



Towards Refactoring of Rule-Based, In-Place Model Transformation Systems

Gabriele Taentzer,
Thorsten Arendt



Claudia Ermel



Reiko Heckel





Towards Refactoring of Rule-Based, In-Place Model Transformation Systems

Gabriele Taentzer,
Thorsten Arendt



Claudia Ermel

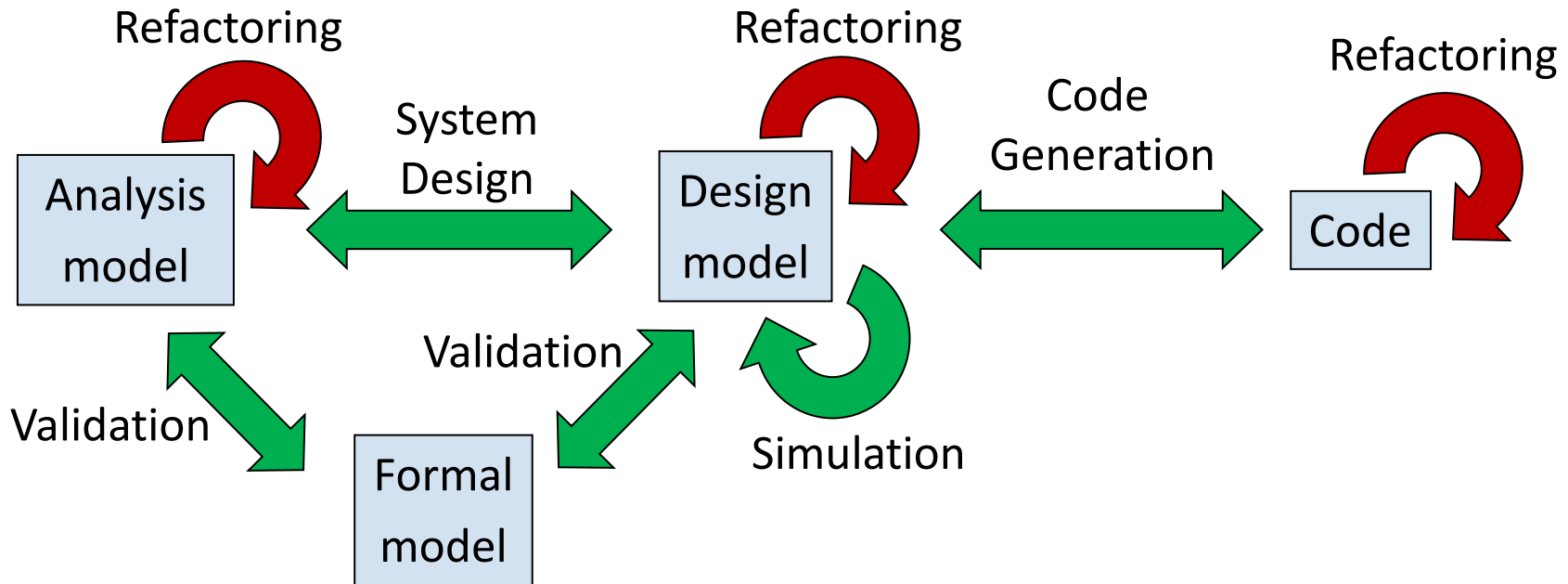


Reiko Heckel



- **Motivation**
 - Improving the Quality of Model Transformations
- **Quality Aspects of Model Transformation Systems**
 - Selected Smells
- **Selected Refactorings**
 - Merge Rules Differing in Types Only
 - Extract Precondition
 - Unify Rules with Same Actions
- **Some Remarks on Semantics Preservation**
- **Implementation: Using the EMF Model Transformation Tool Henshin**
- **Conclusion and Future Work**

