  - Be aware that it does not pass Ecore validation
    - We provide (compatible) variants in anATLyzer
      - ATLStatic.ecore – Fully compatible, without validation errors
      - ATLModified.ecore – Almost compatible reorganization

ATL abstract syntax

- ATL
  - Module
  - Rule structure, bindings
  - Imperative features
- OCL
  - OCL Expression and subclasses
  - OCL Model
  - OCL Model Element
- Primitive types

ATL abstract syntax

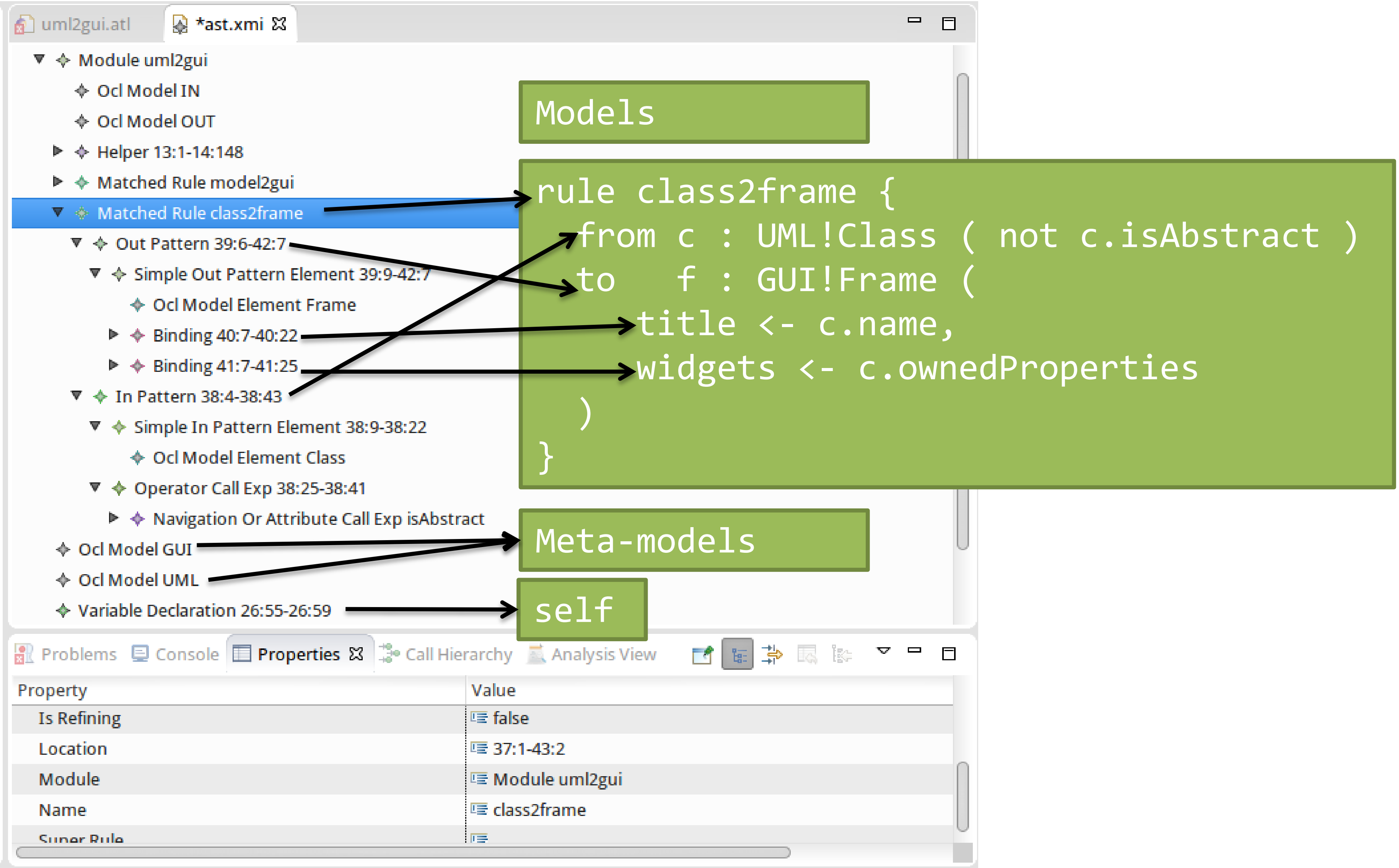
- Best way to learn and understand
  - Serialize a transformation to XMI
    - Use AnATLyzer facility (Right-click -> anATLyzer -> Serialize)
    - Use Ant task:

```
<target name="run">
  <atl.loadModel modelHandler="EMF" name="ATL" metamodel="MOF"  path="ATL.ecore" />
  <atl.loadModel name="ast" metamodel="ATL" path="simple_trafo.atl">
     <injector name="ATL"/>
  </atl.loadModel>

  <atl.saveModel model="ast" path="simple_trafo.xmi" />
</target>
```

  - Explore the model with the tree editor

ATL abstract syntax



- Module uml2gui
  - Ocl Model IN
  - Ocl Model OUT
  - Helper 13:1-14:148
  - Matched Rule model2gui
  - Matched Rule class2frame
    - Out Pattern 39:6-42:7
      - Simple Out Pattern Element 39:9-42:7
        - Ocl Model Element Frame
        - Binding 40:7-40:22
        - Binding 41:7-41:25
    - In Pattern 38:4-38:43
      - Simple In Pattern Element 38:9-38:22
        - Ocl Model Element Class
      - Operator Call Exp 38:25-38:41
        - Navigation Or Attribute Call Exp isAbstract
  - Ocl Model GUI
  - Ocl Model UML
  - Variable Declaration 26:55-26:59

