Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations

Authors: T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer.

Presenter: Mike Maksimov

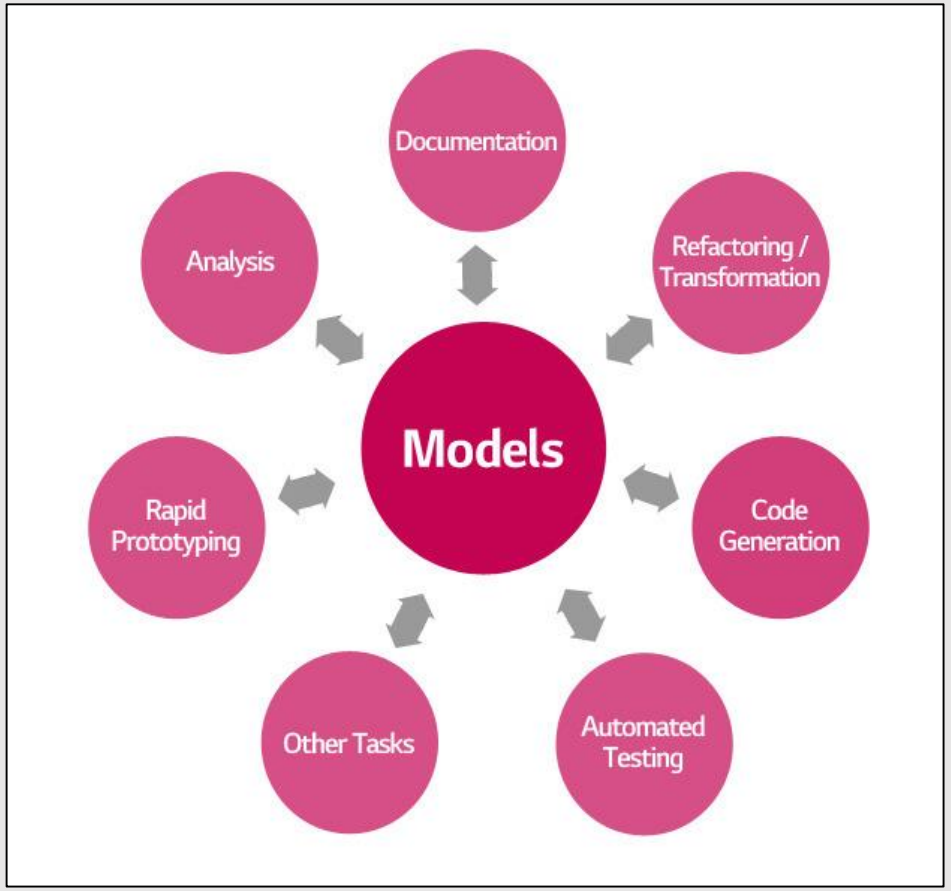
Introduction to MDD

Model Driven Development (MDD)

- Promising paradigm in software engineering
- Ideal means of abstraction
- Manages complexity of software systems

Model Transformation

- Essential activity in MDD
- Transformation types
  - *In-place, Out-Place.*
  - *Endogenous, Exogenous.*



Introduction to EMF

Eclipse Modelling Framework (EMF)

- Modeling and code generation capabilities
- Describes structural aspects of models only

Existing in-place transformation languages

- They are too simple
- Not declarative enough
- Do not support formal reasoning
- Example languages
  - *Kermeta, EMF Tiger, EWL, Moment2.*

Henshin Overview

Henshin - Transformation Language:

- Powerful language offering formal reasoning
- Operates directly on EMF models
- In-place endogenous and exogenous support
- Based on graph transformations

Henshin - Characteristics:

- Rules
- Application Conditions
- Transformation Units

Henshin Transformation Meta-model

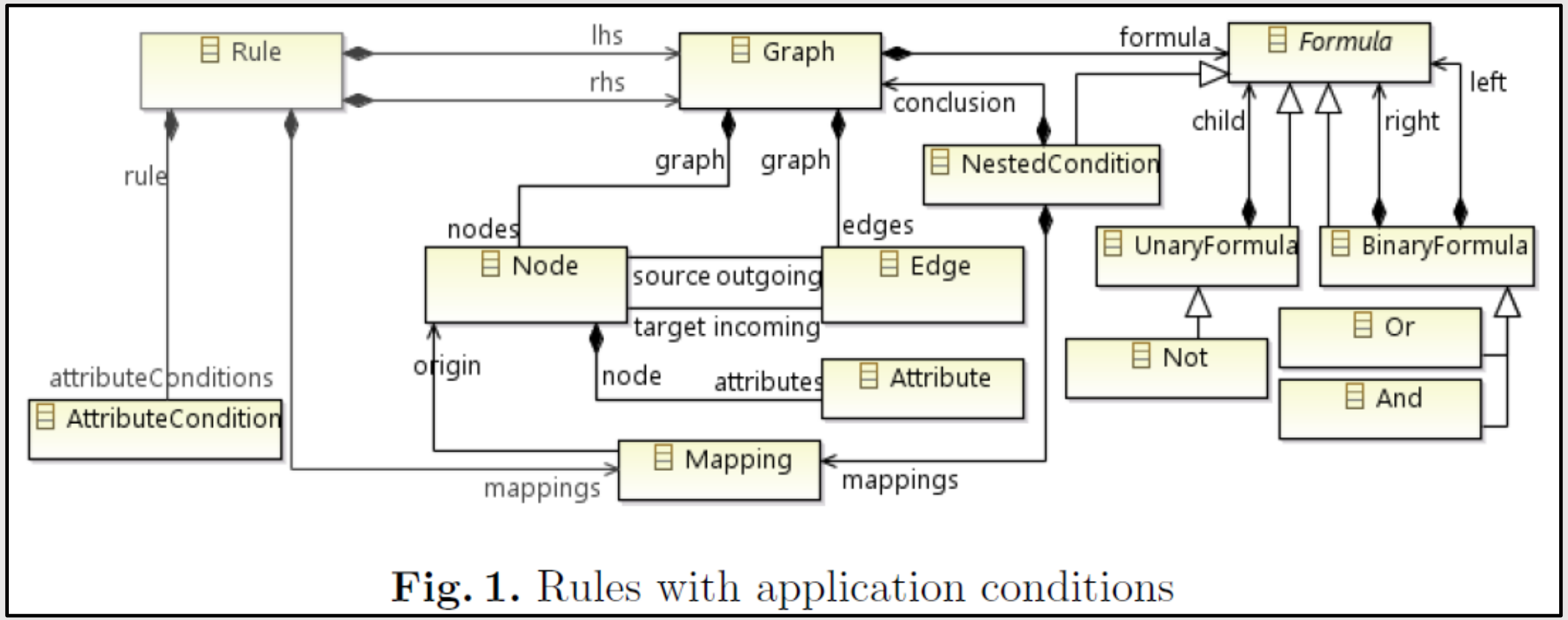
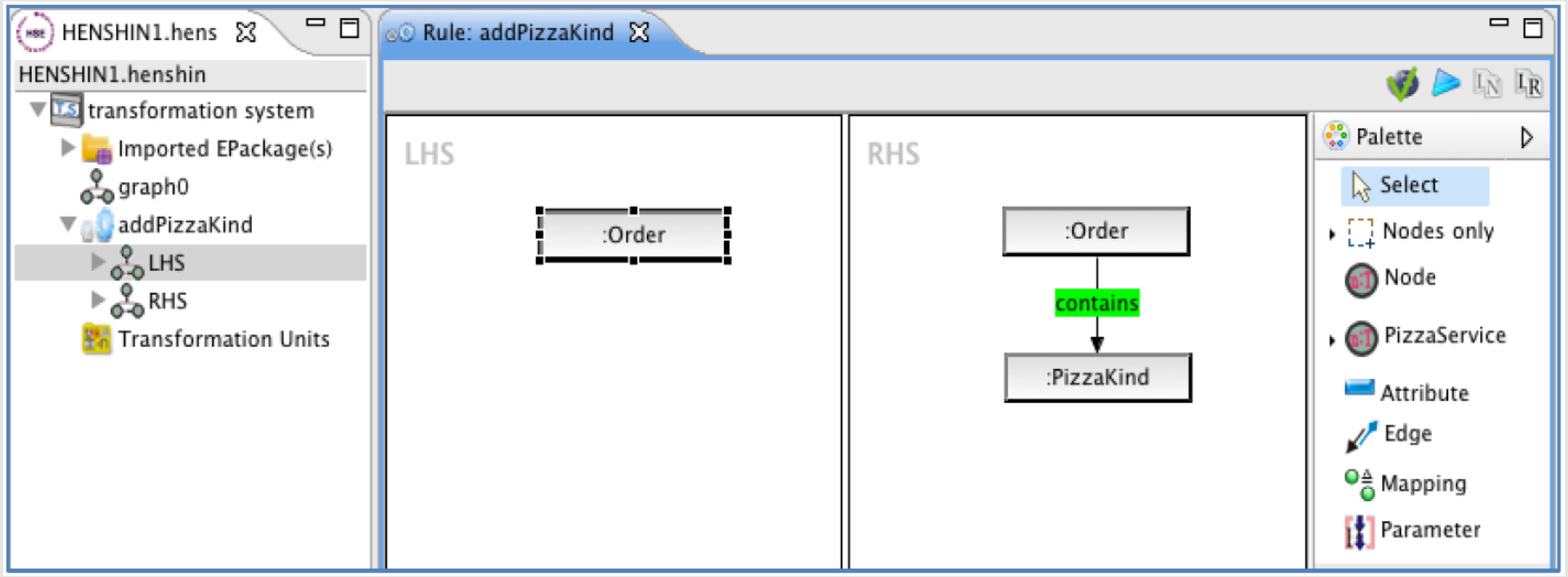


Fig. 1. Rules with application conditions

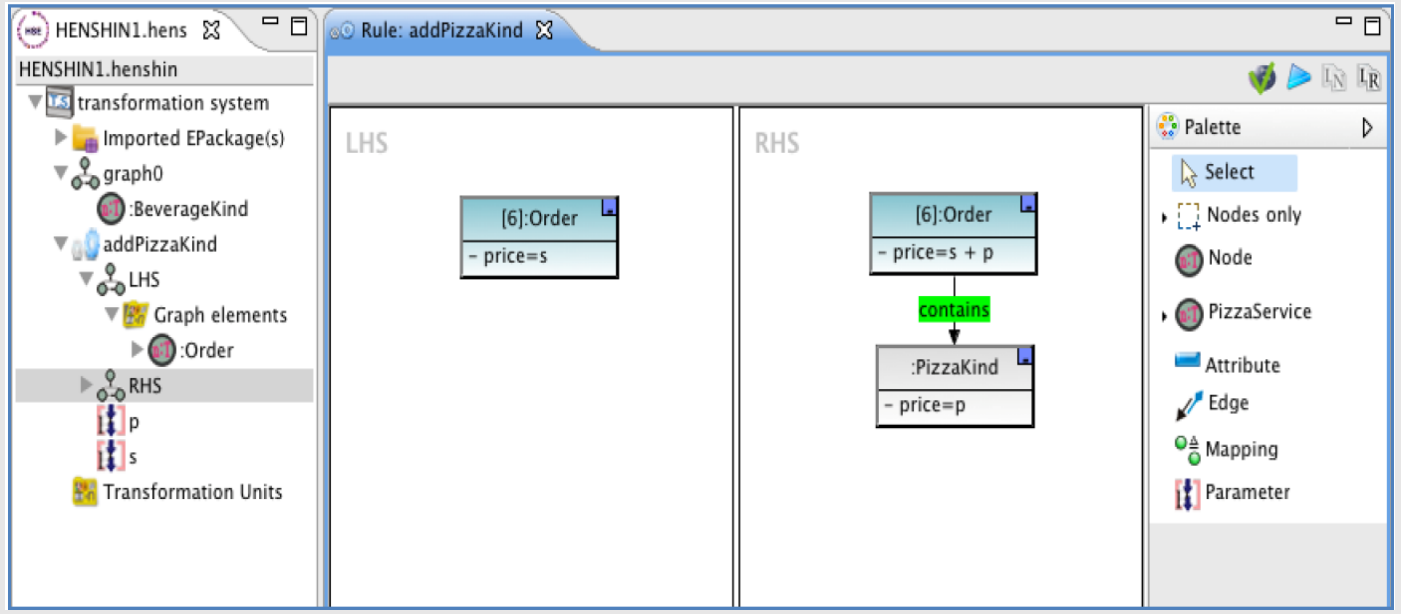
Transformation Rules

- Rules consist of LHS and RHS graphs
- The LHS matches to the source graph
- RHS replaces the matched source graph

