Henshin: A Model Transformation Language and its Use for Search-Based Model Optimisation in MDEOptimiser

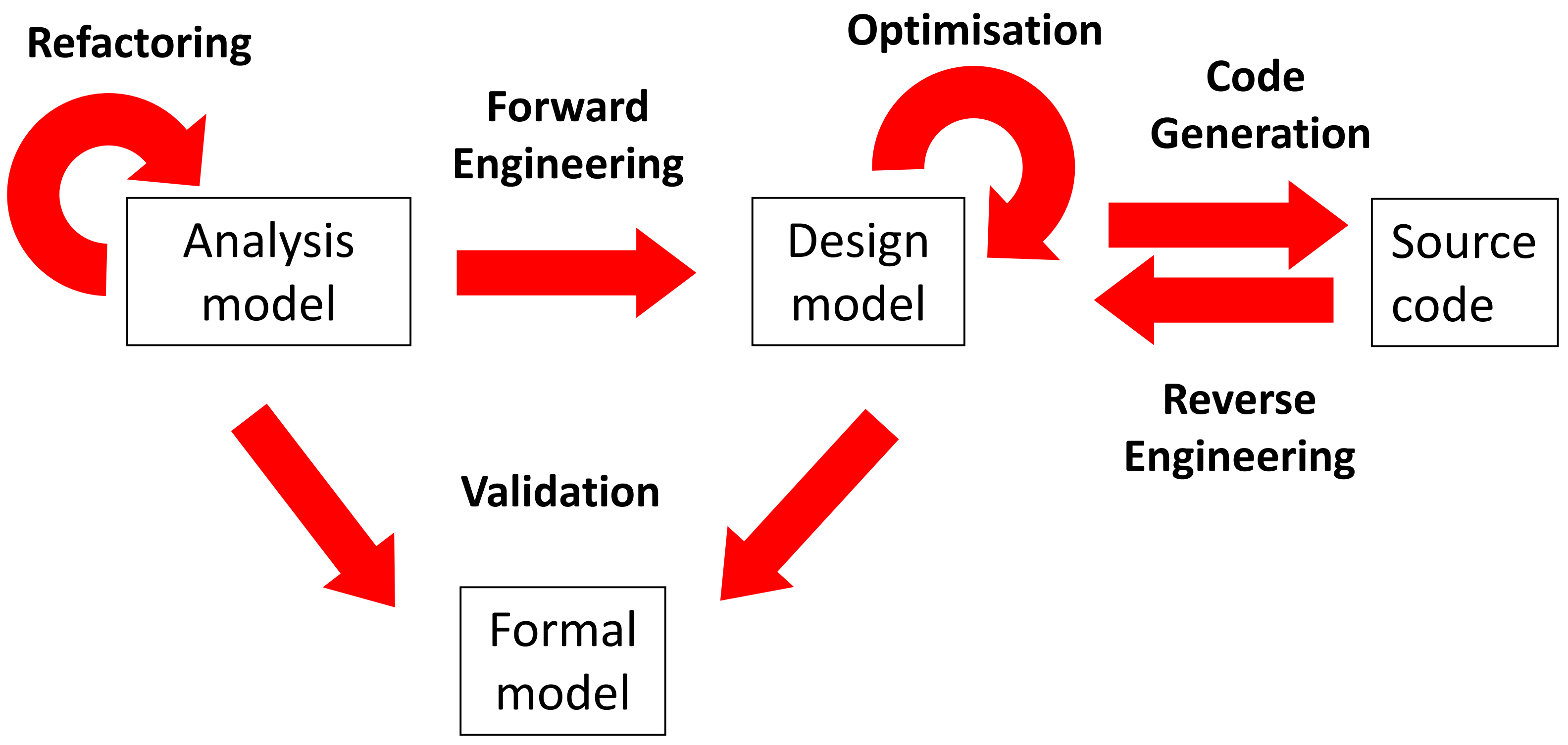
Part 1

Daniel Strüber[1], Alexandru Burdusel[2], Stefan John[3], Steffen Zschaler[2]

[1] Universität Koblenz-Landau,
[2] King's College London, [3] Philipps-Universität Marburg

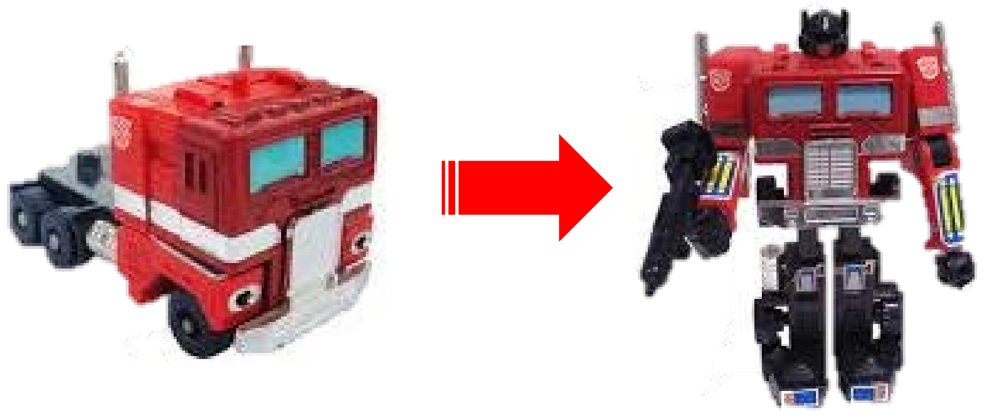
Fachtagung Modellierung
February 21, 2018

Model-driven software engineering: Transformations everywhere



Henshin

- Intuitive model transformation language with graphical syntax
- Supports various kinds of transformations
- Based on graph transformation theory
    - Rule-based
    - Expressive: advanced concepts



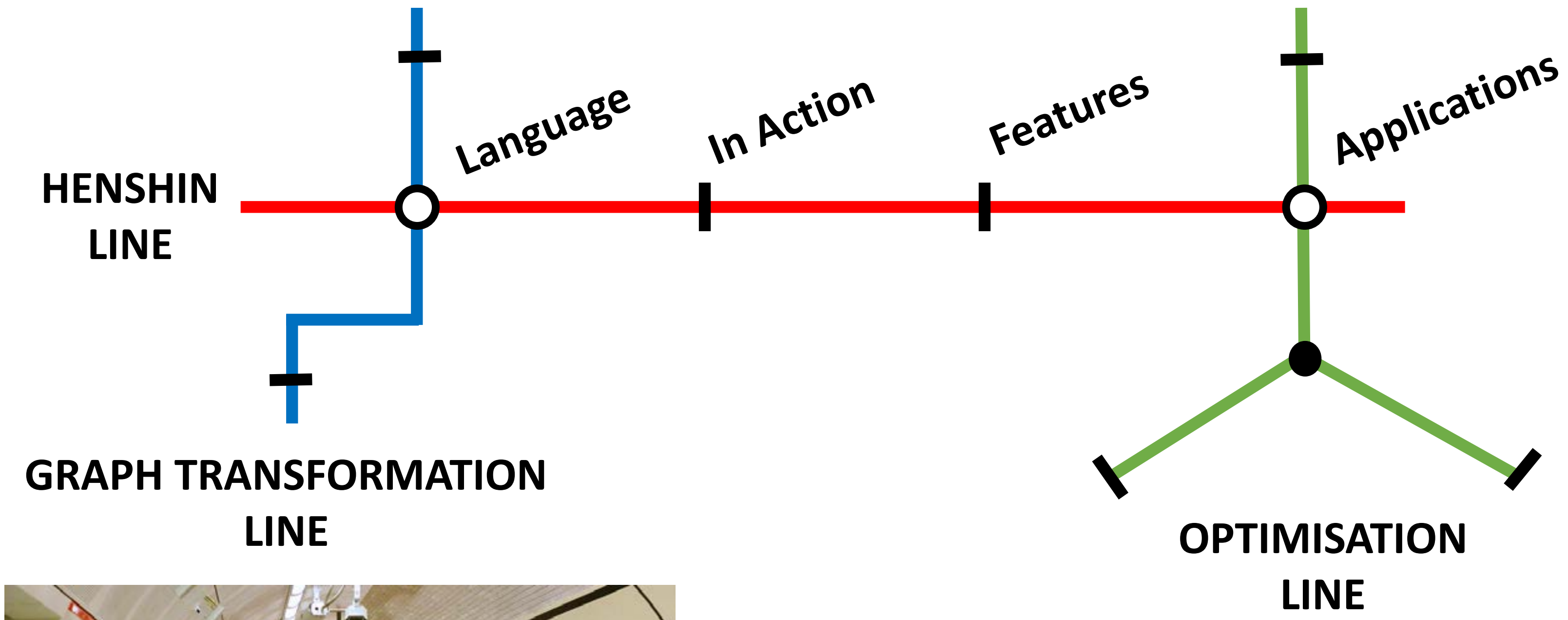
Henshin: Japanese for *Transformation*

Overview

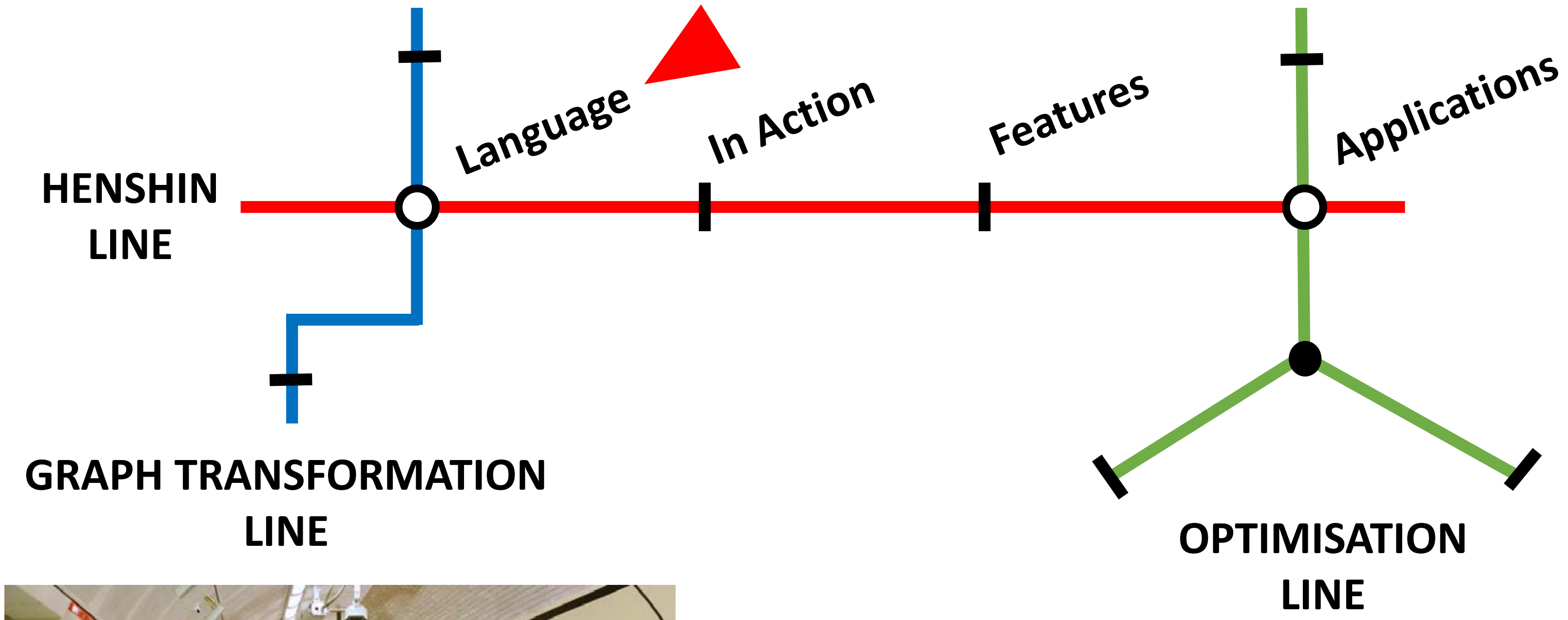
- Part 1: Henshin: A Guided Tour
  - Language
  - In Action (interactive)
  - Features
  - Applications

- Part 2: Henshin in Search-Based Model Optimization
  - Background
  - MDEOptimiser
  - Case 1: Class Responsibility Assignment (interactive)
  - Case 2: SCRUM Planning (interactive)

Henshin: A Guided Tour



Henshin: A Guided Tour

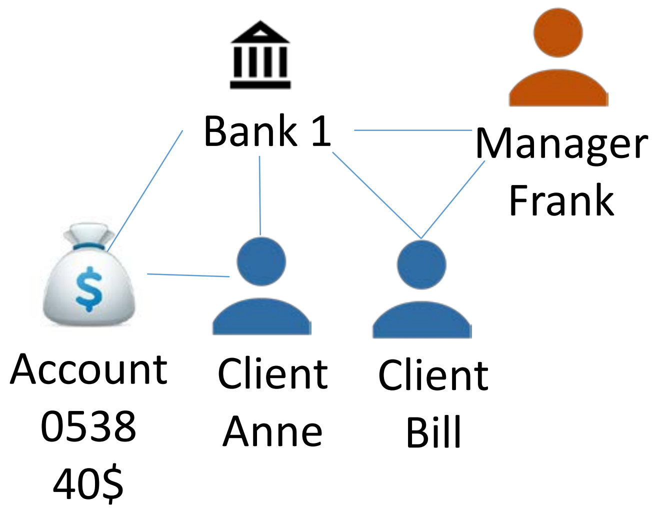


Language: Running example

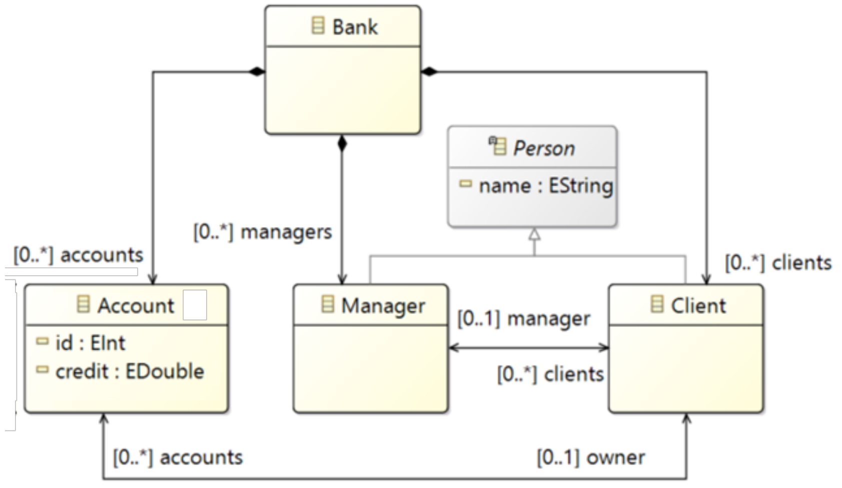
Specify banking processes to analyse and simulate them