Models

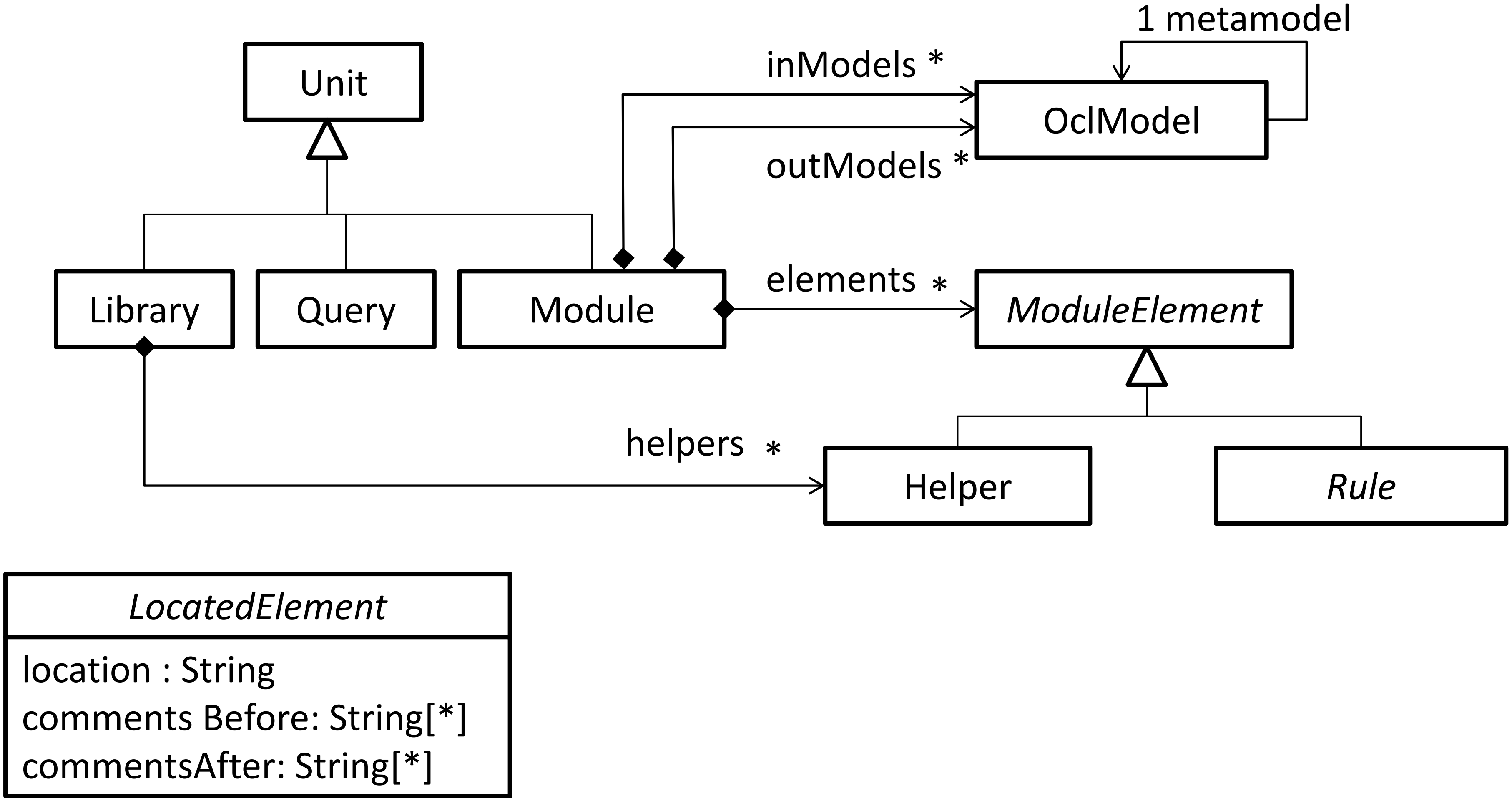
```
rule class2frame {
  from c : UML!Class ( not c.isAbstract )
  to   f : GUI!Frame (
     title <- c.name,
     widgets <- c.ownedProperties
  )
}
```

Meta-models

self

Property	Value
Is Refining	false
Location	37:1-43:2
Module	Module uml2gui
Name	class2frame
Super Rule	

