Towards Systematic Mutations for and with ATL Model Transformations

Javier Troya, Alexander Bergmayr, Loli Burgueño, Manuel Wimmer

10th International Workshop on Mutation Analysis
Mutation 2015, Graz, Austria
13.04.2015

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology

GISUM/Atenea Research Group
ETS Ingeniería Informática
Unviersidad de Málaga

ICST 2015

*This work is co-funded by the European Commission under the ICT Policy Support Programme, grant no. 317859

Model-Driven Engineering (MDE)

Basics

- **Models are used as first-class artifacts**
  - **A model represents (is an abstraction of) a system**
  - **Models conform to a metamodel**
  - **Models may be transformed into other models**

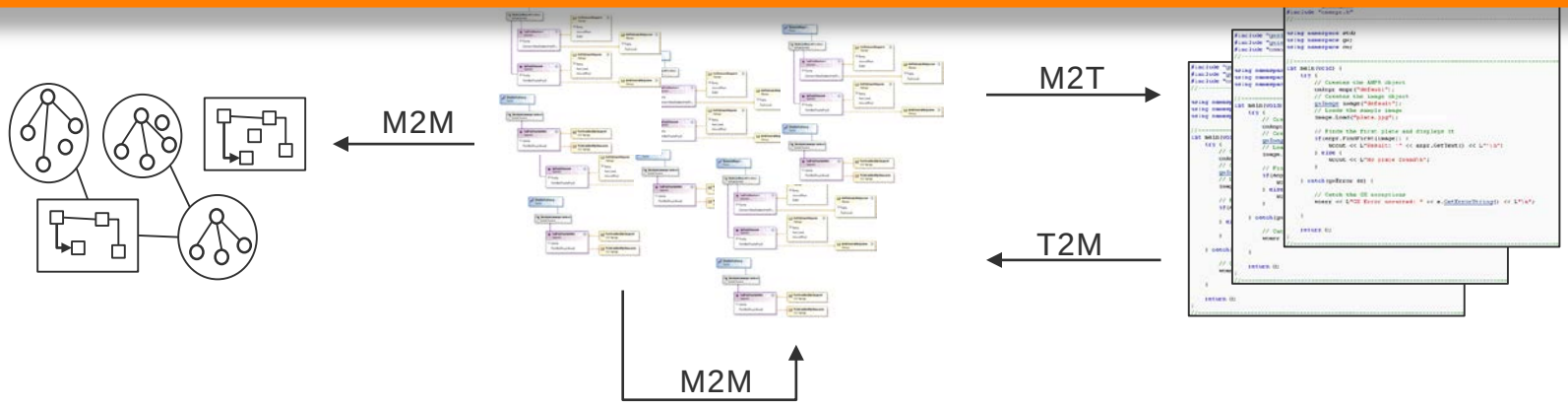
represents

Model Transformations

- Model Transformations are at the heart of MDE
- Text-to-Model, Model-to-Model and Model-to-Text Transformations are used with different purposes
  - Reverse-engineer models from code
  - Forward-engineer code from models

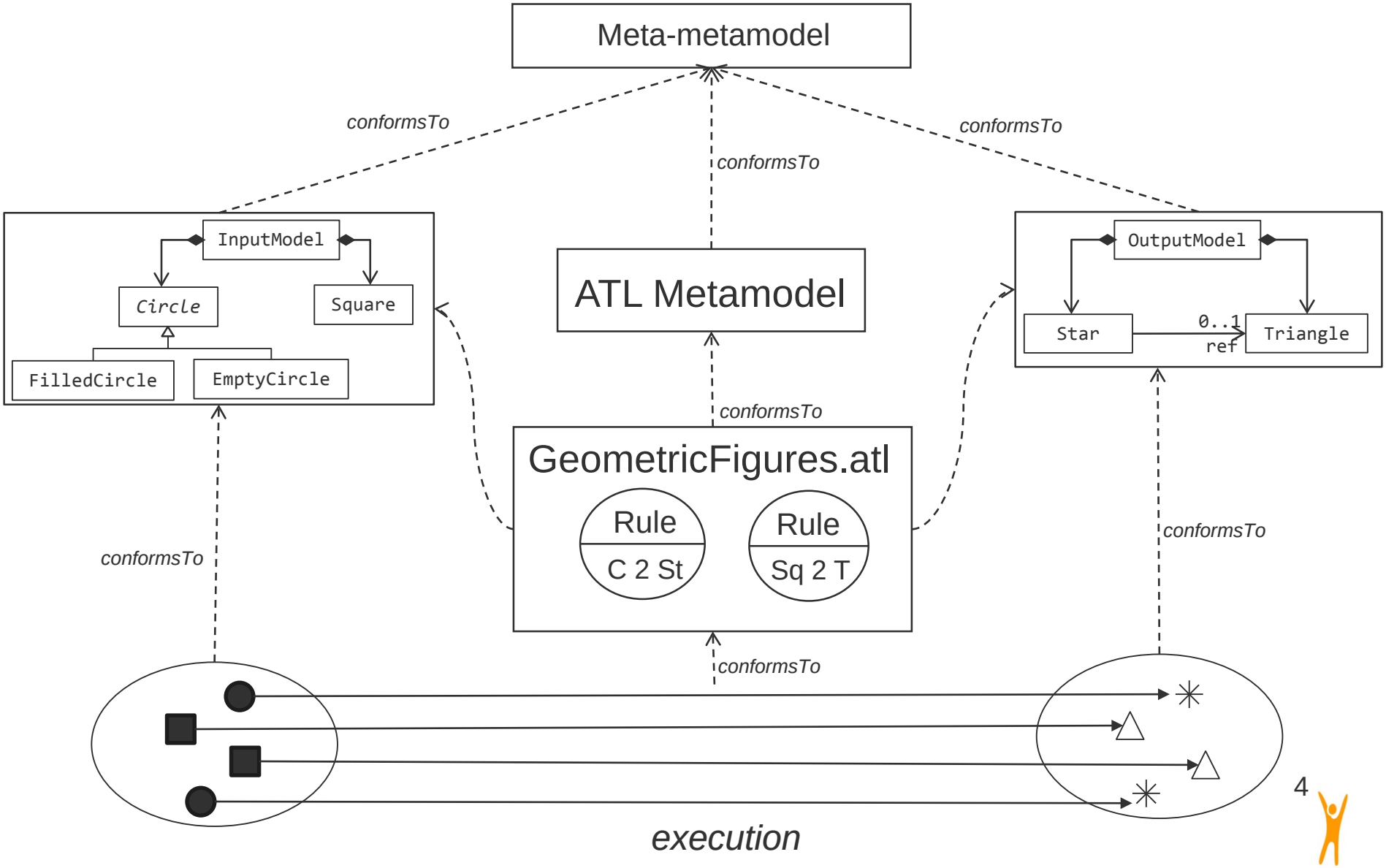
The quality of the generated artifacts is highly affected by the quality of the developed model transformations

