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

Part I

INTRODUCTION TO ATL

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

Motivation

- Some documentation, but not much and a bit disperse
  - http://www.eclipse.org/atl/documentation/
- You have to learn by:
  - Trial and error
  - Code excerpts in papers
  - Looking up the forums
  - Browsing the source code

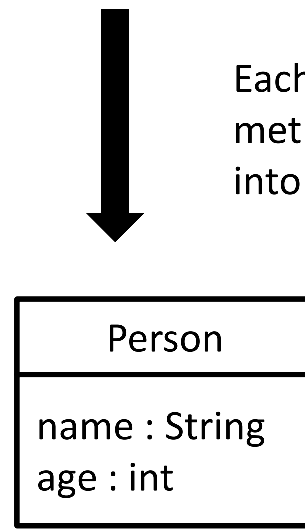
Motivation

Java-like meta-model


```
class Person {
  public void setName(String name) {...}
  public String getName() { ... }

  public void setAge(int v) { ... }
  public int getAge() { ... }
}
```

Each pair of “get” / “set” methods is transformed into a property

UML-like meta-model



Motivation

```
rule JavaClass2Classifier {
   from j : JAVA!JavaClass
     to c : CD!Class (
       name <- j.name,
       features <- j.operations ¿?
     )
}

rule get_set2attribute {
   from get : JAVA!Operation, set : JAVA!Operation (
     get.name.startsWith('get') and set.name.startsWith('set') and
     get.name.substring(3, get.name.size()) =
       set.name.substring(3, get.name.size())
   )
   to feature : CD!Property (
       name <- get.name,
       type <- get.type
   )
}
```

How can we resolve the properties created with get_set2attribute?

Motivation

- If you know the answer probably you can come and help me with the talk ☺
- We need to understand how ATL works to make the most of it.
- This is what this course is about
  - To understand how ATL works
  - To learn how to use ATL effectively

Introduction to ATL

A grasp of the language

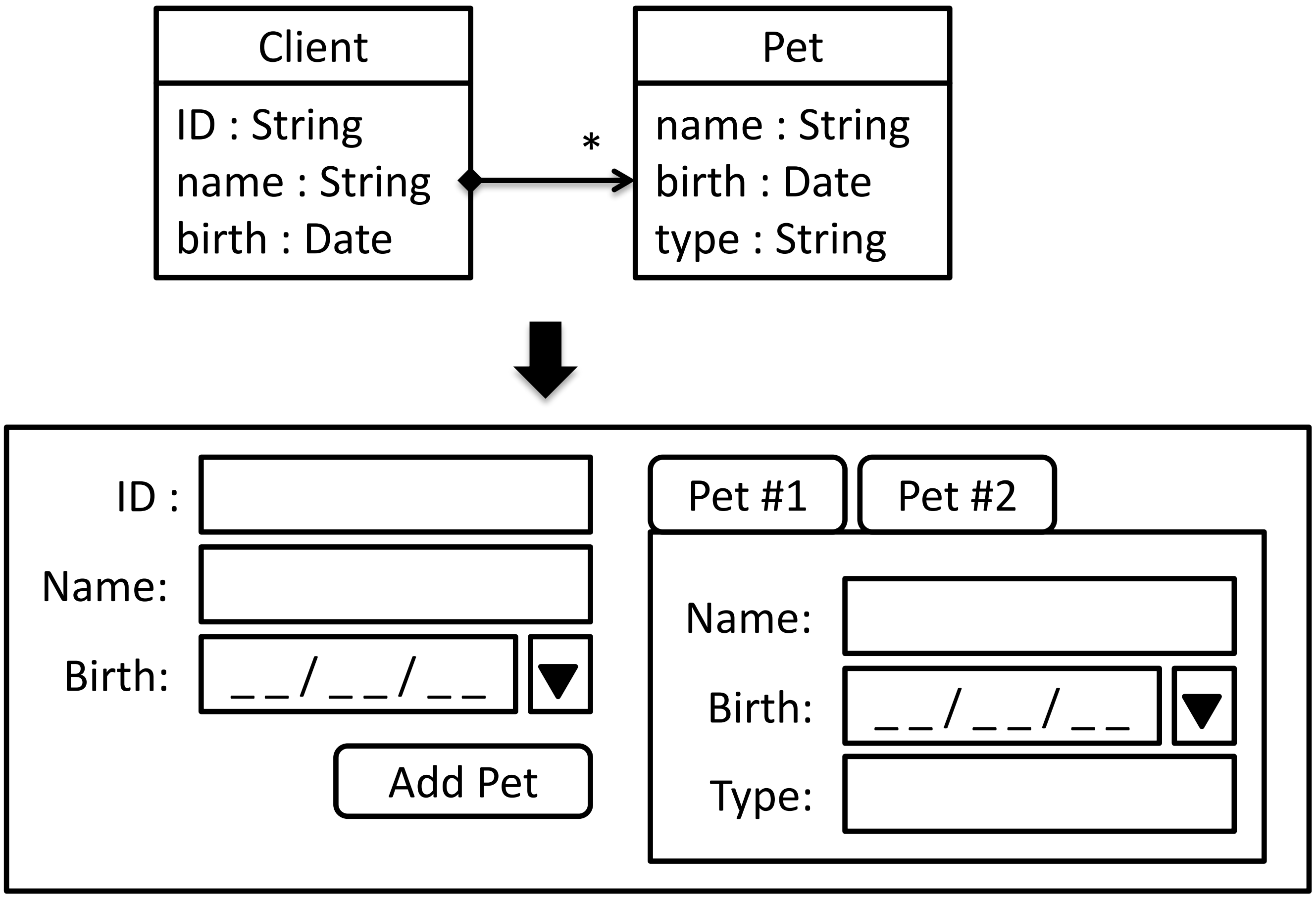
ATL Language

- ATL characteristics
  - Designed for model-to-model transformations
  - Source models are read-only
  - Target models are write-only *
  - Implicit reference resolution
  - Model navigation in OCL
  - Helpers
  - Limited imperative constructs

