Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework

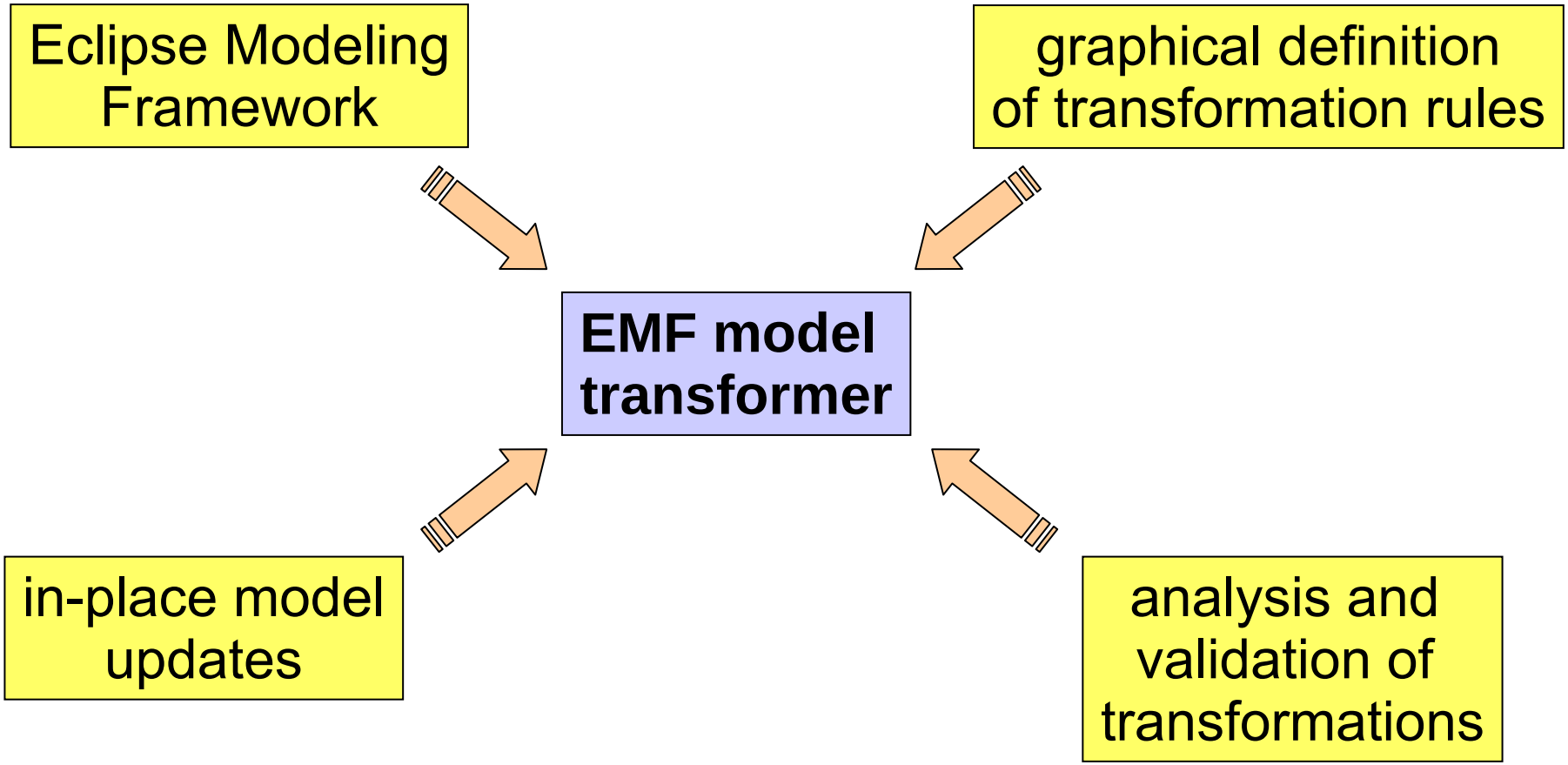
Enrico Biermann, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer, Eduard Weiss

Technical University of Berlin
University of Leicester

A fictious dialog on models

- We made good experiences with model-driven software development. It‘s much faster!
- Good to hear! That‘s what I told you from the beginning.
- But I think our models could be better.
- You could test the model smell and check out model refactoring.
- Okay! That means we can improve our models by transformation?
- Yes, you need some in-place model transformer.