- For this reason, it is important to test the transformations



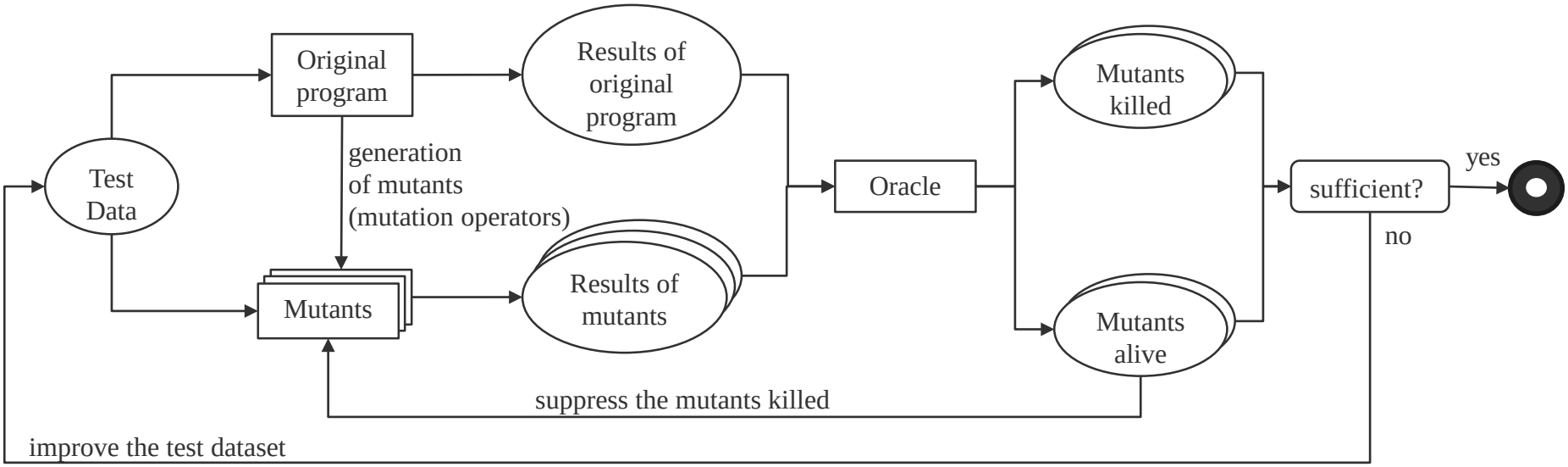
Model-to-Model Transformation

Example



Mutation Testing

General Schema



(adapted from [1])

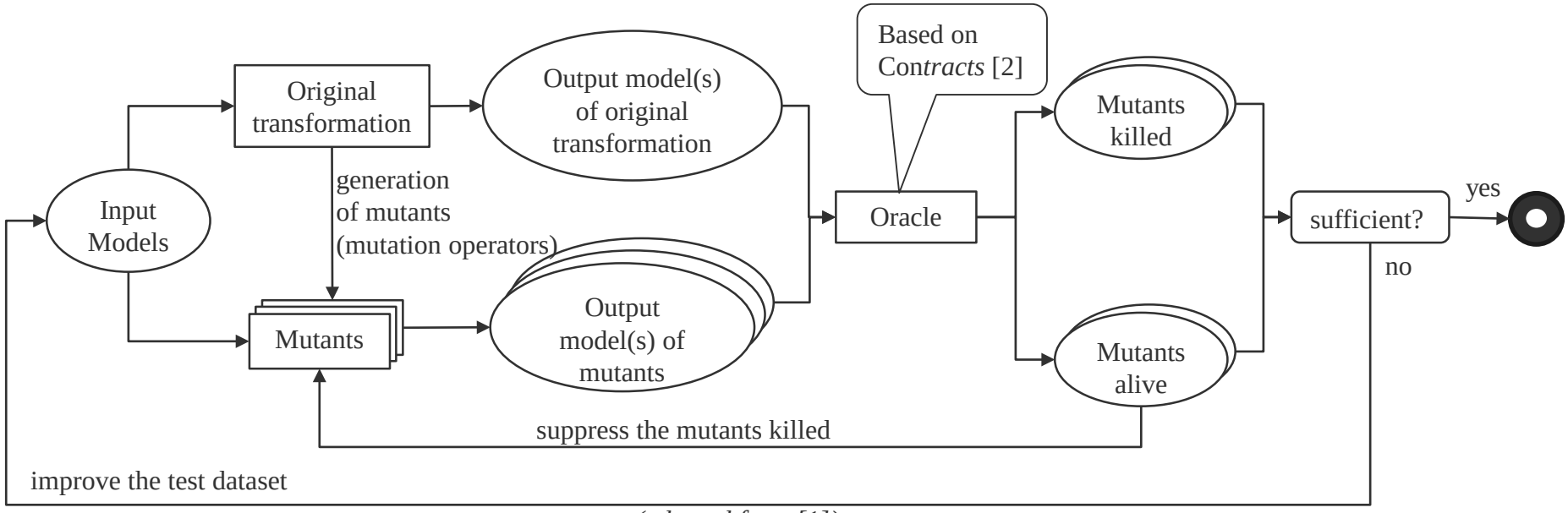
[1] J.-M. Mottu, B. Baudry, and Y. Le Traon, “Mutation Analysis Testing for Model Transformations”, in ECMDA-FA LNCS 4066. Springer, 2006, pp. 376–390.

Mutation Testing

General Schema

- Mutation Testing is gaining importance in Model-Driven Engineering
  - First Interactive Workshop on Combining Mutation Testing and Model Transformation.

    Workshop of STAF Conferences 2014 - http://stafconferences.info/



(adapted from [1])

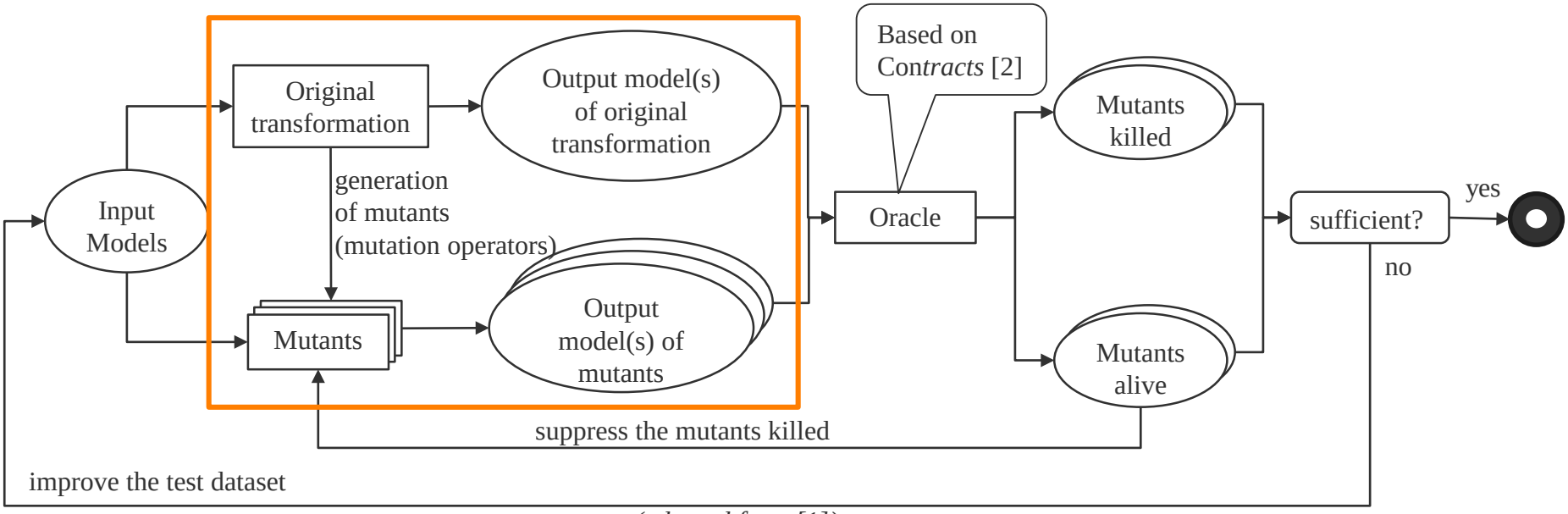
[1] J.-M. Mottu, B. Baudry, and Y. Le Traon, “Mutation Analysis Testing for Model Transformations”, in ECMDA-FA LNCS 4066. Springer, 2006, pp. 376–390.