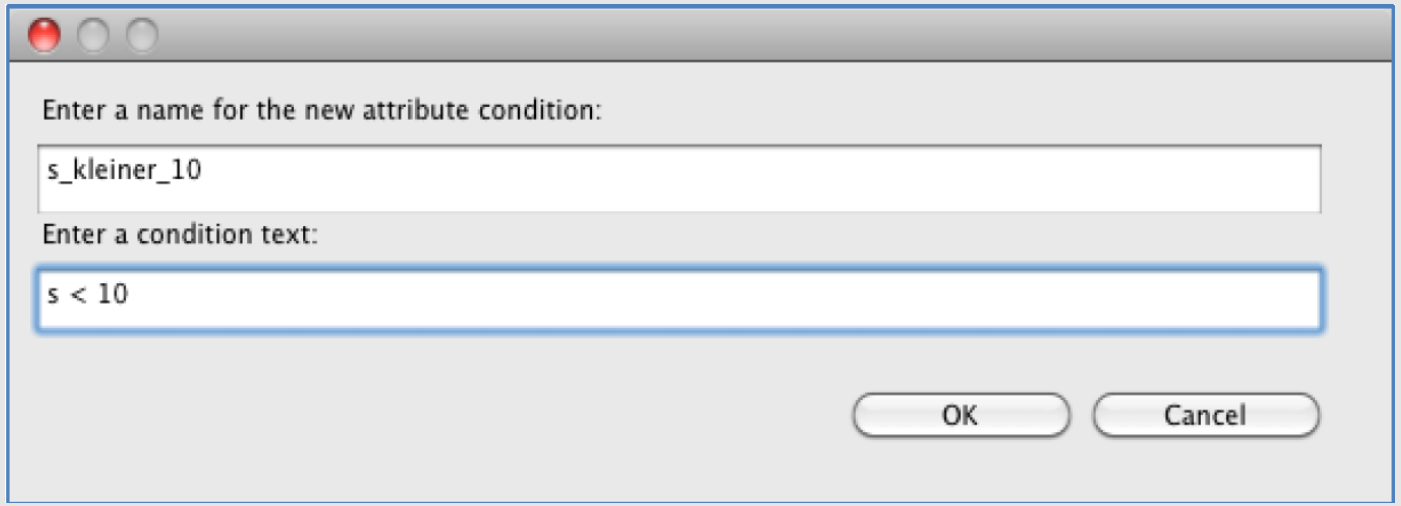


Attributes and Conditions

- Henshin supports the addition of attributes

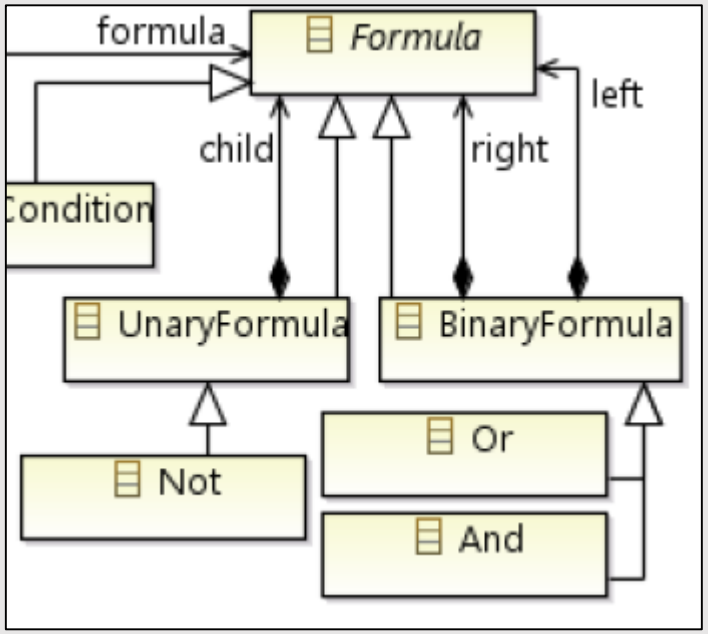


- Attribute conditions can be added on rules



Application Conditions

- Determine where a rule should be applied
- Set graph conditions using first order logical formulas
- Positive Application Conditions (PACs)
  - *Require the presence of elements or relationships*
- Negative Application Conditions (NACs)
  - *Forbid the presence of elements or relationships*
- Can be nested



Transformation Units

- Control flow is specified using units
- Support object and parameter passing
- Types
  - *Sequential Units*
  - *Priority Units*
  - *Independent Units*
  - *Loop Units*
  - *Conditional Units*
  - *Amalgamation Units*