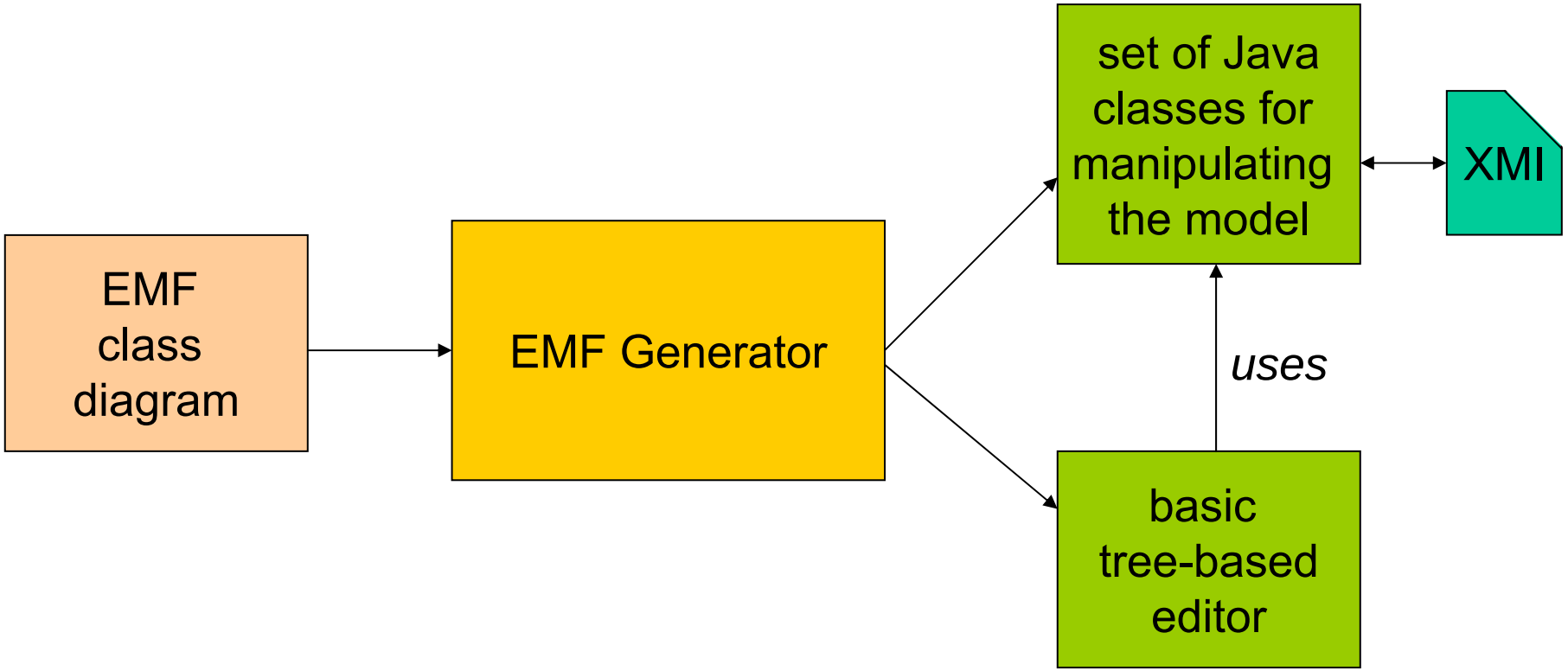
Synopsis


Eclipse Modeling Framework
graphical definition of transformation rules
EMF model transformer
in-place model updates
analysis and validation of transformations

Outline

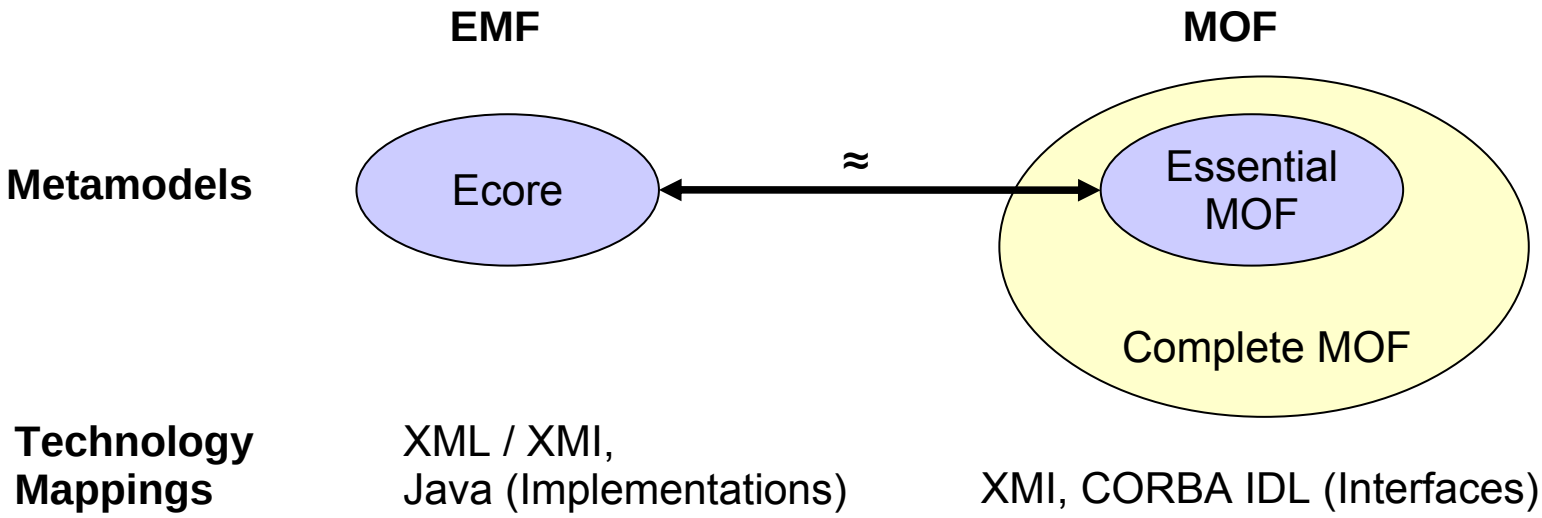
- What is the Eclipse Modeling Framework (EMF)?
- EMF transformation concepts
- Running example: EMF model refactoring
- Execution of EMF transformations
  - Interpreter mode
  - Compiler mode
- Related work
- Conclusions

Eclipse Modeling Framework (EMF)