* This is not actually true

Transformation example

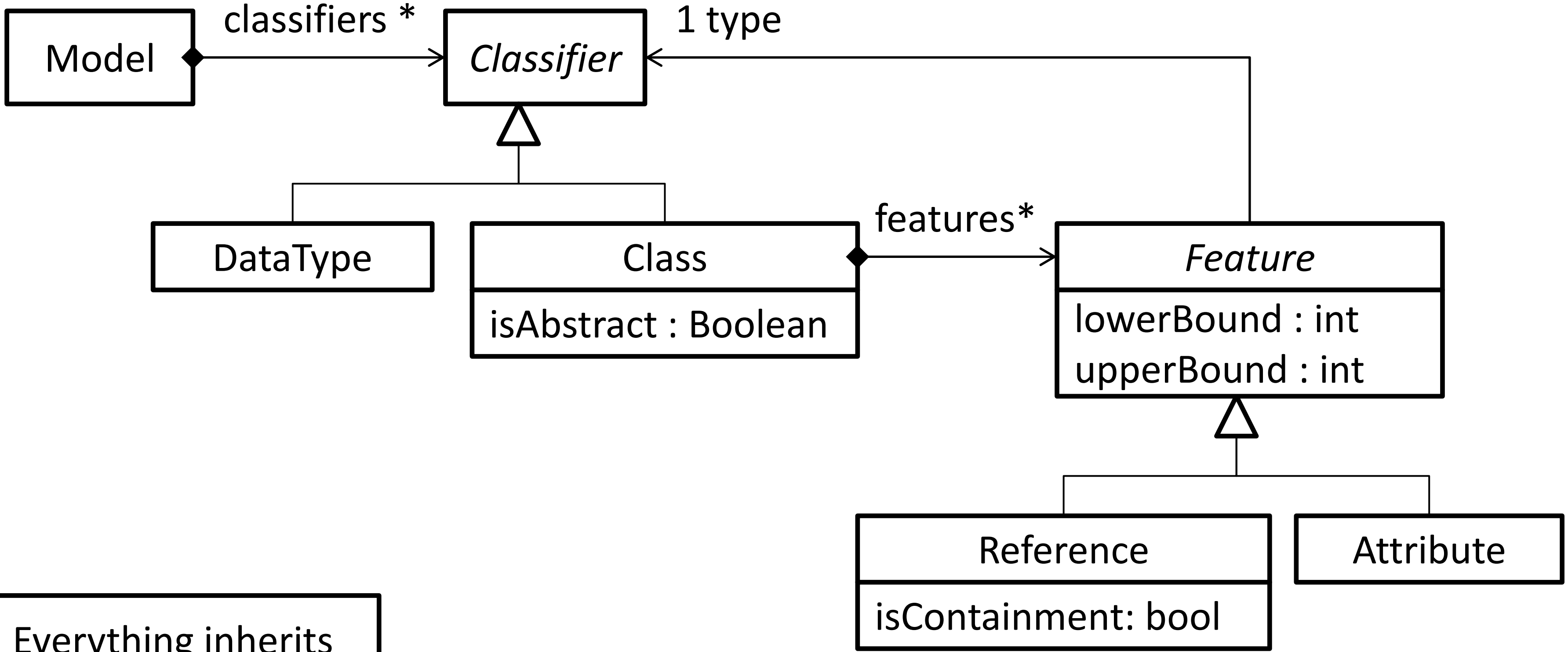
- Class Diagram to UI



Transformation example

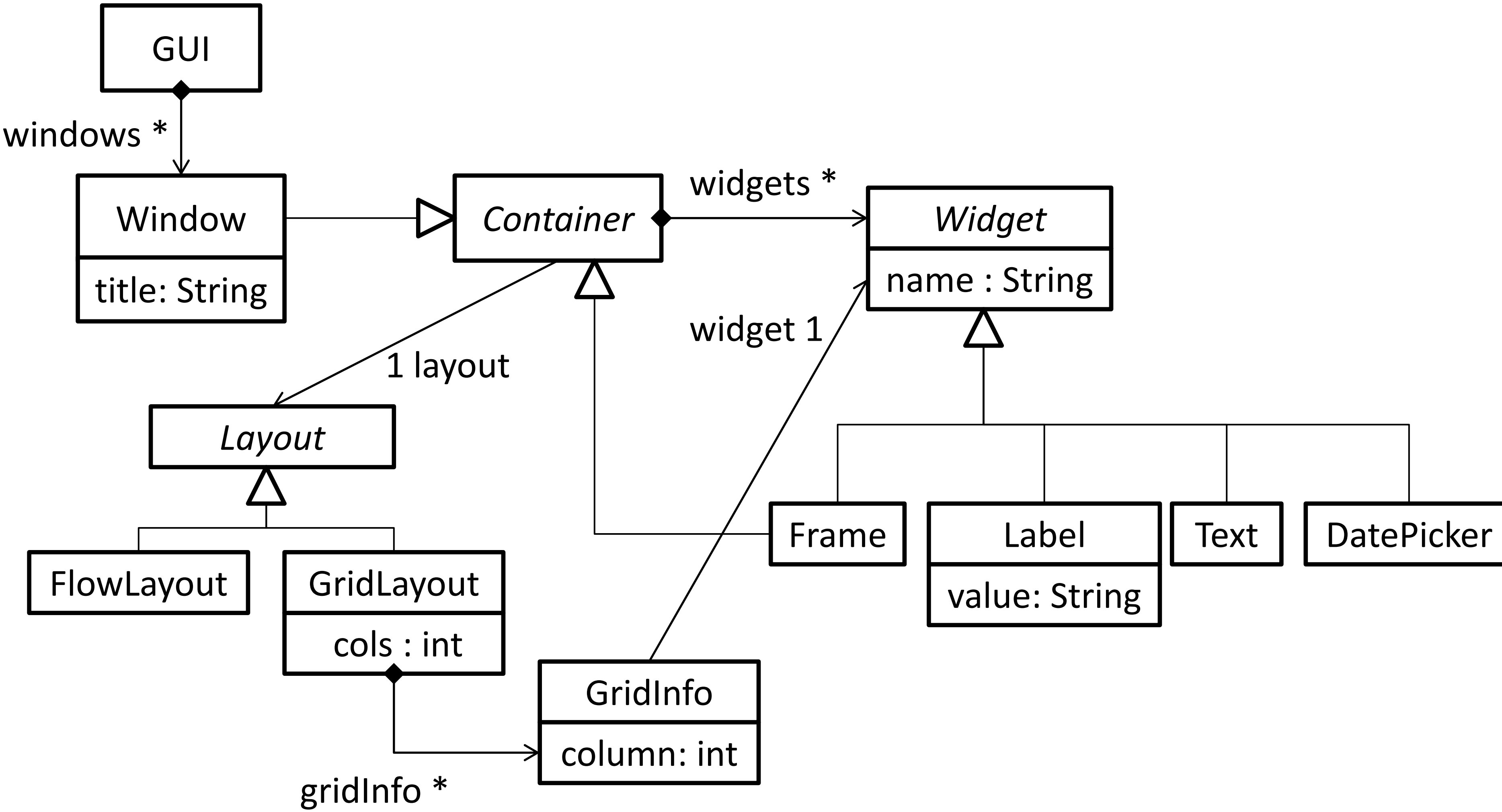
- PUT THE META-MODELS
- Mapping at the high-level
  - Model would be a Window
  - Class would be Frame
  - Each property a UI element
    - String properties would be text widgets
    - Date properties would be a date picker
    - References can be converted to buttons, etc.

Class diagram meta-model



Everything inherits from NamedElement

GUI meta-model



cd2gui_01.atl

ATL transformation

```
module "cd2gui";
create OUT: GUI from IN: CD;

helper context CD!Attribute def : isText() : Boolean =
  self.type.name = 'String';


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

rule attribute2text {
  from p : CD!Attribute ( p.isText() )
  to t : GUI!Text
}
```

Module declaration

Helper

In pattern

Bindings

Out pattern

Matched rule

ATL Transformation

- This transformation:
  - Creates a `Frame` widget for each class
  - Creates a `Text` widget for each attribute whose type is `String`
  - Links `Text` widgets to the frame (via the `widget` reference)
- This is the basic schema of any ATL transformation

What else?

- Tooling
  - The compiler
  - The development environment
- Language
  - Different kinds of rules
  - OCL navigation
  - Helpers

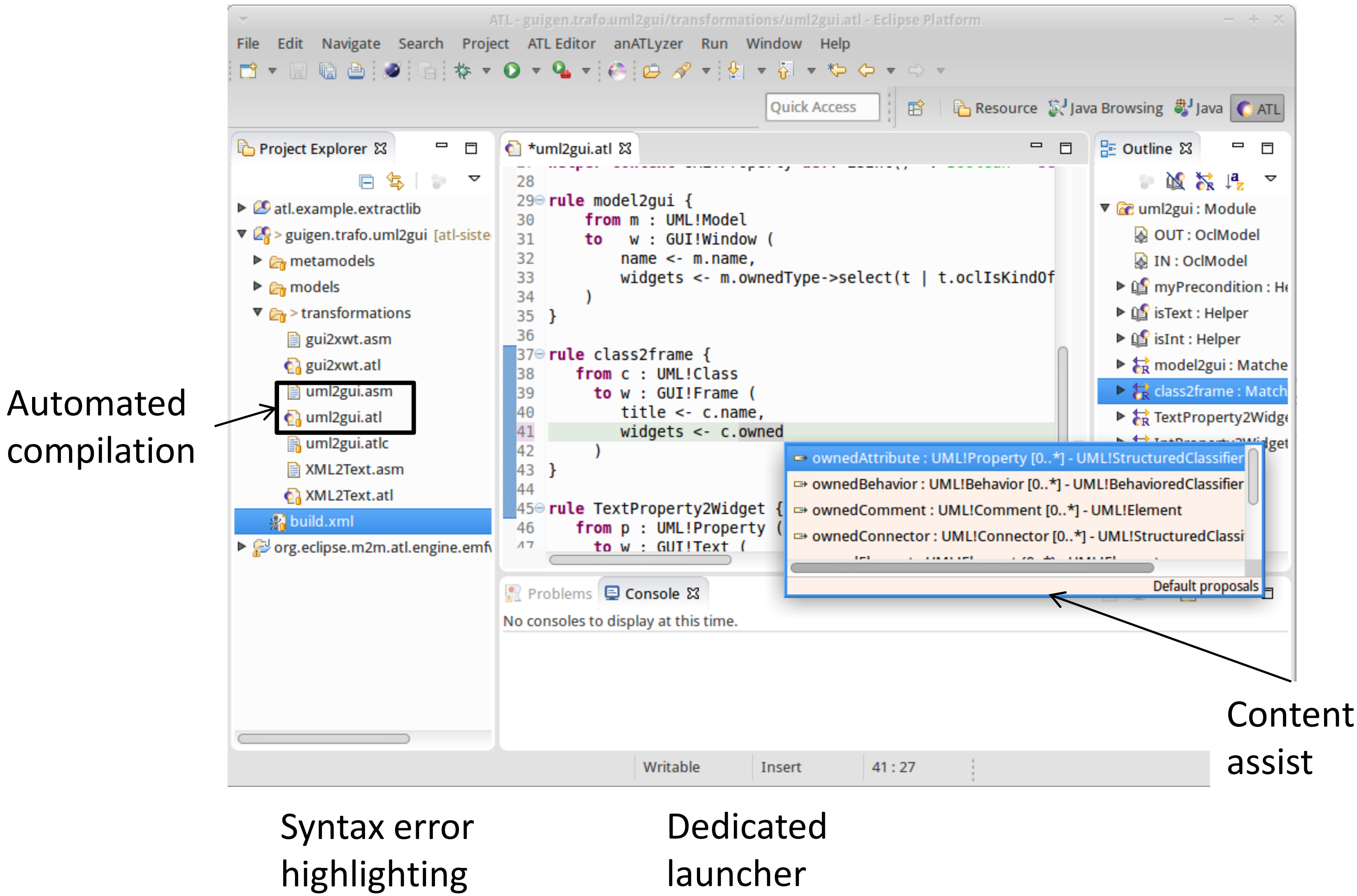
Introduction to ATL

Tooling

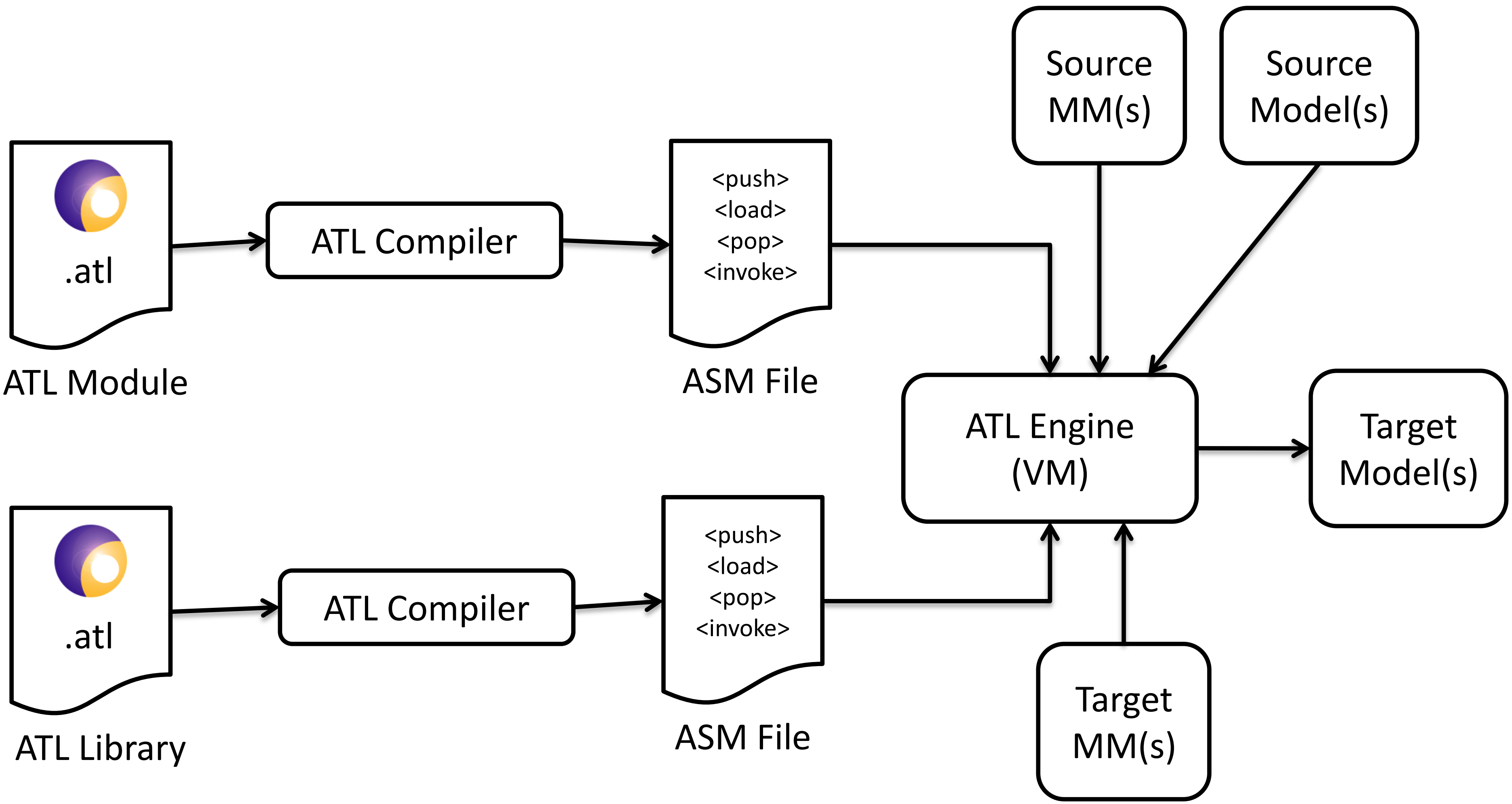
ATL Plug-in

- Features
  - ATL perspective
    - Register meta-model button
  - Editor with syntax highlighting
  - Automatic compilation
  - Autocompletion + Code templates
    - CTRL + SPACE
  - Outline view
  - ATL Console
  - Launching transformations