1. Create an account
2. Transfer money
3. Delete an account
4. Batch-delete accounts

Example model

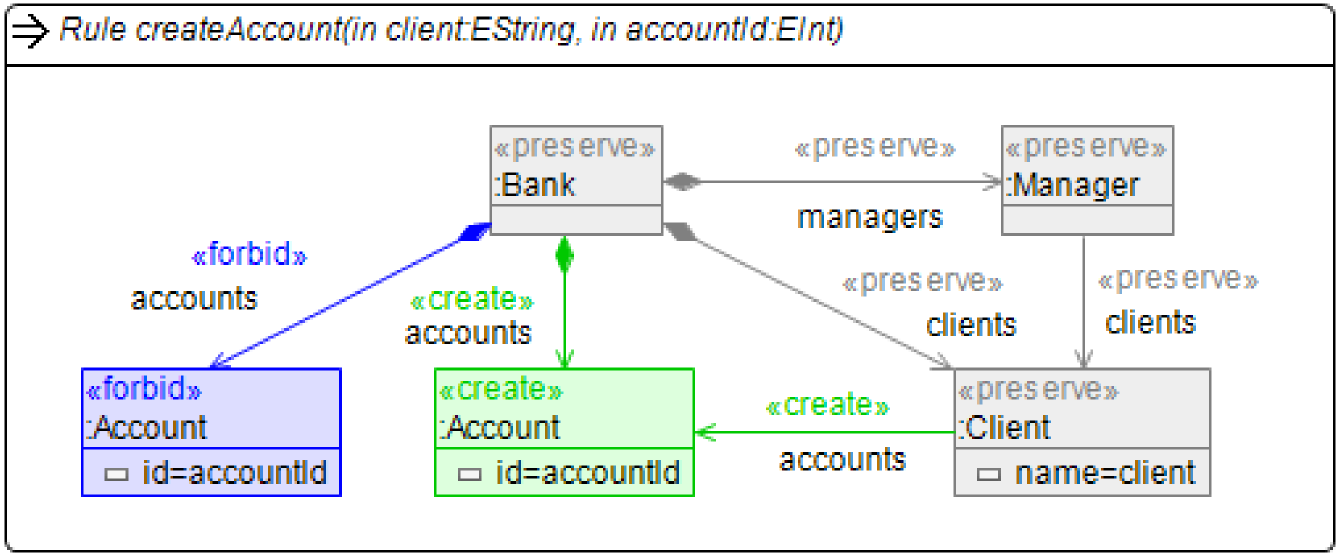


Example meta-model (in EMF)



Graph-transformation-based language Example 1: createAccount

Example rule

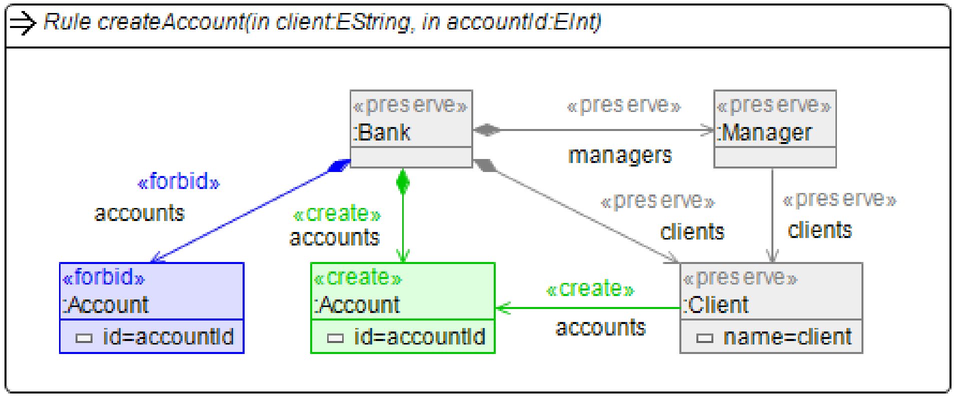


Rule createAccount(in client:EString, in accountId:EInt)
«preserve» :Bank
«preserve» managers
«preserve» :Manager
«forbid» accounts
«create» accounts
«preserve» clients
«preserve» clients
«forbid» :Account id=accountId
«create» :Account id=accountId
«create» accounts
«preserve» :Client name=client

create	Newly created by rule
delete	Removed by rule
preserve	Context for creations and deletions
forbid	Prevents rule from being applied
require	Additional required parts
parameters	Data passed into and from rule (in, out, inout)

Example 1: Create an account

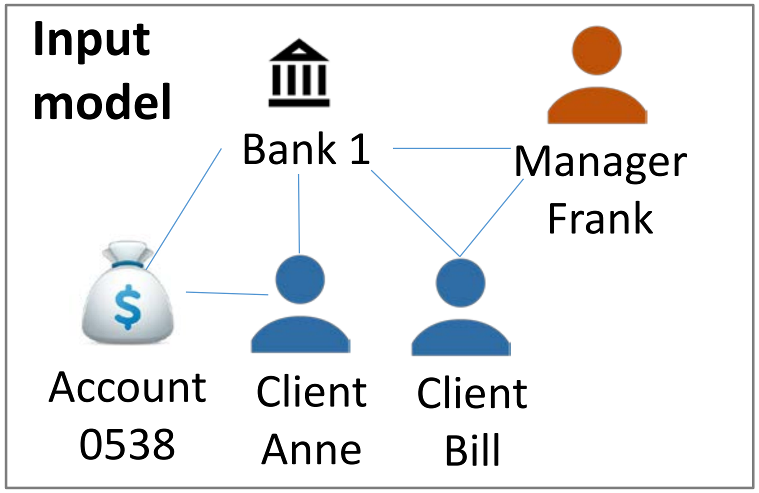
Example application of rule



with parameter values

client = **“Bill“**

accountID = **0539**

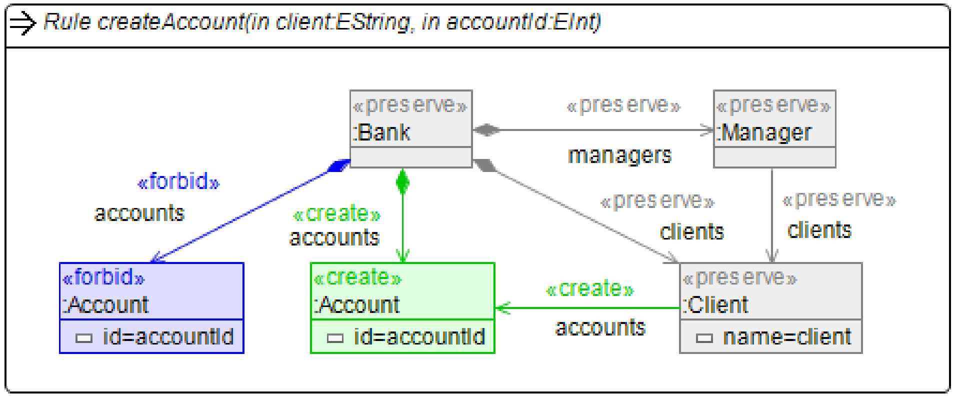


Output model

?

Example 1: Create an account

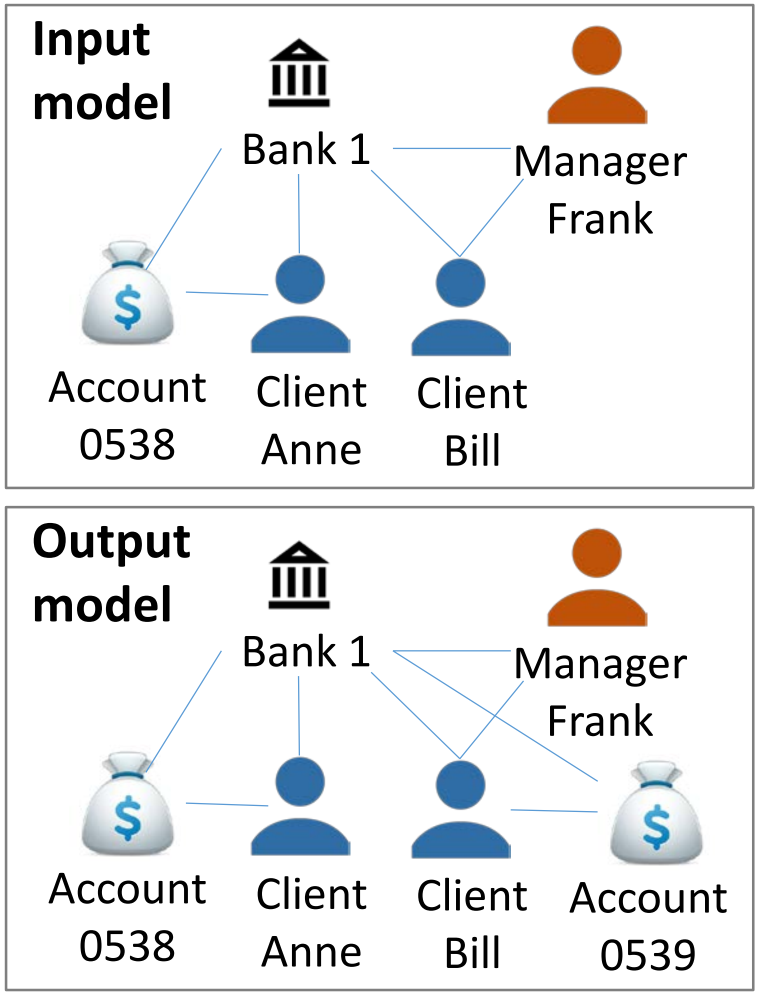
Example application of rule



with parameter values

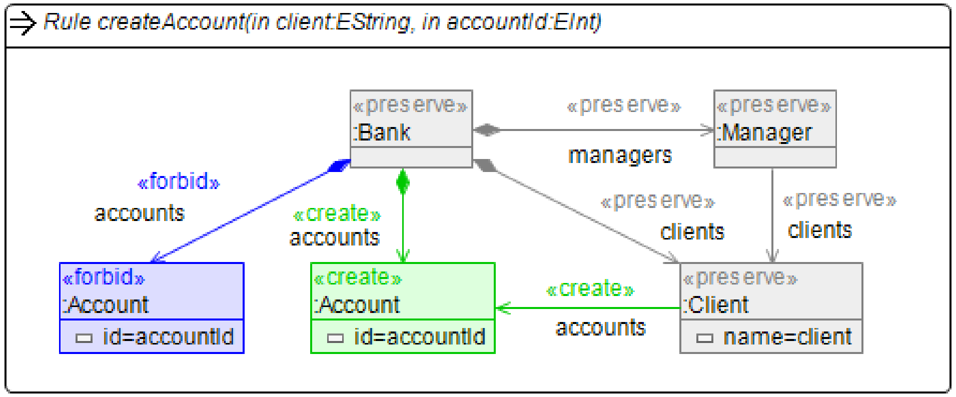
client = **"Bill"**

accountID = **0539**



Example 1: Create an account

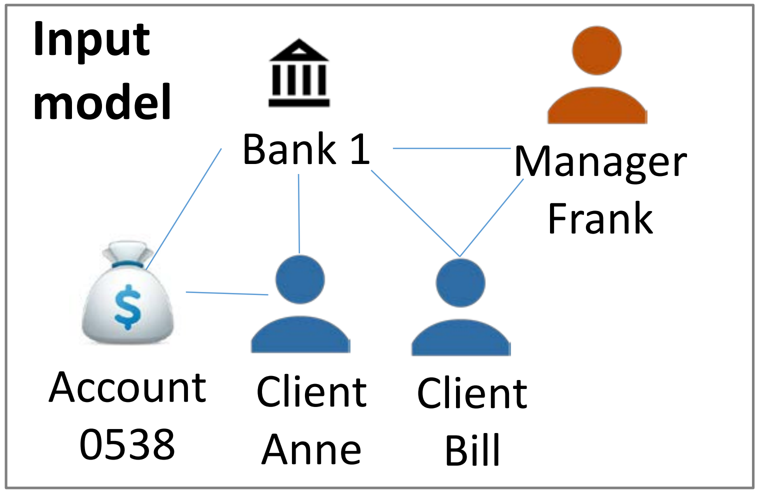
Example application of rule



with parameter values

client = **“Bill“**

accountID = **0538**

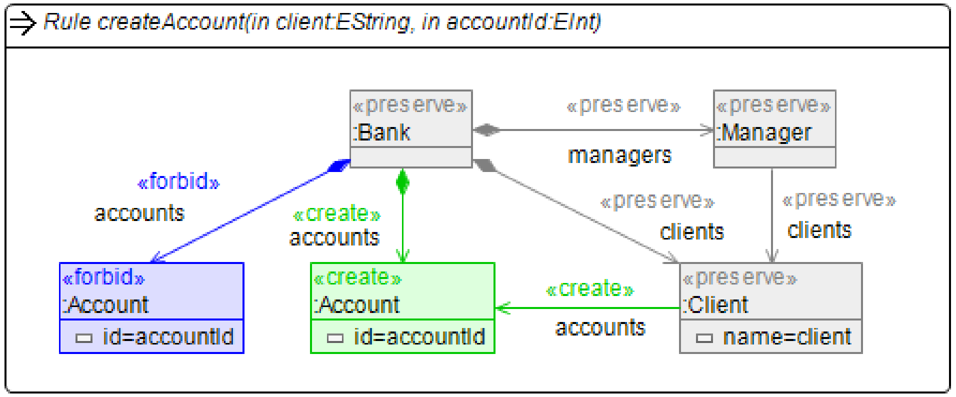


Output model

?