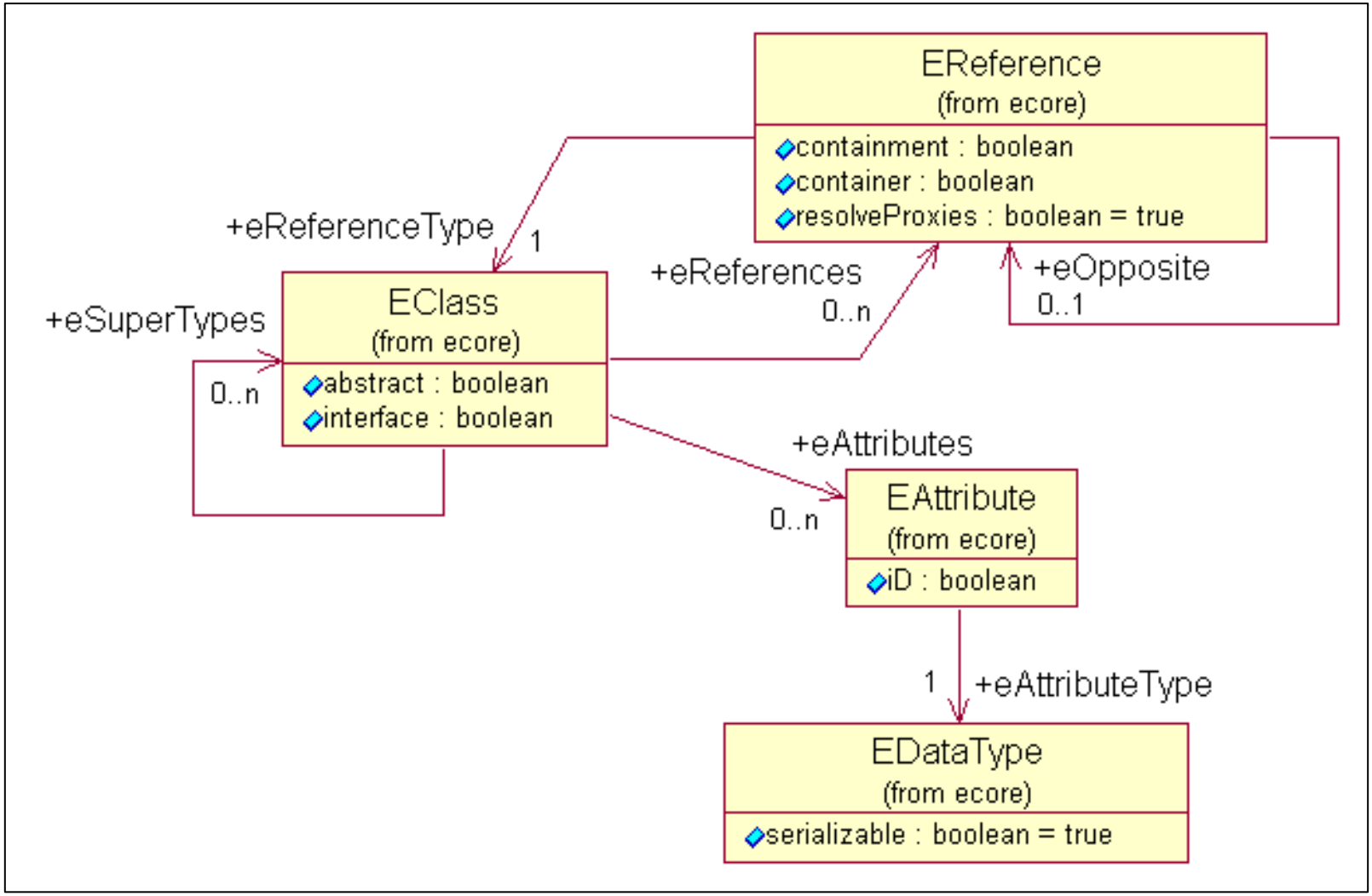


EMF and MetaObject Facility (MOF)

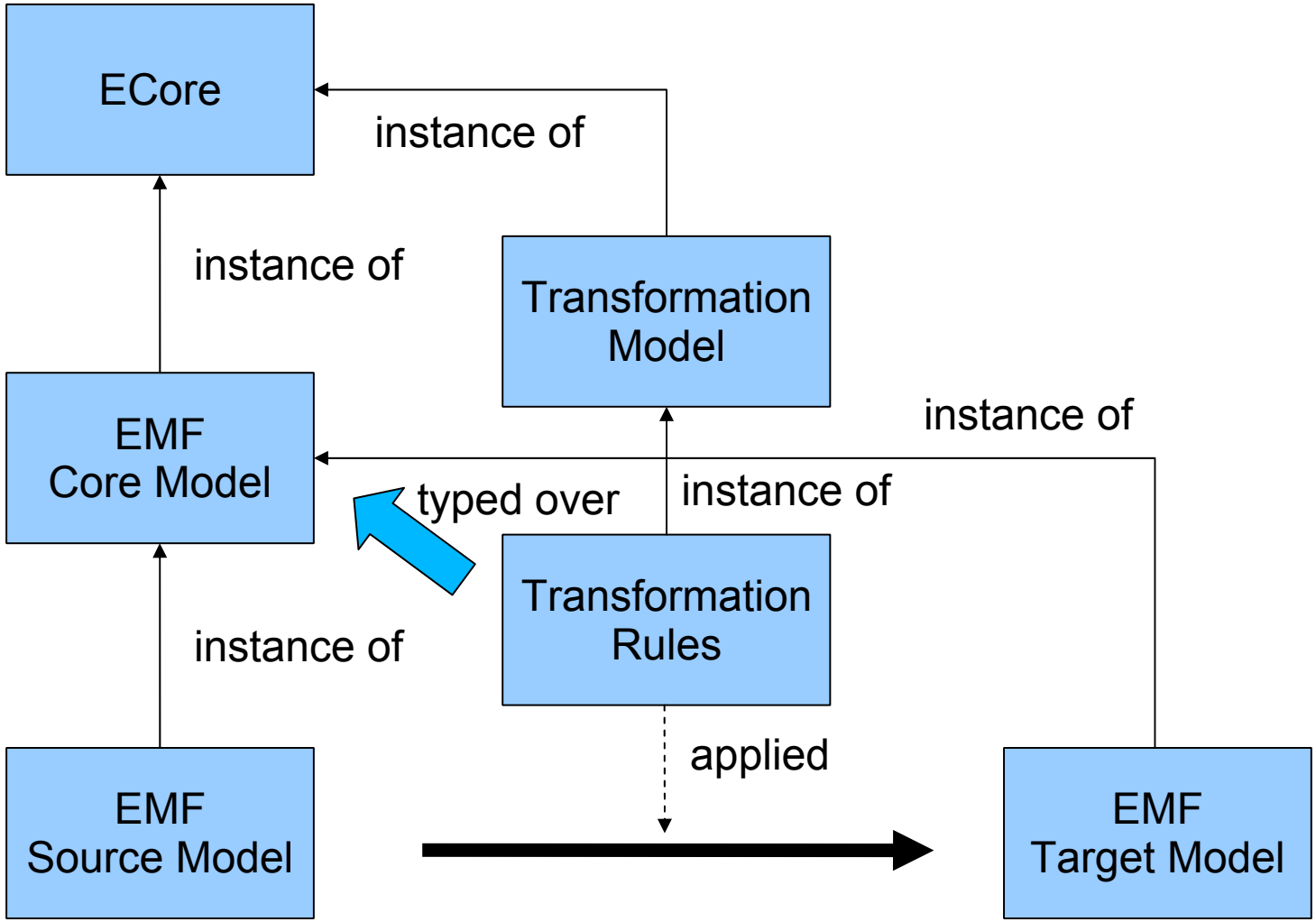
- EMF consists of a metamodel and technology mappings like MetaObject Facility (MOF).
- MOF is the basis for a couple of language definitions by the Object Management Group (OMG), including UML 2.