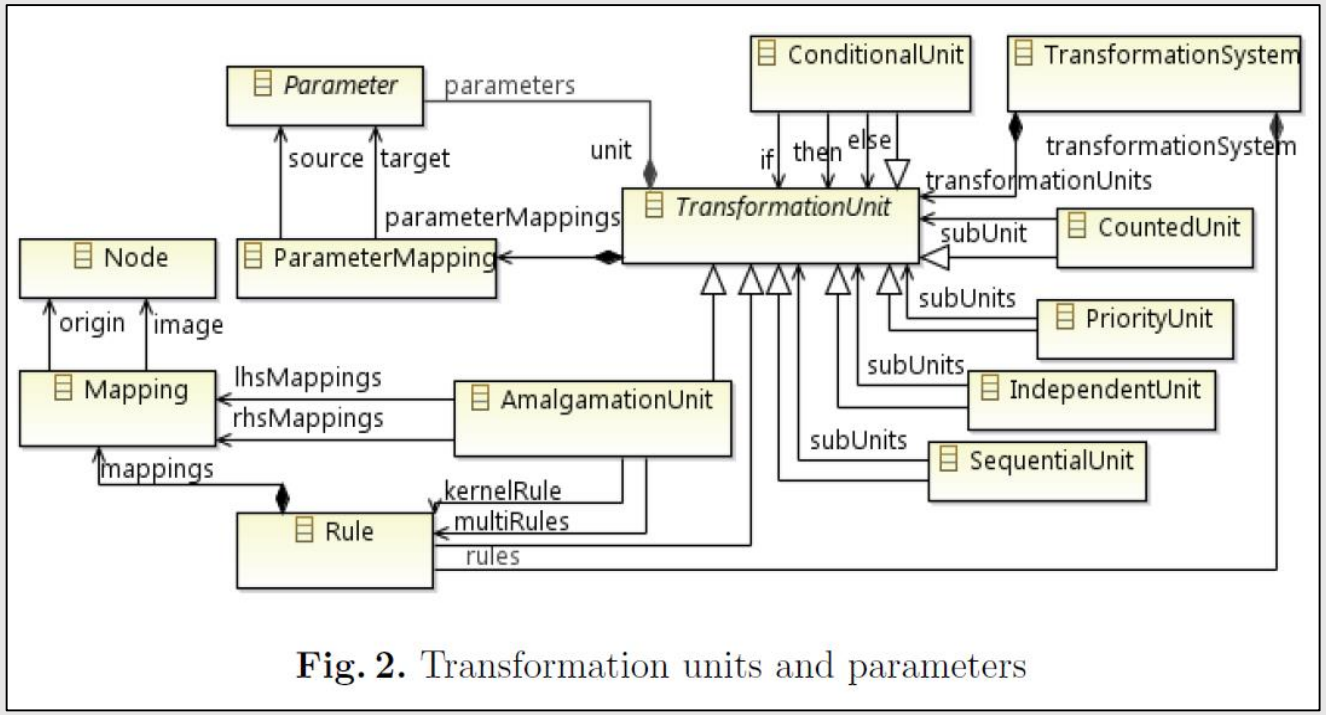
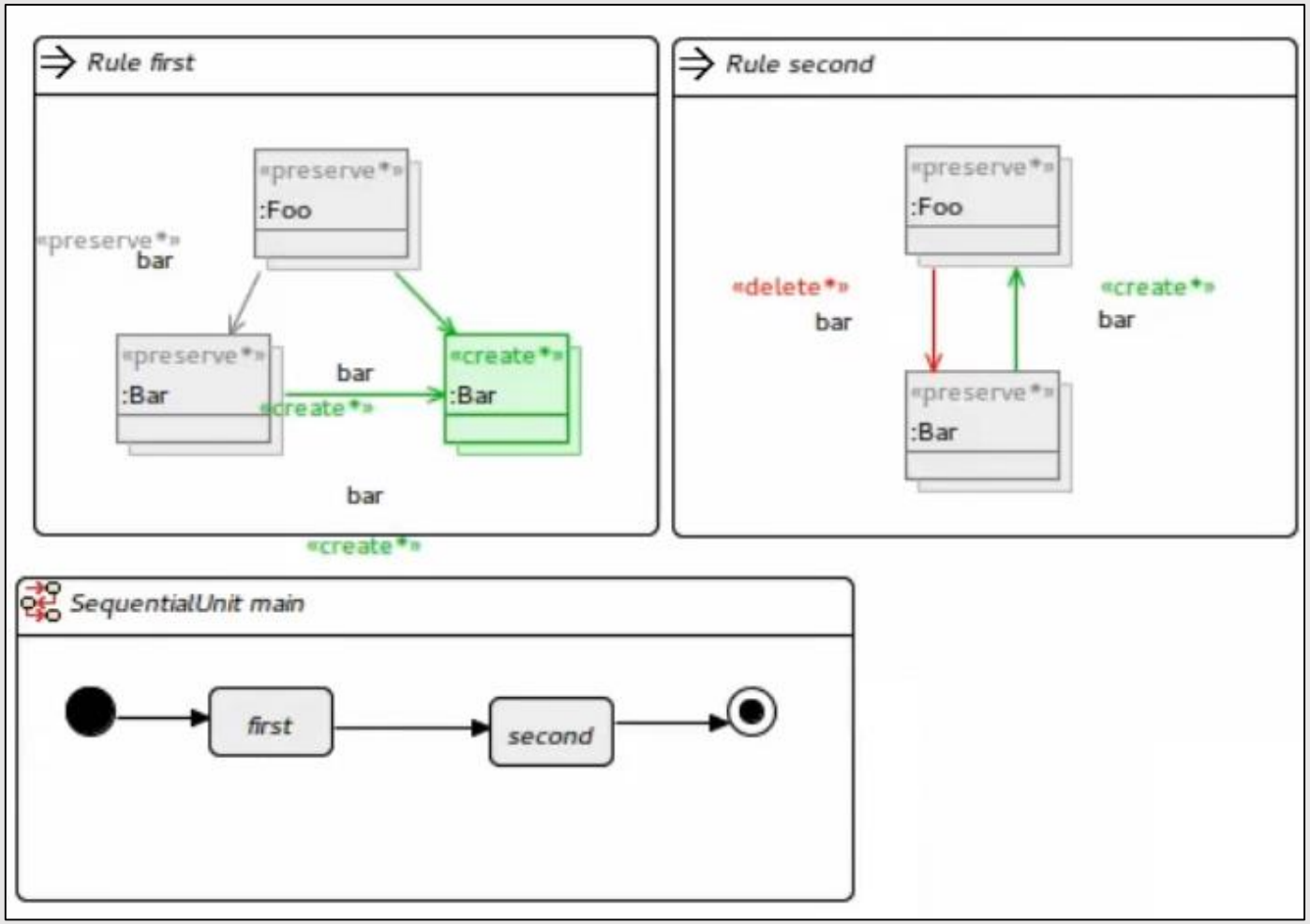