ATL Editor


ATL - guigen.trafo.uml2gui/transformations/uml2gui.atl - Eclipse Platform
File Edit Navigate Search Project ATL Editor anATLyzer Run Window Help
Quick Access
Resource
Java Browsing
Java
ATL
Project Explorer
atl.example.extractlib
guigen.trafo.uml2gui
metamodels
models
transformations
gui2xwt.asm
gui2xwt.atl
uml2gui.asm
uml2gui.atl
uml2gui.atlc
XML2Text.asm
XML2Text.atl
build.xml
org.eclipse.m2m.atl.engine.emf
*uml2gui.atl
rule model2gui {
from m : UML!Model
to w : GUI!Window (
name <- m.name,
widgets <- m.ownedType->select(t | t.oclIsKindOf
)
}
rule class2frame {
from c : UML!Class
to w : GUI!Frame (
title <- c.name,
widgets <- c.owned
)
}
rule TextProperty2Widget {
from p : UML!Property (
to w : GUI!Text (
ownedAttribute : UML!Property [0..*] - UML!StructuredClassifier
ownedBehavior : UML!Behavior [0..*] - UML!BehavioredClassifier
ownedComment : UML!Comment [0..*] - UML!Element
ownedConnector : UML!Connector [0..*] - UML!StructuredClassi
Default proposals
Outline
uml2gui : Module
OUT : OclModel
IN : OclModel
myPrecondition : He
isText : Helper
isInt : Helper
model2gui : Matche
class2frame : Match
TextProperty2Widge
Problems
Console
No consoles to display at this time.
Writable
Insert
41 : 27
Automated compilation
Content assist
Syntax error highlighting
Dedicated launcher

Compilation & Execution



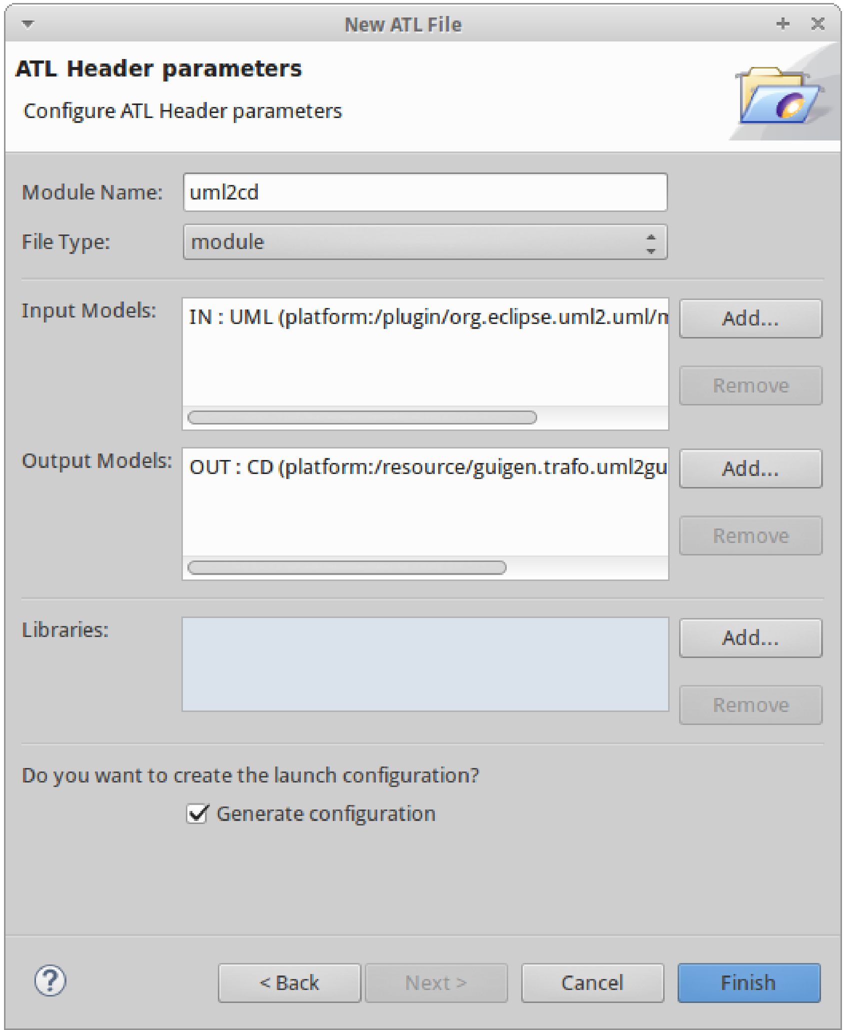
Project structure

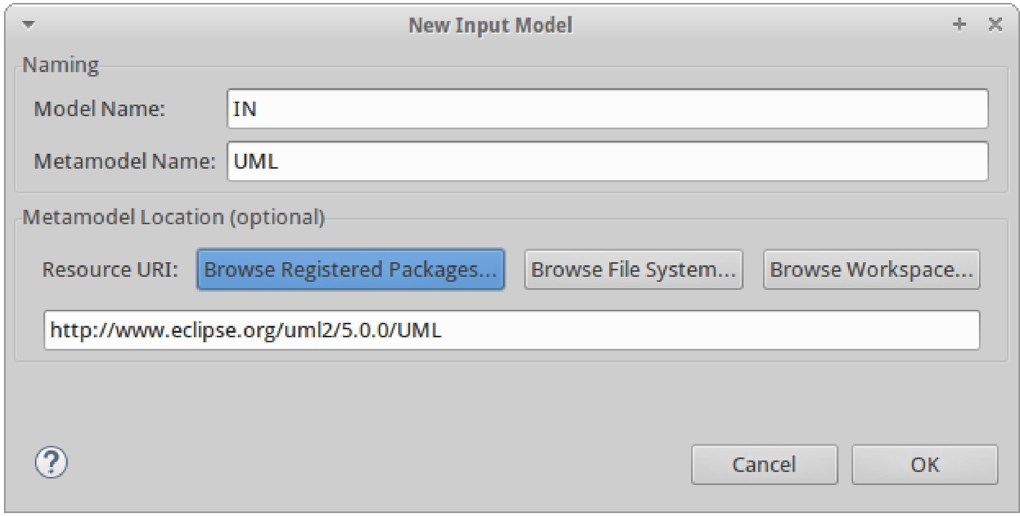
- File -> Project .. -> ATL Project
  - The projects are created with no structure
  - Possible structure

```
myProject
  + launching
  + metamodels
  + models
  + output
  + transformations
```