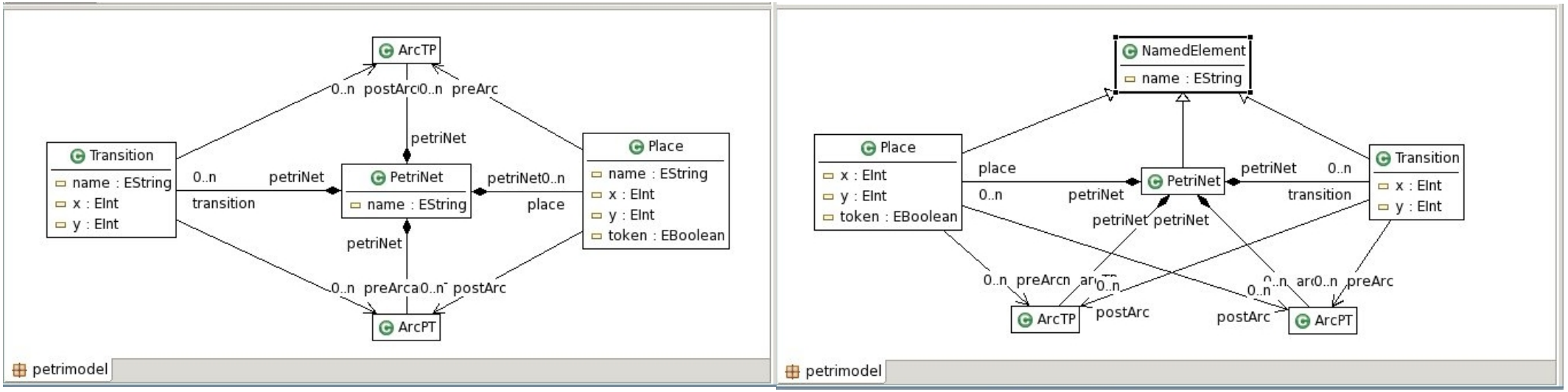
Eclipse Modeling Framework (EMF)



Transformation Model for EMF



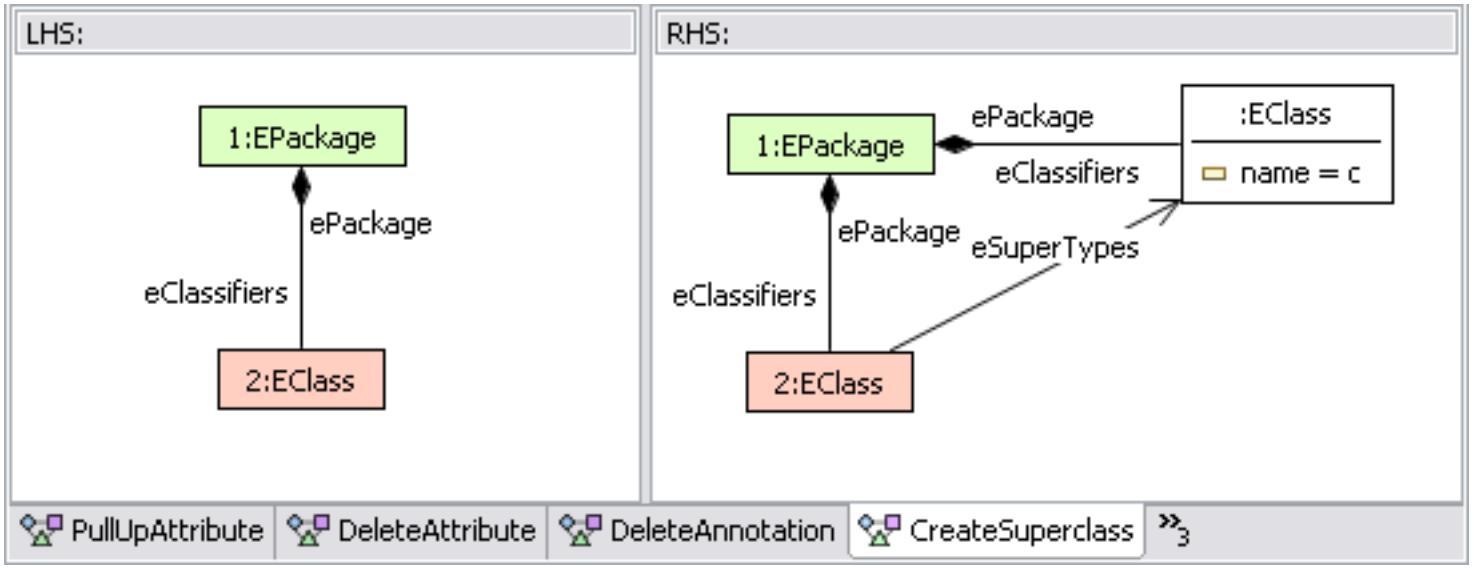
Sample EMF Model Refactoring



Two refactorings:

- new superclass „NamedElement“ for classes „Place“, „PetriNet“, and „Transition“
- pull up attribute „name“ to class „NamedElement“