[2] L. Burgueño, J. Troya, M. Wimmer, A. Vallecillo, “Static Fault Localization in Model Transformations”, in IEEE TSE, 2014. http://dx.doi.org/10.1109/TSE.2014.2375201

Mutation Testing

General Schema

- Our objective is to create mutants for ATL Model Transformations
  1. Identify mutation operators
  2. Identify the possible changes produced by mutants in output models
  3. Develop a way to automatically generate mutants, since manually generating them appears unfeasible



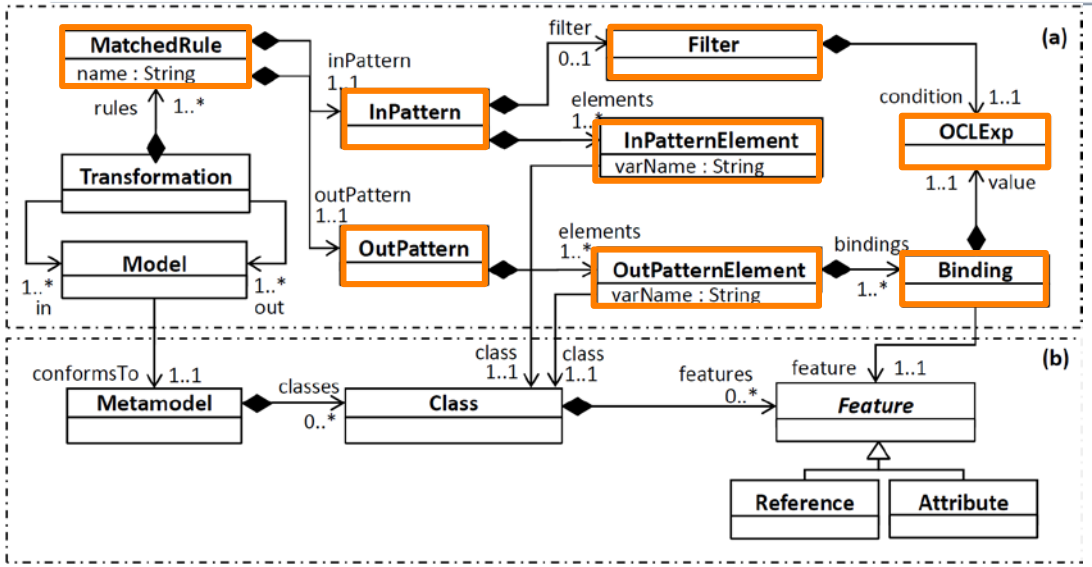
(adapted from [1])

[1] J.-M. Mottu, B. Baudry, and Y. Le Traon, "Mutation Analysis Testing for Model Transformations", in ECMDA-FA LNCS 4066. Springer, 2006, pp. 376–390.

[2] L. Burgueño, J. Troya, M. Wimmer, A. Vallecillo, "Static Fault Localization in Model Transformations", in IEEE TSE, 2014. http://dx.doi.org/10.1109/TSE.2014.2375201

ATL

1- Identify Mutation Operators



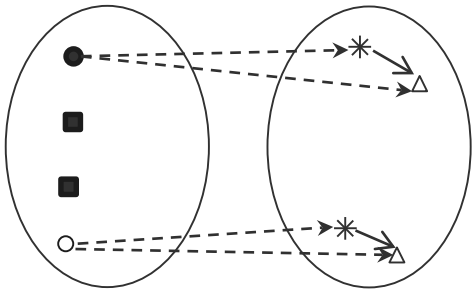

```
rule Place {
    from
      c : MM_IN!Circle
        (c.oclIsTypeOf(MM_IN!FilledCircle))
    to
      n : MM_OUT!Star (
          name <- c.name,
          ref <- t
      ),
      t : MM_OUT!Triangle (
          name <- c.name + '_triangle'
      )
}
```

Concept	Mutation Operator
Matched Rule	Addition Deletion Name Change
In Pattern Element	Addition Deletion Class Change Name Change
Filter	Addition Deletion Condition Change
Out Pattern Element	Addition Deletion Class Change Name Change
Binding	Addition Deletion Value Change Feature Change

On the Generation of Mutants for ATL Transformations

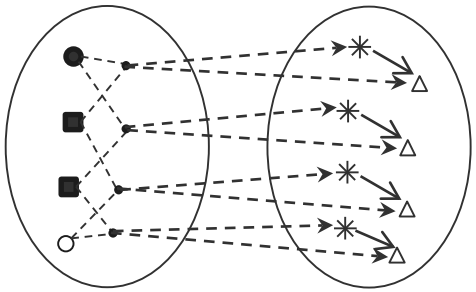
2 - Identify the possible changes produced by mutants in output models

- Effect of Adding an *InPatternElement* in a rule



Original transformation

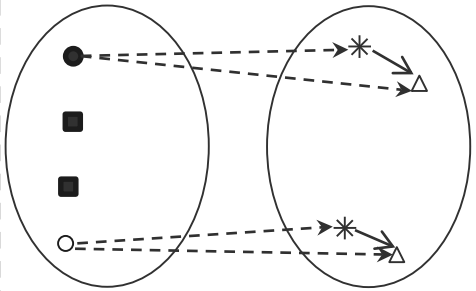
```
rule Rule{
   from
     c : MM_IN!Circle
   to
     s : MM_OUT!Star (
        ref <- t
     ),
     t : MM_OUT!Triangle()
}
```



InPatternElement added to rule

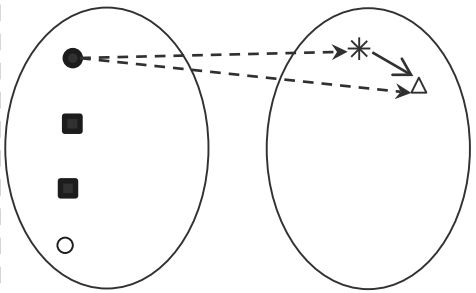
```
rule Rule_Mutated{
   from
     c : MM_IN!Circle,
     a : MM_IN!Square
   to
     s : MM_OUT!Star (
        ref <- t
     ),
     t : MM_OUT!Triangle()
}
```

- Effect of Adding a *Filter* in a rule



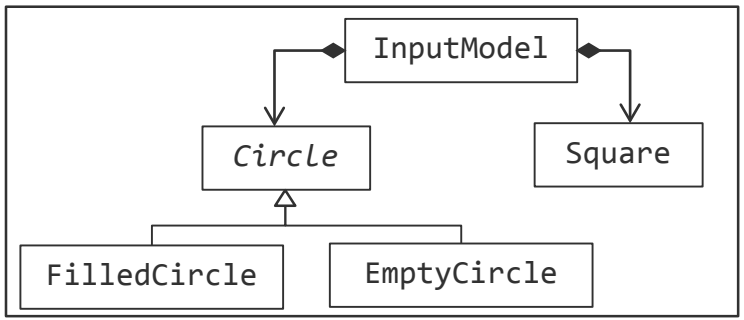
Original transformation

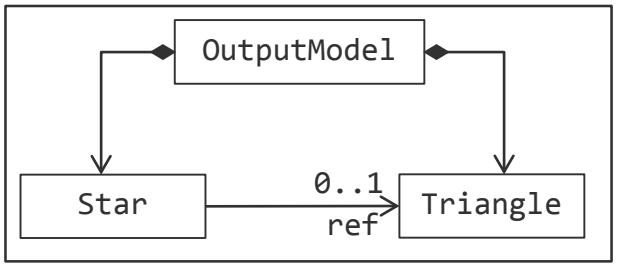
```
rule Rule{
   from
     c : MM_IN!Circle
   to
     s : MM_OUT!Star (
        ref <- t
     ),
     t : MM_OUT!Triangle()
}
```



Filter added to rule

```
rule Rule{
   from
     c : MM_IN!Circle
        (c.oclIsTypeOf
         (MM_IN!FilledCircle))
   to
     s : MM_OUT!Star (
        ref <- t
     ),
     t : MM_OUT!Triangle()
}
```

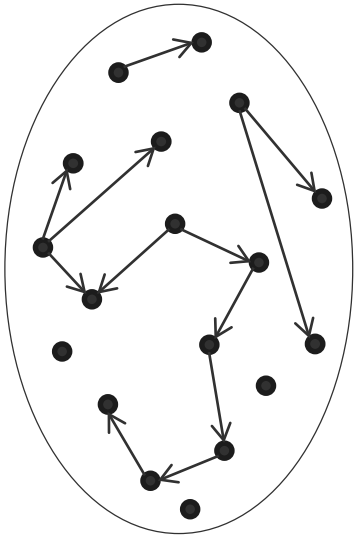




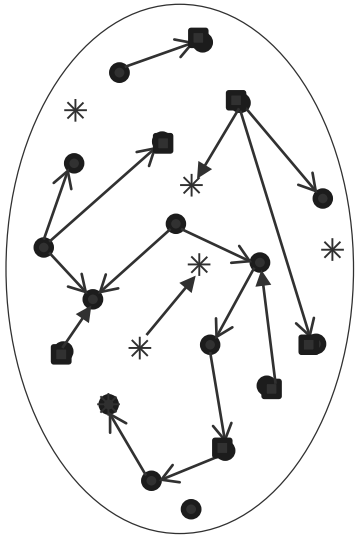
On the Generation of Mutants for ATL Transformations