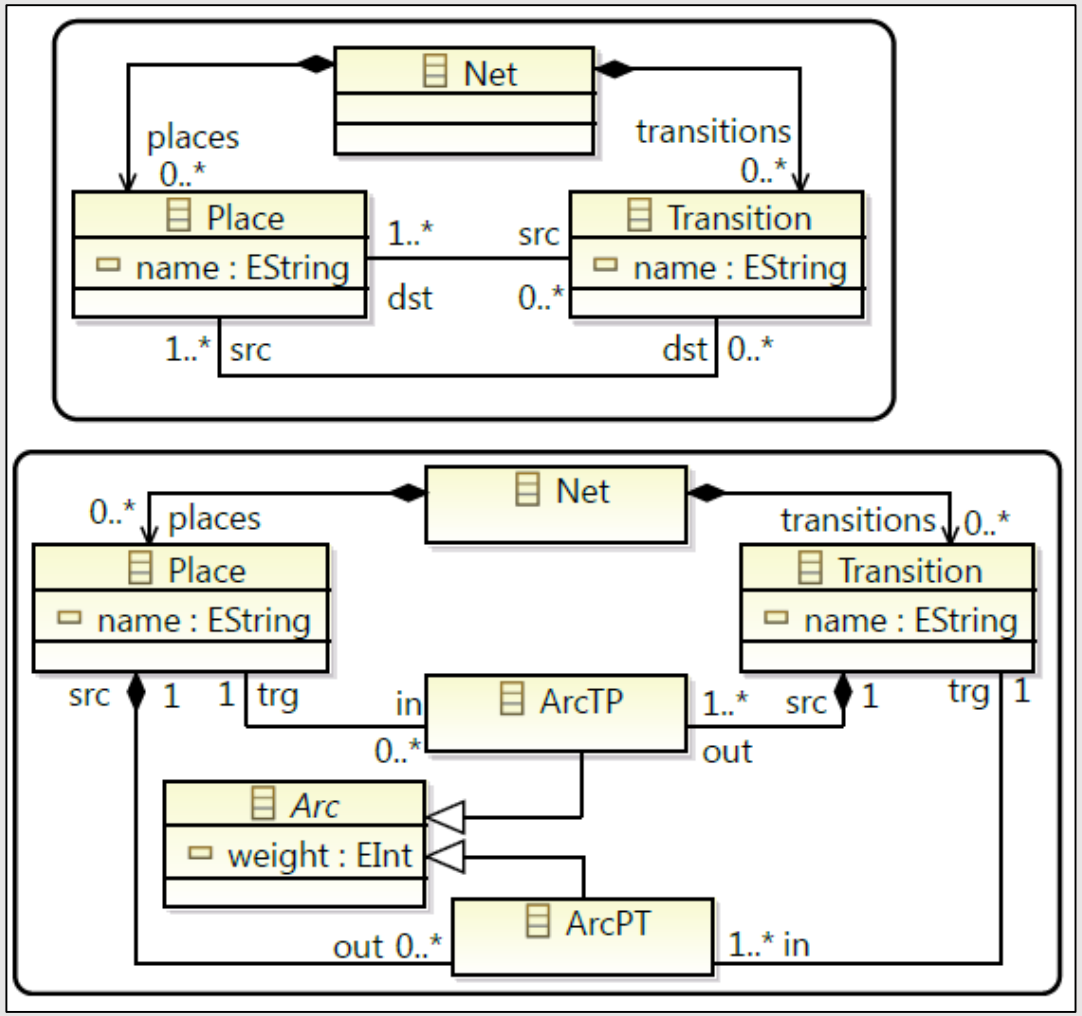
Fig. 2. Transformation units and parameters

Transformation Rules & Units: Example



Henshin In Meta-Model Evolution

- In MDD models are the key artifacts and they evolve over time
- Henshin supports transformation rules for both meta-models and instances
- Example of an evolving Petri Net meta-model on the right