New ATL transformation

- File -> New ... -> ATL File



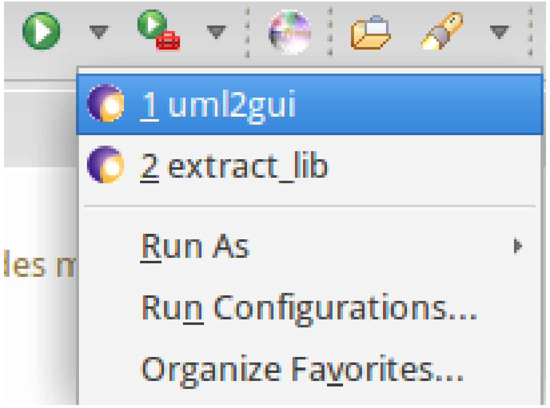


Launching

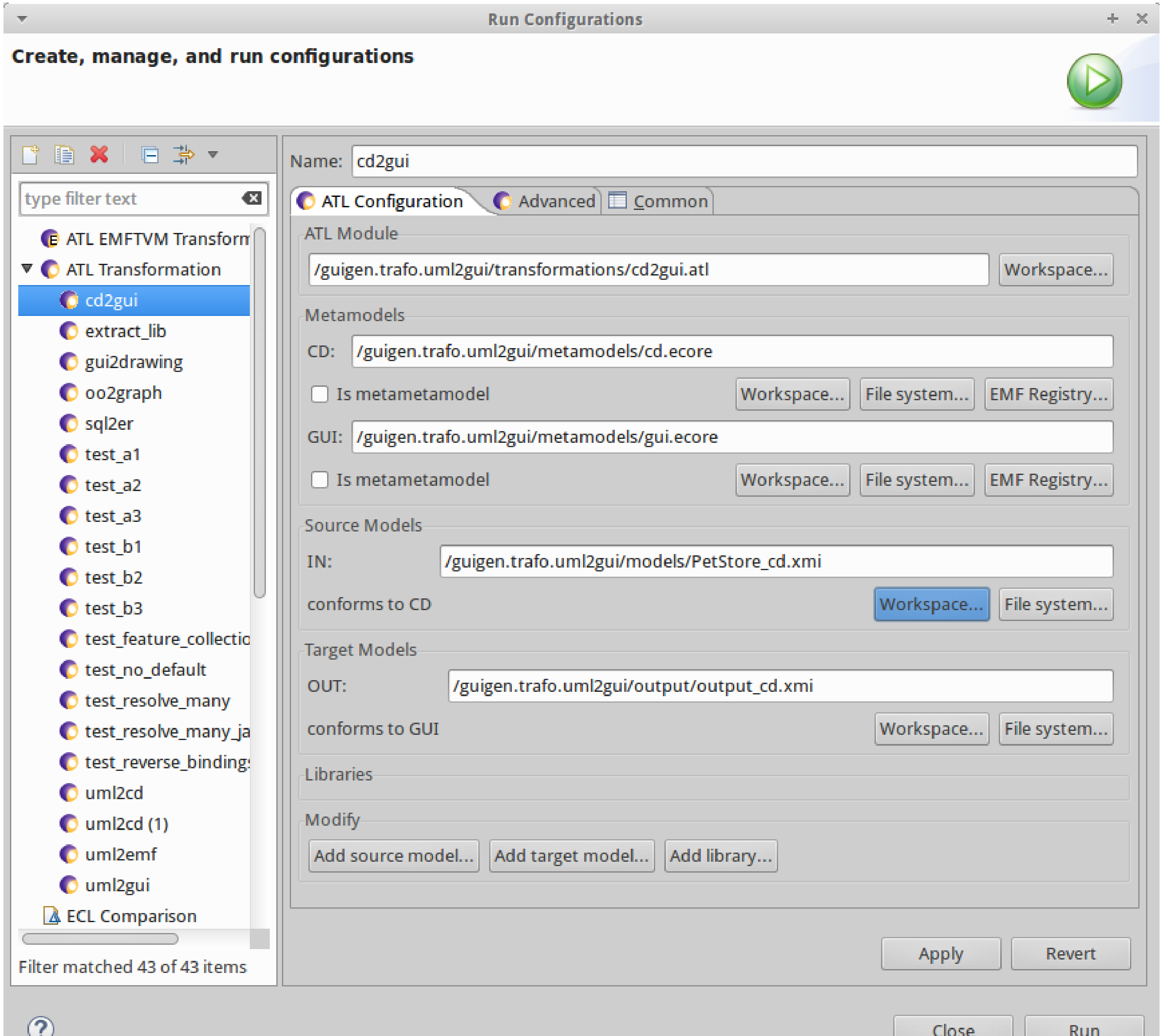
- Dedicated launcher
- ANT Tasks
  - http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Tools#ATL_ant_tasks
- Programatically
  - ATL Plug-in project
  - We will see this later

Launching

- Dedicated launcher
  - Based on Eclipse infrastructure
  - Accessible via the “play button”
- Right-click on the ATL file
  - Run as... -> ATL Transformation
  - Meta-model information is automatically filled in if you have the proper annotations



Launching


Run Configurations
Create, manage, and run configurations
type filter text
ATL EMFTVM Transform
ATL Transformation
cd2gui
extract_lib
gui2drawing
oo2graph
sql2er
test_a1
test_a2
test_a3
test_b1
test_b2
test_b3
test_feature_collectio
test_no_default
test_resolve_many
test_resolve_many_ja
test_reverse_binding
uml2cd
uml2cd (1)
uml2emf
uml2gui
ECL Comparison
Filter matched 43 of 43 items
Name: cd2gui
ATL Configuration
Advanced
Common
ATL Module
/guigen.trafo.uml2gui/transformations/cd2gui.atl
Workspace...
Metamodels
CD: /guigen.trafo.uml2gui/metamodels/cd.ecore
Is metametamodel
Workspace... File system... EMF Registry...
GUI: /guigen.trafo.uml2gui/metamodels/gui.ecore
Is metametamodel
Workspace... File system... EMF Registry...
Source Models
IN: /guigen.trafo.uml2gui/models/PetStore_cd.xmi
conforms to CD
Workspace... File system...
Target Models
OUT: /guigen.trafo.uml2gui/output/output_cd.xmi
conforms to GUI
Workspace... File system...
Libraries
Modify
Add source model... Add target model... Add library...
Apply
Revert
Close
Run

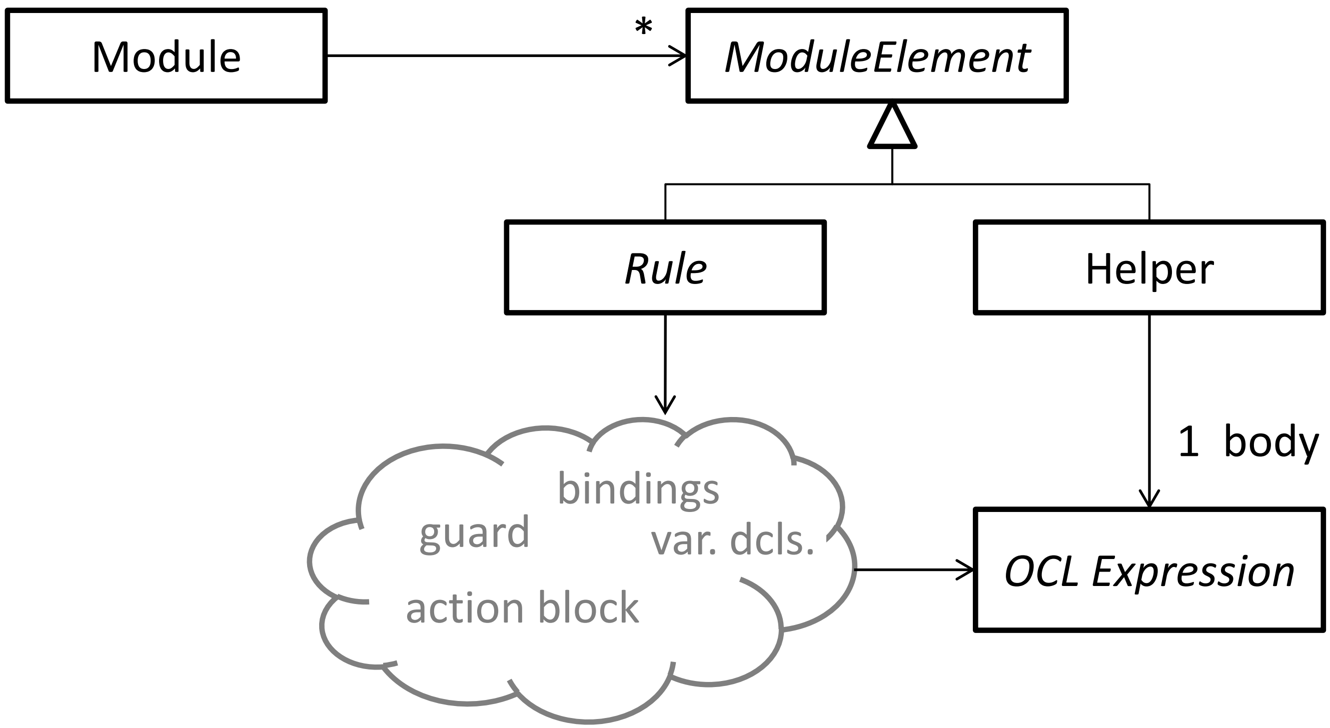
Launching

- Opening the output model
  - Not that easy…
- XMI files does not include schemaLocation information
- Registering meta-models is a must
  - The ATL perspective must be active to have access to the register meta-model button
  - Right-click on the target meta-model file
    - Register meta-model

Introduction to ATL

Basic constructs

ATL


Module
*
ModuleElement
Rule
Helper
bindings
guard
var. dcls.
action block
1 body
OCL Expression

Module definition

- Name
  - No need to coincide with the file name
    - (need to be the same for EMFTVM)
  - Dots not allowed. Several words, with “ “

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

module “uml to class diagram”;
create OUT : CD from IN : UML;
```

Module definition

- Meta-model references
  - Not compulsory, but recommended
  - Enables auto-completion (+ anATLyzer)
  - @nsURI for registered meta-models
  - @path for workspace files

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

module "uml to class diagram";
create OUT : CD from IN : UML;
```

Module definition

- Compiler directive
  - -- @atlcompiler atl2004
  - -- @atlcompiler atl2006
  - -- @atlcompiler atl2010
  - -- @atlcompiler emftvm

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

module "uml to class diagram";
create OUT : CD from IN : UML;
```

Rules

- Matched rule
- Lazy rule
- Unique lazy rule
- Called rule
- Entry point rule
- Endpoint rule

Rules

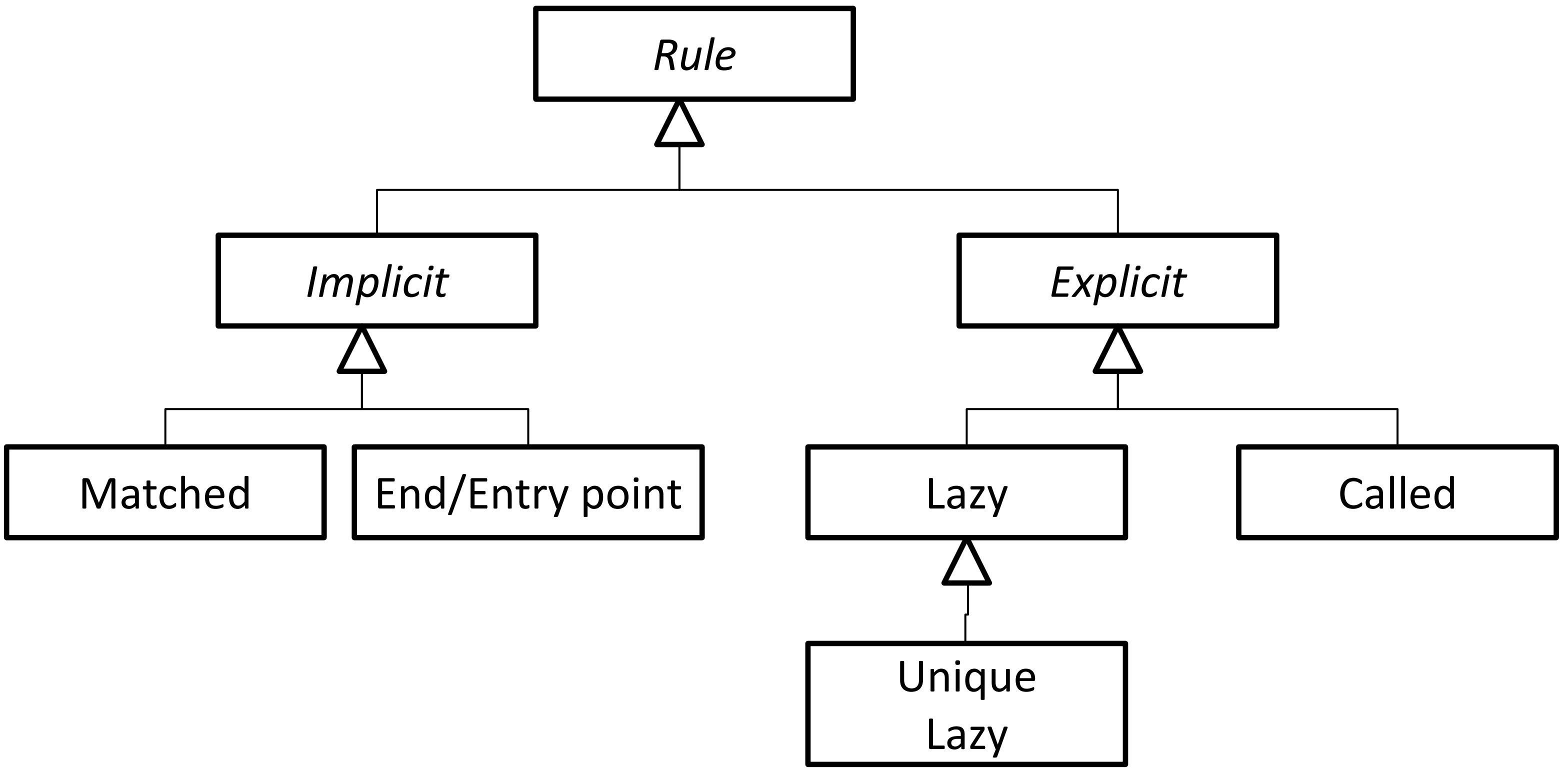
- Matched rule
- Lazy rule
- Unique lazy rule
- Called rule
- Entry point rule
- Endpoint rule

In this part

Later

Rules – Conceptual model

- According to the invocation mode *



* This classification has nothing to do with the abstract syntax of the language

Matched rules

- Structure
  - Input pattern (from)
    - Optional filter/guard
  - Output pattern (to)
    - Contains bindings (**<-**)
  - Imperative block (do)
    - Optional. Discouraged.
- Behaviour
  - Executed implicitly, at the top level
  - Target elements created automatically
  - Target features initialized with *bindings*

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

Matched rules

- Multiple input elements
  - Cartesian product of instances of the input pattern types
  - Intuitively:

```
for x in T1.allInstances()
  for y in T2.allInstances()
    check guard(x, y)
```

```
rule model_class2frame {
  from m : CD!Model,
       c : CD!Class ( m.classifiers->includes(c) )
  to   f : GUI!Frame (
     title <- m.name + '_' + c.name,
     widgets <- c.features
  )
}
```

* This is not a very efficient way of doing this...

Matched rules

- Multiple output pattern elements
  - Comma-separated
  - Can be linked to any of the other output elements

```
rule model2gui {
  from m : CD!Model
    to w : GUI!Window (
       title <- m.name,
       layout <- vflow,
       widgets <- m.classifiers
     ), g : GUI!GUI (
        windows <- w
     ), vflow : GUI!FlowLayout (
       direction <- #vertical
     )
}
```

Bindings

- Structure
  - Left part
    - Target feature
  - Right part
    - OCL expression
- Behaviour
  - Right part is flattened
  - Primitive bindings
    - Left is primitive type
    - Right is primitive value
    - Direct assignment
  - Object bindings
    - Left type is meta-class
    - Right value is object

Primitive binding

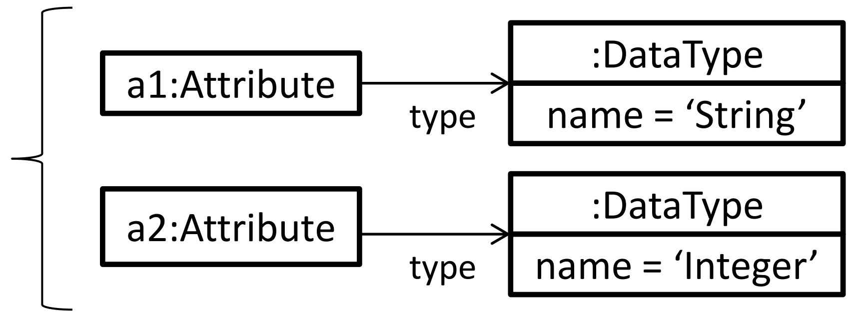
```
title <- c.name,
```

Object binding

```
widgets <- c.features
```

Binding resolution

```
widgets <- c.features
```



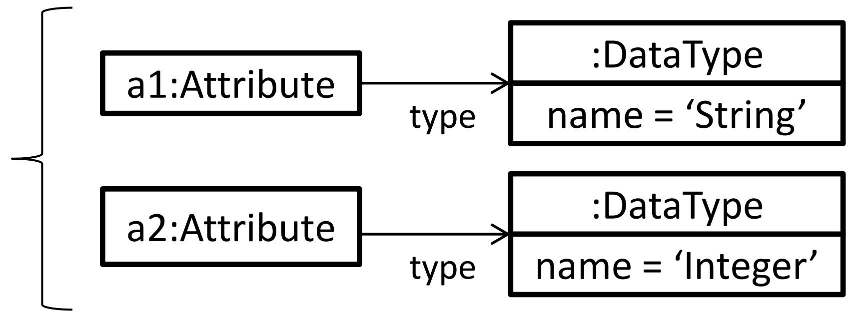
a1 matched by the input pattern?

```
rule attribute2text {
  from p : CD!Attribute ( p.isText() )
  to   t : GUI!Text
}
```

Object instantiated
Assigned to feature

Binding resolution

```
widgets <- c.features
```



a2 matched by the input pattern?

```
rule attribute2int {
  from p : CD!Attribute ( p.isInt() )
  to   t : GUI!Text
}
```

Object instantiated
Assigned to feature

Binding assignment

- Semantics
  - Multi-valued
    - Addition of elements
      - widgets <- xxx,
        widgets <- yyy

        Assign both elements
  - Mono-valued
    - The second assignment wins
    - I assume bindings are executed in order, but all ATL papers claim that the order is not guaranteed (i.e., because ATL is a declarative language...)

Binding assignment

- Given that the semantics is a bit confusing...
- Idiomatic way of dealing with multi-valued bindings and multiple-resolutions

```
feature <- expr1->union(expr2)
```

  – You could also use including for single elements

```
feature <- expr1->including(expr2)
```

  – Take into account that union and including may be expensive operations...

Resolving elements explicitly

- Problem: We want to attach a label each widget.
  - Solution: add an additional *out pattern element*

```
rule attribute2text {
  from p : CD!Attribute ( p.isText() )
  to   t : GUI!Text ( ... ),
       l : GUI!Label ( ... )
}
```

Resolving elements explicitly

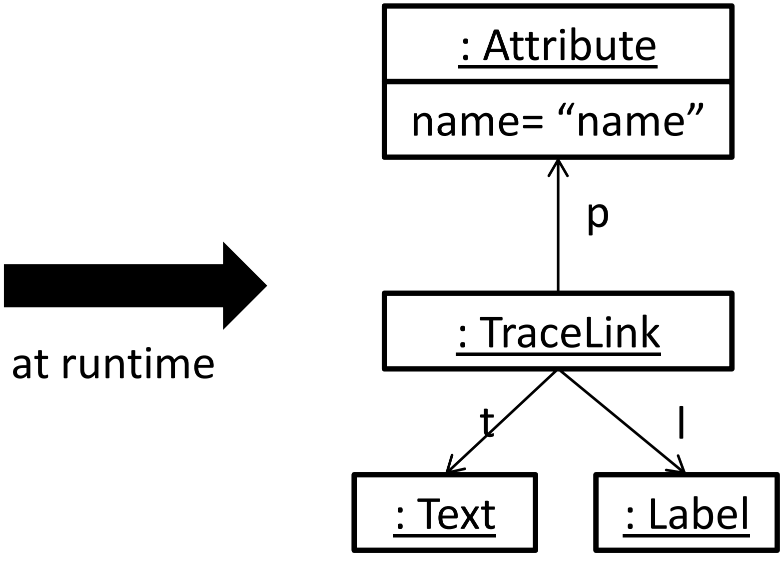
- Next problem, we need to link the label to its container
  - Remember, ATL only resolves the first element
  - Solution: `resolveTemp`

- `thisModule.resolveTemp(obj, ‘varName’)`
  - Performs the trace lookup for `obj` explicitly
  - Retrieves the element created with the output pattern element whose variable name is `‘varName’`

Resolving elements explicitly

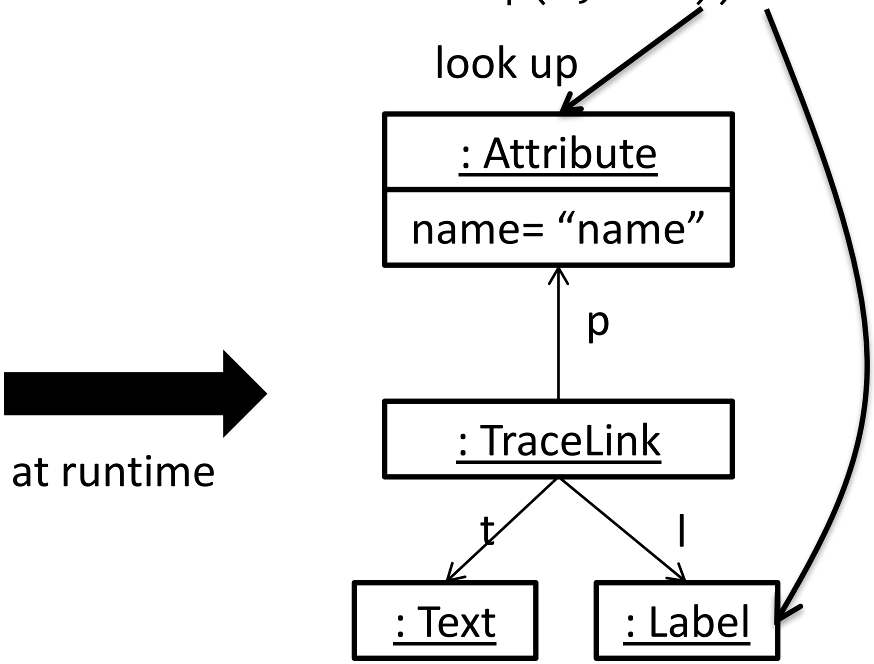
```
rule class2frame {
  from c : CD!Class ( not c.isAbstract )
  to   f : GUI!Frame (
    title <- c.name,
    widgets <- c.features,
    widgets <- c.features->collect(a | thisModule.resolveTemp(a, 'l'))
  )
}
```

```
rule attribute2text {
  from p : CD!Attribute ( p.isText() )
  to   t : GUI!Text ( ... ),
       l : GUI!Label ( ... )
}
```



Resolving elements explicitly