Model Transformations and Refactorings: Central Technologies in MDD

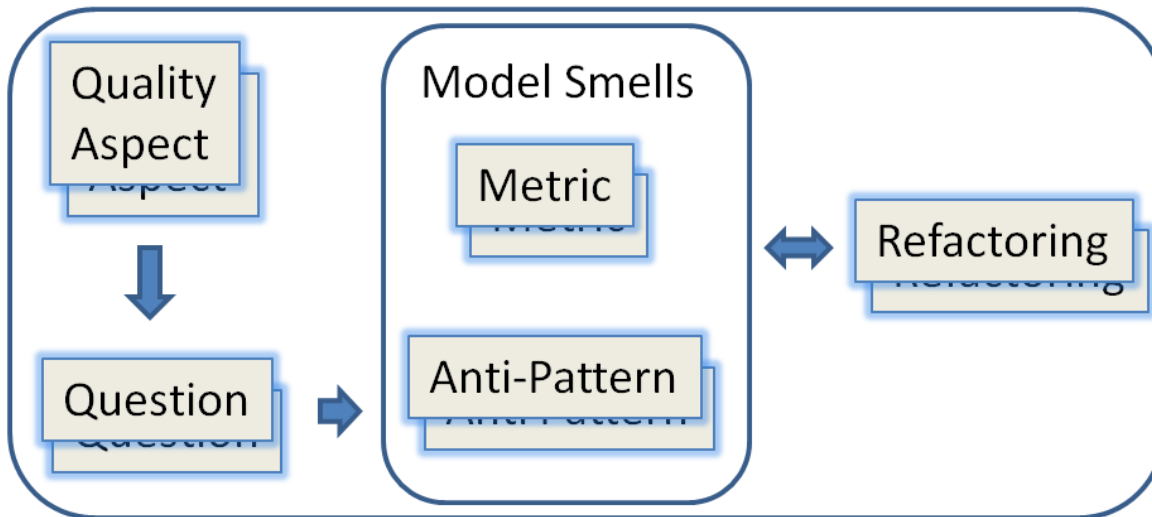


- ❑ **Challenge:** Evolution, maintainability and quality assurance of model transformations
- ❑ **Solution Idea:**
 - Consider rule-based, in-place model transformation systems
 - Use Rule-Based Transformation for Refactoring of Model Trafos

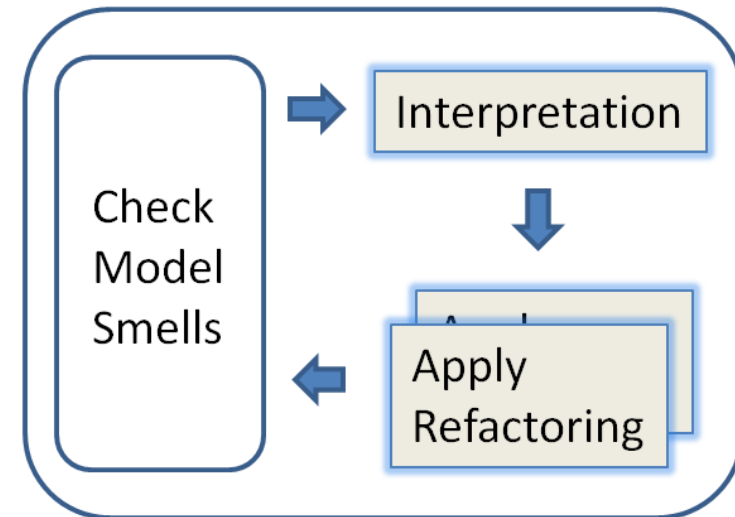
A Quality Assurance Process for Models



Process Specification



Process Application



Aim: Apply the quality assurance process to rule-based model transformation specifications

Correctness

- Concerning syntax and semantics

Conciseness

- Presentation at the “right” level of abstraction

Changeability

- Rapid and continuous evolution possible

Comprehensibility

- Understandable by intended users

Large Rule

- **Detection:** number of elements (objects, relations, pre-conditions, actions, ...)
- **Affected quality aspects:** conciseness, comprehensibility

Redundant Attributes and References

- **Detection:** No. of equal attributes and references
- **Affected quality aspect:** conciseness, comprehensibility, changeability

Redundant Rules

- **Detection:** No. of rule pairs differing in types only
- **Affected quality aspect:** conciseness, comprehensibility, changeability

Unused Object Type

- **Detection:** number of rules using a specific object type
- **Affected quality aspect:** correctness, conciseness, completeness

Delete and Created the Same Object

- **Detection:** find corresponding patterns in rules
- **Affected quality aspect:** conciseness, comprehensibility

Rules with Common Subrules

- **Detection:** find corresponding subpatterns in rules
- **Affected quality aspect:** conciseness, changeability, comprehensibility

Refactoring changes a model transformation system such that its structure is improved while its semantics is preserved.