Henshin In Meta-Model Evolution

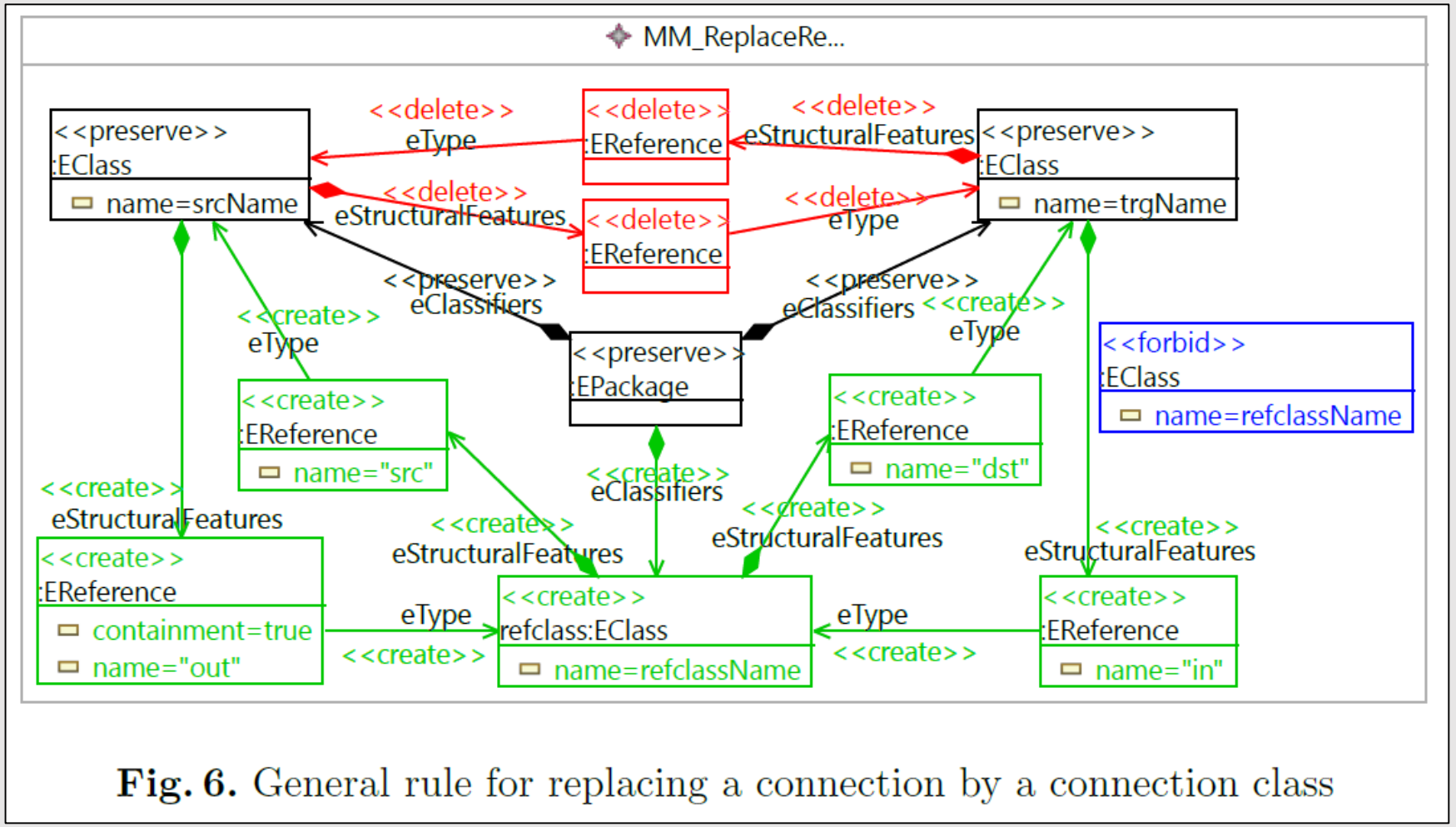
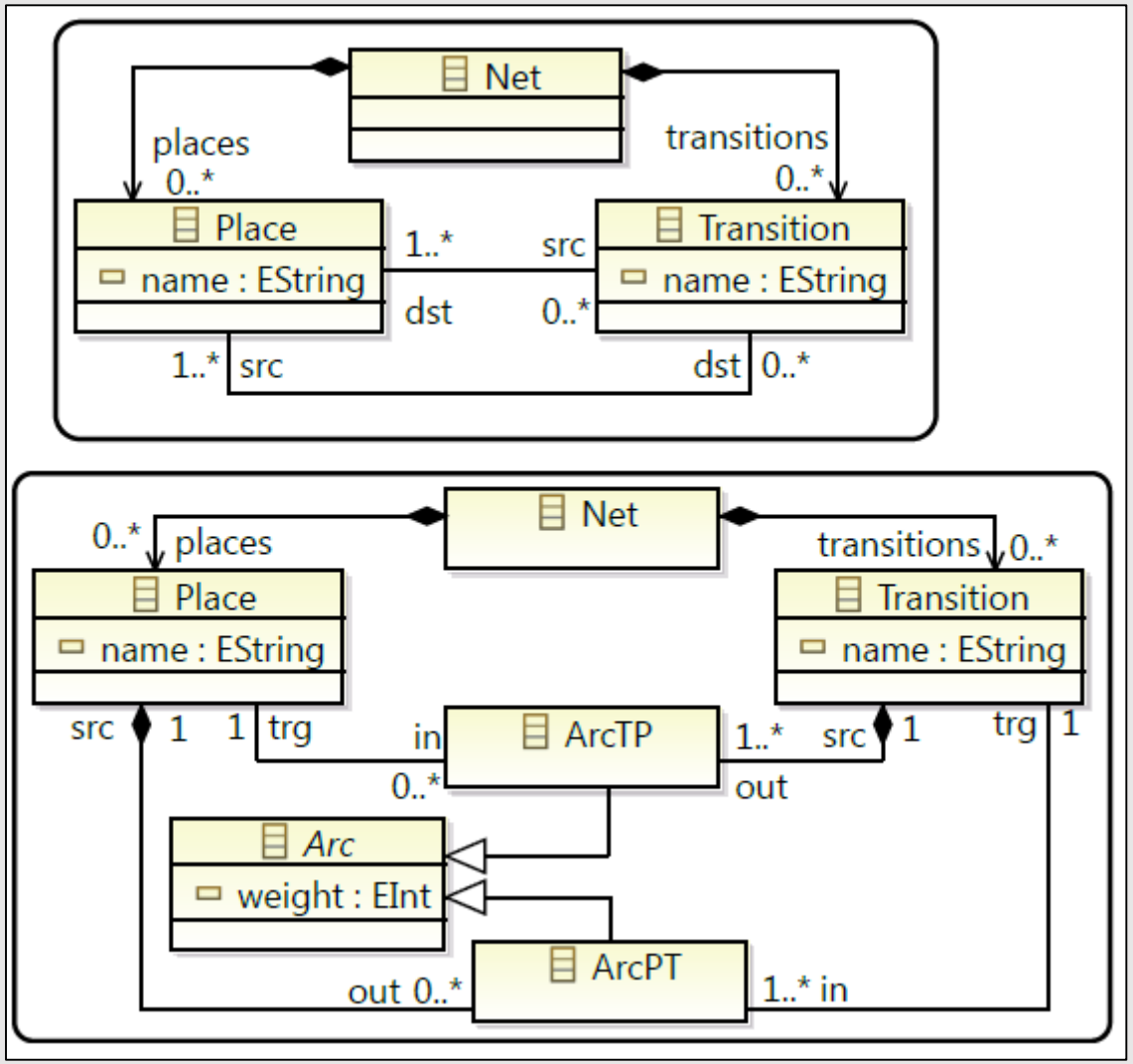