```
rule class2frame {
  from c : CD!Class ( not c.isAbstract )
  to   f : GUI!Frame (
    title <- c.name,
    widgets <- c.features,
    widgets <- c.features->collect(a | thisModule.resolveTemp(a, ‘l’))
  )
}


rule attribute2text {
  from p : CD!Attribute ( p.isText() )
  to   t : GUI!Text ( ... ),
       l : GUI!Label ( ... )
}
```



Resolving elements explicitly

- What if the source element cannot be resolved?
  – It returns OclUndefined
- What if the output pattern element name (e.g., ‘l’) does not exist?
  – It returns OclUndefined

OCL

- **Object constraint language**
  - OMG Specification. Current version 2.4 *
  - Initially defined to write constraints on UML models
    - Invariants (well-formedness rules)
    - Operation pre/post conditions
  - Scope extended for e.g.,:
    - Model navigation in model transformation languages
    - Well-formedness rules in DSLs (i.e., validation rules)
    - Transformation contracts

* http://www.omg.org/spec/OCL/2.4/
* Object Constraint Language (OCL): A Definitive Guide. Jordi Cabot, Martin Gogolla

OCL

- Characteristics
  - Side-effect free (i.e., there are no assignments)
  - No statements, only expressions
  - Collection navigation operators
    - Collection navigation in a “functional style”
  - An OCL expression is typed w.r.t. a meta-model

OCL

- ATL implements its own variant
  - Somewhat out of date with respect to newer versions
    - e.g., lack of closure operation
  - OCL is statically typed, ATL/OCL is not!
  - Model elements named with syntax `MM!Type`

OCL

- Example: get all attributes of type String in a class diagram

```
aModel.classifiers->
   select(c | c.oclIsKindOf(CD!Class))->
   collect(c | c.features->select(f | f.oclIsKindOf(CD!Attribute) )->
   flatten()->
   select(a | if a.type.oclIsUndefined() then
                  a.type.name = 'String'
              else
                  false
              endif)
```

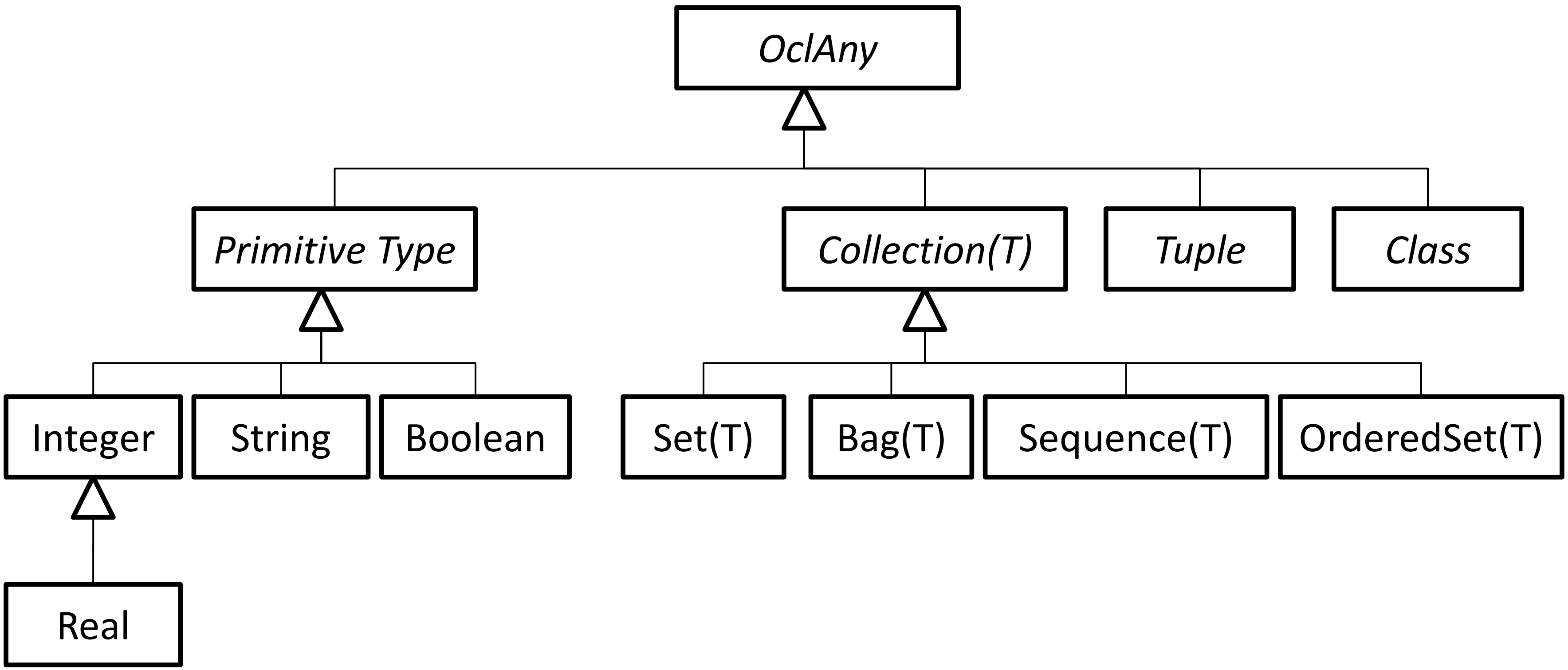
This example assumes that Attribute.type is an optional property

OCL

- Details about the supported operations available in the ATL guide
- https://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language

OCL

- Type hierarchy



OCL

- Literal values

```
let i : Integer = 1024 in ...
let r : Real    = 3.1415 in ...
let s : String  = ‘a string’ in ...
let b : Boolean = true in ...

let s : Set(Integer) = Set {1, 2, 3} in ...
let s : Sequence(Integer) = Sequence {1, 2, 3, 3} in ...
let s : Bag(Integer) = Sequence {1, 2, 3, 3, 3} in ...
let s : OrderedSet(Integer) = Sequence {3, 2, 1} in ...
```

Collection operations

- Should be written using “->”
  - `collectionExp->operation()`
- Query operations:
  - `size()`
  - `includes(o : oclAny)`
  - `excludes(o : oclAny)`
  - `count(o : oclAny)`
  - `includesAll(c : Collection)`
  - `excludesAll(c : Collection)`
  - `isEmpty()`
  - `notEmpty()`
  - `sum()`

Collection operations

- Modification operations available for all collection types:
  - `union(c : Collection)`
  - `flatten()`
  - `including(o : OclAny)`
  - `excluding(o : OclAny)`

* ATL and OCL collections are immutable

Collection operations

- There are other operations depending on the collection type
  - **Sequence**: append, prepend, insertAt, subSequence, at, indexOf, first, last
  - **Set**: intersection, - (operator, for difference), symmetricDifference
  - **OrderedSet**: mix Sequence and Set

Iterators

- Syntax:
  - `<source_exp>->iteratorName(itVar | <body>)`
- Available:
  - exists(itVar | <boolean-body>)
  - forAll(itVar | <boolean-body>)
  - isUnique(itVar | <boolean-body>)
  - any(itVar | <boolean-body>)
    - If body never evaluates to true, the operation returns OclUndefined;
  - one(itVar | <boolean-body>)
  - collect(itVar | <any-body>)
    - Implement OCL's collectNested
  - select(itVar | <boolean-body>)
  - reject(itVar | <boolean-body>)
  - sortedBy(itVar | <any-body>)
    - The body must evaluate to a data type with "<" operator

Iterator

- Example: organize widgets by alphabetical order

```
rule class2frame {
  from c : CD!Class ( not c.isAbstract )
  to   f : GUI!Frame (
    title <- c.name,
    widgets <- c.features->sortedBy(f | f.name)
  )
}
```

- Example: obtain a searchable attribute

```
helper context CD!Class def: searchableAtt() : CD!Atribute =
  self.features->reject(f | not f.oclIsKindOf(CD!Attribute))->
                ->any(f | f.isId);
```

Checking “instance of”

- OCL provides two operations to check if an object is instance of a given class
  - `obj.oclIsKindOf(Type)`
    - Is the runtime type of `obj` the same as `Type` or a subtype?
  - `obj.oclIsTypeOf(Type)`
    - Is the runtime type of `obj` exactly the same as `Type`?

```
aClassifier.oclIsKindOf(CD!Class)
aNamedElement.oclIsTypeOf(CD!Property)
```

OclAny

- **refImmediateComposite**
  - Returns the container of an object
    - Equivalent to eContainer in EMF
  - OclUndefined if none

```
from c : CD!Class
  using m : CD!Model = c.refImmediateComposite();
  to    f : GUI!Frame (
     title <- m.name + '_' + c.name,
     widgets <- c.features
  )
}
```

OclAny

- **asSet, asBag, asSequence**
  - For collections: type conversion
    - `Sequence {1, 2, 2}->asSet() => Set{1, 2}`
  - For single elements: wrap an object into a collection
    - `aClass->asSequence  => Sequence { aClass }`
  - Try not to abuse
  - They have a cost, particularly for large collections

OclAny

- **debug**
  - Prints the value of an object, prefixed by a message and returns the same object
  - Syntax: `obj.debug(message : String)`

```
aClass.debug('class: ')  =>   class: IN!Person
```

ATL tries to pretty print...

```
aClass.debug(aClass) => org.eclipse.emf.ecore.impl.Dynamic
                        EObjectImpl@14e987ac (eClass:
                        org.eclipse.emf.ecore.impl.EClassI
                        mpl@11851c1d (name: Class)
```

Often you want the identify

Model elements

- **allInstances, allInstancesFrom(model : String)**
  - Retrieve all instances of a given OclModelElement

```
CD!Class.allInstances()
```

  - If the same meta-model is used for two input models:

```
CD!Class.allInstances()
=
CD!Class.allInstancesFrom('IN1')->union(
CD!Class.allInstancesFrom('IN2') )
```

Helpers

- “Methods” attached to (meta-model) types at runtime
- Two types
  - Module helpers
  - Context helpers
- Two modes
  - Operation
  - Attribute

Helpers

Module helpers

- Global helpers
  - Or methods attached to “this transformation module”

Context helpers

- Methods attached at runtime

```
thisModule.propsByName(‘age’)

helper def: propsByName(name : String) : Set(CD!Attribute) =
  CD!Attribute.allInstances()->select(p | p.name = name);
```

```
aClass.hasProperty(‘age’)

helper context CD!Class def: hasFeature(name : String): Boolean =
  self.features->exists(p | p.name = name);
```

Helpers

- **Module helpers**
  - Global helpers
  - Methods attached to “this transformation module”

```
thisModule.propsByName('age')

helper def: propsByName(name : String) : Set(CD!Attribute) =
  CD!Attribute.allInstances()->select(p | p.name = name);
```

- **Context helpers**
  - Methods attached at runtime to a meta-class

```
aClass.hasProperty('age')

helper context CD!Class def: hasFeature(name : String): Boolean =
  self.features->exists(p | p.name = name);
```

Helpers

- **Context helpers**
  - Methods attached at runtime
  - Polymorphic calls
  - **self** variable refers to the current object

```
aFeature.isContainment()

helper context CD!Attribute def: isContainment(): Boolean = false;

helper context CD!Reference def: isContainment(): Boolean =
    self.containment;
```

Helpers

- Attribute
  - No parameters. Syntactically there are no ()
  - Overrides meta-model features
  - Memoized
- Operation
  - Regular method

Helpers

- When to use Attribute or Operation?
  - Think if caching makes sense. Is it going to be reused the called value?
- Example:
  - Which one is faster?