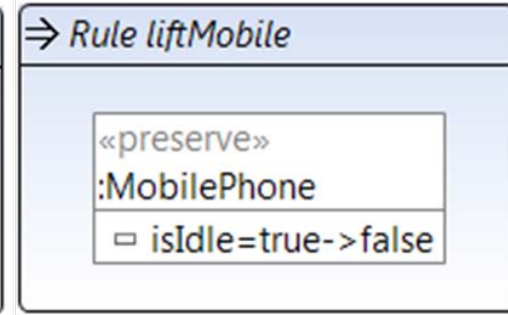
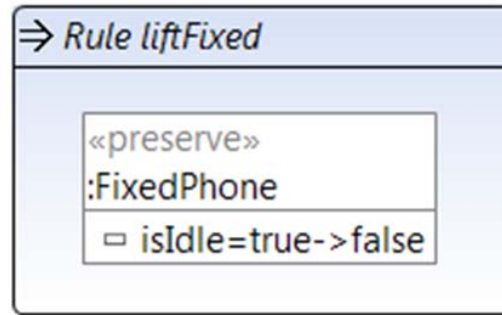
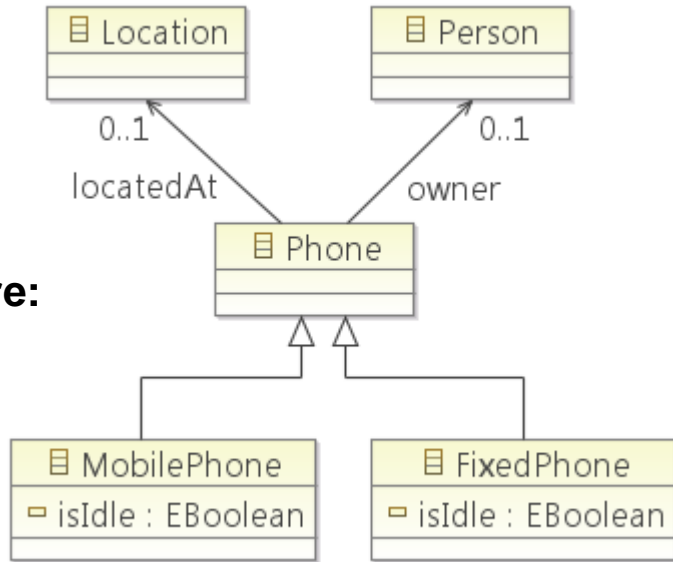
Selected Refactorings, improving

- **typing**: pull up attribute, push down attribute, extract supertype, remove supertype, extract node type, loop-edges-to Boolean-attributes
- **rules**: extract pre-condition, split rule into set of action rules, merge rules differing in types only, move versus delete and create, unify rules with same actions
- **feature usage**: inline pre-condition, Boolean-attribute-to-loop-edges,

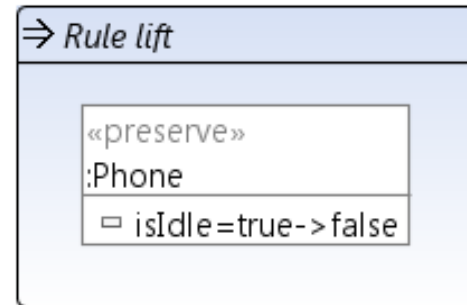
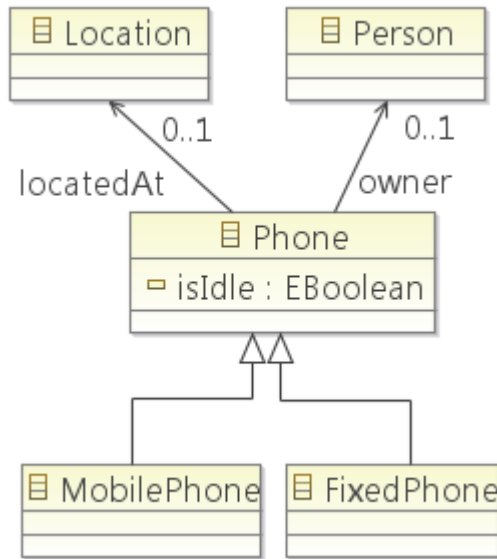
Refactoring „Merge Rules Differing in Types Only“



Before:



After:



Transformed Model	Original Model
◆ Phone System	◆ Phone System
◆ Mobile Phone	◆ Mobile Phone
◆ Fixed Phone false	◆ Fixed Phone true
◆ Person	◆ Person
◆ Location	◆ Location

Refactoring „Merge Rules Differing in Types Only“



Input parameters: Names of rules to be merged

Pre-condition: Rules differ in one object type only; the set of found object types contains all subclasses of a common superclass

Strategy:

- Identify all varying object types with a common superclass
- Construct new rule by taking one existing rule and replacing the identified subclass by the identified superclass; rename rule
- Delete all remaining original rules