ATL abstract syntax



* Everything inherits from LocatedElement

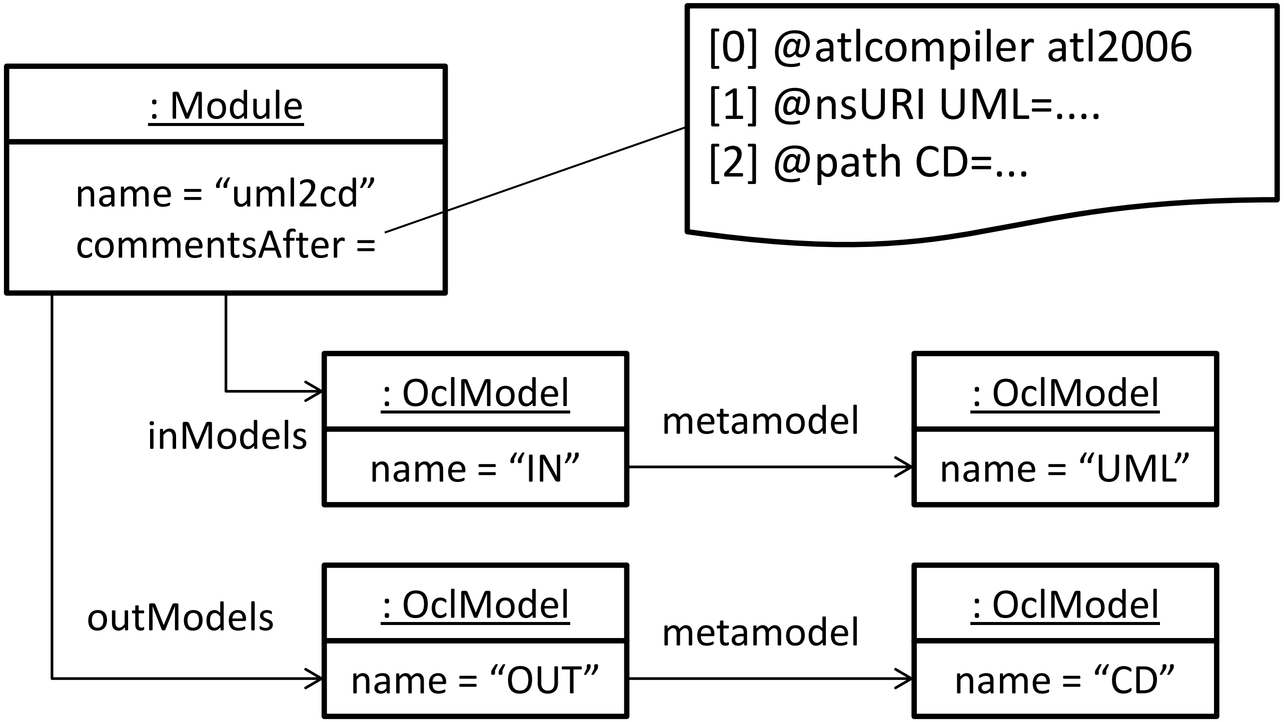
ATL abstract syntax

- Considerations:
  - OclModel.metamodel cardinality is [1]
    - Meta-model is not strongly satisfiable (cannot be instantiated)
  - Location is a string with the format `row:column`
  - `Library` and `Query` do not declare models/meta-models

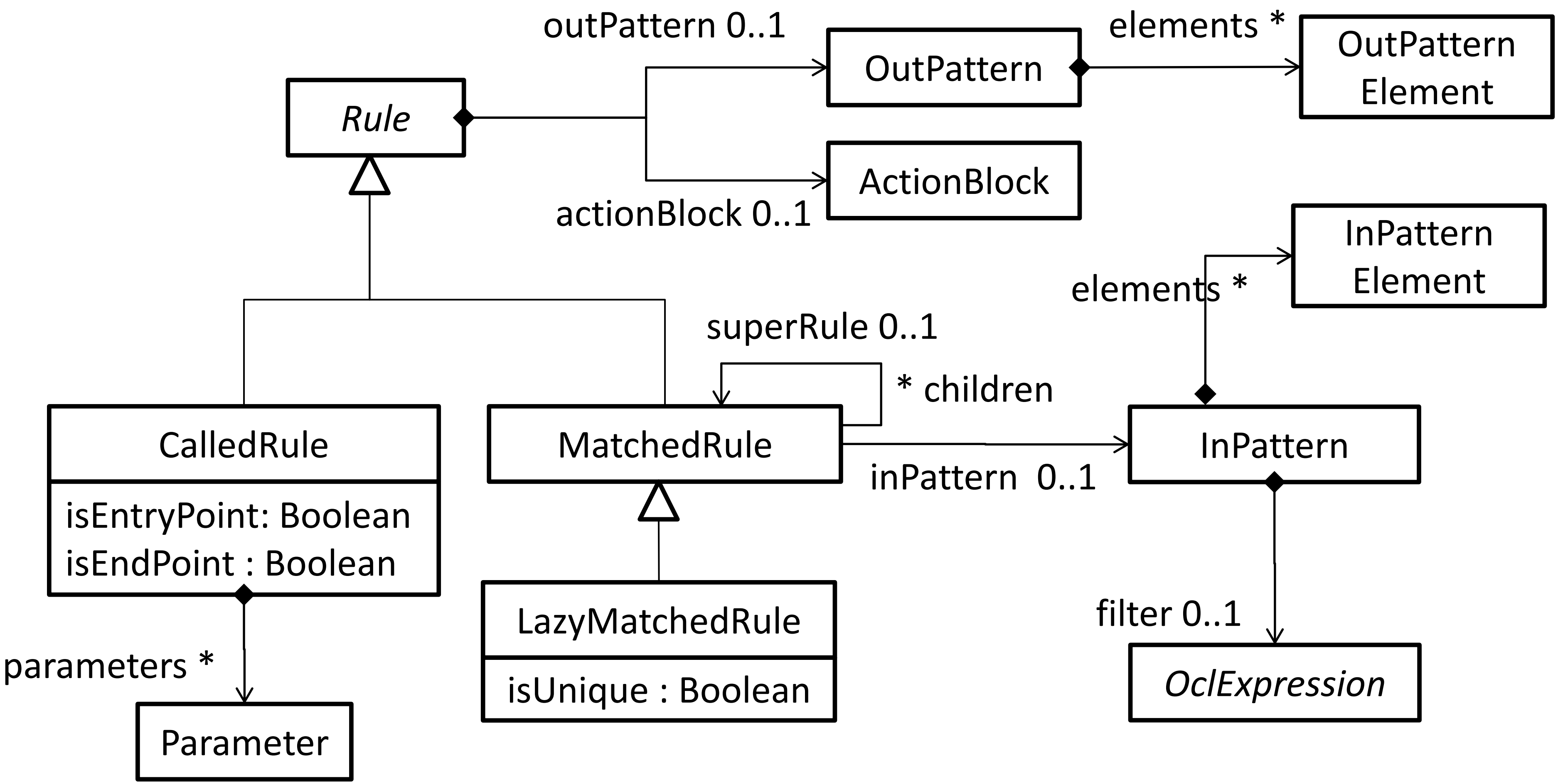
Module abstract syntax

```
-- @atlcompiler atl2006
-- @nsURI UML=http://www.eclipse.org/uml2/5.0.0/UML
-- @path CD=/guigen.trafo.uml2gui/metamodels/cd.ecore

module “uml2cd”;
create OUT : CD from IN : UML;
```