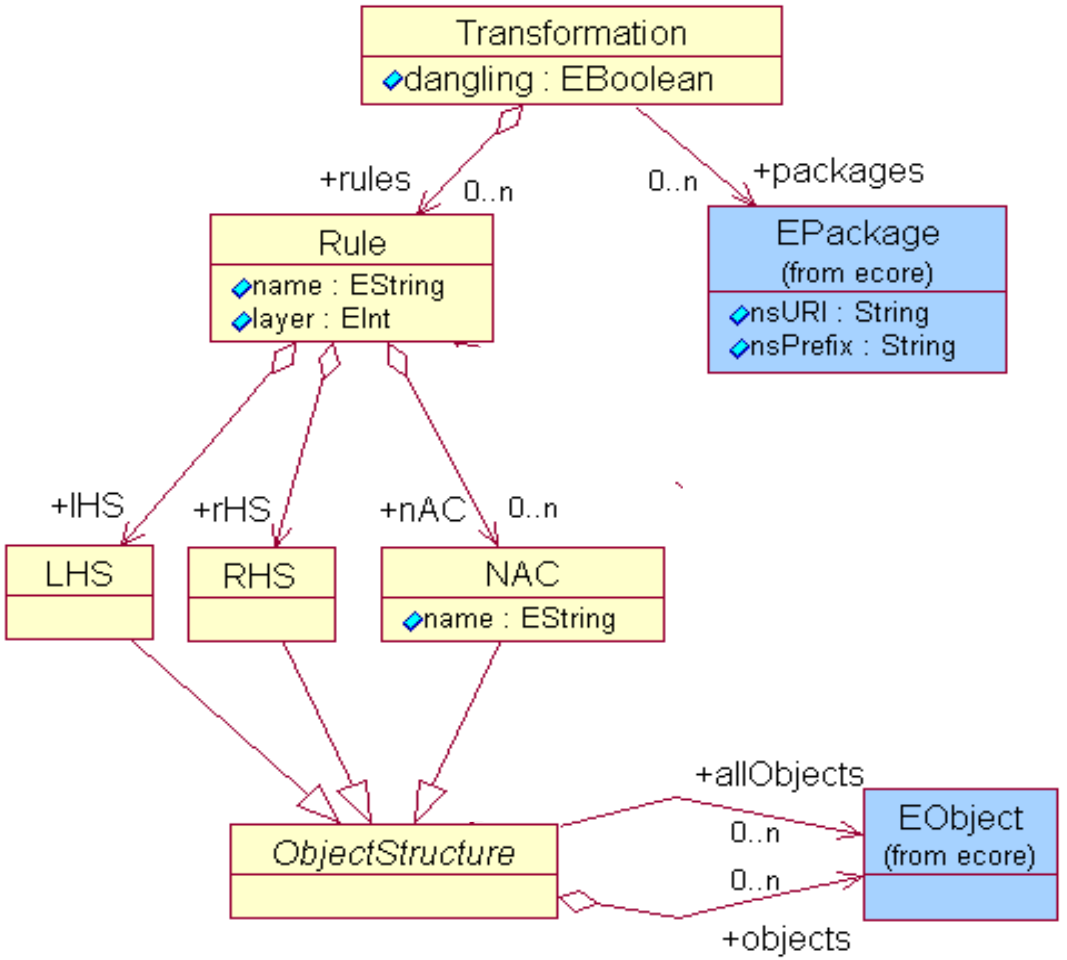
A simple refactoring rule for Ecore



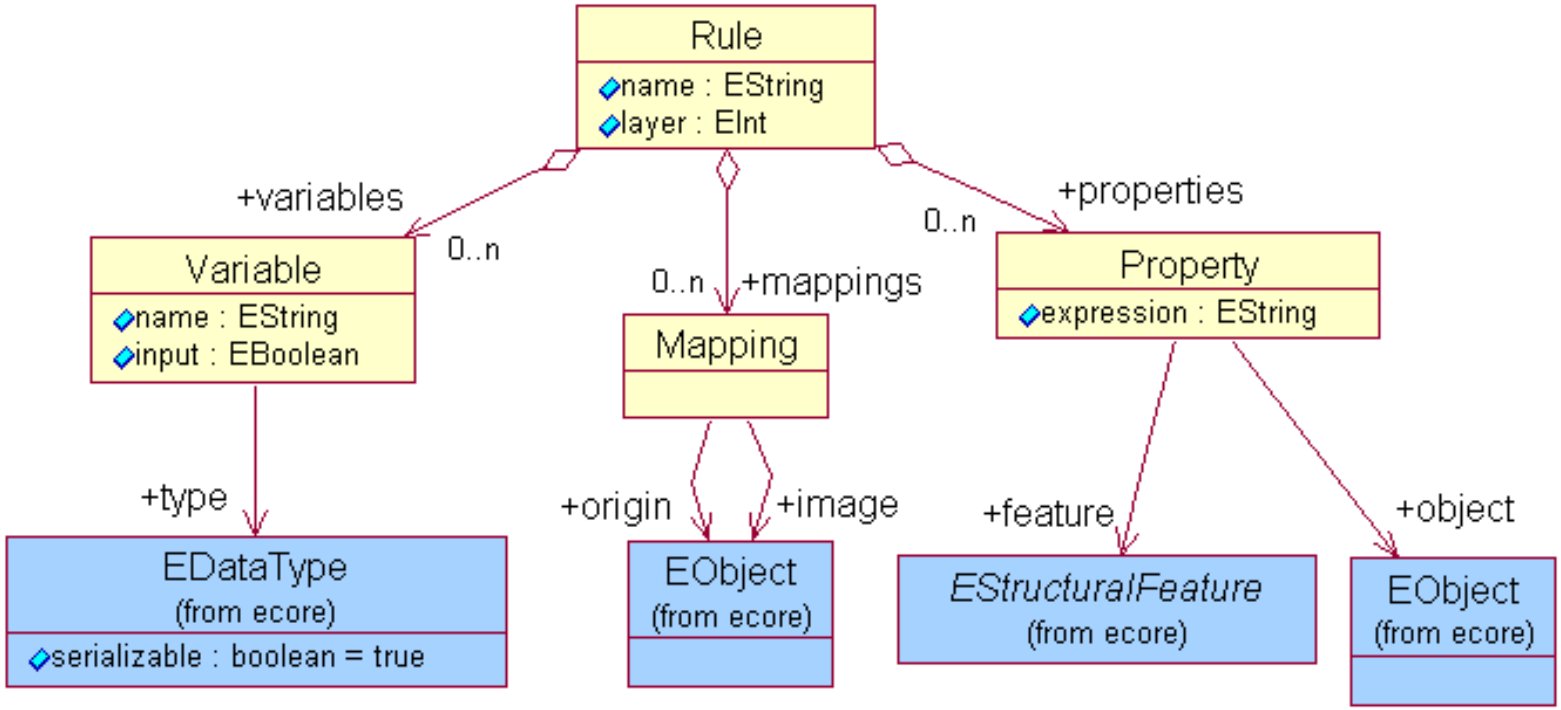
Create a new superclass for a given class

Transformation Model for EMF

- influenced by graph transformation due to validation facilities
- LHS checks existence of pattern
- NACs check non-existence of any pattern
- difference between LHS and RHS defines actions