2 - Identify the possible changes produced by mutants in output models

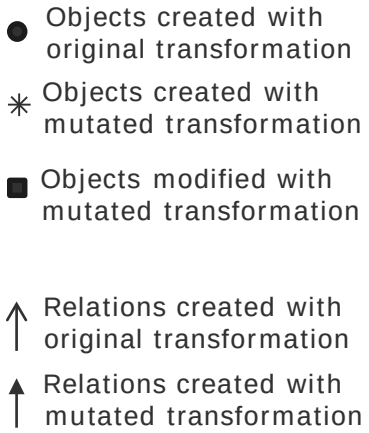
- **OA**: Object Addition
- **OR**: Object Replacement
- **OD**: Object Deletion
- **RD**: Relationship Deleted
- **OPI**: Object Property Initialized
- **OPN**: Object Property set to Null
- **OPM**: Object Property Modified
- **RA**: Relationship Added



(a) Original output model



(b) Output model produced by a mutated transformation



On the Generation of Mutants for ATL Transformations

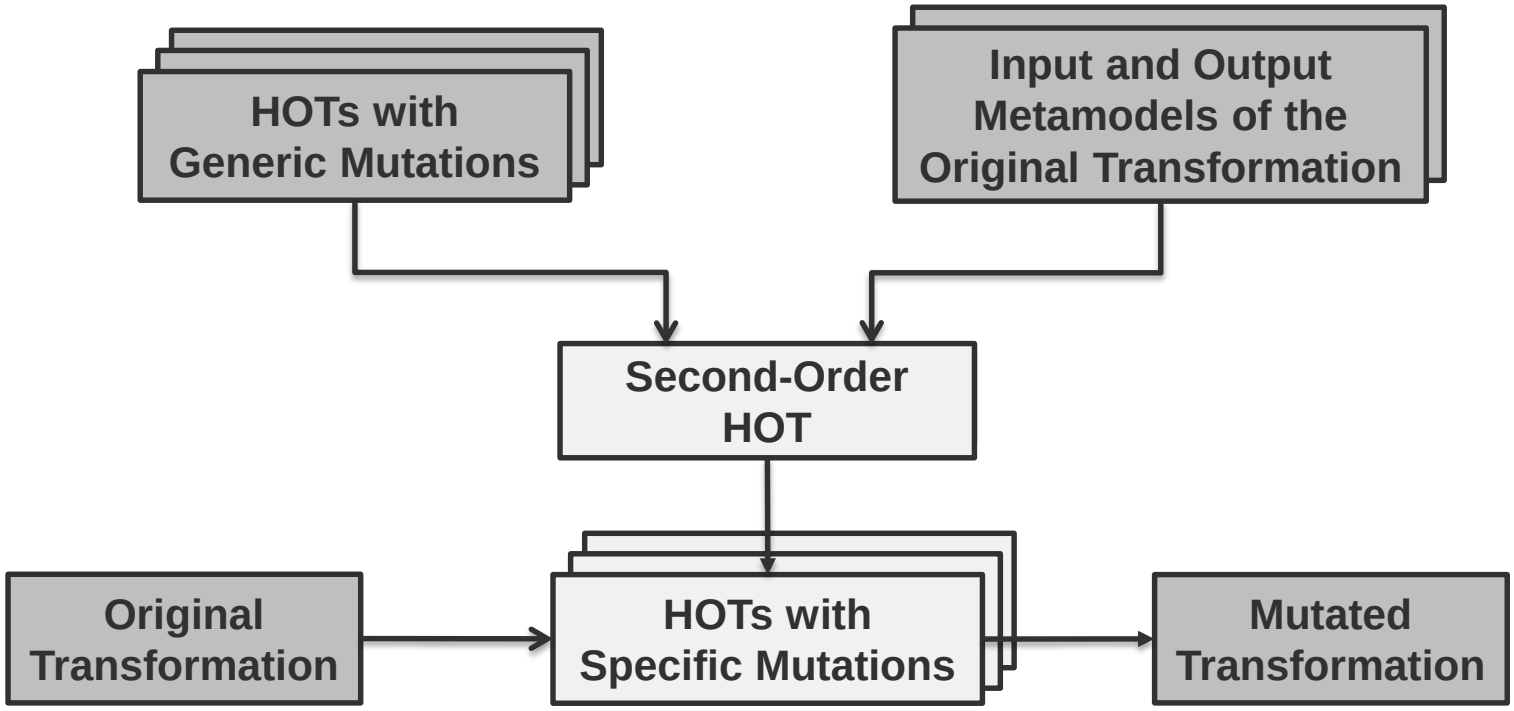
1 & 2 – Idenitfy Mutation Operators and their Effects in the Output Models

Concept	Mutation Operator	Consequence
Matched Rule	Addition Deletion Name Change	OA;[RA] OD;[RD]
In Pattern Element	Addition Deletion Class Change Name Change	OA;[RA] OD;[RA] OD;OA;[RD];[RA]
Filter	Addition Deletion Condition Change	OD OA OA;OD
Out Pattern Element	Addition Deletion Class Change Name Change	OA;[RA] OD;[RD] OR;[RA];[RD]
Binding	Addition Deletion Value Change Feature Change	OPI;[RA] OPN;[RD] OPM;[RA];[RD]

- **OA**: Object Addition
- **OD**: Object Deletion
- **OPI**: Object Property Initialized
- **OPN**: Object Property set to Null
- **OPM**: Object Property Modified
- **OR**: Object Replacement
- **RA**: Relationship Added
- **RD**: Relationship Deleted
- **[ ]**: It may happen or not

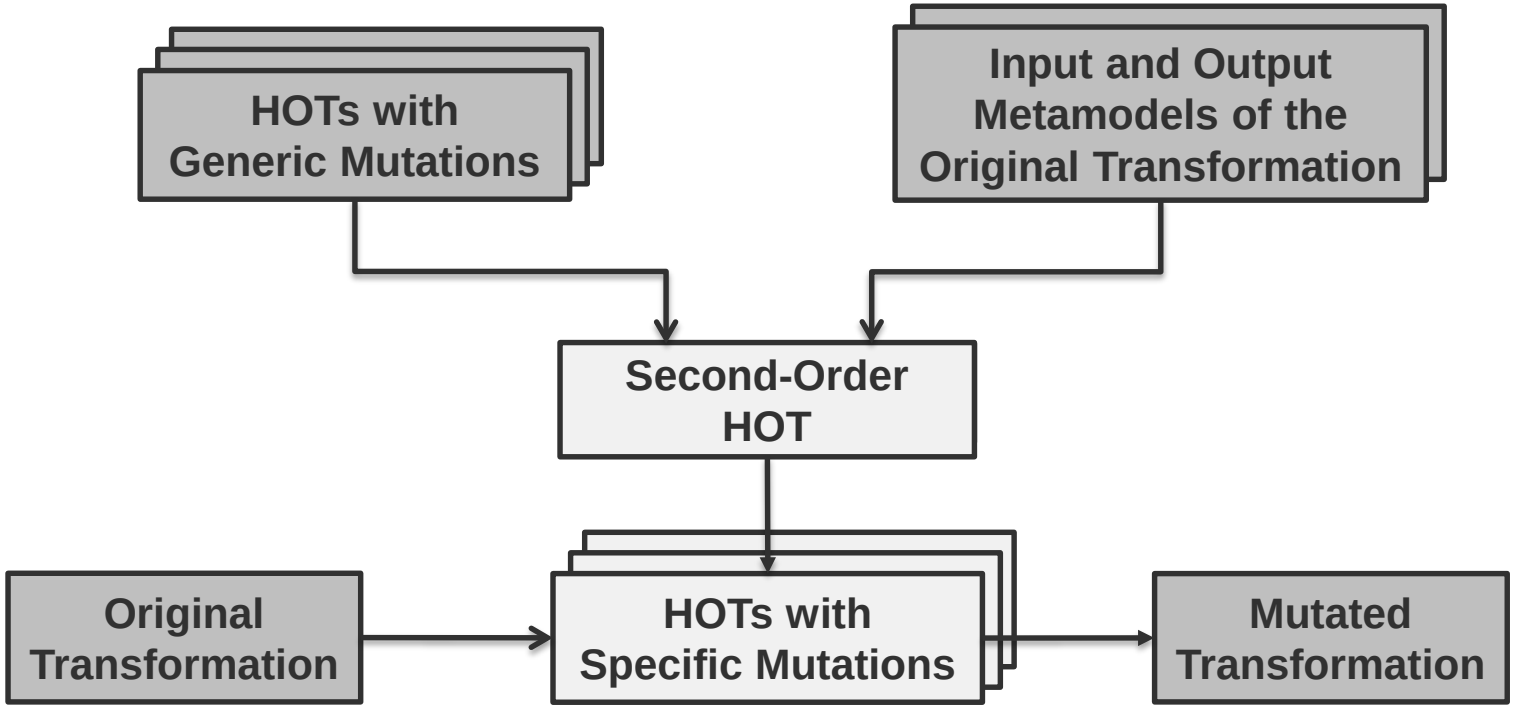
On the Generation of Mutants for ATL Transformations