```
helper context CD!Class def: allFeatures() : Sequence(CD!Feature) =
   self.superclasses->collect(c | c.allFeatures())->flatten()->
                              union(self.features);

helper context CD!Class def: allFeatures : Sequence(CD!Feature) =
   self.superclasses->collect(c | c.allFeatures)->flatten()->
                              union(self.features);
```

Helpers

	Operation	**Attribute**
Module	function	static final field
Context	method	final field (lazily initialized)

Lazy rules

- Rules which are explicitly invoked
  - Same structure as matched rules
  - No trace links are generated
- Can be invoked many times over the same source element

```
thisModule.aLazyRule(obj1, obj2)
```

Input pattern elements are passed as parameters

```
rule aLazyRule {
  from c : CD!Class, p : CD!Property
  to   t : GUI!Text ( ... )
}
```

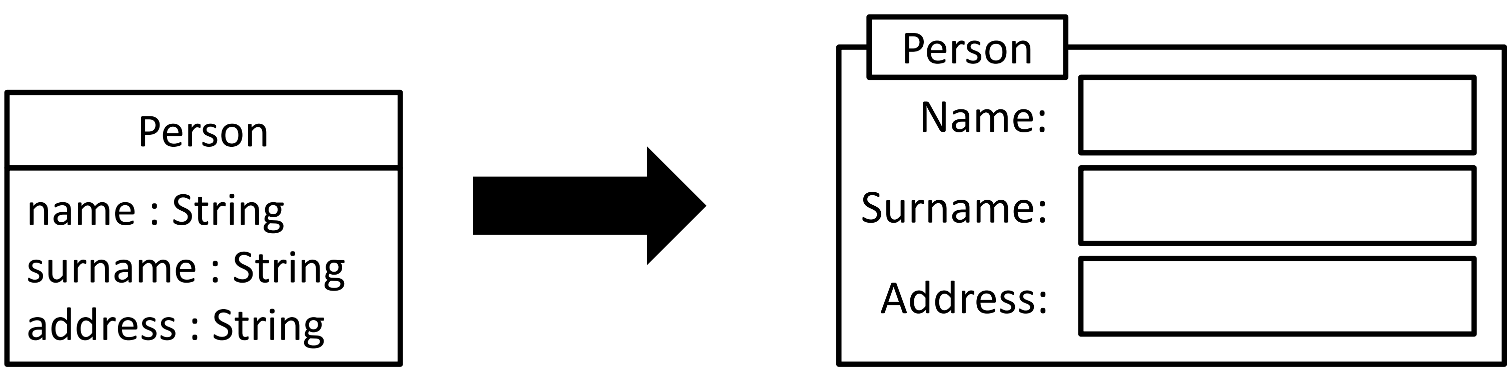
Object instantiated
Assigned to feature

Lazy rules

- Lazy rules vs. Matched rules
- Use matched rules
  - Start the design with matched rules
  - They are a good fit for relative direct mappings
- Use lazy rules
  - If you need to create a target element from a source element many times
  - If you need dynamic mappings

Lazy rules

- Example. Consider generating a layout for each frame.
- Approach #1: Fixed layout.
  - Grid layout with two columns



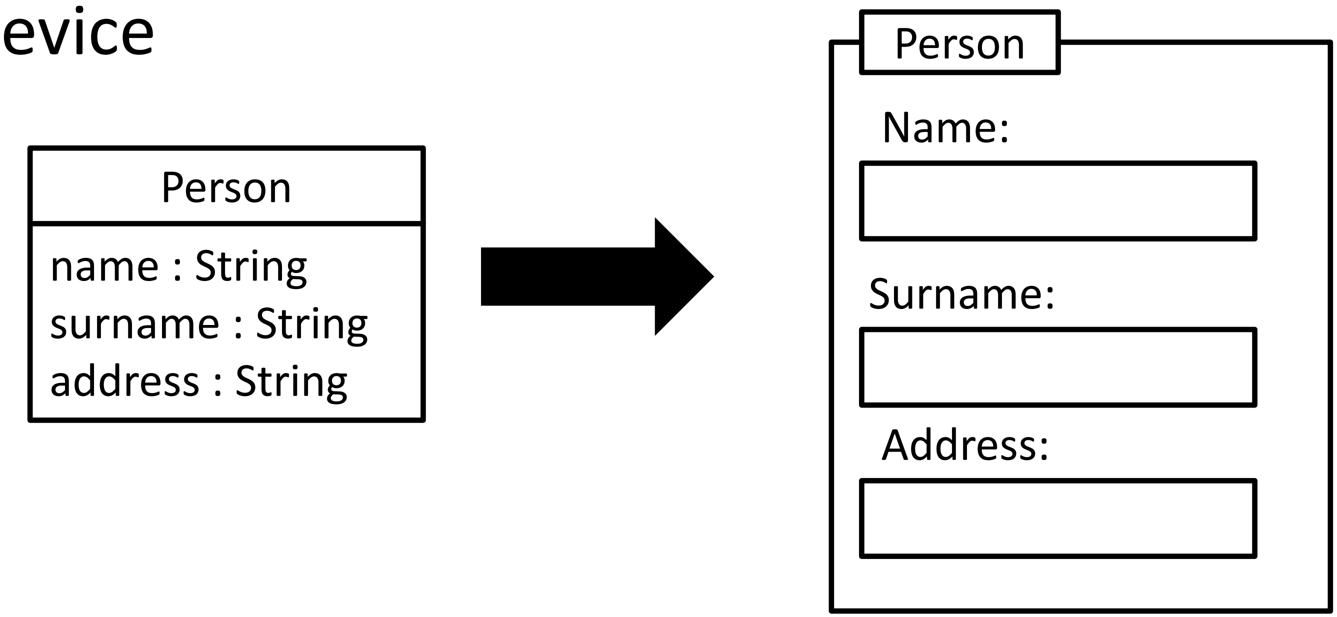
cd2gui_04.atl

```
rule class2frame {
  from c: CD!Class ( not c.isAbstract )
  to   f: GUI!Frame (
      ...,
      layout <- grid
  ), grid: GUI!GridLayout (
         numColumns <- 2,
         info <- c.features -> collect(a | thisModule.resolveTemp(a, 'g1')),
         info <- c.features -> collect(a | thisModule.resolveTemp(a, 'g2'))
  )
}

rule attribute2text {
  from a: CD!Attribute ( a.isText() )
  to   t: GUI!Text,
       l: GUI!Label,
       g1: GUI!GridInfo (
         column <- 1,
         widget <- t
       ),
       g2: GUI!GridInfo (
         column <- 2,
         widget <- l
       )
}
```

The layout is hardcoded in the output patterns

Lazy rules

- Approach #2: “Dynamic selection” of layout.
  - Decide based on the elements of the model the best layout or with some parameter flag
  - For example, we want a vertical flow for a mobile device



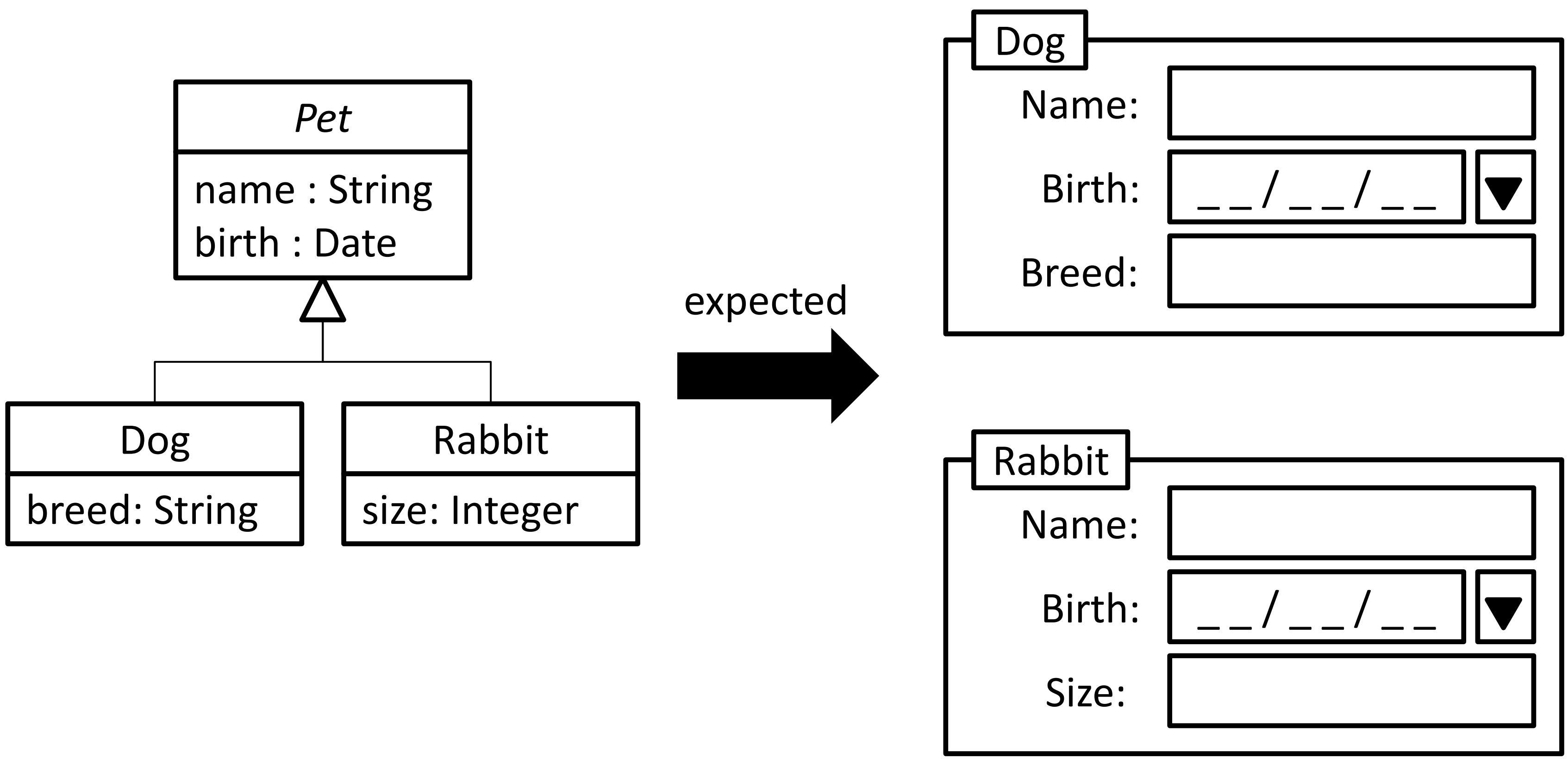
```
helper def : isMobile : Boolean = true;