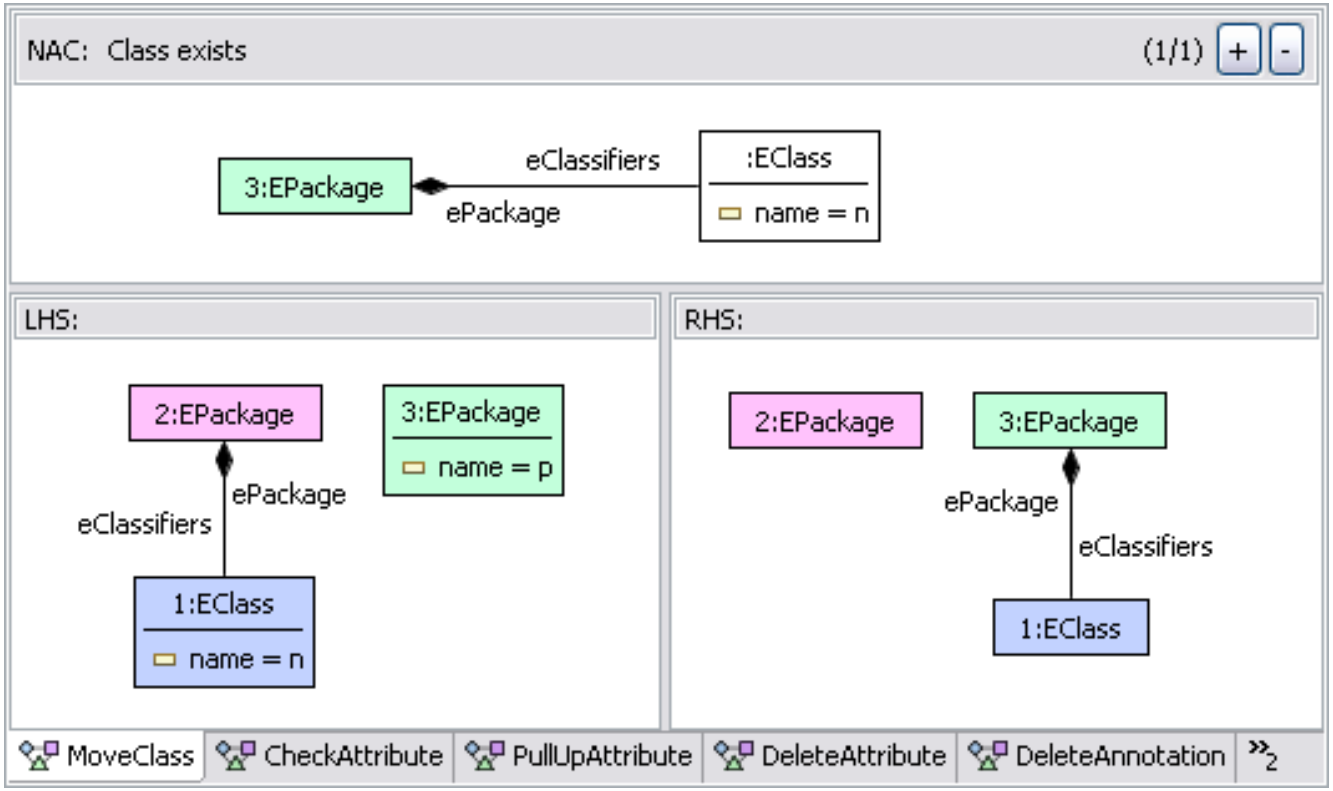


Transformation Model for EMF



- mappings are defined as LHS -> RHS or LHS -> NAC
- non-injective mappings, i.e. gluings, are allowed