Example 1: Create an account

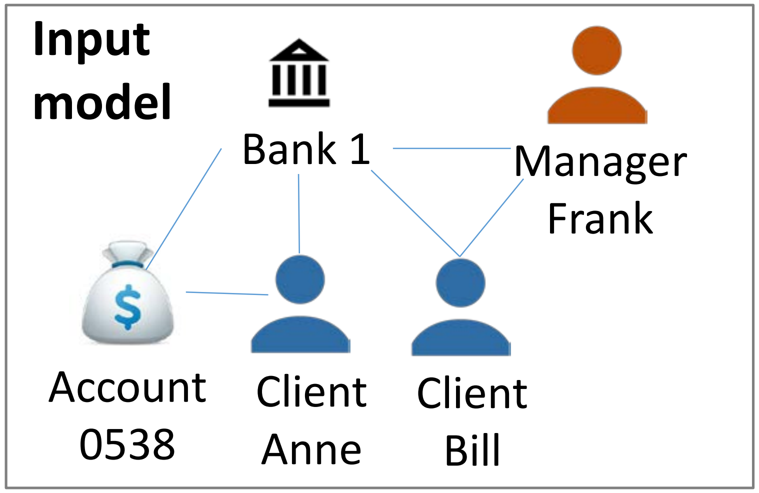
Example application of rule



with parameter values

client = **“Bill“**

accountID = **0538**

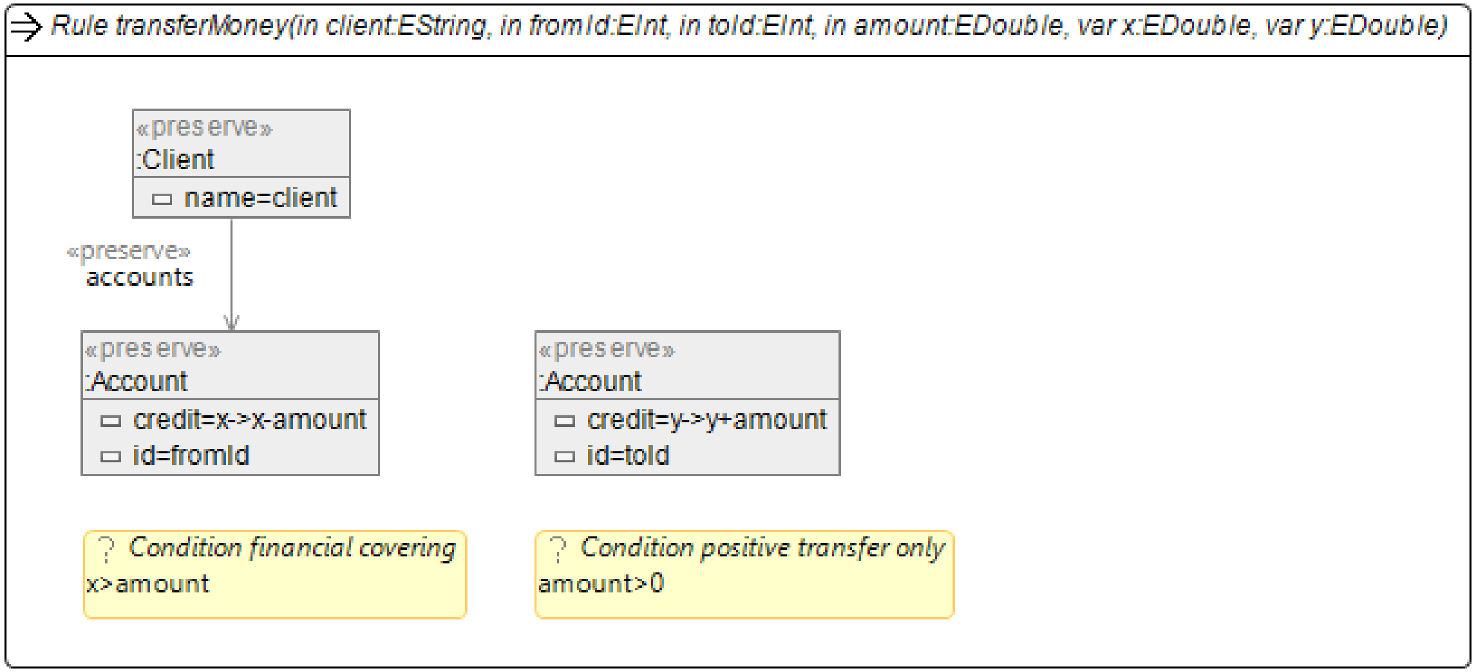


Output model

No rule application possible

Example 2: Transfer money

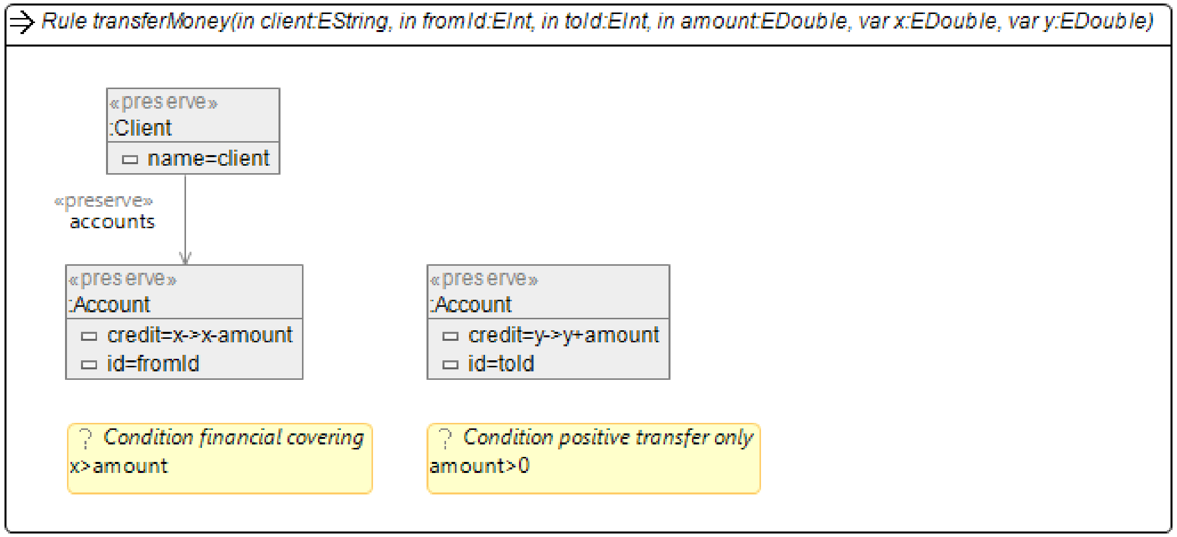
Example rule


Rule transferMoney(in client:EString, in fromId:EInt, in toId:EInt, in amount:EDouble, var x:EDouble, var y:EDouble)
«preserve» :Client
name=client
«preserve» accounts
«preserve» :Account
credit=x->x-amount
id=fromId
«preserve» :Account
credit=y->y+amount
id=toId
Condition financial covering
x>amount
Condition positive transfer only
amount>0

Variables (var keyword) used inside rules to propagate values
Attribute manipulated using parameters, variables and ->
Conditions can restrict rule applications

Example 2: Transfer money

Example application of rule



Rule transferMoney(in client:EString, in fromId:EInt, in toId:EInt, in amount:EDouble, var x:EDouble, var y:EDouble)
«preserve» :Client
name=client
«preserve» accounts
«preserve» :Account
credit=x->x-amount
id=fromId
«preserve» :Account
credit=y->y+amount
id=toId
Condition financial covering
x>amount
Condition positive transfer only
amount>0

with parameters

client = **“Anne“**

fromID = **0538**

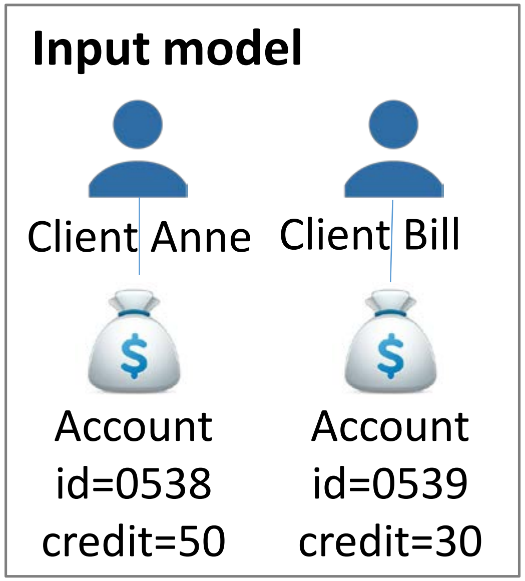
toID = **0539**

amount = **10**

variables:

set automatically

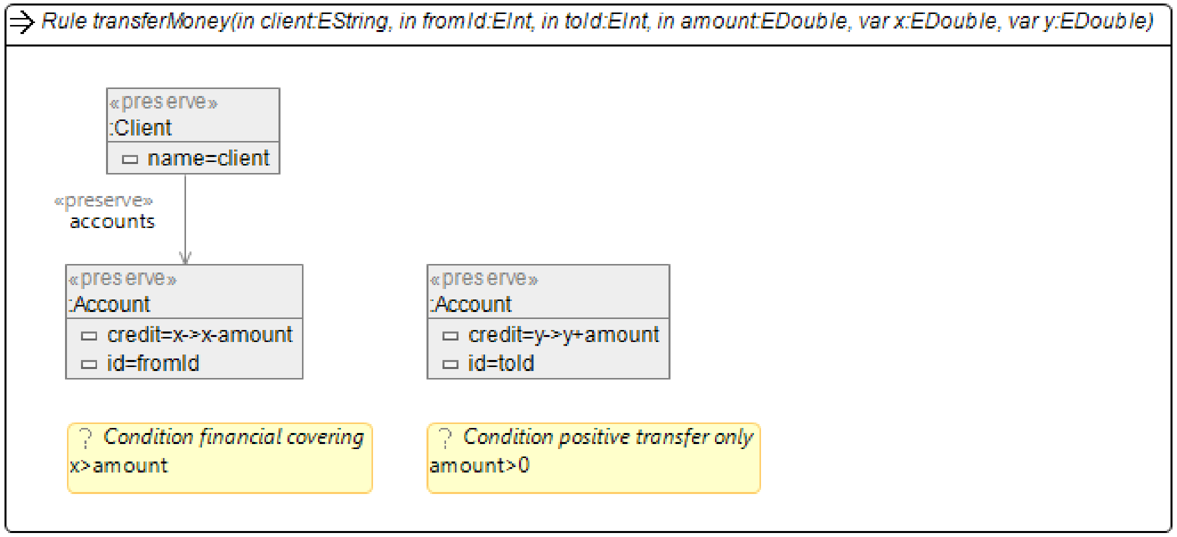
on rule application



Output model ?