3 – Develop a way to automatically produce mutants



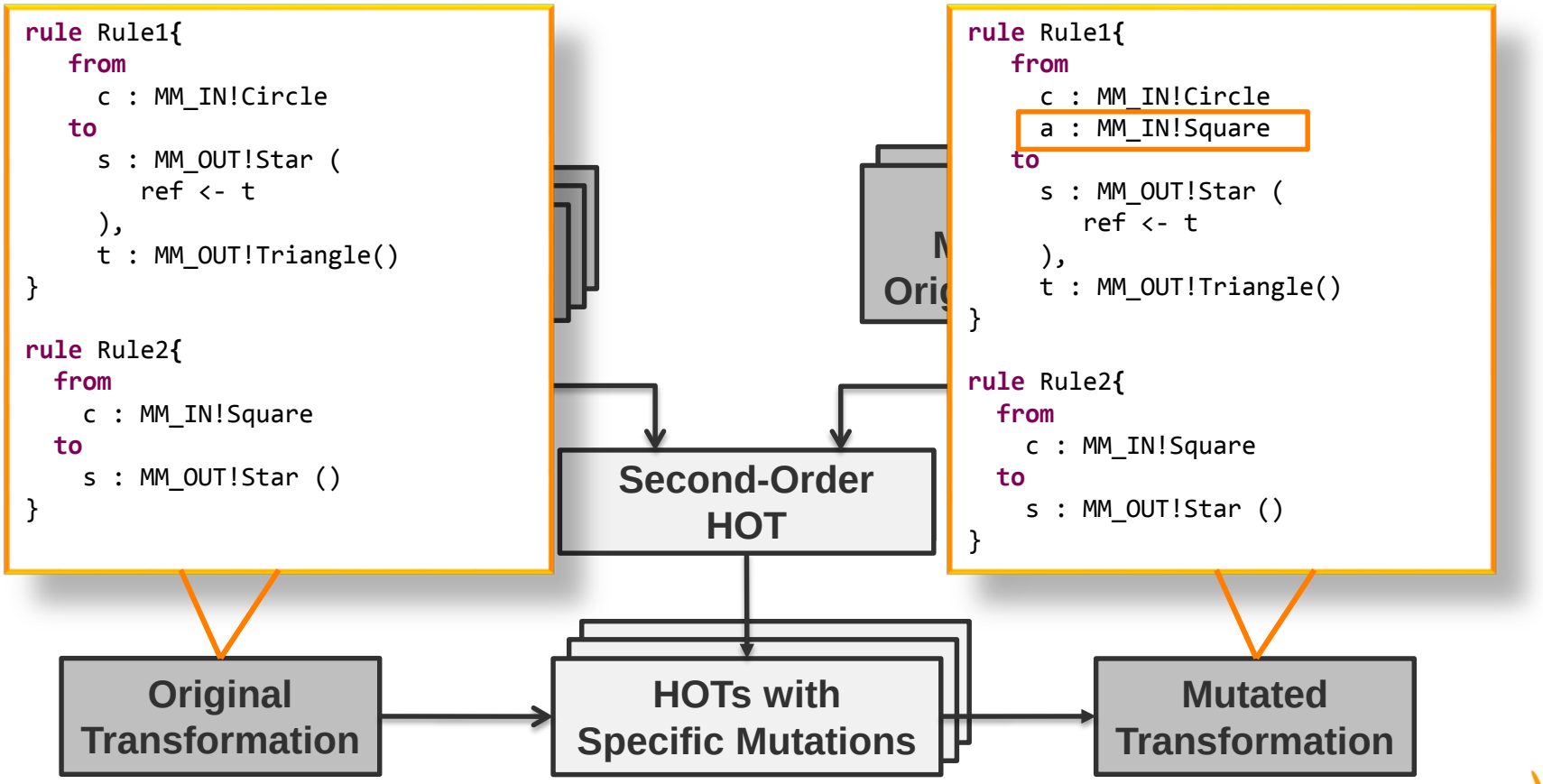
On the Generation of Mutants for ATL Transformations

- We want to automatically produce a mutant where a rule has an extra *InPatternElement*
  - We want to be able to produce such mutant for any ATL transformation
  - We do not have information about the specific transformation to mutate



On the Generation of Mutants for ATL Transformations

3 – Develop a way to automatically produce mutants



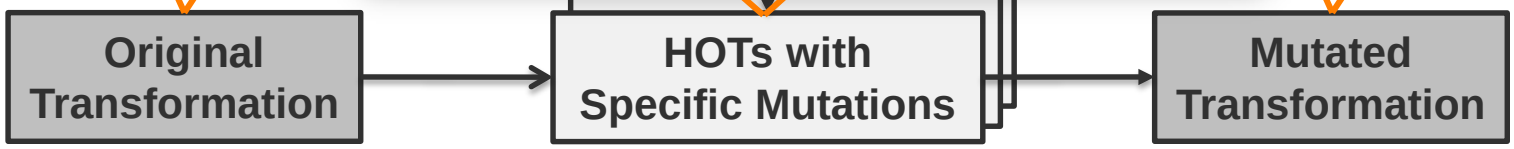
On the Generation of Mutants for ATL Transformations

3 – Develop a way to

```
rule Rule1{
   from
     c : MM_IN!Circle
   to
     s : MM_OUT!Star (
        ref <- t
     ),
     t : MM_OUT!Triangl
}

rule Rule2{
  from
    c : MM_IN!Square
  to
    s : MM_OUT!Star ()
}
```

```
-- @atlcompiler emftvm

module AddInPatternElement_FirstRule;
create OUT : ATL refining IN : ATL;

-- Sequence for giving new variable names to
-- new pattern elements that are created
helper def : varNames : Sequence(String) = Sequence{
          'a','aa','b','bb','c','cc','d','dd','e',
           'ee','f','ff','...'};

rule AddInPatternElement {
 --We will add the SIPE only in the first rule
  from s : ATL!InPattern
    (ATL!Rule.allInstances()->first() = s."rule")
  to
    t : ATL!InPattern(
        elements <- s.elements -> append(newIPE)
    ),
    newIPE : ATL!InPatternElement (
       varName <- thisModule.varNames->
                     any(n | ATL!PatternElement.allInstances()
                      ->collect(pe|pe.varName)->excludes(n)),
       type <- newOCLType
     ),
    -- The type is composed of a model and a name: model!name
    newOCLType : ATL!OclModelElement(
      model <- s.elements->first().type.model,
      name <- 'Square'
    )
}
```