Rule structure

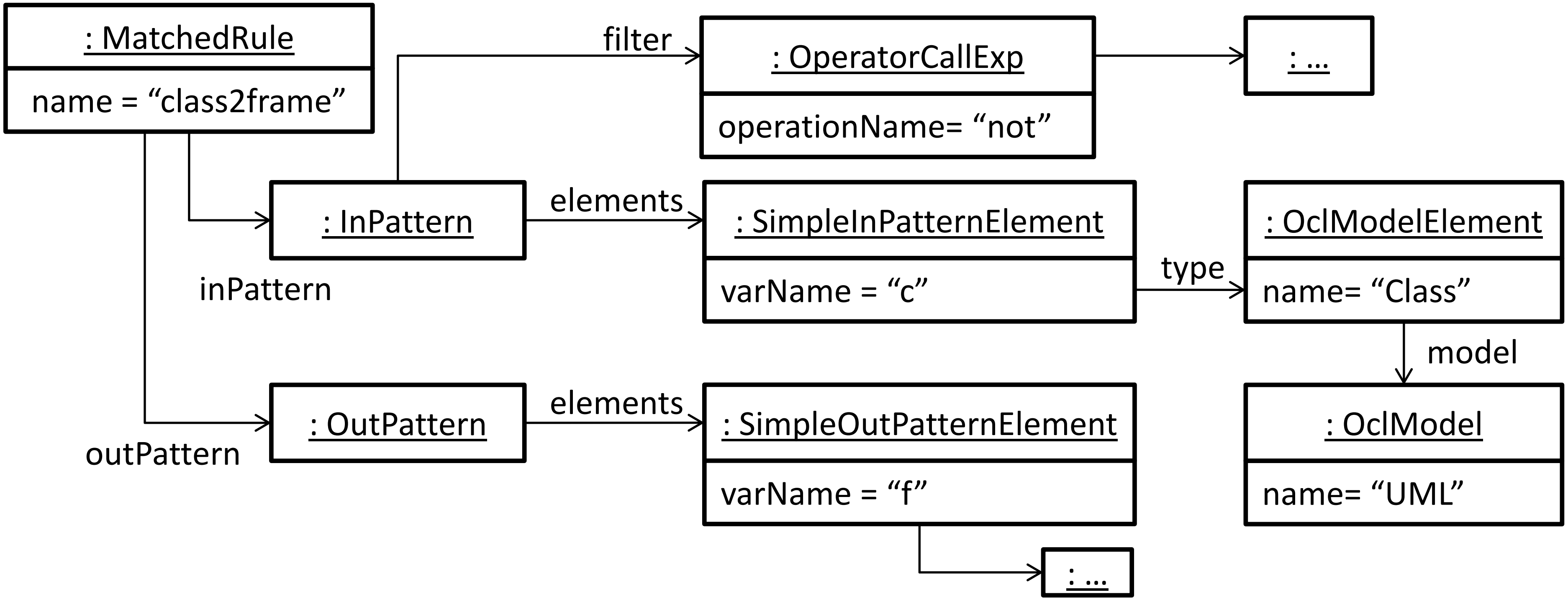


ATL abstract syntax

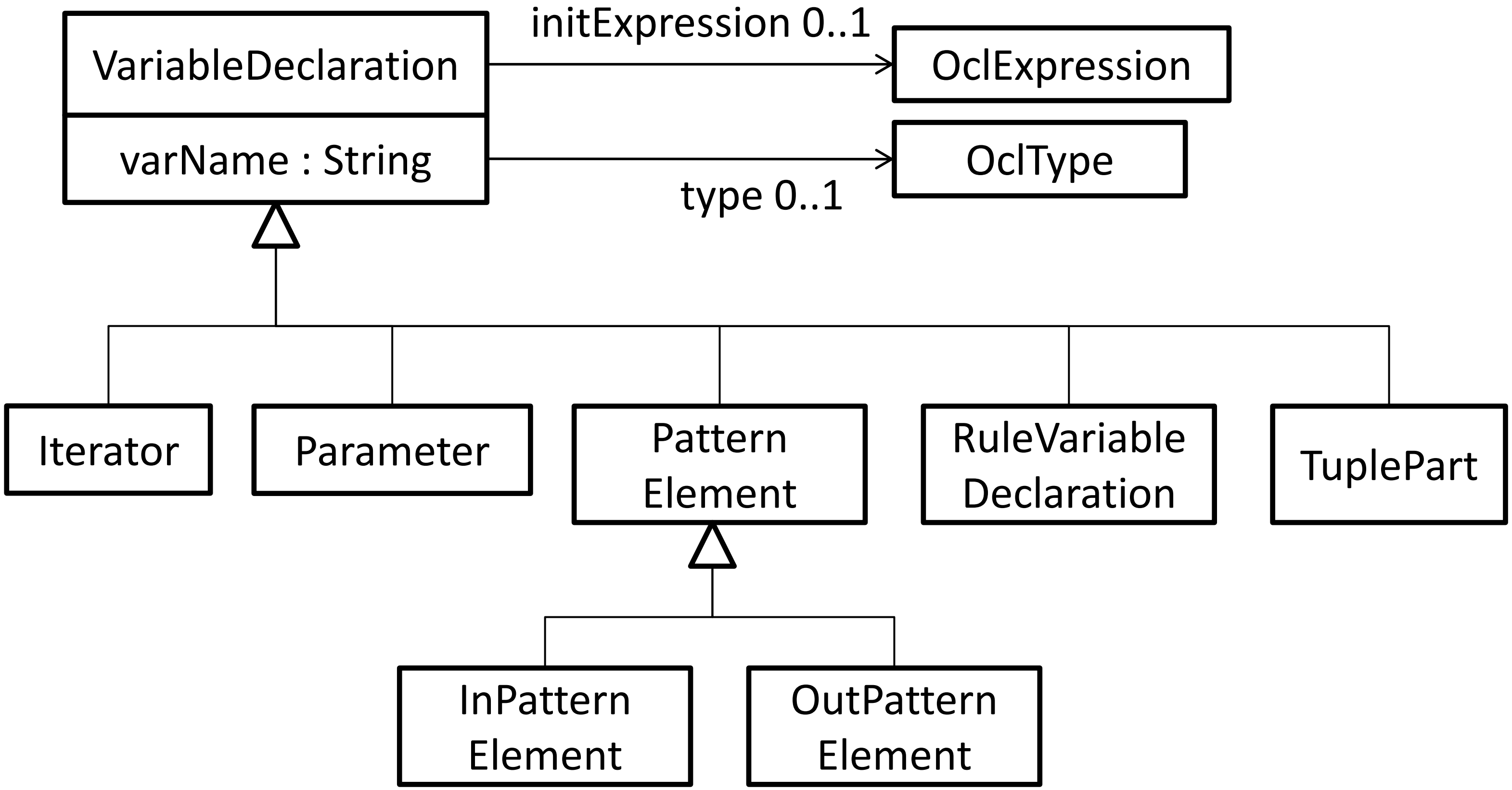
- Considerations:
  - Rule hierarchy is not natural
  - Input and output patterns are optional
  - OclExpression in filter must evaluate to Boolean

Matched rule example

```
rule class2frame {
  from c : UML!Class ( not c.isAbstract )
  to   f : GUI!Frame (
     title <- c.name,
     widgets <- c.ownedAttribute
  ) }
```



Variable declarations



Variable declarations

- Considerations
  - type is optional in VariableDeclaration. In practice it is compulsory in e.g., InPatternElement
  - type must be OclModelElement in InPatternElement