Example 2: Transfer money

Example application of rule


Rule transferMoney(in client:EString, in fromId:EInt, in toId:EInt, in amount:EDouble, var x:EDouble, var y:EDouble)
«preserve» :Client
name=client
«preserve» accounts
«preserve» :Account
credit=x->x-amount
id=fromId
«preserve» :Account
credit=y->y+amount
id=toId
Condition financial covering
x>amount
Condition positive transfer only
amount>0

with parameters

client = **"Anne"**

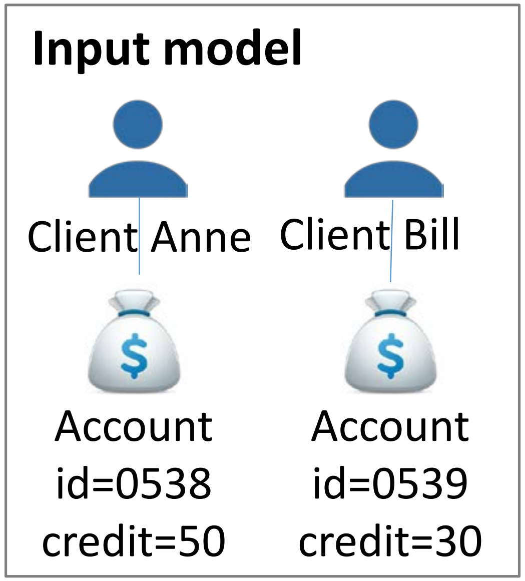
fromID = **0538**

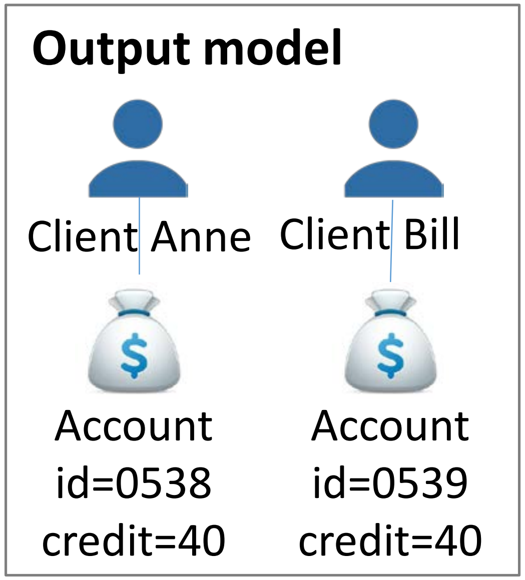
toID = **0539**

amount = **10**

variables:

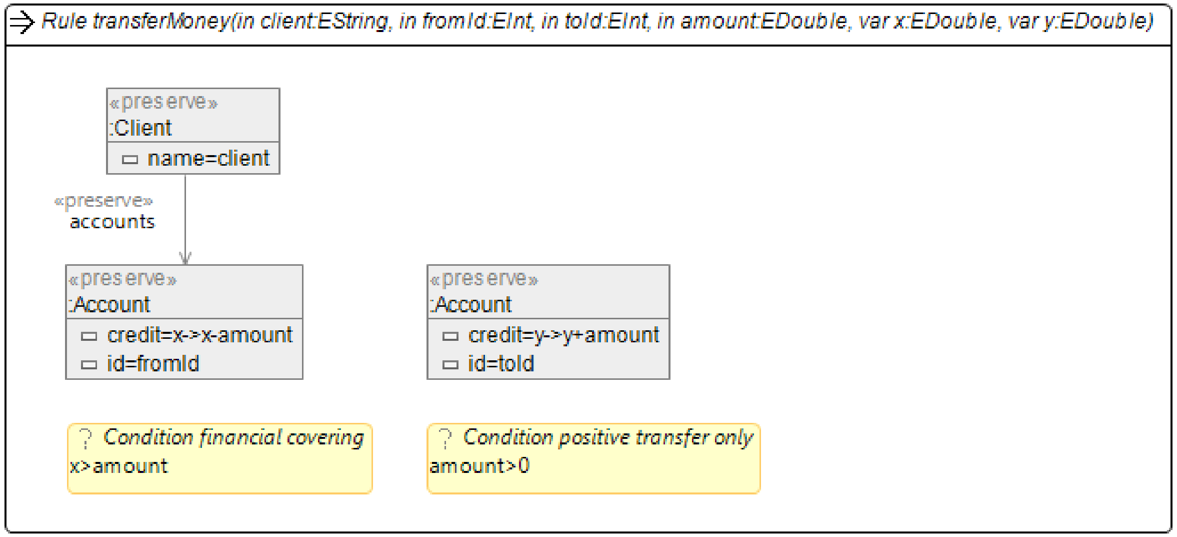
set automatically on rule application





Example 2: Transfer money

Example application of rule



Rule transferMoney(in client:EString, in fromId:EInt, in toId:EInt, in amount:EDouble, var x:EDouble, var y:EDouble)

«preserve» :Client
name=client

«preserve» accounts

«preserve» :Account
credit=x->x-amount
id=fromId

«preserve» :Account
credit=y->y+amount
id=toId

Condition financial covering
x>amount

Condition positive transfer only
amount>0

with parameters

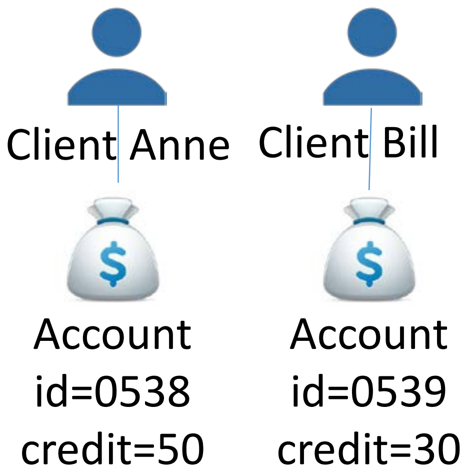
client = **“Anne“**

fromID = **0538**

toID = **0539**

amount = **-30**

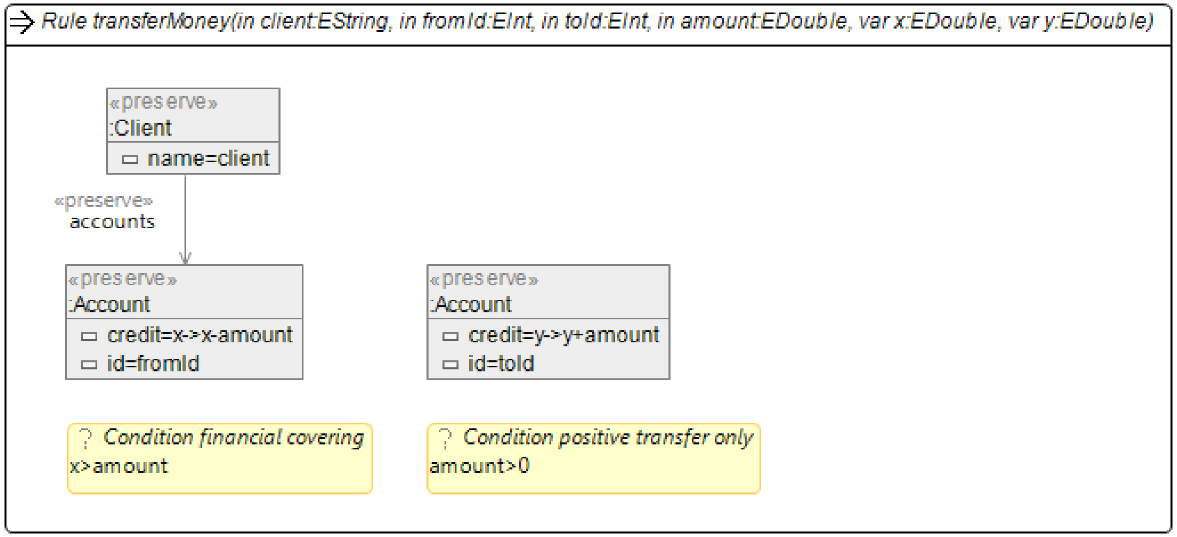
Input model



Output model ?

Example 2: Transfer money

Example application of rule


Rule transferMoney(in client:EString, in fromId:EInt, in toId:EInt, in amount:EDouble, var x:EDouble, var y:EDouble)
«preserve» :Client
name=client
«preserve» accounts
«preserve» :Account
credit=x->x-amount
id=fromId
«preserve» :Account
credit=y->y+amount
id=toId
Condition financial covering
x>amount
Condition positive transfer only
amount>0

with parameters

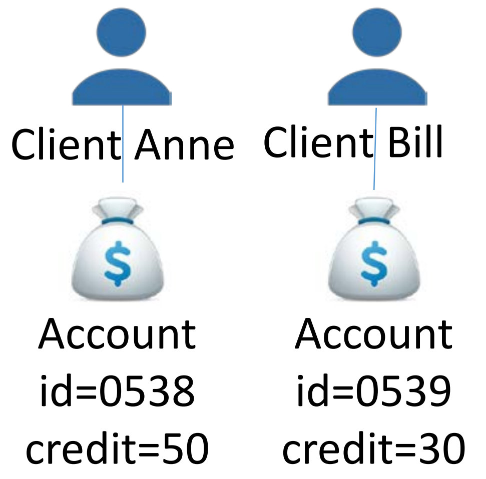
client = **“Anne“**

fromID = **0538**

toID = **0539**

amount = **-30**

Input model

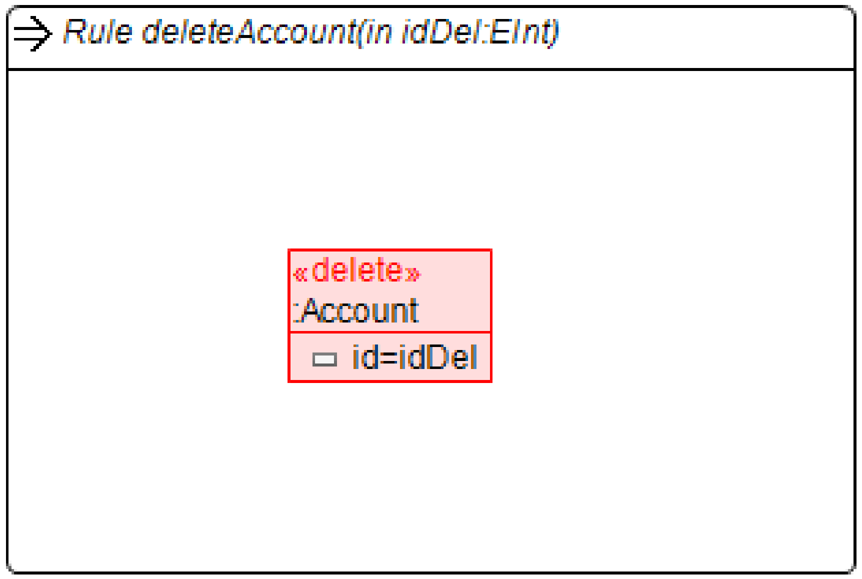


Output model

No rule application possible

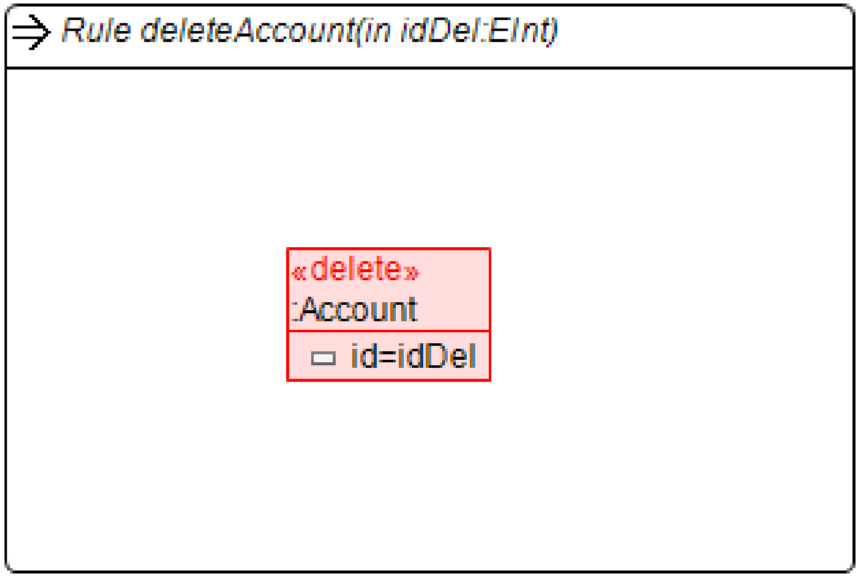
Example 3: Delete an account

Example rule (first draft)



Example 3: Delete an account

Example rule (first draft)

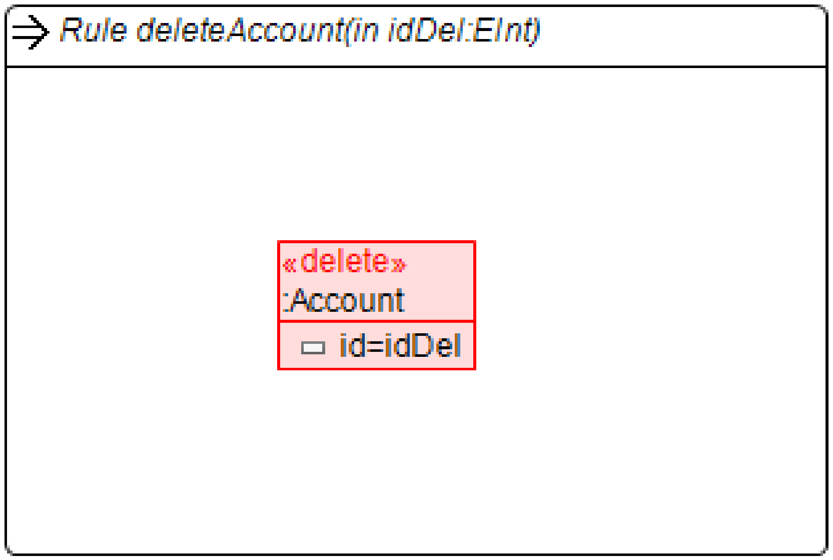


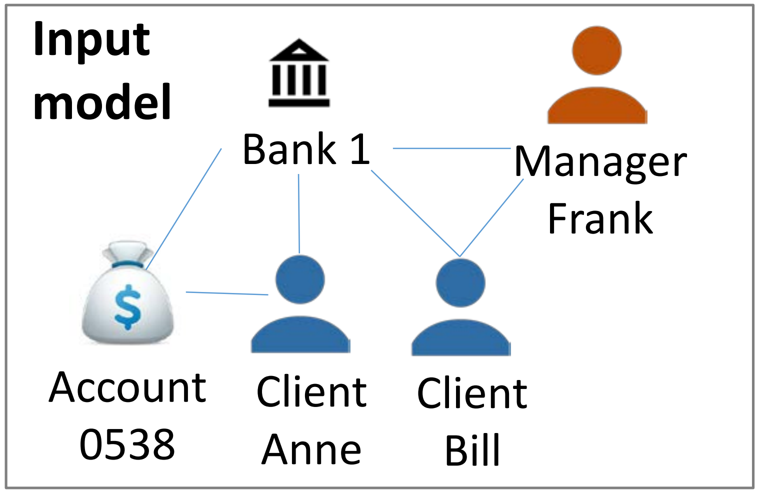
Want to delete an account which is given by its ID.

Is this rule sufficient?