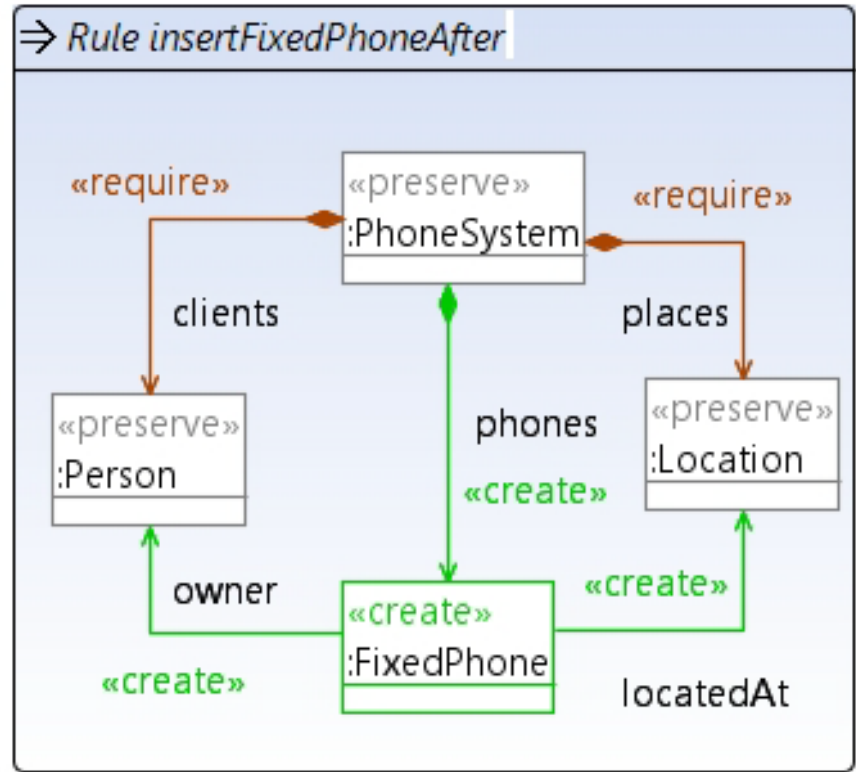
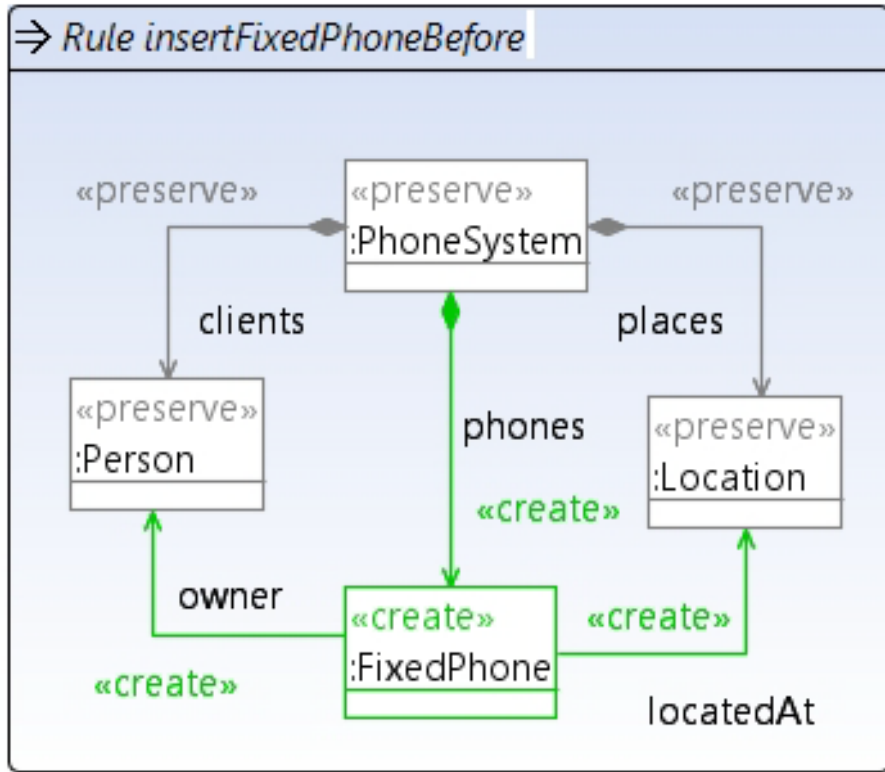
Post-condition:

- Original rules are replaced by one new rule using superclass type

Quality improvement: conciseness, comprehensibility

Semantics: is preserved

Refactoring „Extract Pre-condition“



Input parameters: Rule name

Pre-condition: none

Strategy:

- Determine preserved part
- Create new PAC and put preserved part in
- Reduce the rule's preserved part

Post-condition:

- Preserved part is minimal

Quality improvement:

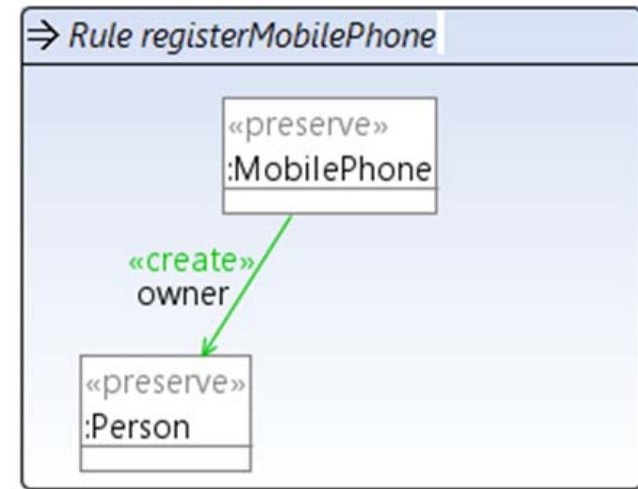
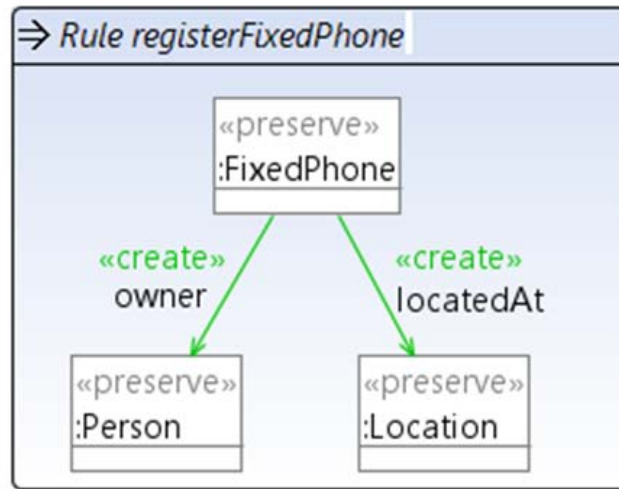
- conciseness, comprehensibility

Semantics: is preserved

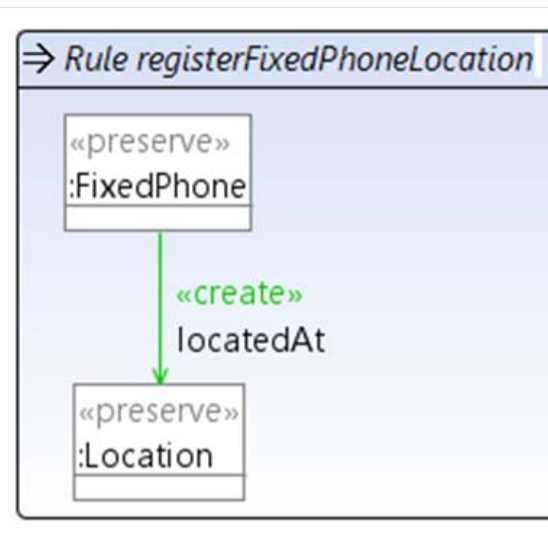
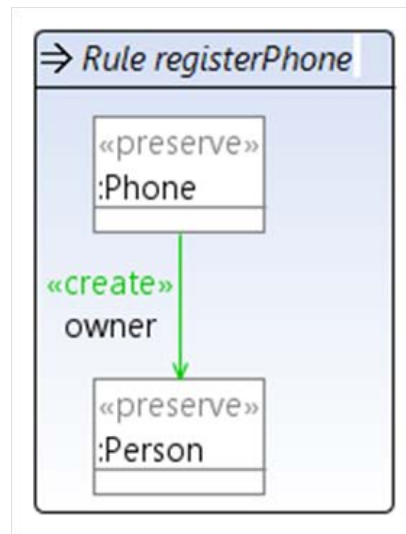
Refactoring „Unify Rules with Same Actions“



Before:



After:



Refactoring „ *Unify Rules with Same Actions*“

Input parameters: Set of rule names

Pre-condition: none

Strategy:

- Identify set of actions being shared by the set of input rules
- Create new rule scheme with common action (kernel rule) and remaining actions (remainder rules)
- Apply kernel rule before remainder rules

Post-condition:

- new kernel rule (common actions) and remainder rules

Quality improvement:

- conciseness

Semantics: more trafo rule sequences (more interleaving of rules)

Refactoring $r : MTS_1 \rightarrow MTS_2$ induces mapping of models (graphs).

Depending on the kind of semantics, e.g.

- **generated language** $L(MTS)$,
- **transformation relation** $MT(MTS)$ over input-output models,
- **operational semantics** $LTS(MTS)$,

preserving them means different things.