  - initExpression is used in IterateExp and LetExp
  - type is used in InPatternElement, OutPatternElement, Parameter and LetExp

Variable declarations

Parameter

Variable Declaration

```
helper context UML!Class
          def: attrByName(n : String) : UML!Property =
  let attrs : Sequence(UML!Property) = self.ownedAttribute
   in attrs->any(a | a.name = n);
```

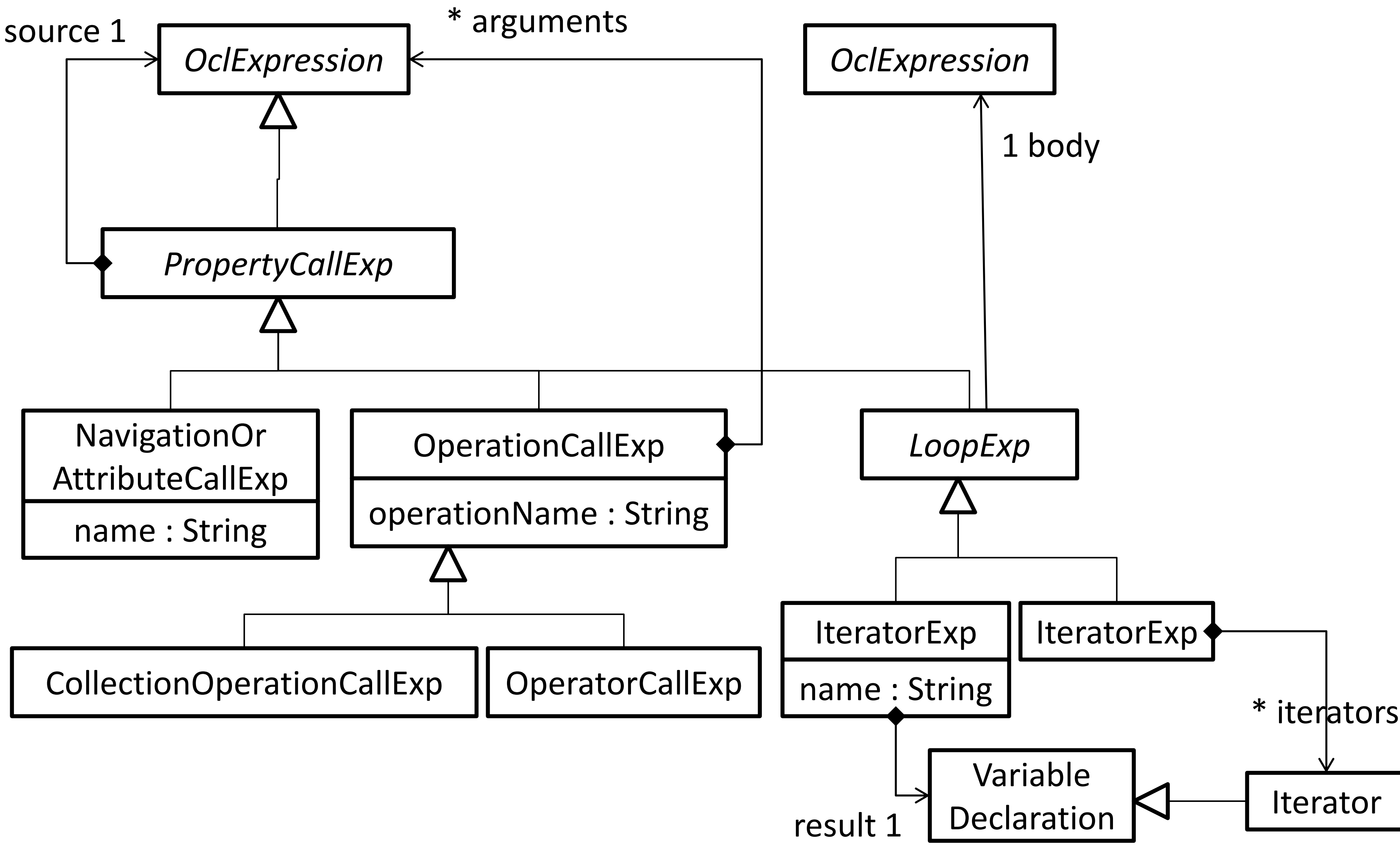
RuleVariable Declaration

```
rule class2frame {
  from c : UML!Class ( not c.isAbstract )
  using {
    attrs : Sequence(UML!Property) = c.ownedAttribute;
  }
  to   f : GUI!Frame (
    widgets <- attrs->
        collect(a | Tuple {class=c, attr = a} )->...
  )
}
```