Example 3: Delete an account

Example rule (first draft)



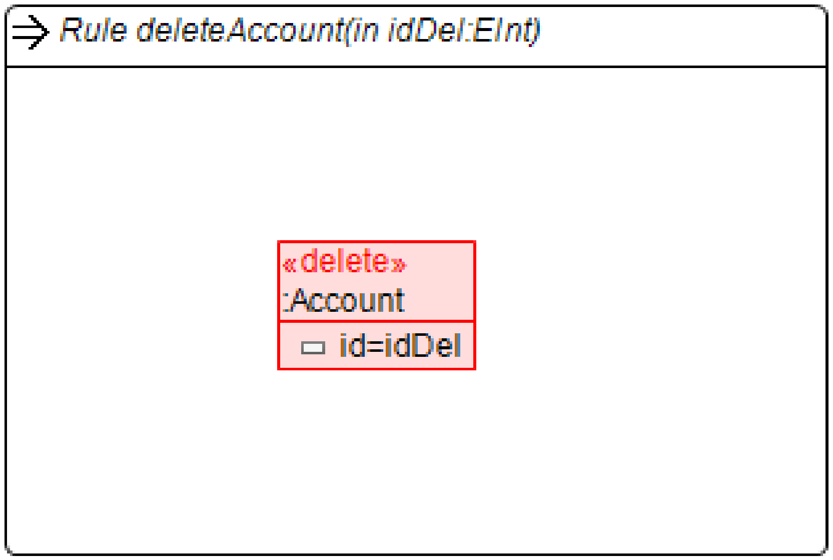


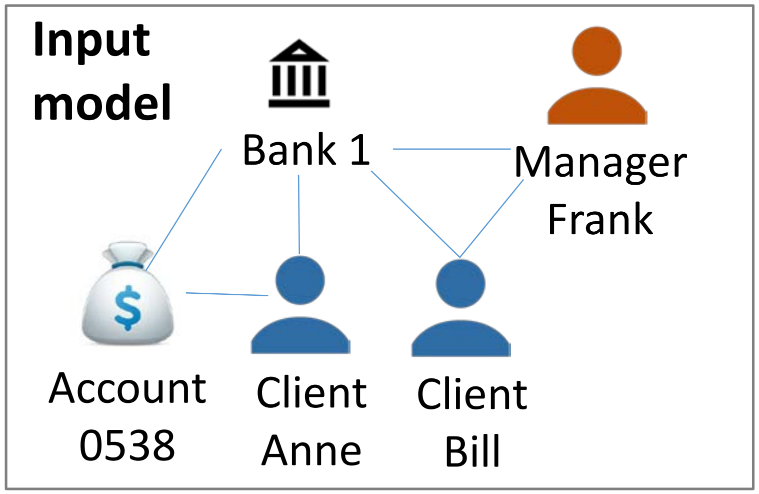
Output model

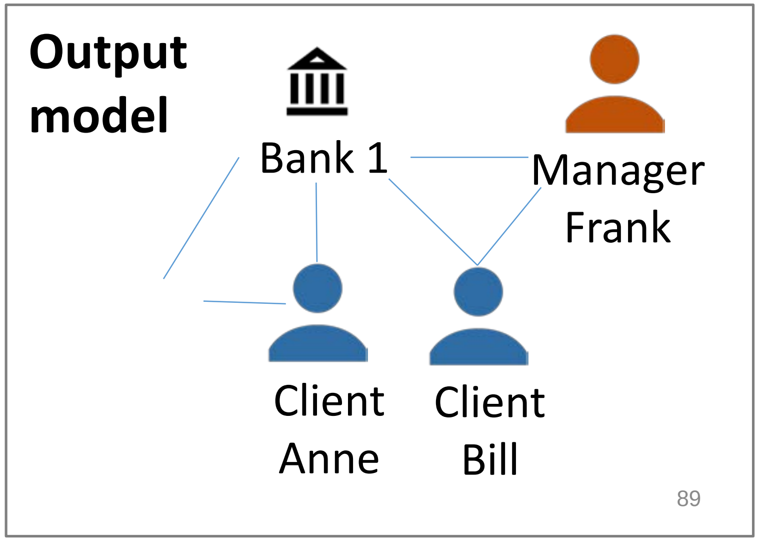
?

Example 3: Delete an account

Example rule (first draft)

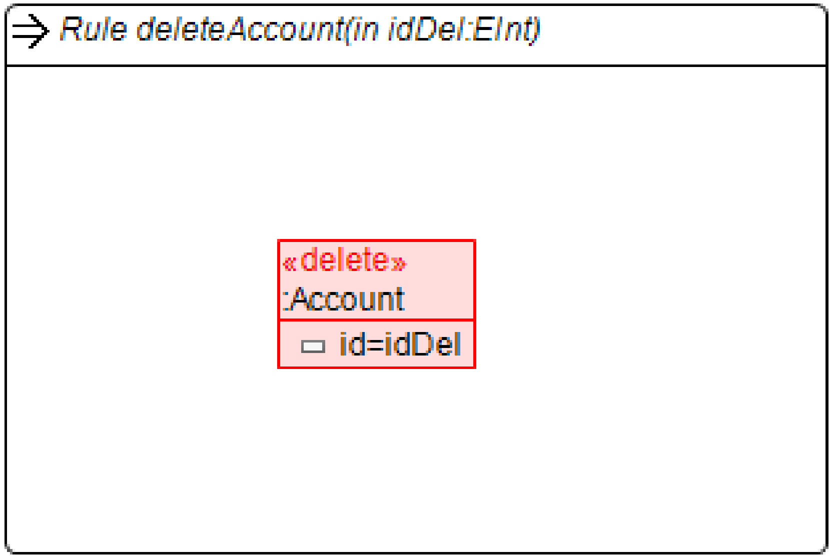




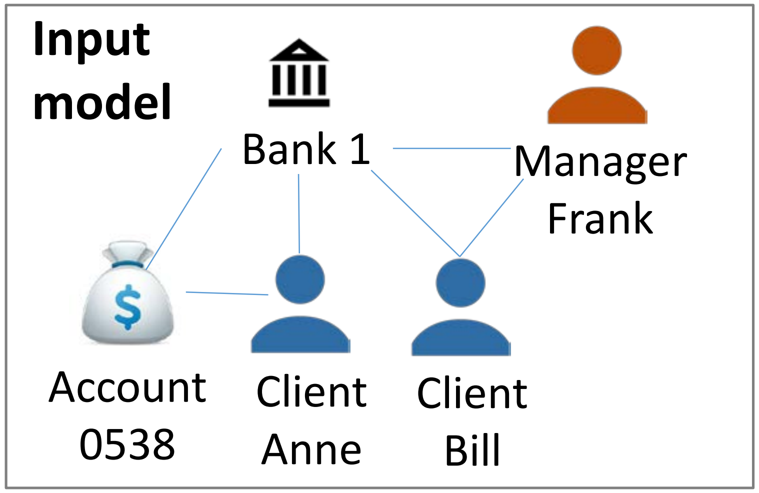


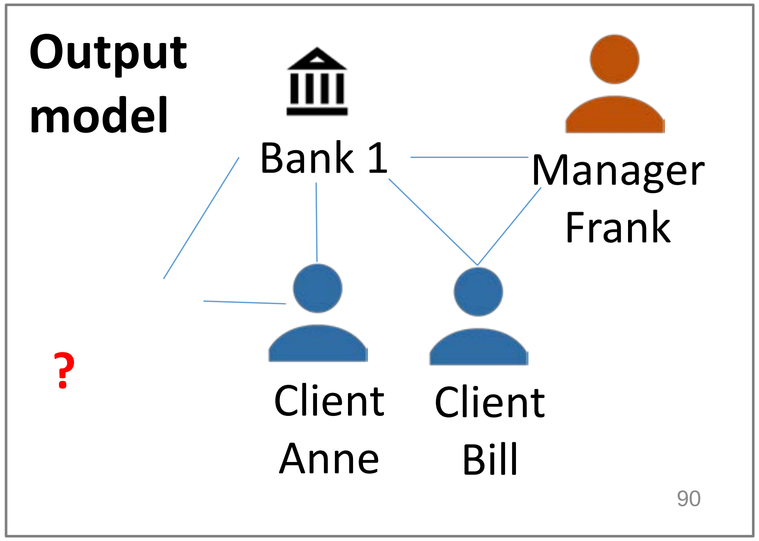
Example 3: Delete an account

Example rule (first draft)



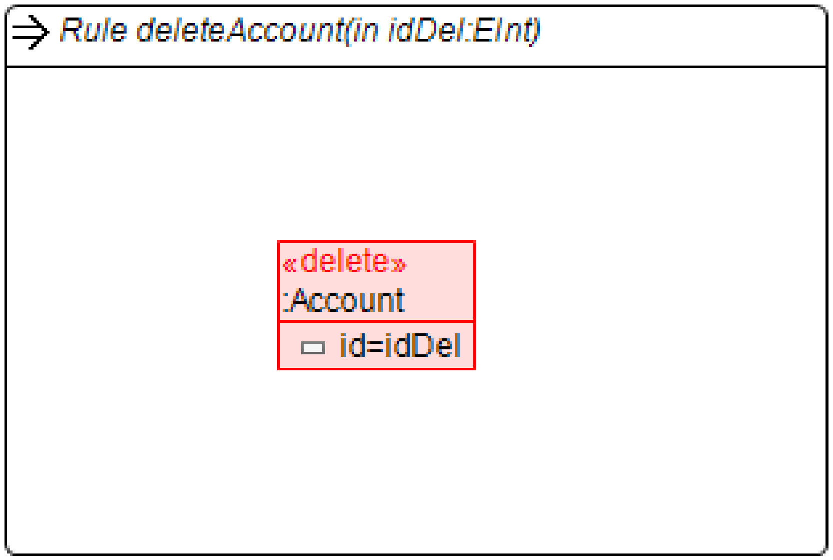
By deleting the node only, without incident edges, these edges would be left behind dangling.
Henshin ensures this won‘t happen

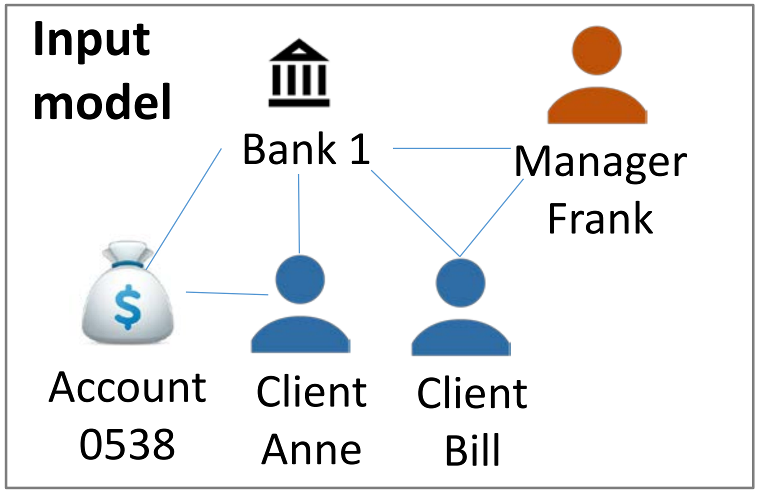




Example 3: Delete an account

Example rule (first draft)





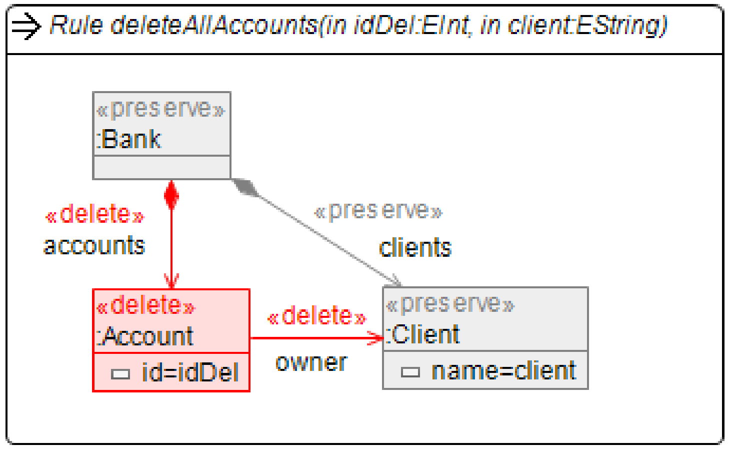
Output model

No rule application possible

By deleting the node only, without incident edges, these edges would be left behind dangling.
Henshin ensures this won‘t happen

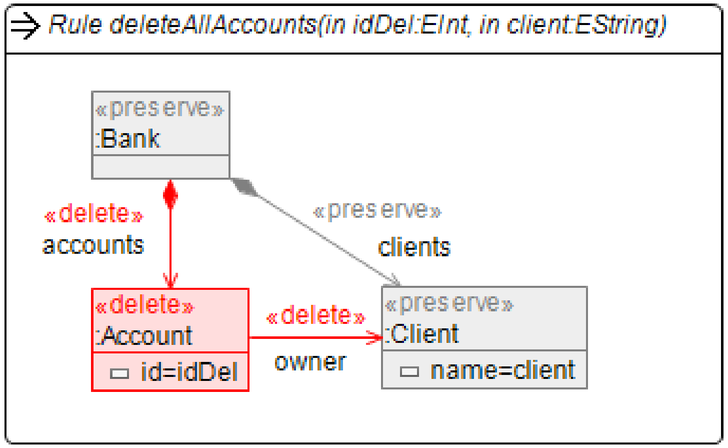
Example 3: Delete an account

Example rule (improved)