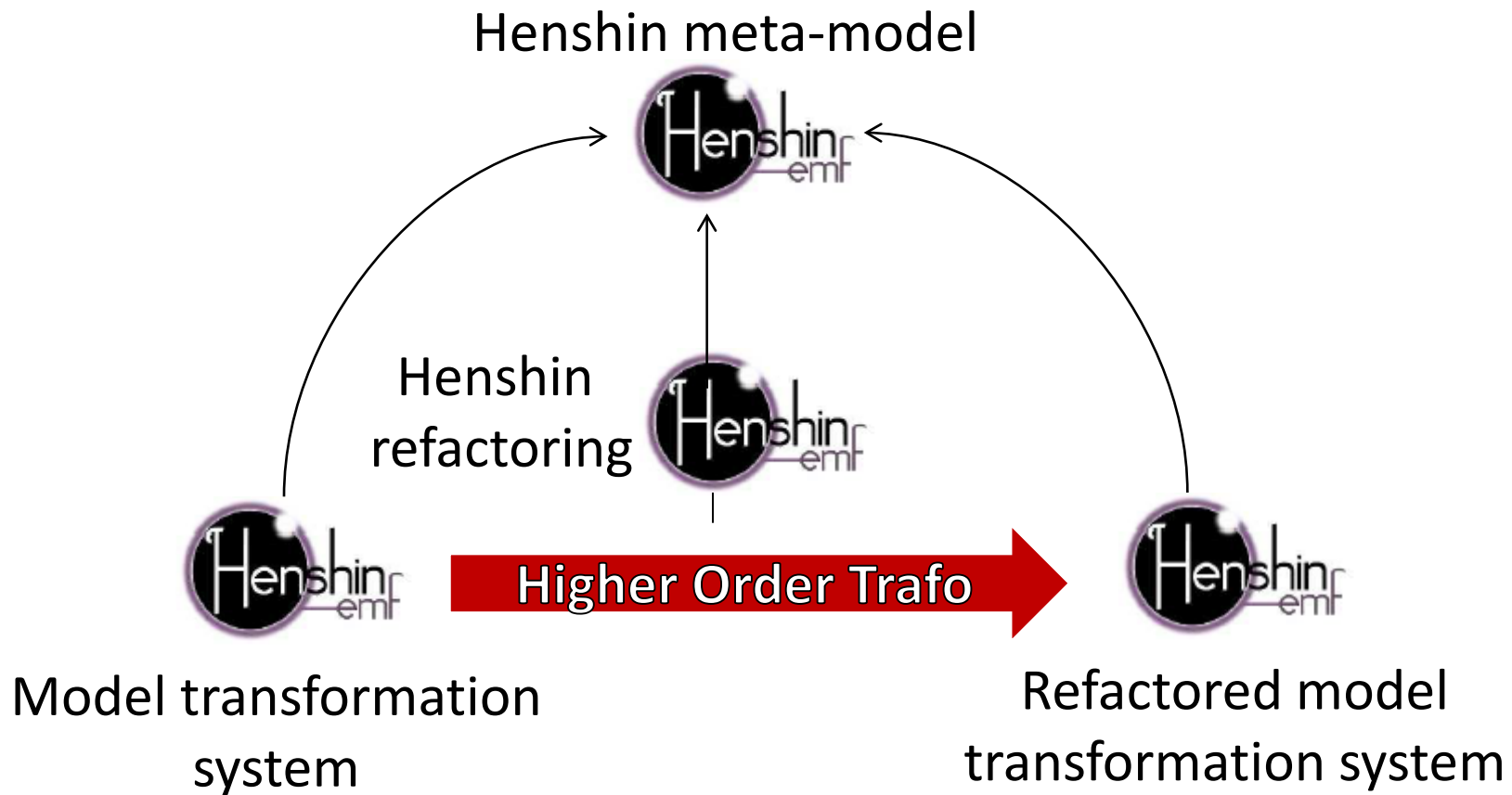
For example, r preserves the

- **generated language** if $r(L(MTS_1)) = L(MTS_2)$
- **transformation relation** if $r(MT(MTS_1)) = MT(MTS_2)$
- **operational semantics** if r is suitable observational equivalence relating $LTS(MTS_1)$ and $LTS(MTS_2)$