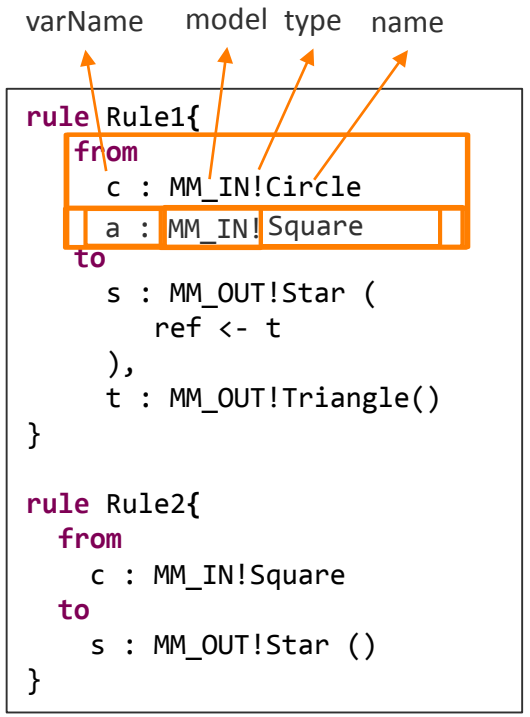
```
!Circle
!Square

T!Star (
 t

T!Triangle()

Square

!Star ()
```



On the Generation of Mutants for ATL Transformations

3 – Develop a way to automatically produce mutants: Scenario by Example

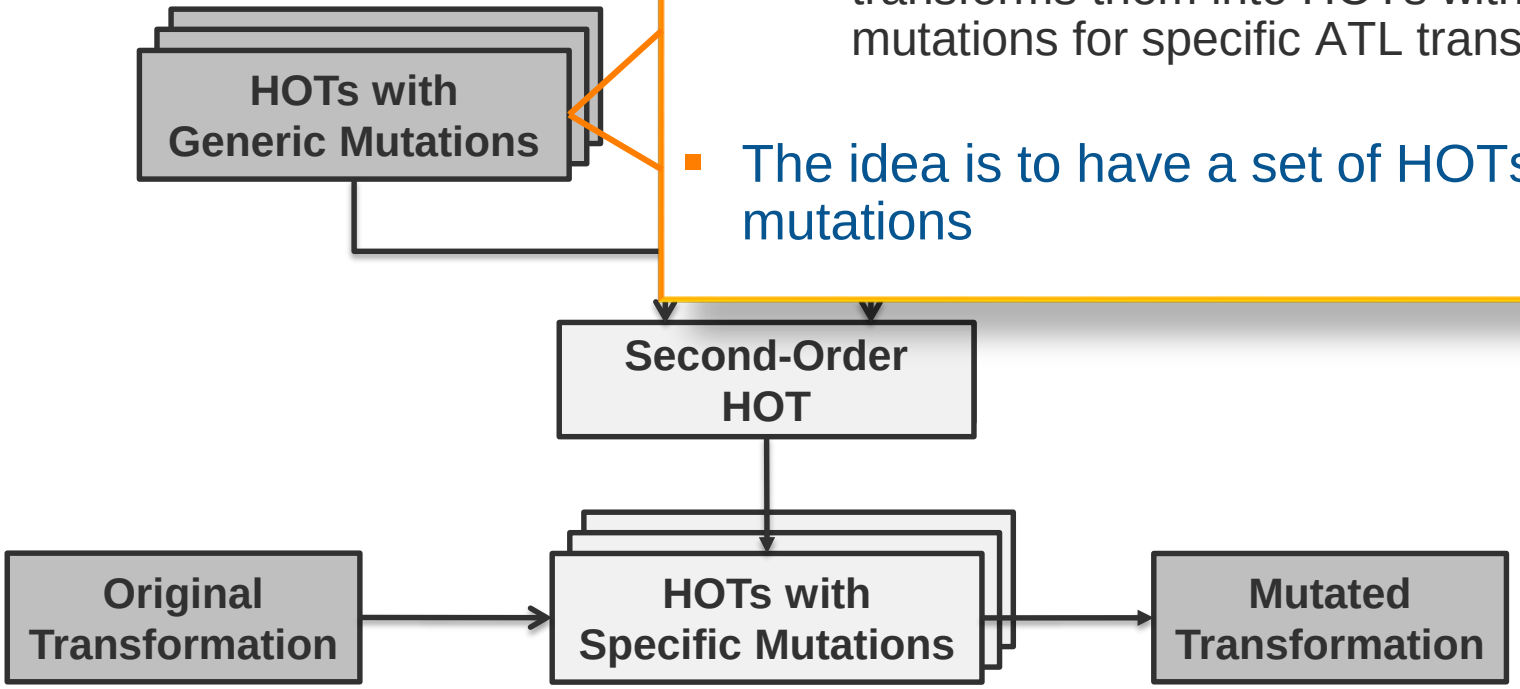
```
-- @atlcompiler emftvm

module AddInPatternElement_FirstRule;
create OUT : ATL refining IN : ATL;

-- Sequence for giving new variable names to
-- new pattern elements that are created
helper def : varNames : Sequence(String) = Sequence{
          'a','aa','b','bb','c','cc','d','dd','e','ee','f','ff','...'};

rule AddInPatternElement {
 --We will add the SIPE only in the first rule
  from s : ATL!InPattern
    (ATL!Rule.allInstances()->first() = s."rule")
  to
    t : ATL!InPattern(
        elements <- s.elements -> append(newIPE)
    ),
    newIPE : ATL!InPatternElement (
       -- We have to give a variable name that no PatternElement has
       varName <- thisModule.varNames->any(n | ATL!PatternElement.allInstances()
                                    ->collect(pe|pe.varName)->excludes(n)),
       type <- newOCLType
     ),
    -- The type is composed of a model and a name: model!name
    newOCLType : ATL!OclModelElement(
      model <- s.elements->first().type.model,
      name <- 'Square'
    )
}
```



On the Generation of Mutants for ATL Transformations

3 – Develop a way to automatically produce mutants

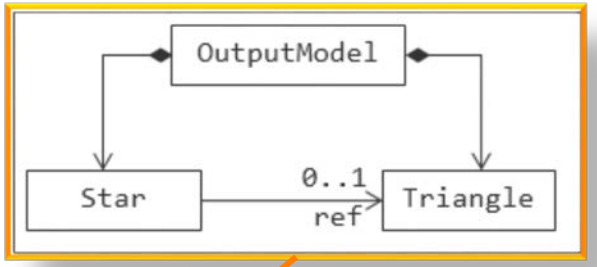
- We say that a mutation is generic when it is not defined in the context of a specific transformation
  - These mutations can then be used to mutate any model transformation
  - They are the input of a second-order HOT that transforms them into HOTs with specific mutations for specific ATL transformations
- The idea is to have a set of HOTs with generic mutations