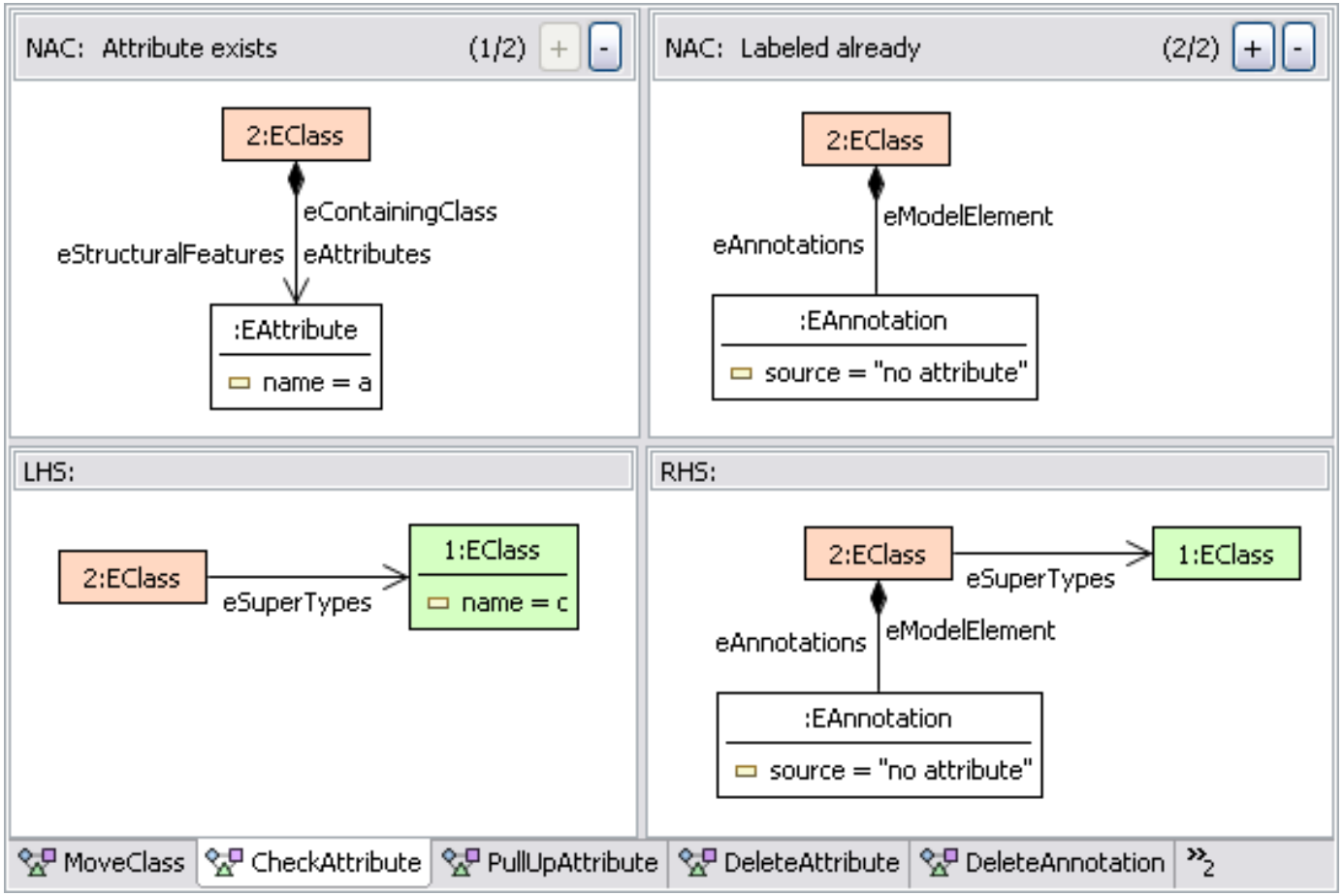
A simple refactoring rule for Ecore



Move a class from one package to another

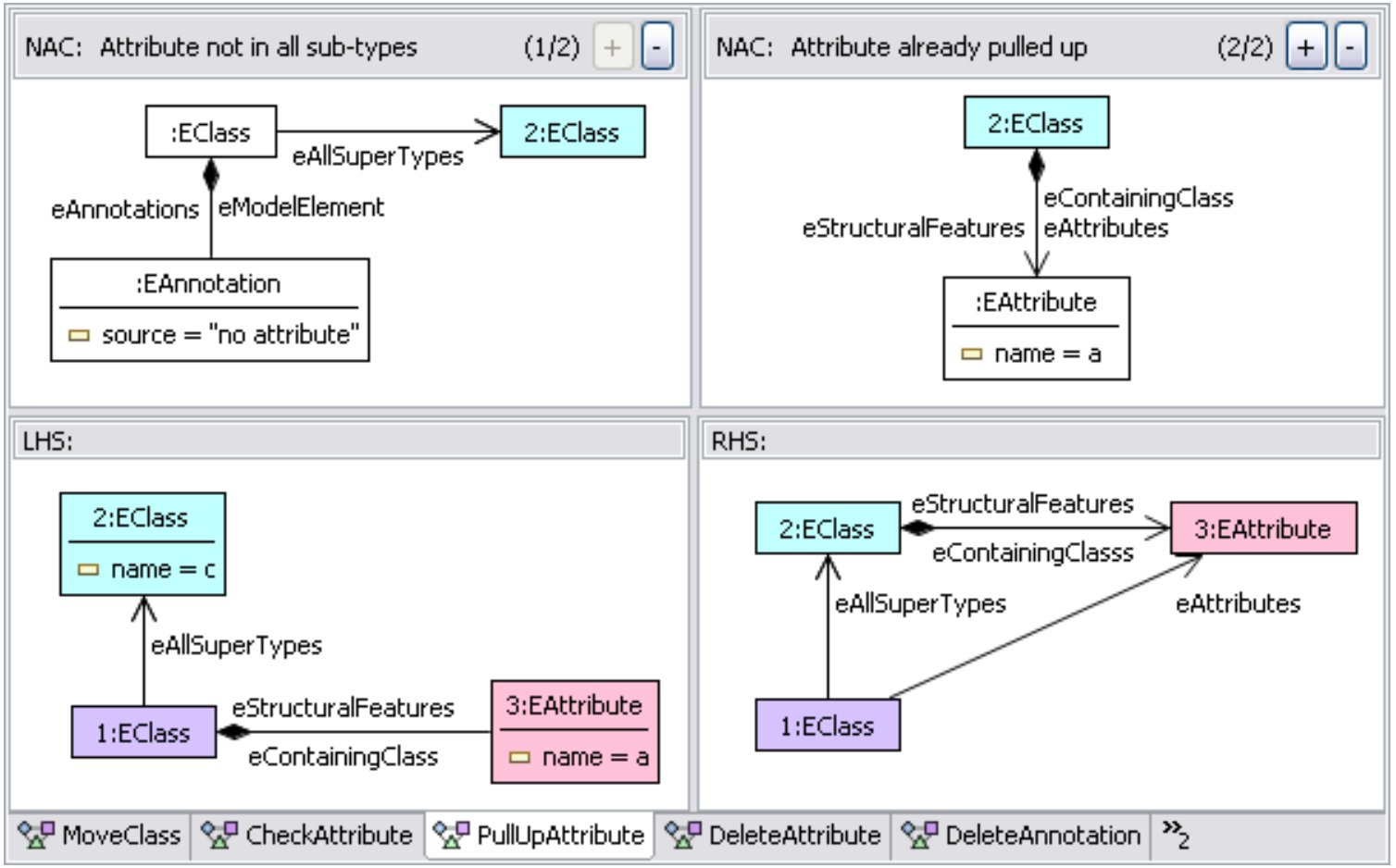
Similar: move attribute, create superclass, connect superclass, rename parameter, add /remove parameter, etc.

A complex Ecore refactoring – Pull up attribute



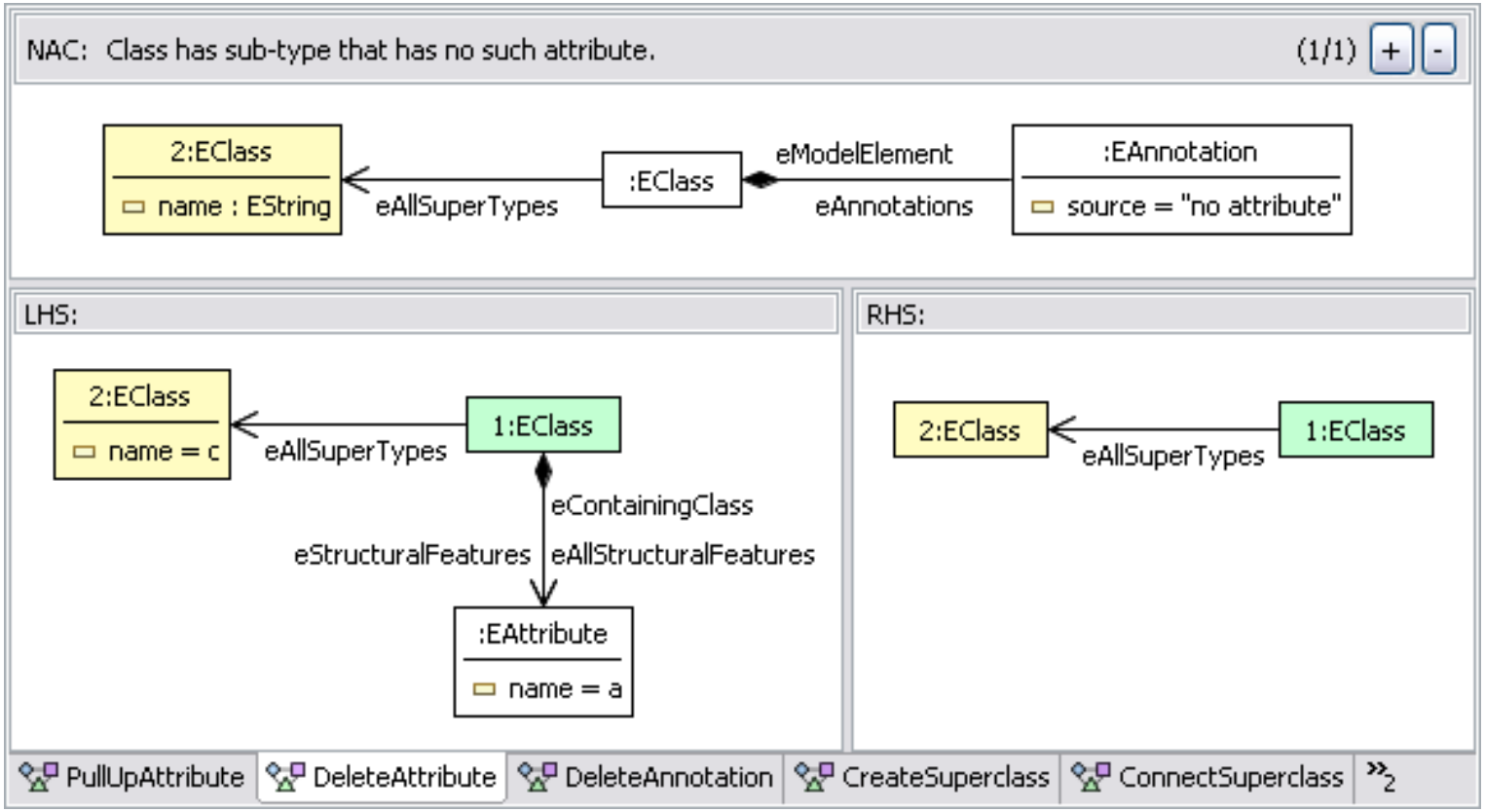
1. Check if all subclasses contain the attribute to be pulled up