rule class2frame {
  from c: CD!Class ( not c.isAbstract )
  to   f: GUI!Frame (
     ...,
     layout <- if thisModule.isMobile then
                  thisModule.createVFlow(c)
          else
                  thisModule.createGrid(c)
          endif  ) }

lazy rule createVFlow {
   from c : CD!Class
     to layout : GUI!FlowLayout (
        direction <- #vertical
     ) }

lazy rule createGrid {
   from c : CD!Class
     to grid: GUI!GridLayout (
        numColumns <- 2,
        info <- c.features -> collect(a | thisModule.resolveTemp(a, 'g1')),
        info <- c.features -> collect(a | thisModule.resolveTemp(a, 'g2'))
   )
}
```

Approach #1 was not enough because the layout was fixed in the target pattern.

Nevertheless, you can always code all possibilities in the filters of the matched rules...

Lazy rules

- Example. Consider both owned and inherited features of a class.



cd2gui_06.atl

CD_PetStore_inh.xmi

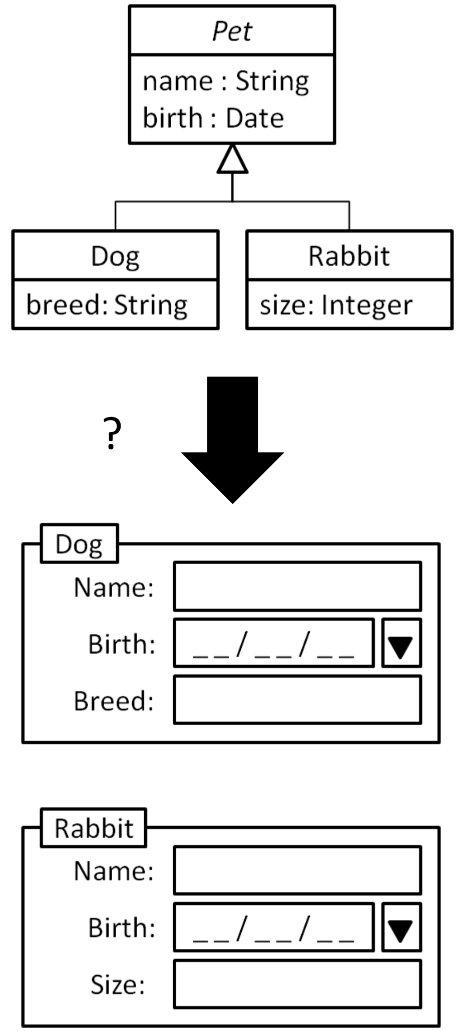
Lazy rules

- Change is straightforward:

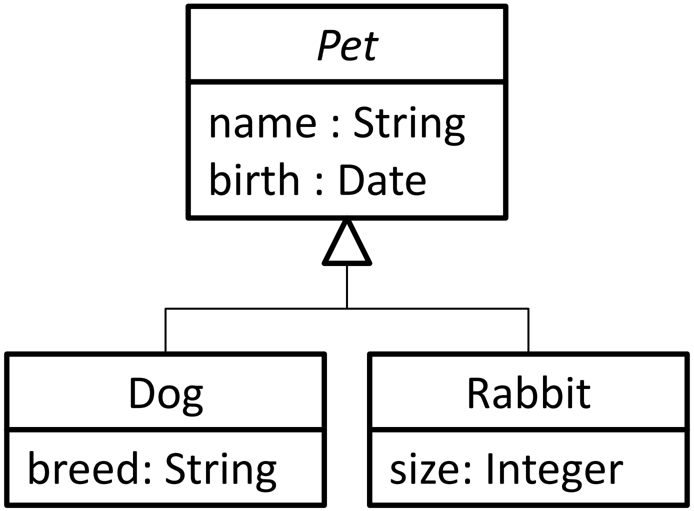
```
helper context CD!Class def: allFeatures : Sequence(CD!Feature) =
  self.superclasses->collect(c | c.allFeatures)->flatten()
                   ->union(self.features);

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

Lazy rules

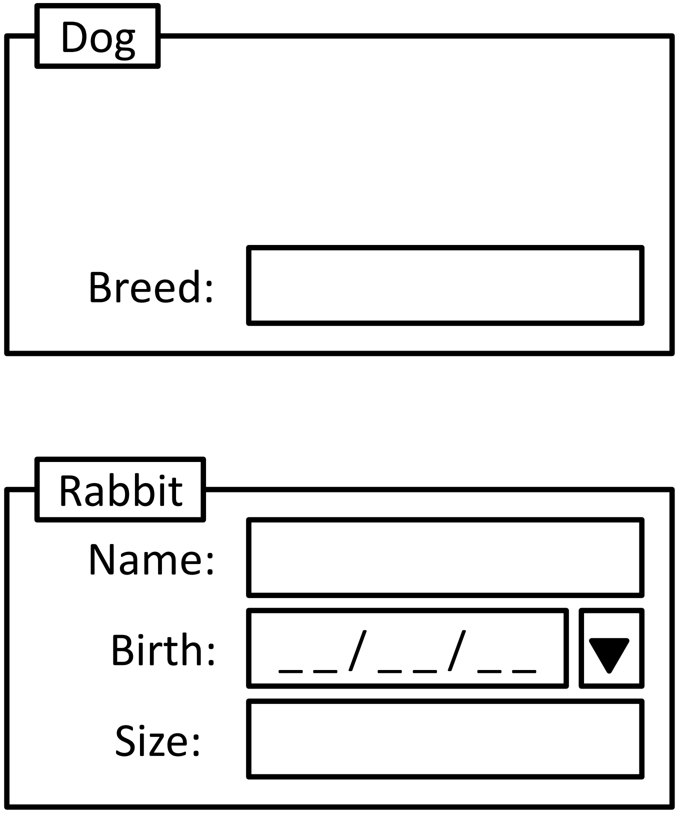
```
rule class2frame {
  from c : CD!Class ( not c.isAbstract )
  to   f : GUI!Frame (
     title <- c.name,
     widgets <- c.allFeatures
  )
}

rule attribute2text {
  from p : CD!Attribute
       ( p.isText() )
  to   t : GUI!Text
}

rule attribute2date {
  from p : CD!Attribute ( p.isDate() )
  to   t : GUI!DatePicker
}
```



Lazy rules



you get



Why?

Lazy rules

- Child stealing
  - Each instance of an Atttribute generates one Widget instance
    - The last binding wins
  - We need one instance per concrete subclass

cd2gui_07.atl

CD_PetStore_inh.xmi

Lazy rules

```
rule class2frame {
  from c : CD!Class ( not c.isAbstract )
  to   f : GUI!Frame (
     title <- c.name,
     widgets <- c.allAttributes->collect(f |
        if f.isText()       then thisModule.attribute2text(f)
        else if f.isInt()   then thisModule.attribute2int(f)
        else if f.isDate()  then thisModule.attribute2date(f)
        else                     OclUndefined endif endif endif
     )
  )
}
```

You need to “pattern match” explicitly. We will use abstract rules later to solve this.

```
lazy rule attribute2text {
  from p : CD!Attribute
  to   t : GUI!Text
}
```

```
lazy rule attribute2date {
  from p : CD!Attribute
  to   t : GUI!DatePicker
}
```

What about labels? We need another lazy rule **attribute2label**

Unique lazy rules

- Similar to lazy rules, but they keep trace links
  - Useful if a matched rule is subordinated to the execution of others
  - Required if the target element of a lazy rule must be reused
    - For example, the previous modification did not considered the layout.
    - We need to create GridInfo elements (with called rules) and make them point to the wiget they are layout out.

Called rules

- Rules invoked explicitly (like lazy rules) but with parameters (like a function)
- A called rule has a return value, which is the last value of the **do** section
  - This is important if the result of a called rule is going to be assigned

Unique lazy rules and called rules

- Revisiting the example...

```
rule class2frame {
  from c: CD!Class ( not c.isAbstract )
  to frm: GUI!Frame (
       widgets <- ...
     ),
     grid: GUI!GridLayout (
        numColumns <- 2,
        info <- c.allAttributes->collect(a |
           Sequence { thisModule.widget2gridInfo(a, 1),
                      thisModule.label2gridInfo(a, 2) }
     )
}
```

Called rule invocation

cd2gui_07.atl

CD_PetStore_inh.xmi

Unique lazy rules and called rules

```
rule widget2gridInfo(f : CD!Attribute, i : Integer) {
  to tgt : GUI!GridInfo (
    column <- i,
    widget <- if f.isText()    then thisModule.attribute2text(f)
              else if f.isInt()  then thisModule.attribute2int(f)
              else if f.isDate() then thisModule.attribute2date(f)
              else OclUndefined.fail_() endif endif endif
    ) do {
      tgt; -- This is the return value!
    }
}
```

Can use lazy rules here?

cd2gui_08.atl

CD_PetStore_inh.xmi

Unique lazy rules and called rules

- In this case, the lazy rules (`attribute2text`, `attribute2int` and `attribute2date`) needs to be converted into **unique lazy rules**
  - This is so because GridInfo.widget is not containment
  - We have to assign the reference contained in Frame.widgets

To finish this part

- The last example is just wrong again :-S
  - By using unique lazy rules we went back to the original problem of child stealing...
- This is difficult to solve
  - We will use some imperative constructs later to deal with it.