Example 3: Delete an account

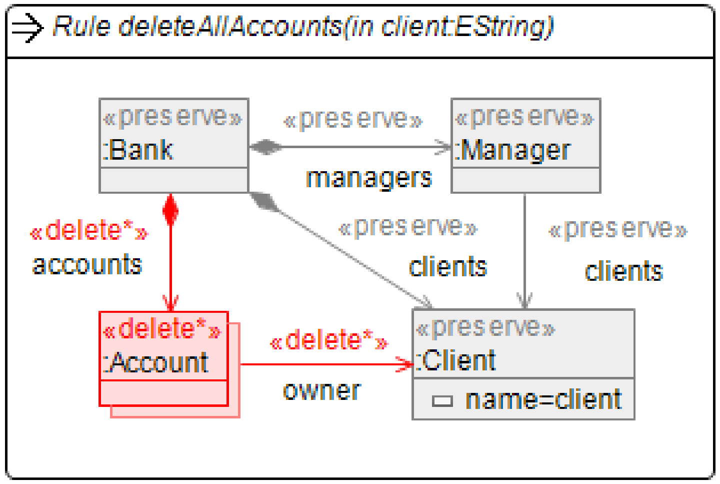
Example rule (improved)



Deletion: When deleting a model element, need to specify all references from and to that element as deleted, too. (**Dangling Condition**)

Example 4: Batch-delete accounts

Example rule



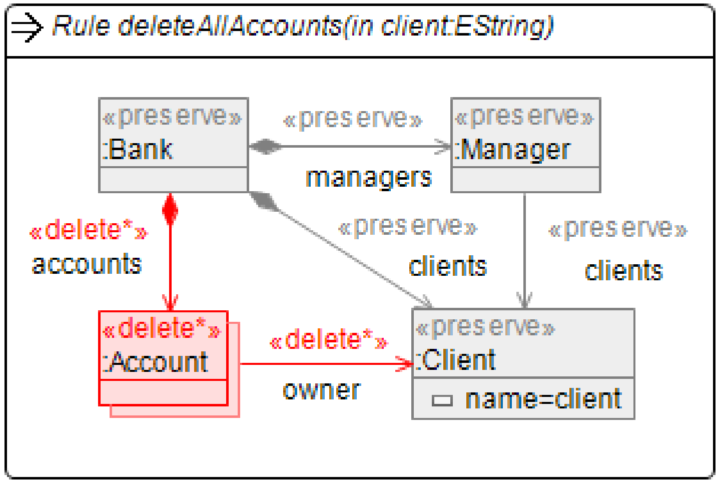
For-all operator: multi-rule (*)

Semantics:

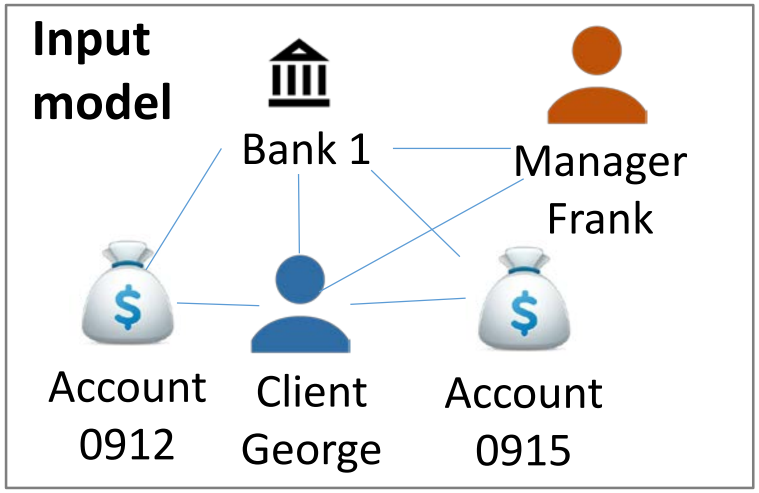
1. apply kernel rule (part without *) once
2. apply multi-rule **as often as possible** at the given place in the input model

Example 4: Batch-delete accounts

Example application of rule



with parameter
client = **“George“**

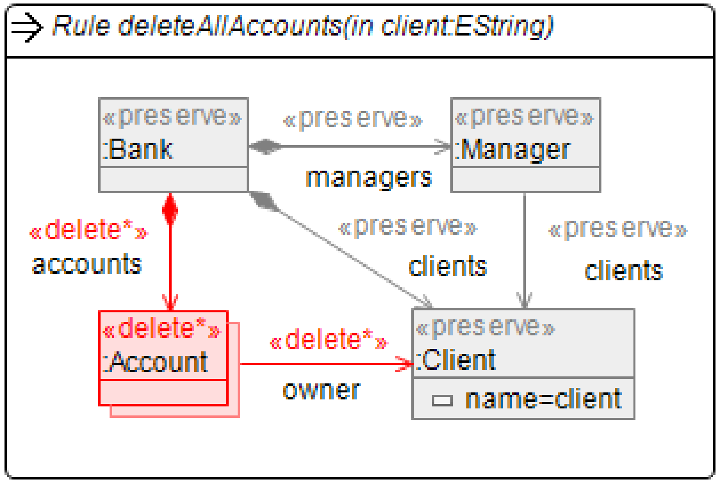


Output model

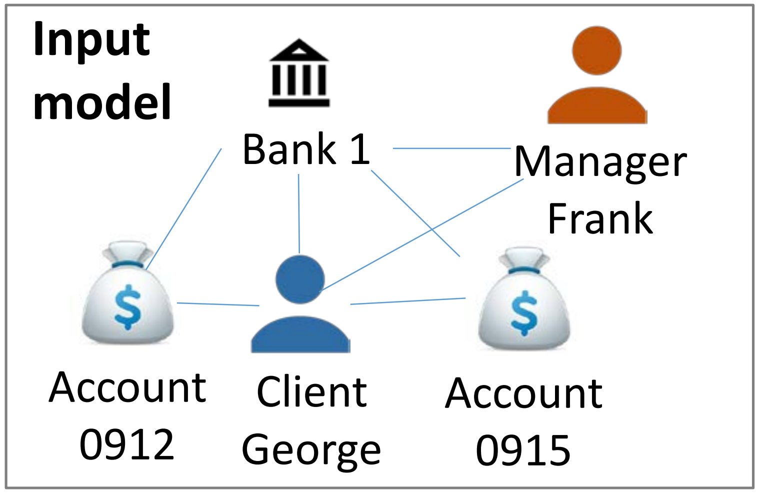
?