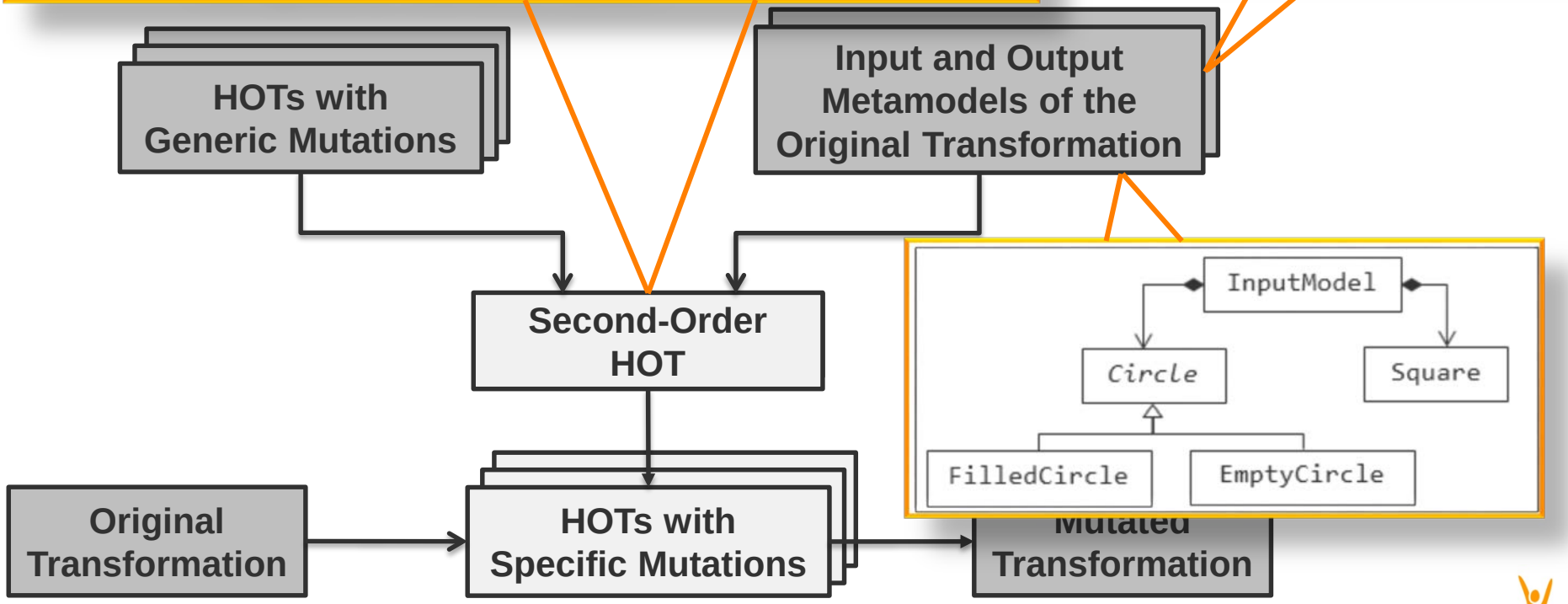


On the Generation of Mutants for ATL Transformations

3 – Develop a way to automatically produce mutants

- It takes HOTs with generic mutations and transforms them into HOTs with specific mutations
  - For this, it is needed to have the metamodels of the specific transformation





On the Generation of Mutants for ATL Transformations

3 – Develop a way to automatically produce mutants: Scenario by Example

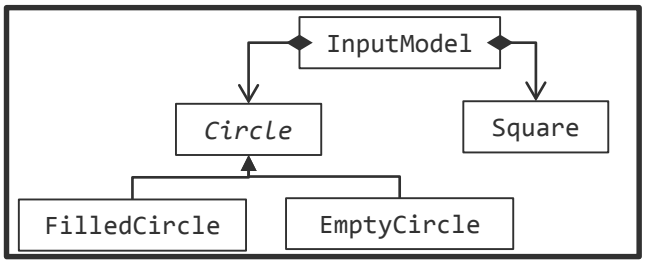
HOT with generic mutation

```
-- @atlcompiler emftvm

module AddInPatternElement_FirstRule;
create OUT : ATL refining IN : ATL;

-- Sequence for giving new variable names to
-- new pattern elements that are created
helper def : varNames : Sequence(String) = Sequence{
          'a','aa','b','bb','c','cc','d','dd','e',
           'ee','f','ff','...'};

rule AddInPatternElement {
 --We will add the SIPE only in the first rule
  from s : ATL!InPattern
    (ATL!Rule.allInstances()->first() = s."rule")
  to
    t : ATL!InPattern(
        elements <- s.elements -> append(newIPE)
    ),
    newIPE : ATL!InPatternElement (
       -- We have to give a variable name that no PatternElement has
       varName <- thisModule.varNames->
                     any(n | ATL!PatternElement.allInstances()
                      ->collect(pe|pe.varName)->excludes(n)),
       type <- newOCLType
     ),
    -- The type is composed of a model and a name: model!name
    newOCLType : ATL!OclModelElement(
      model <- s.elements->first().type.model,
      name <- 'Complete_IPE'
    )
}
```

2nd Order HOT

HOT with specific mutation

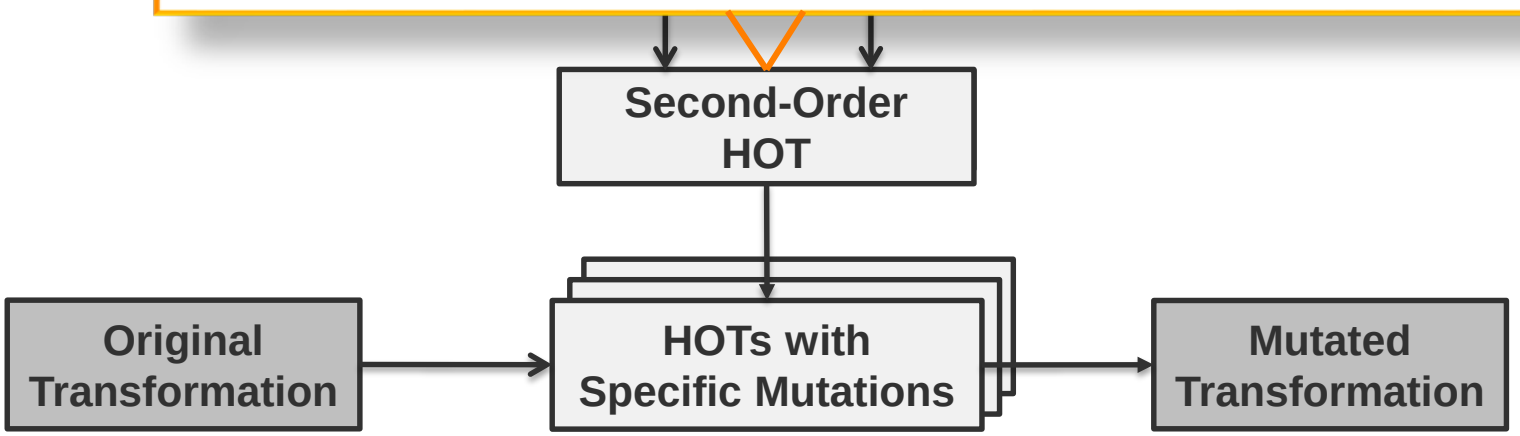
```
-- @atlcompiler emftvm

module AddInPatternElement_FirstRule;
create OUT : ATL refining IN : ATL;

-- Sequence for giving new variable names to
-- new pattern elements that are created
helper def : varNames : Sequence(String) = Sequence{
          'a','aa','b','bb','c','cc','d','dd','e',
           'ee','f','ff','...'};

rule AddInPatternElement {
 --We will add the SIPE only in the first rule
  from s : ATL!InPattern
    (ATL!Rule.allInstances()->first() = s."rule")
  to
    t : ATL!InPattern(
        elements <- s.elements -> append(newIPE)
    ),
    newIPE : ATL!InPatternElement (
       -- We have to give a variable name that no PatternElement has
       varName <- thisModule.varNames->
                     any(n | ATL!PatternElement.allInstances()
                      ->collect(pe|pe.varName)->excludes(n)),
       type <- newOCLType
     ),
    -- The type is composed of a model and a name: model!name
    newOCLType : ATL!OclModelElement(
      model <- s.elements->first().type.model,
      name <- 'Square'
    )
}
```