Iterator

TuplePart

Property calls

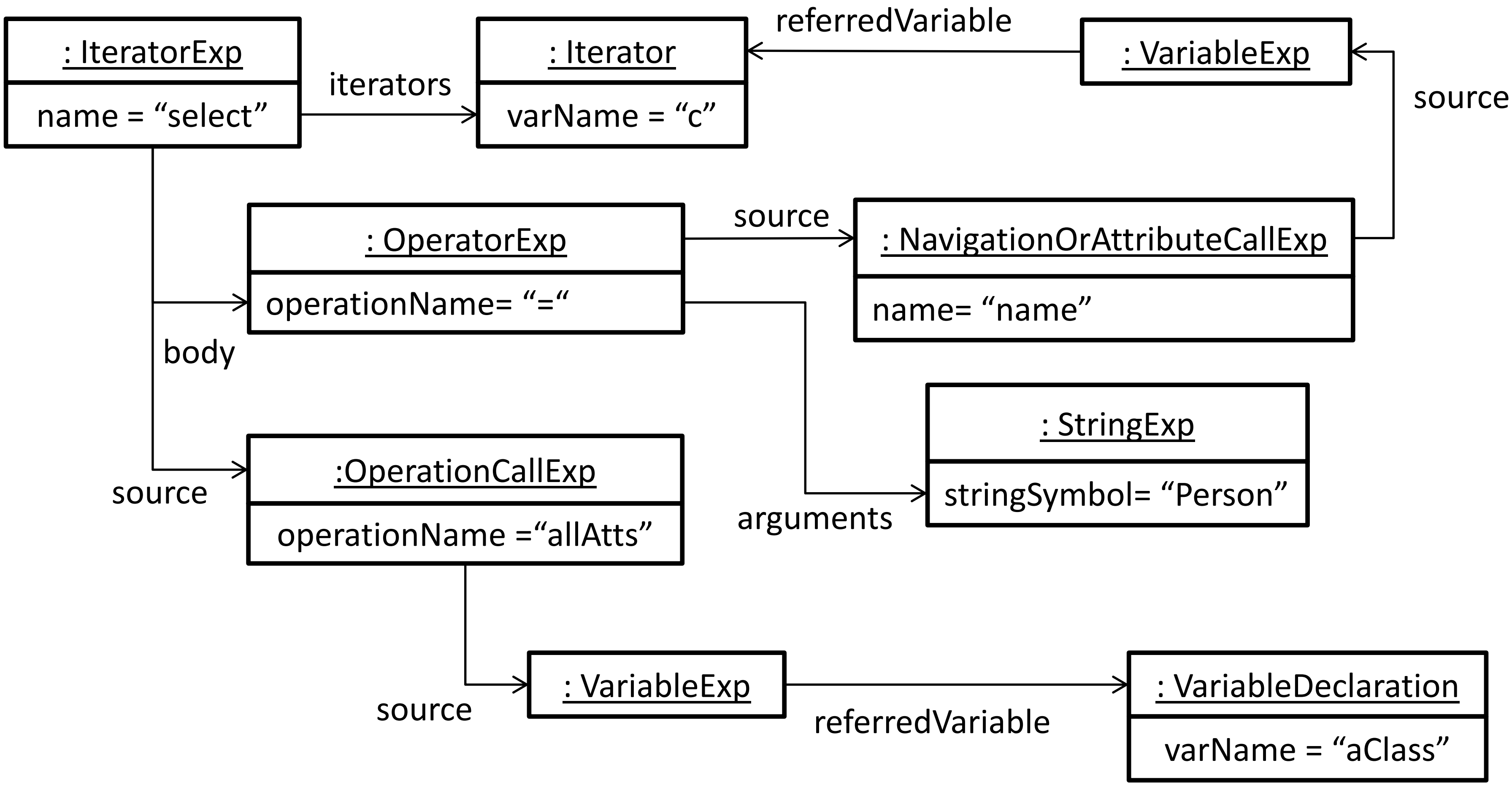


Property calls

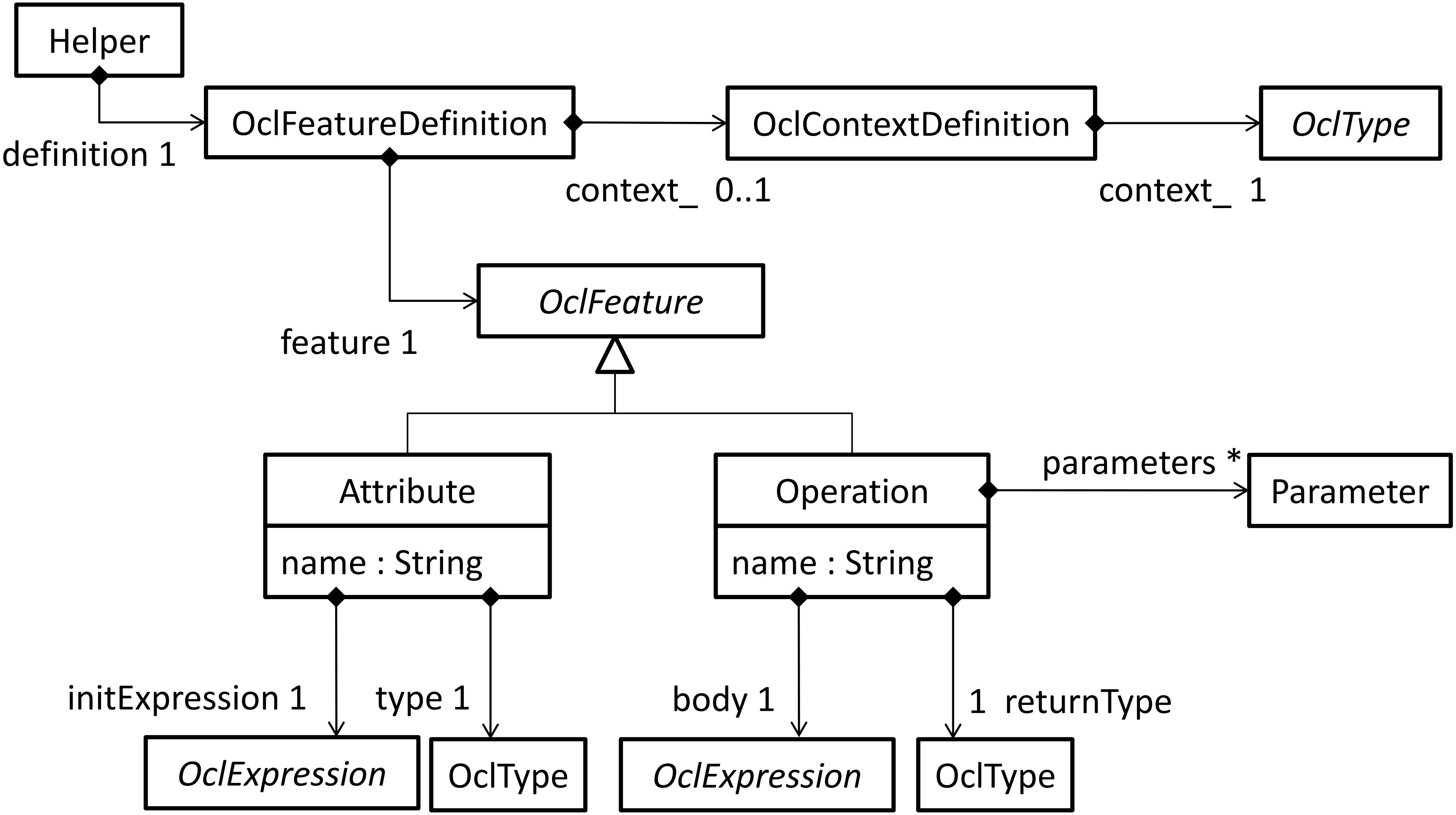
- Considerations
  - Hierarchy is not natural
  - LoopExp syntatically supports many iterators, but in practice only the first one is used
  - Do not confuse IteratorExp with Iterator
  - Expressions are nested via the `source` reference