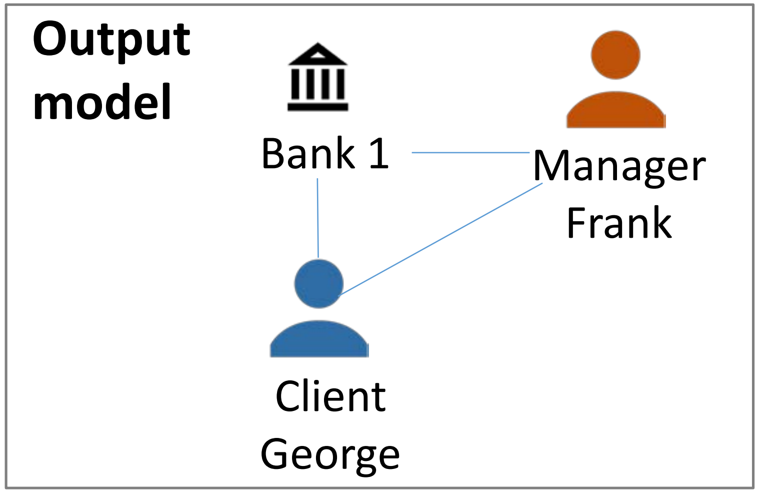
Example 4: Batch-delete accounts

Example application of rule



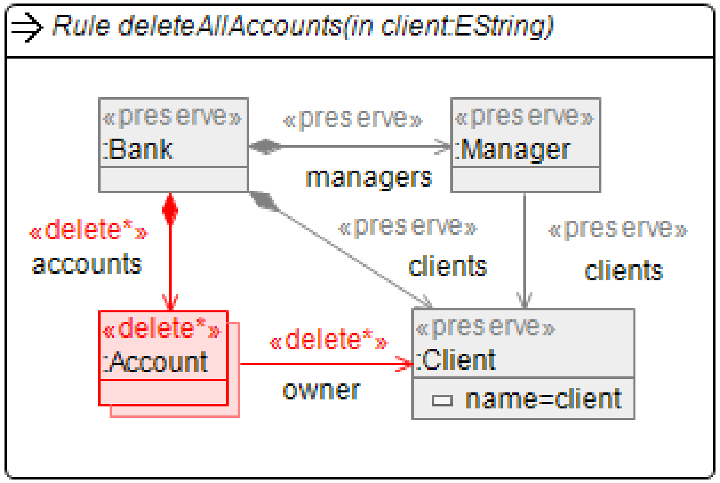
with parameter
client = **“George“**



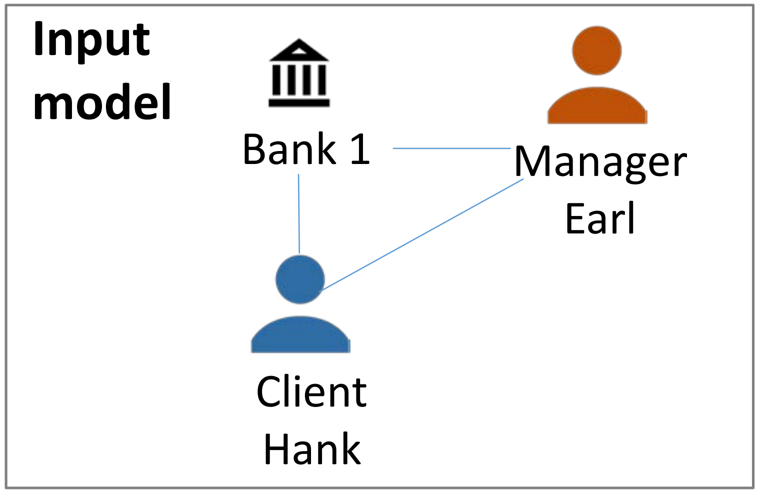


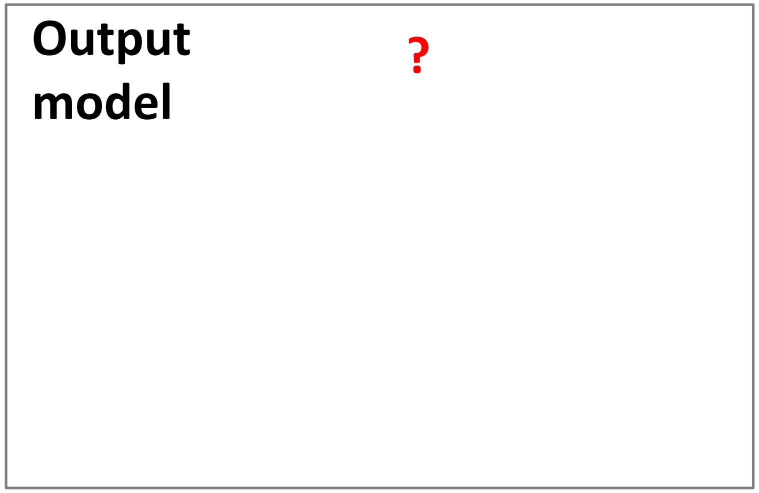
Example 4: Batch-delete accounts

Example application of rule



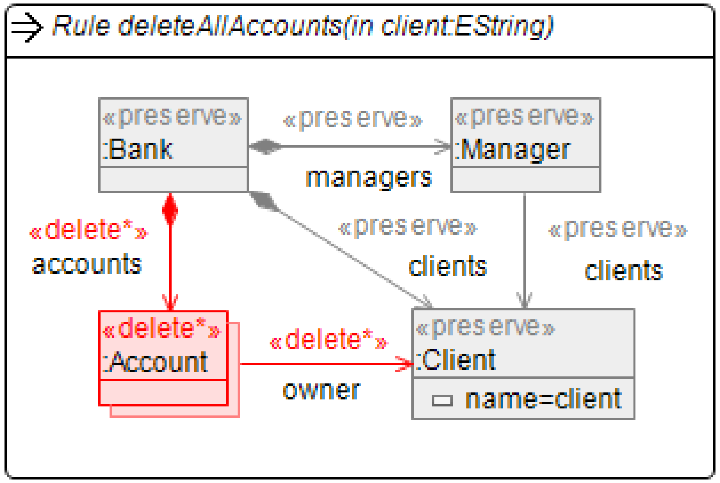
with parameter
client = **“Hank“**



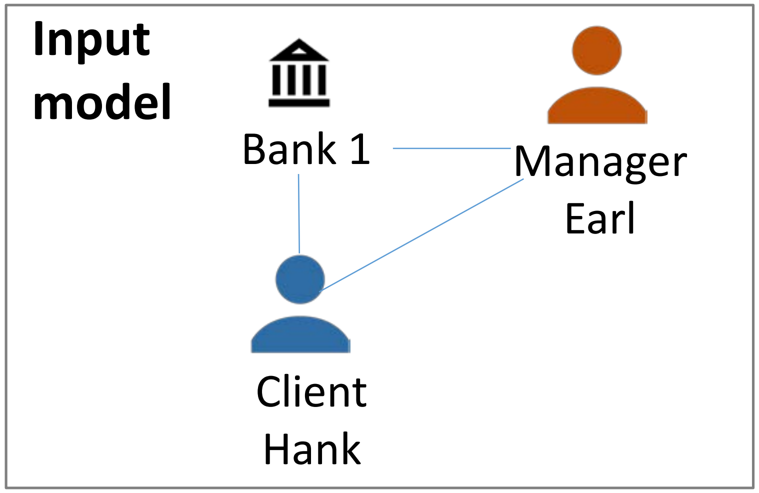


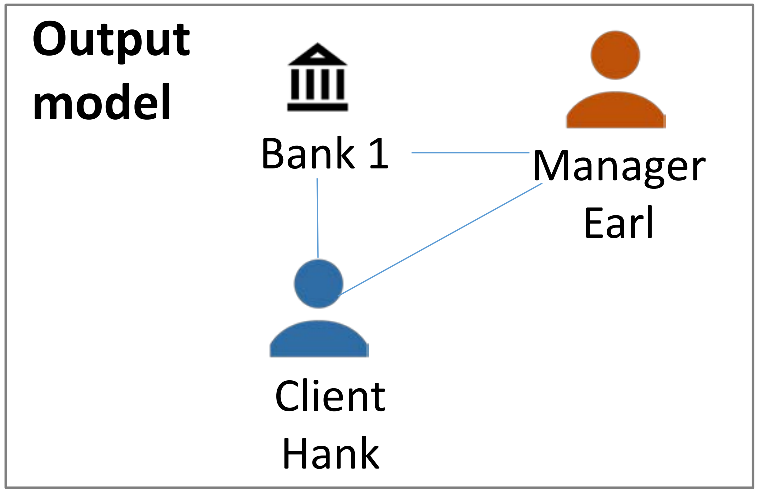
Example 4: Batch-delete accounts

Example application of rule



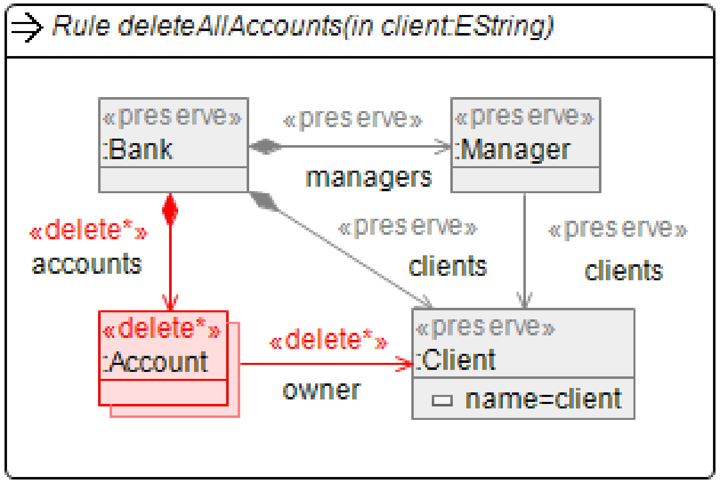
with parameter
client = **“Hank“**





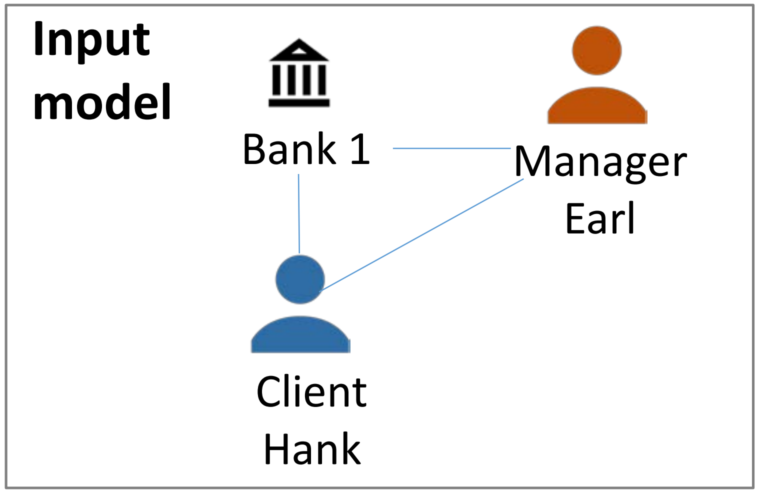
Example 4: Batch-delete accounts

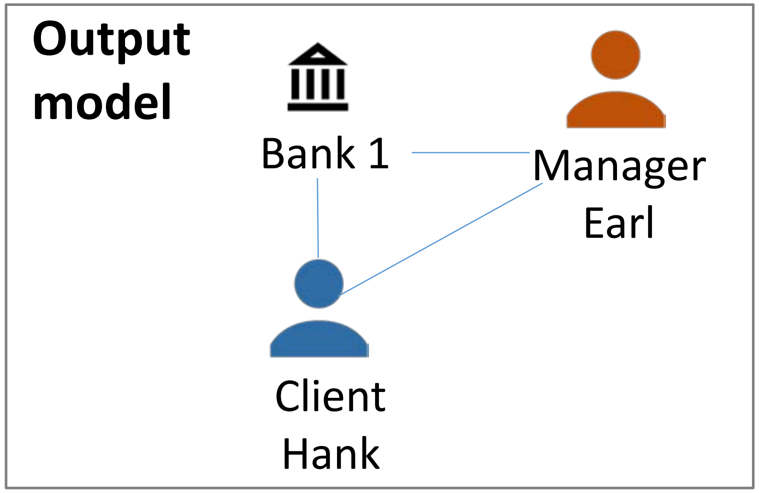
Example application of rule



Semantics:

1. apply kernel rule once
2. apply multi-rule **as often as possible** at the given place



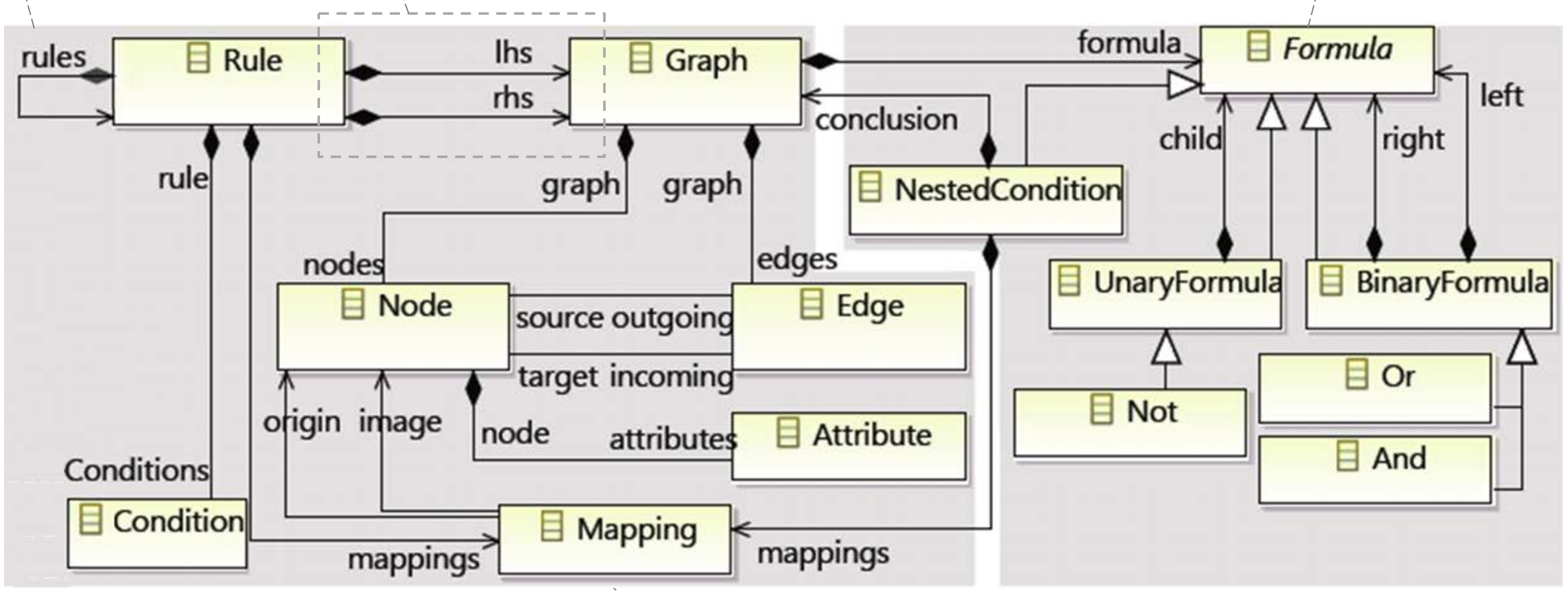


Language definition: meta-model excerpt

Multi-rules

LHS and RHS

Can link negative and positive application conditions using boolean formulas



mark identity of nodes in different graphs (like preserve nodes in LHS and RHS)

Language definition: illustration

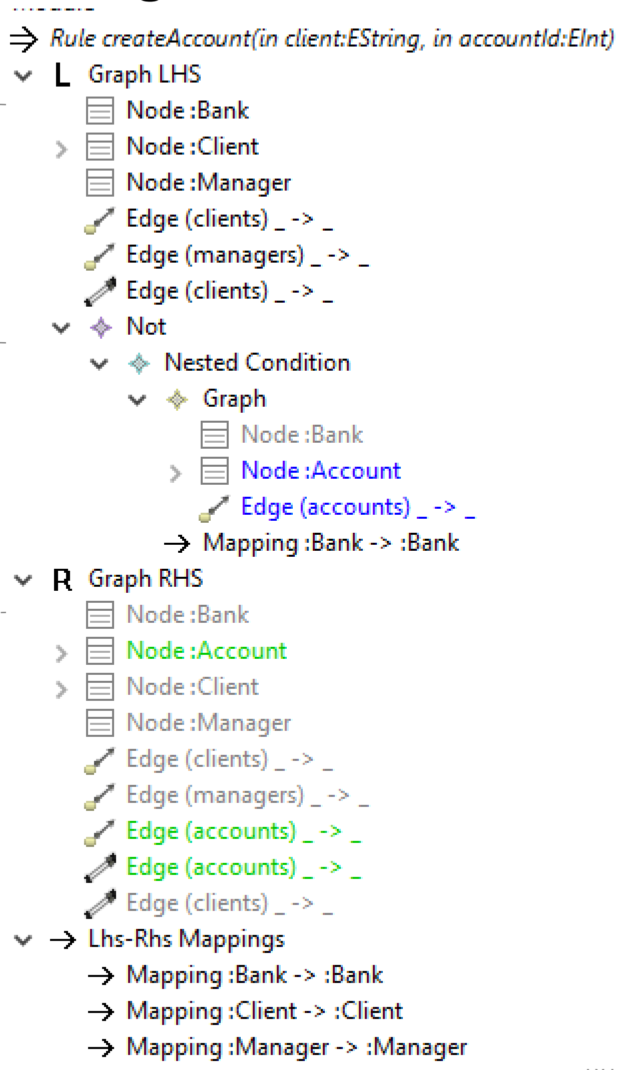
Abstract syntax: Based on left-hand side and right-hand side

Left-hand side (LHS):
Deleted + Preserved elements

Negative application condition

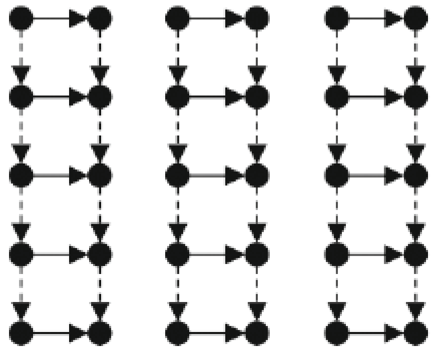
Right-hand side (RHS):
Created + Preserved elements

Mappings



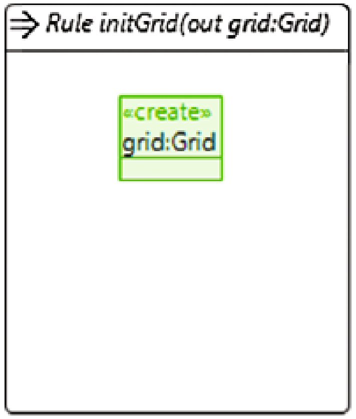
```
Rule createAccount(in client:EString, in accountId:EInt)
  Graph LHS
    Node :Bank
    Node :Client
    Node :Manager
    Edge (clients) _ -> _
    Edge (managers) _ -> _
    Edge (clients) _ -> _
    Not
      Nested Condition
        Graph
          Node :Bank
          Node :Account
          Edge (accounts) _ -> _
        Mapping :Bank -> :Bank
  Graph RHS
    Node :Bank
    Node :Account
    Node :Client
    Node :Manager
    Edge (clients) _ -> _
    Edge (managers) _ -> _
    Edge (accounts) _ -> _
    Edge (accounts) _ -> _
    Edge (clients) _ -> _
  Lhs-Rhs Mappings
    Mapping :Bank -> :Bank
    Mapping :Client -> :Client
    Mapping :Manager -> :Manager
```

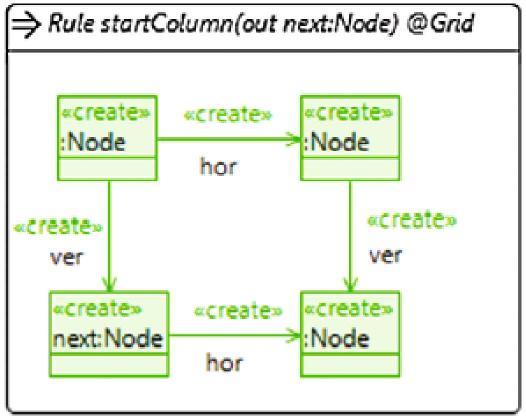
Control flow in transformations

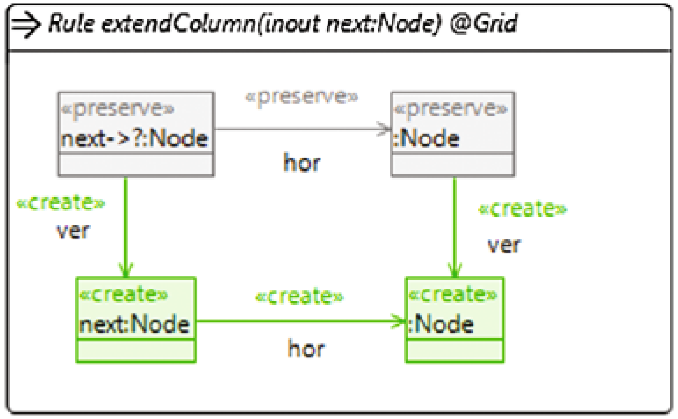


Task: build a sparse grid
[Varró et al. 2005]