Fig. 6. General rule for replacing a connection by a connection class

Henshin In Meta-Model Evolution



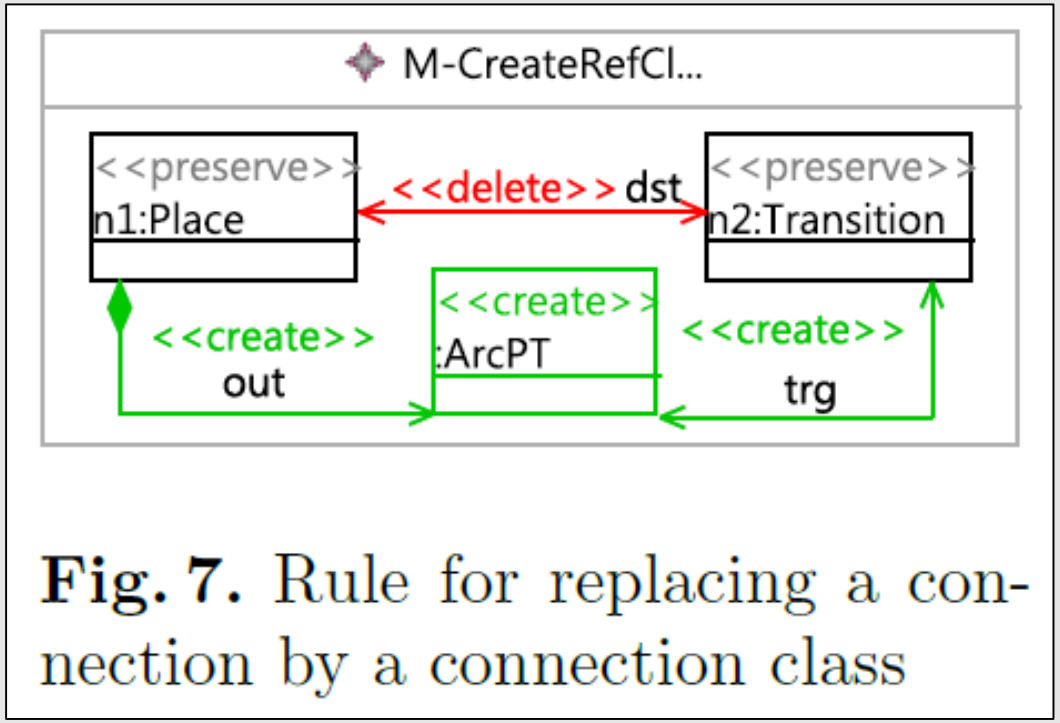


Fig. 7. Rule for replacing a connection by a connection class

Henshin In Meta-Model Refactoring

Model Refactoring Demonstration:

- ***Pull Up Attribute*** example
  - *Moves a common attribute from all subclasses to the superclass*
- ***Pull Up Attribute*** preconditions