Prototypical Implementation

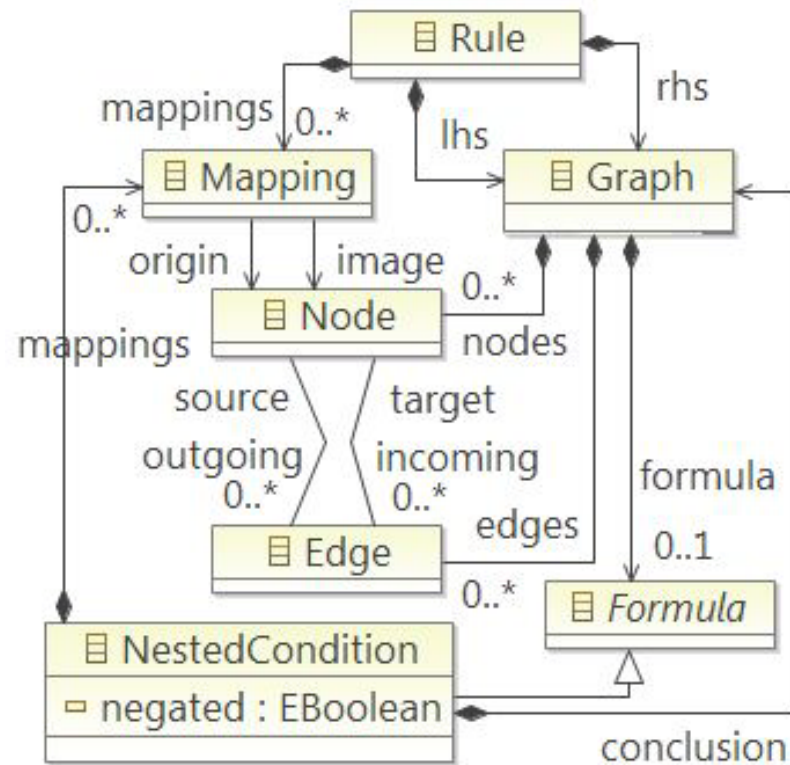


Model transformation language based on graph transformation concepts for the Eclipse Modeling Framework



Henshin Features

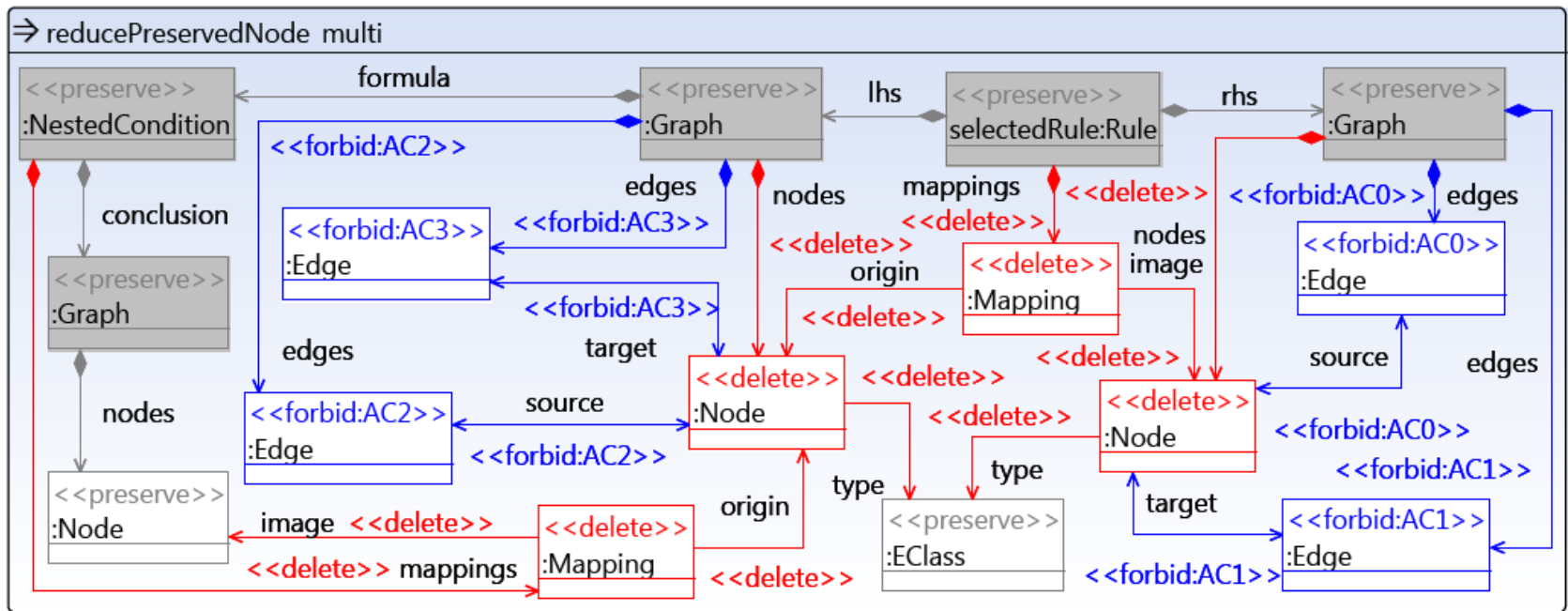
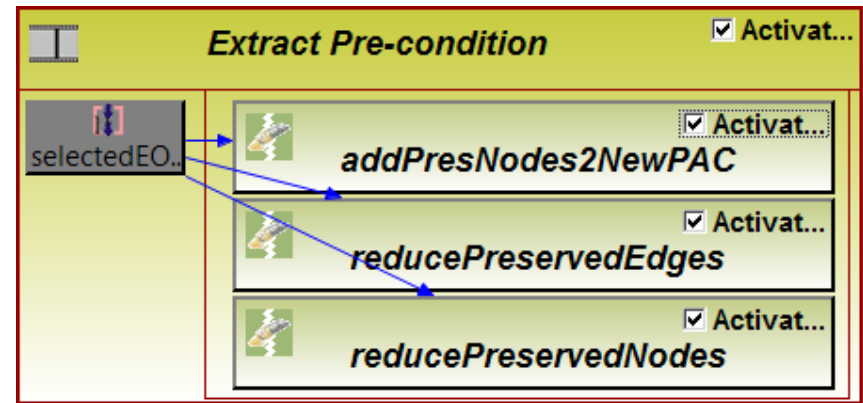
- *in-place*: models are transformed directly
- *rule-based*: transformation rules are defined visually
- *transformation model in EMF*:



- *refactorings*: higher-order transformation on the Henshin model

Example: Extract Pre-condition

Control Structures on Rules
(here: rule sequence)



Integration of refactorings in Henshin editors by EMF Refactor

New topic: Refactoring of model transformation systems

Modeling experiences are made explicit.

Selection of interesting refactorings as starting point

What next?

Startet joint activity to build up a Wiki with interesting smells and refactorings (for graph transformations so far)

The screenshot shows the Wikia website interface. At the top, there is a navigation bar with the Wikia logo, a 'Start a wiki' button, and category dropdowns for 'Video Games', 'Entertainment', and 'Lifestyle'. On the right side of the navigation bar are 'Log in' and 'Sign up' links. The main header area features the title 'Refactorings for Graph Transformation Systems Wiki' in large blue text. Below the title are links for 'Popular pages' and 'Community'. To the right of the title are buttons for 'Random Page' and 'Wiki Activity'. A secondary navigation bar includes a 'Home' link with an 'Edit' dropdown, a '5 PAGES ON THIS WIKI' indicator, an 'Add a Page' button, a 'Talk' link with a '0' count, a 'Like' button with a '0' count, and a search box labeled 'Search this wiki'. Below this is a 'Contents [show]' button. The main content area starts with a 'Welcome to the Refactorings for Graph Transformation Systems Wiki' message with an 'Edit' link. This is followed by a paragraph: 'This wiki collects refactoring definitions and specifications for graph transformation systems (GTS). These refactorings can be used to improve some quality aspects of a specific GTS and serve as a first step towards an overall quality assurance of graph transformation systems.' Below this is a 'Describe your topic' section with an 'Edit' link, containing a paragraph: 'Graph transformations are used in various fields of computer science research. As a consequence, graph transformations systems must be of high quality in order to be applied properly. One possible technique to improve this quality is refactoring as used in several other research domains. In this wiki, refactorings for graph transformation systems are collected and systematically specified.' The final section is 'Current refactorings' with an 'Edit' link, listing four items: 'Extract Pre-condition', 'Loop Edges to Boolean Attributes', 'Pull Up Attribute', and 'Split Rule into Set of Action Rules'.

wikia Start a wiki Video Games Entertainment Lifestyle Log in Sign up

Refactorings for Graph Transformation Systems Wiki

Popular pages | Community

Random Page Wiki Activity

Extract Pre-condition

Edit 0 Comments Like 0

Contents [show]

Description

This refactoring reduces the preserved part of a rule and extracts it as positive application condition.

Input parameters

- name of the rule (respectively the rule itself)

Example

A customer takes an item out of the shelf. The rule mainly consists of context which has to be determined. We extract this context into a positive application condition which makes the rule considerably smaller (see figure).

The diagram illustrates the refactoring process. The top part shows a rule named 'PutInCart of Shopping'. The LHS (Left Hand Side) consists of nodes 5:Customer, 3:Shelf, 6:Item, and 4:Cart. The RHS (Right Hand Side) consists of nodes 5:Customer, 3:Shelf, 6:Item, and 4:Cart. The bottom part shows a refactored rule named 'PutInCart2 of Shopping'. The Context (PAC) contains nodes Customer, Shelf, and 2:Cart. The LHS (Left Hand Side) consists of nodes 2:Cart and 1:Item. The RHS (Right Hand Side) consists of nodes 2:Cart and 1:Item.

New topic: Refactoring of model transformation systems

Modeling experiences are made explicit

Selection of interesting refactorings as starting point

What next?

Started joint activity to build up a Wiki with interesting smells and refactorings (for graph transformations so far)

Cover further model transformation features: join forces with Wimmer et al. (including also constraints, e.g. OCL)

M. Wimmer, S. Martinez, F. Jouault, J. Cabot: **A Catalogue of Refactorings for Model-to-Model Transformations**. Journal of Object Technology, August 2012

**THANK YOU
FOR YOUR
ATTENTION**