Property calls

```
aClass.allAtts()->select(c | c.name = ‘Person’)
```



Helpers

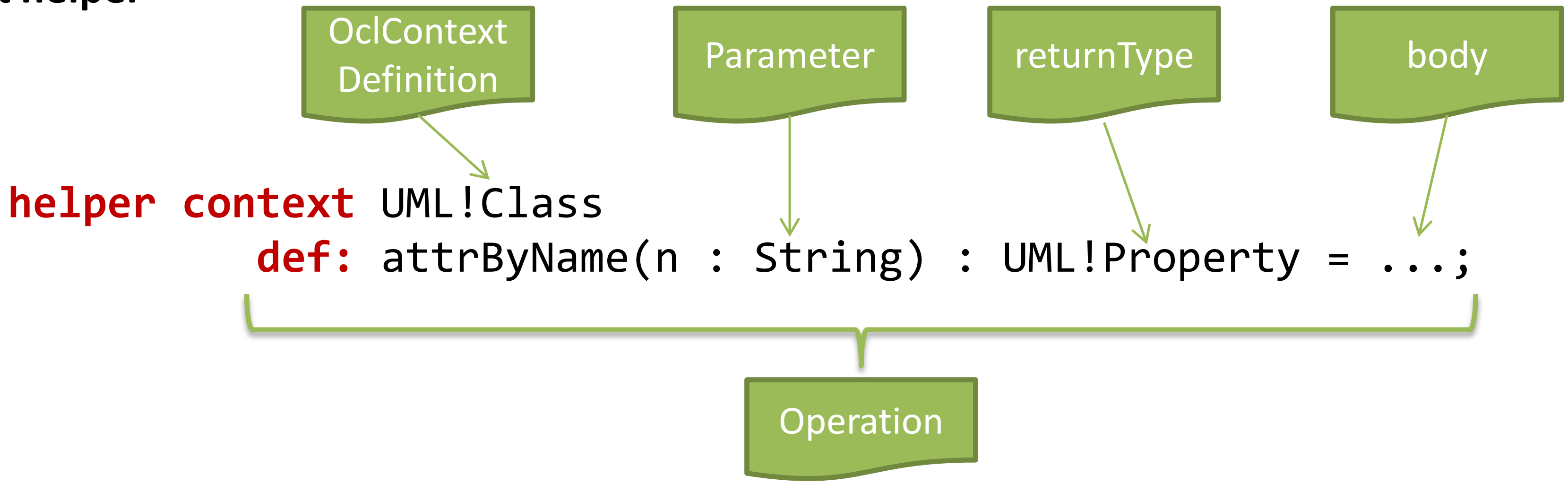

Helper
definition 1
OclFeatureDefinition
context_ 0..1
OclContextDefinition
context_ 1
OclType
feature 1
OclFeature
Attribute
name : String
Operation
name : String
parameters *
Parameter
initExpression 1
type 1
body 1
1 returnType
OclExpression
OclType
OclExpression
OclType

Helpers

- Considerations
  - The structure is sub-optimal
    - Requires many “if” to consider operation vs attribute
  - At the syntax level there is no link to the call sites

Helpers

Context helper



OclContext Definition

Parameter

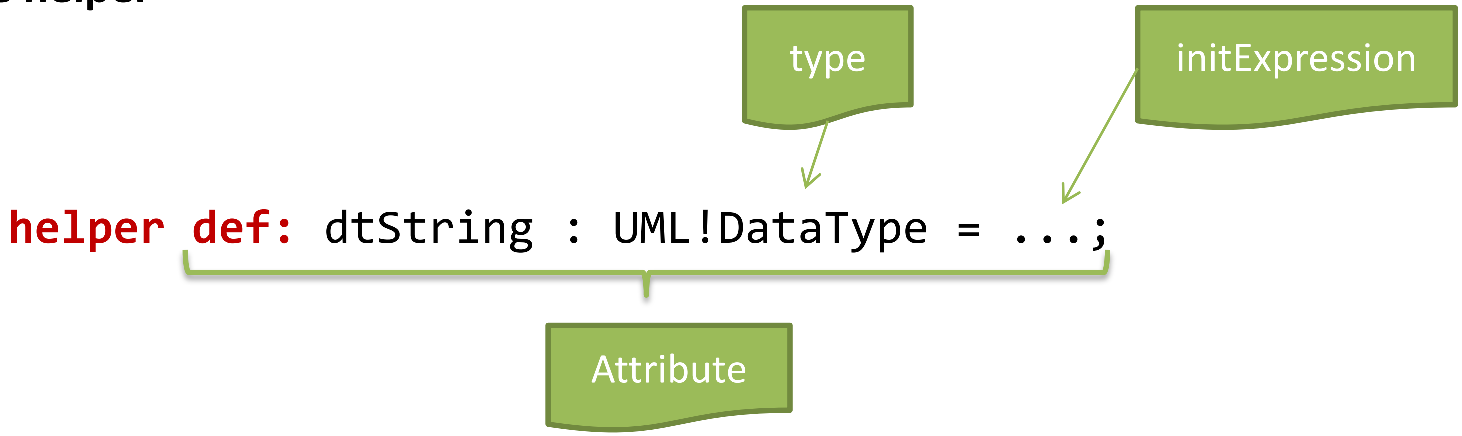
returnType

body

```
helper context UML!Class
         def: attrByName(n : String) : UML!Property = ...;
```

Operation

Module helper



type

initExpression

```
helper def: dtString : UML!DataType = ...;
```

Attribute

Serialization

- Using Ant Tasks

```
<target name="run">
  <atl.loadModel modelHandler="EMF" name="ATL" metamodel="MOF"  path="ATL.ecore" />
  <atl.loadModel name="ast" metamodel="ATL" path="simple_trafo.xmi" />

  <atl.saveModel model="ast" path="simple_trafo.serialized.atl" />
    <extractor name="ATL"/>
  </atl.saveModel>
</target>
```

- Programatically
  - AtlParser class
  - ATLSerialize from anATLyzer

Exercise

- Write a hot to inject debug expressions to visualize the execution flow of any transformation
  - Extend rule filters to output `‘matching <rule-name>’`
  - Extend bindings to output `‘binding <feature-name>’`
  - Remember that `<expr>.debug(‘message’)` returns the original value of <expr>

Exercise

```
rule model2gui {
  from m : CD!Model
    to w : GUI!Window (
       title <- m.name
    )
}
```

⬇

```
rule model2gui {
  from m : CD!Model
   ( true.debug(‘match model2gui’) )
    to w : GUI!Window (
       title <- m.name.debug(‘binding title’),
    )
}
```

`/atl.example.autodebug/autodebug_emftvm.atl`

Exercise

- Complete the code in:
  - Project: `/atl.example.autodebug/`
  - File: `autodebug_emftvm.atl`
- To consider the generation of output messages for bindings