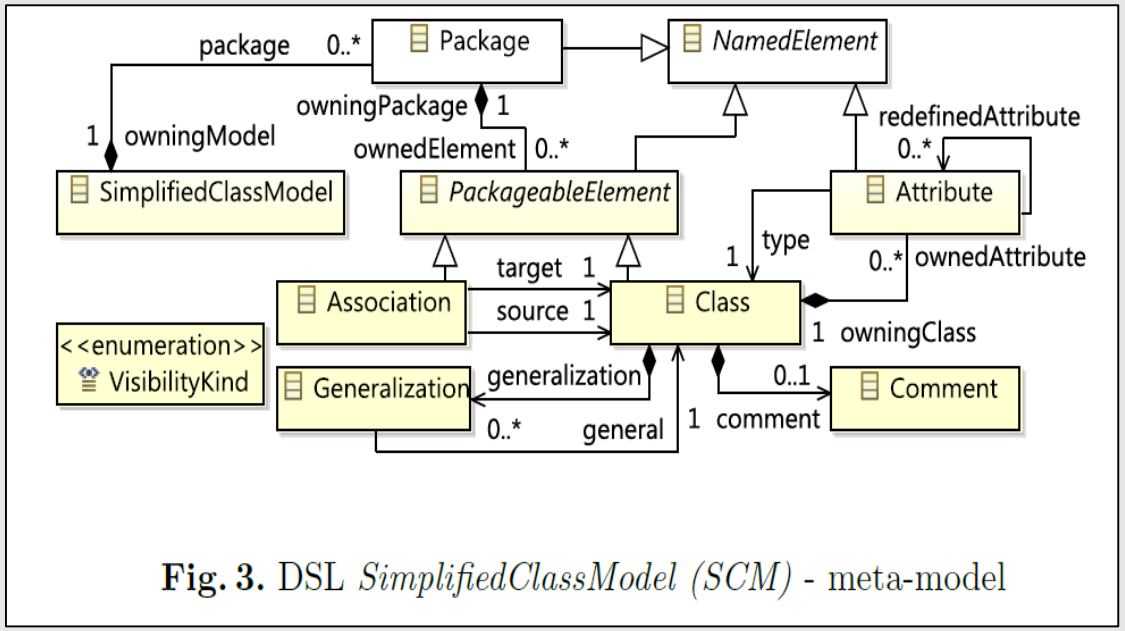


Fig. 3. DSL *SimplifiedClassModel (SCM)* - meta-model

Henshin In Meta-Model Refactoring

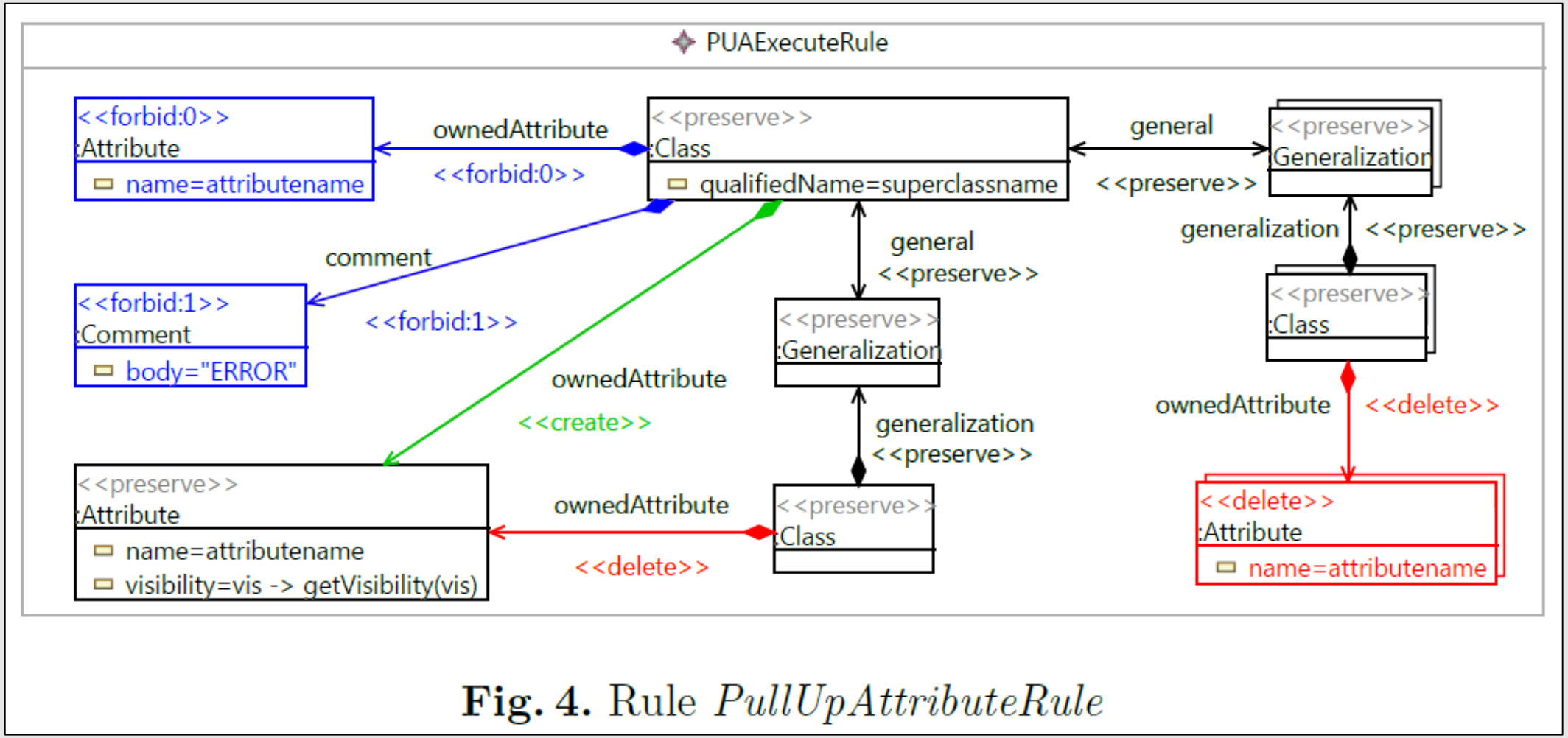


Fig. 4. Rule *PullUpAttributeRule*

Henshin Tool Environment

Henshin Editors:

- Tree Based Editor (Generated by EMF)
- GMF Graphical Editor

Tool Characteristics:

- Integrated view between LHS & RHS
- Built in state space generator
- Support for model checking
- Integrated OCL validator

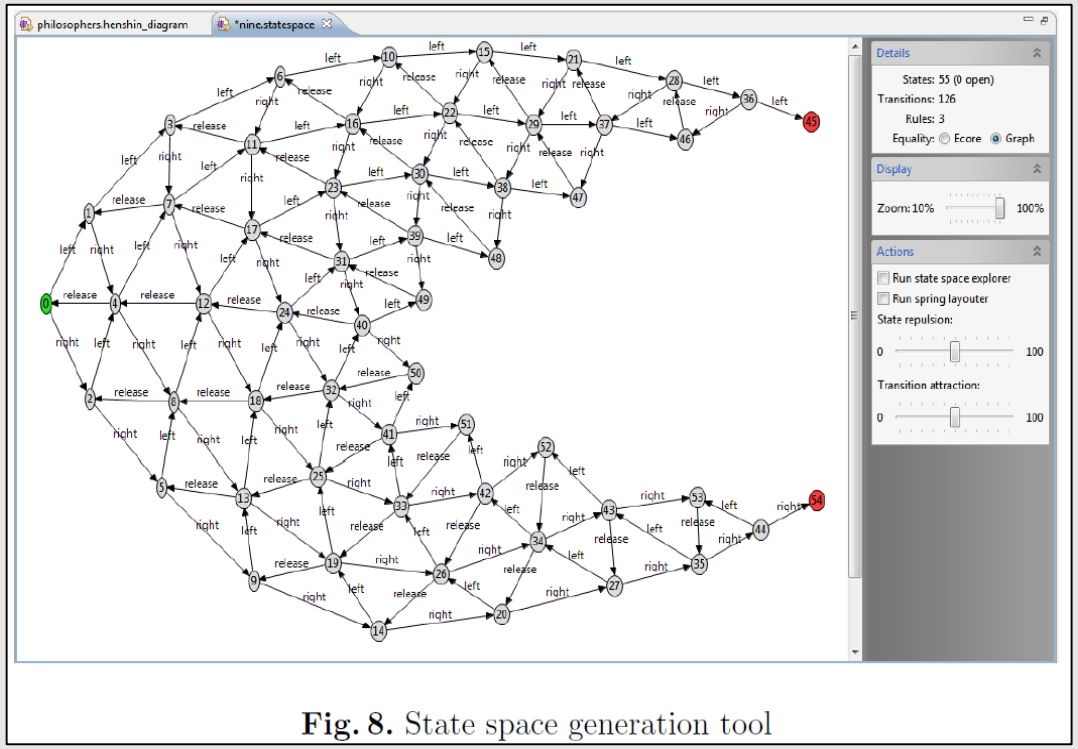


Fig. 8. State space generation tool

Discussion – Pros & Cons

1. Do you think that Henshin is limited in the fact that it only supports EMF models?
2. Do you prefer the split LFH and RHS interface or the joint variation?
3. Was the paper well written in your opinion?
4. Enough support for exogenous transformations?