On the Generation of Mutants for ATL Transformations

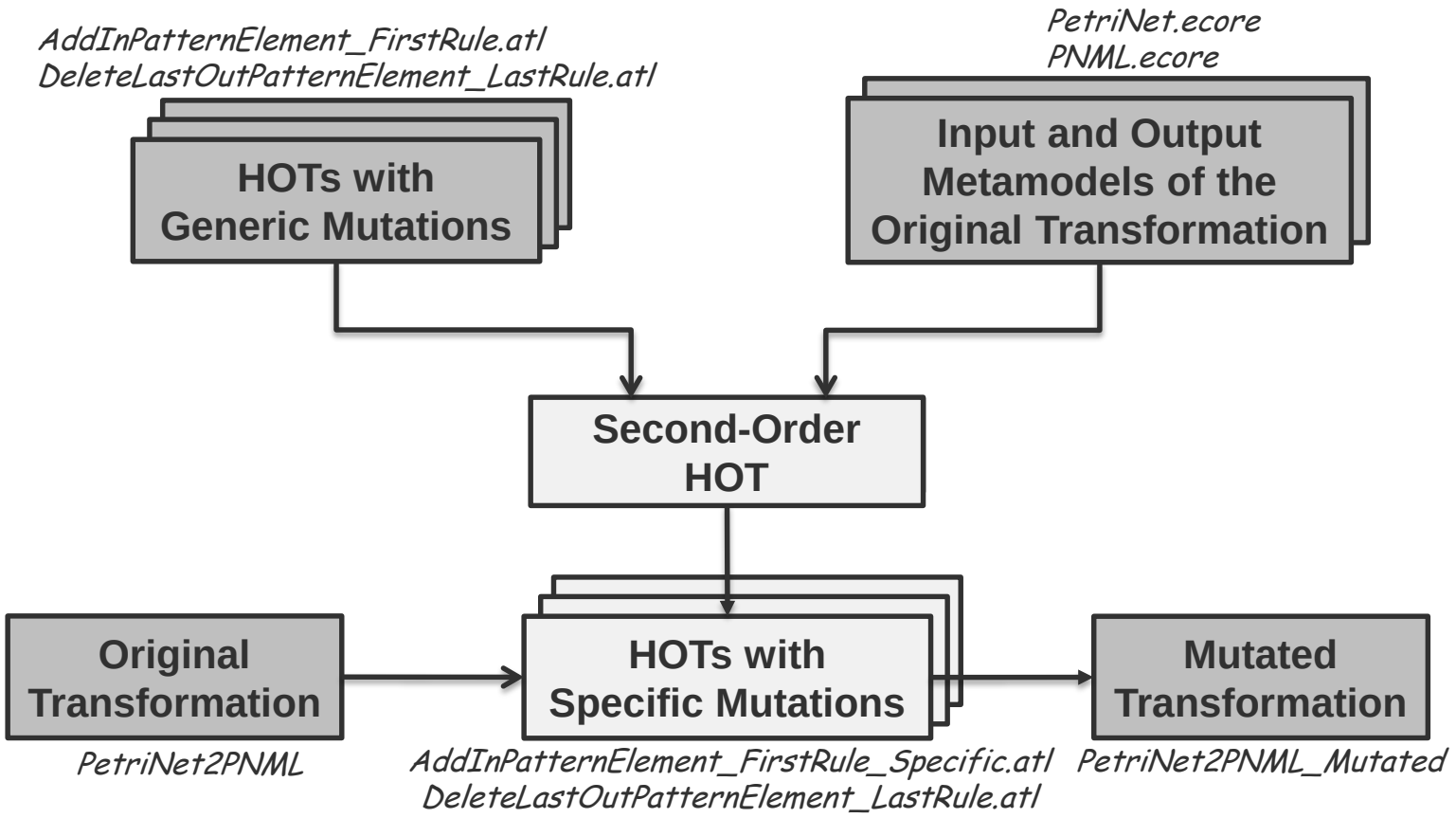
3 – Develop a way to automatically produce mutants

```
-- @atlcompiler emftvm

module SecondOrderHOT;
create OUT : ATL refining IN : ATL, IN_MM : IN_MM, OUT_MM : OUT_MM;

helper def : random() : Real =
"#native"!"java::util::Random".newInstance().nextDouble();

-- A StringExp is one of the types that can conform to the value part of a Binding
-- Since the generic mutation transformation adds 'CompleteIt_IM' in the value part,
-- a StringExp is created, whose stringSymbol is 'Complete_IPE'
rule CompleteInMMNames {
  from s : ATL!StringExp (s.stringSymbol = 'Complete_IPE')
  using {
    classes : Sequence(IN_MM!EClass) = IN_MM!EClass.allInstancesFrom('IN_MM')->select(c|not c.abstract);
  }
  to t : ATL!StringExp(
    -- The idea is to have in the following a random class from the input model
    stringSymbol <- classes->at((thisModule.random()*classes->size()).floor()+1).name
  )
}
```



On the Generation of Mutants for ATL Transformations

On the Automatic Generation of Mutants: Current Status of the Prototype*



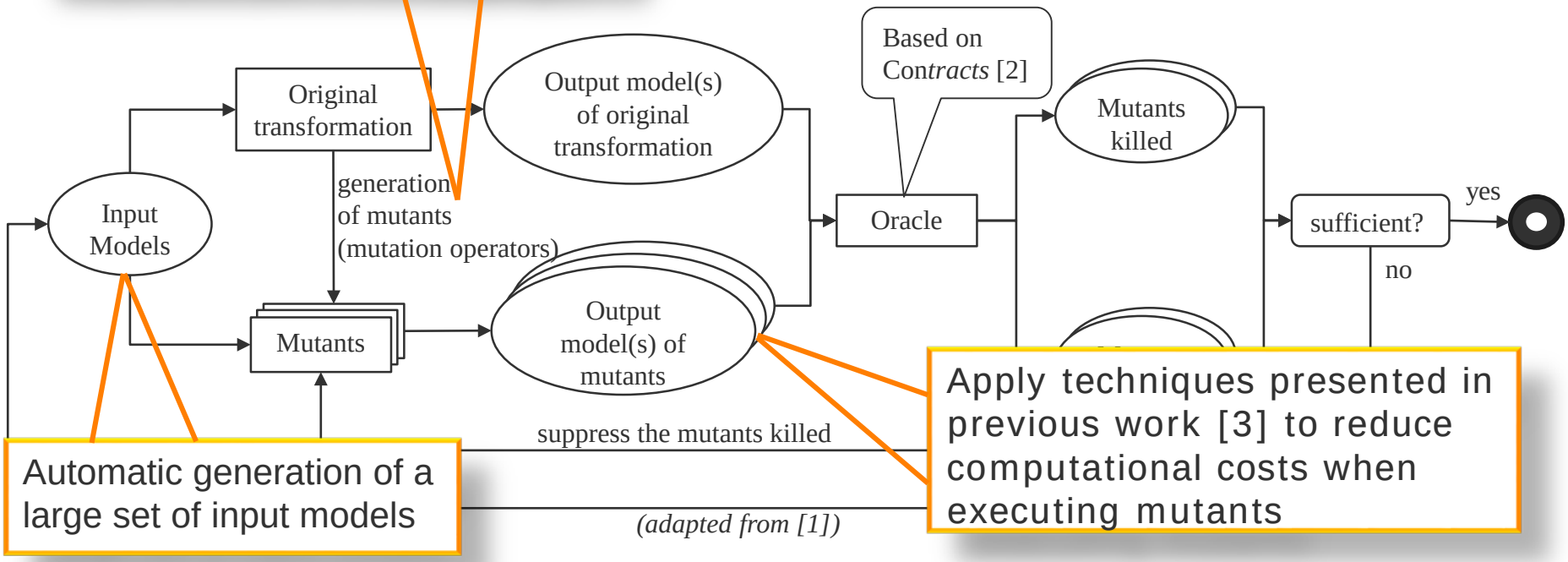
* Available on http://atenea.lcc.uma.es/index.php/Main_Page/Resources/Mutations

Future Work

Several lines of work to explore next

Identify more mutation operators (by studying the OCL package of the ATL metamodel and the imperative part of the language)

Study how the selection of mutations to apply can be driven by the user



Automatic generation of a large set of input models

Apply techniques presented in previous work [3] to reduce computational costs when executing mutants

[1] J.-M. Mottu, B. Baudry, and Y. Le Traon, “Mutation Analysis Testing for Model Transformations”, in ECMDA-FA LNCS 4066. Springer, 2006, pp. 376–390.

[2] L. Burgueño, J. Troya, M. Wimmer, A. Vallecillo, “Static Fault Localization in Model Transformations”, in IEEE TSE, 2014. http://dx.doi.org/10.1109/TSE.2014.2375201

[3] Alex Bergmayr, Javier Troya, Manuel Wimmer. “From Out-place Transformation Evolution to In-place Model Patching”. Proc. of Automated Software Engineering (ASE’14), ACM, 2014, pp 647-652

Towards Systematic Mutations for and with ATL Model Transformations

Javier Troya, Alexander Bergmayr, Loli Burgueño, Manuel Wimmer

THANKS!

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology

GISUM/Atenea Research Group
ETS Ingeniería Informática
Unviersidad de Málaga

ICST 2015

*This work is co-funded by the European Commission under the ICT Policy Support Programme, grant no. 317859