A complex Ecore refactoring – Pull up attribute



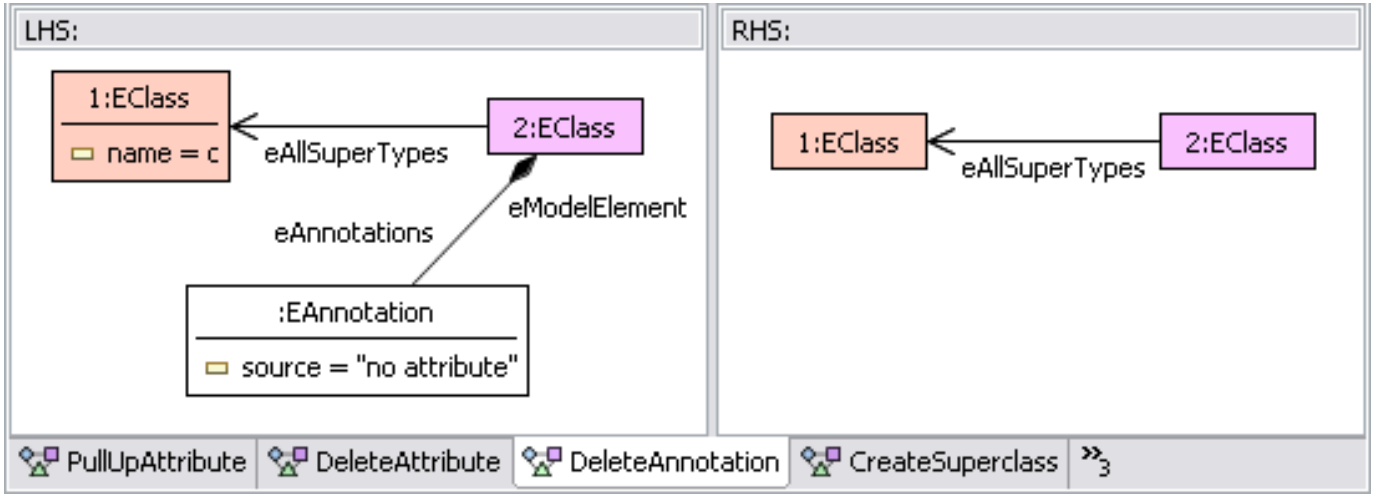
2. If all subclasses have the attribute, then pull it up

A complex Ecore refactoring – Pull up attribute



3. After pulling up, delete the attribute in all subclasses.

A complex Ecore refactoring – Pull up attribute



4. If there are still annotations, delete them.

How to perform EMF model transformations?

Interpreter:

1. Create an AGG type graph from an EPackage
2. Convert the EMF rules to AGG rules
3. Convert an EMF instance to AGG graph
4. Apply a rule in AGG / validate rules
5. Translate AGG graph back to EMF
6. Check and repair consistency

Compiler:

1. Create Java code from rules, one class per rule
2. Execute rules on EMF model instances or on EMF models (instances of Ecore meta-model)

Code Snippet for Rule Application

```
ResourceSet resourceSet = new ResourceSetImpl();
Resource resource =resourceSet.getResource(URI.createFileURI(
                                    "src/Ecore/instances/petrimodel.ecore"), true);
EPackage model = (EPackage) resource.getContents().get(0);
// create the new superclass
CreateSuperclassRule createSC = new CreateSuperclassRule(model);

createSC.setParA(true);

createSC.setParC("PetriNet");

createSC.setParS("NamedElement");

createSC.execute();
```

Consistency of EMF Refactorings

- coherent refactoring of model and code
  - no problem if purely model-driven
- refactoring result is syntactically correct
- refactoring result fullfills validation properties
  - for validation based on graph transformation:

    consistence of EMF transformation with graph transformation

    conflict and dependency analysis of rule applications, termination checks, etc.

Related Work

EMF-related:

- Merlin
- Eclipse GMT-Project:
  - Tefkat
  - ATL
  - UMLX
  - ViaTra2
- MOMENT

Graph transformation-related:

- AToM3
- GReAT
- VMTS
- ViaTra2
- Gmorph
- MOFLON
- MOTMOT

Conclusion and Future Work

- EMF model transformations as in-place model updates
- Interpreter/Compiler for EMF transformations
- Visual definition of transformation rules
- Consistency of EMF transformation with graph transformation offers validation facilities

Future work:

- Complete and integrate transformation engine and editor
- Implement checks for possible containment constraint violations
- Consider different applications:
  - Complex Editing Operations (Eclipse GMF), Simulation, Optimizations