Three rules for extending the grid

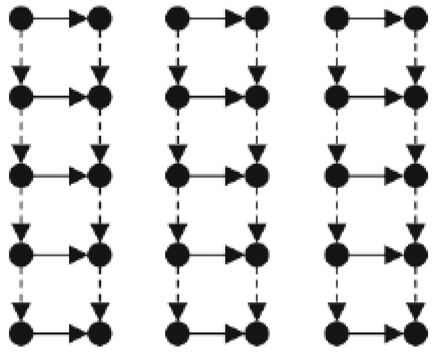






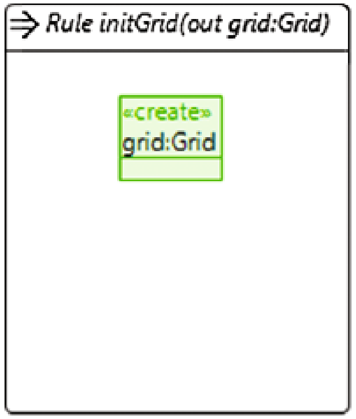
@Grid = additional container node

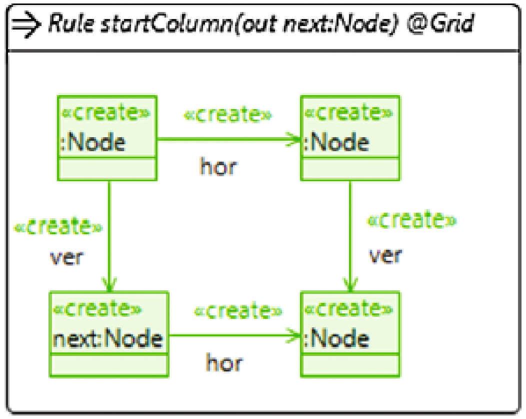
Control flow in transformations

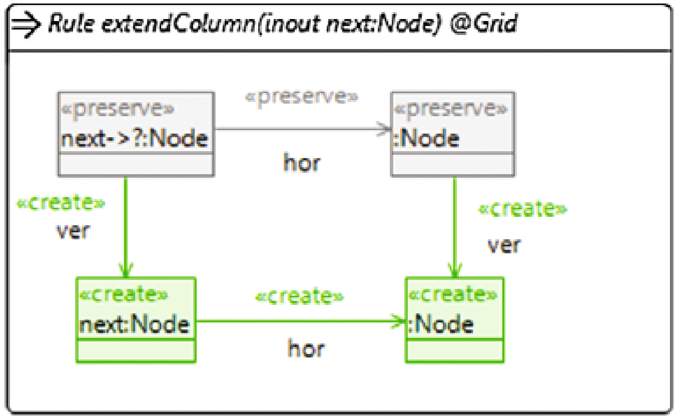


Task: build a sparse grid
[Varró et al. 2005]

Three rules for extending the grid





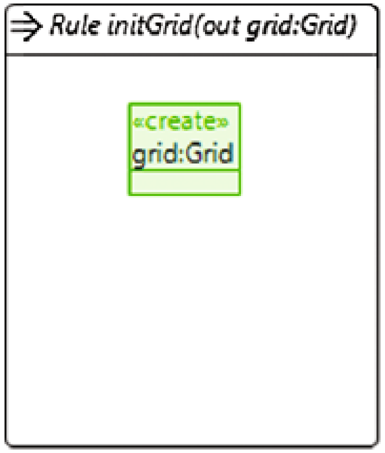


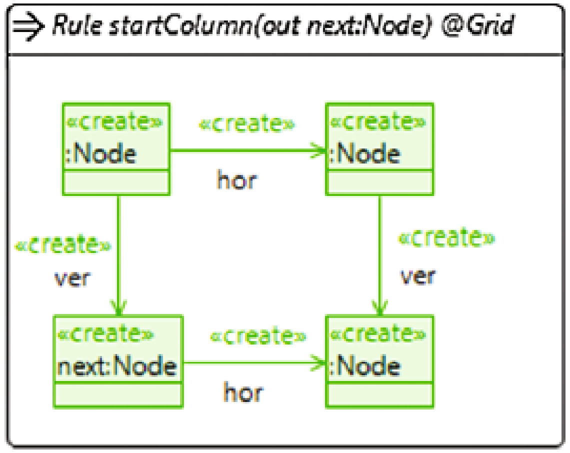
@Grid = additional container node

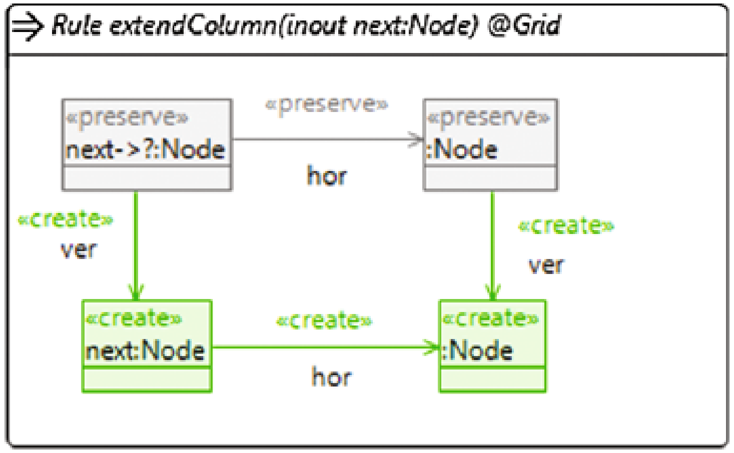
But, how to orchestrate the rules?

Control flow in transformations: units

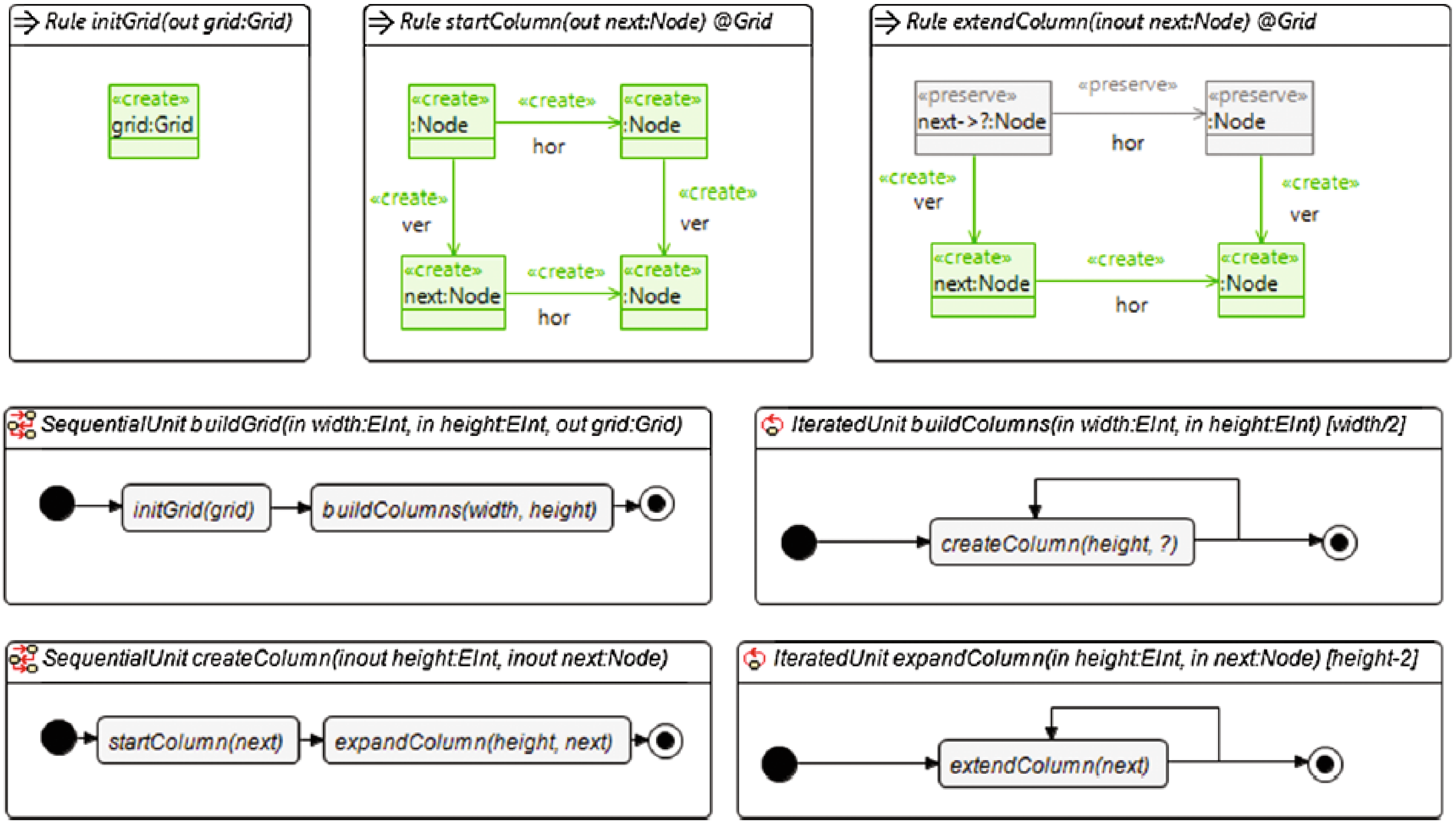
Control flow in transformations: units



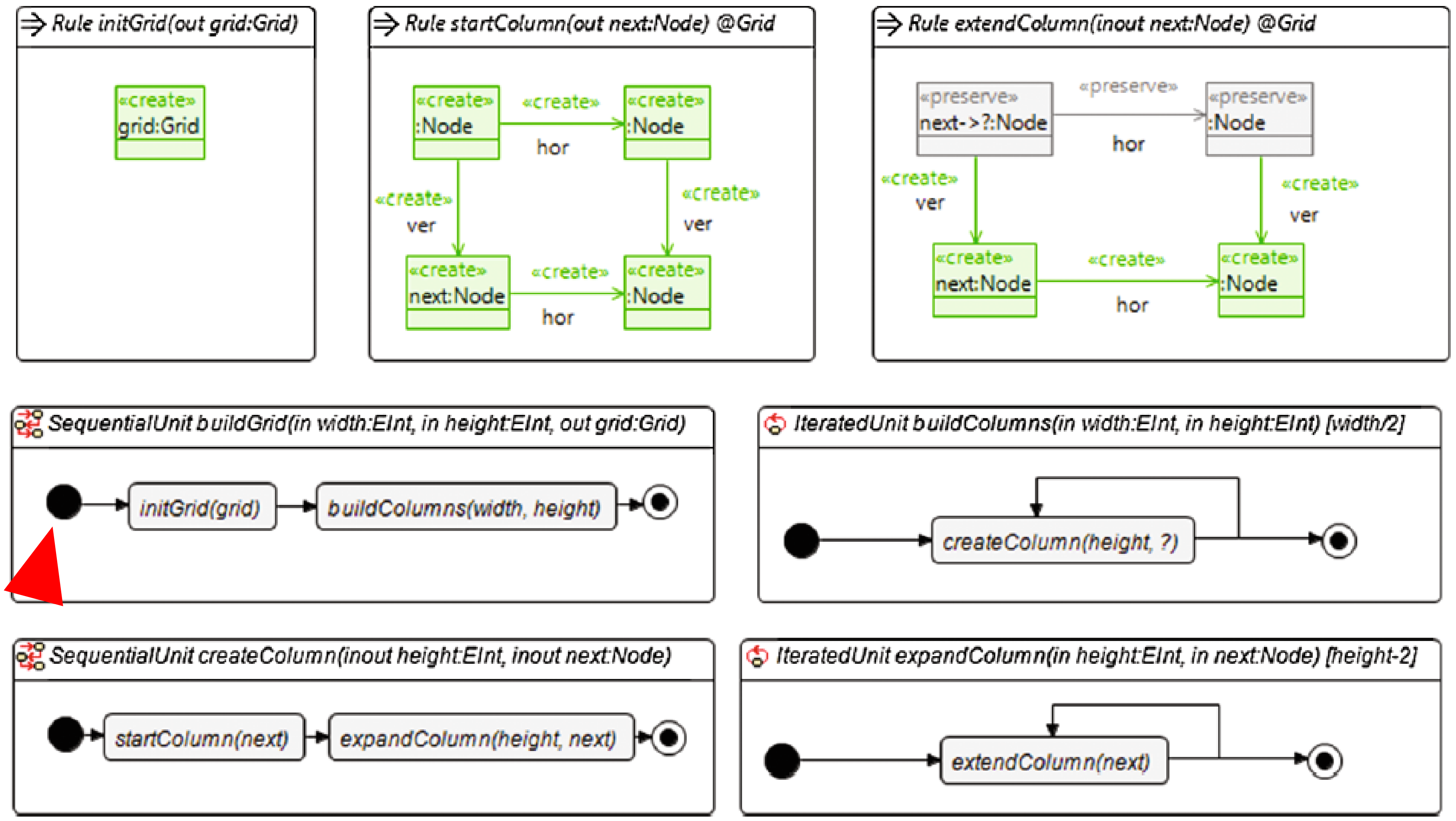




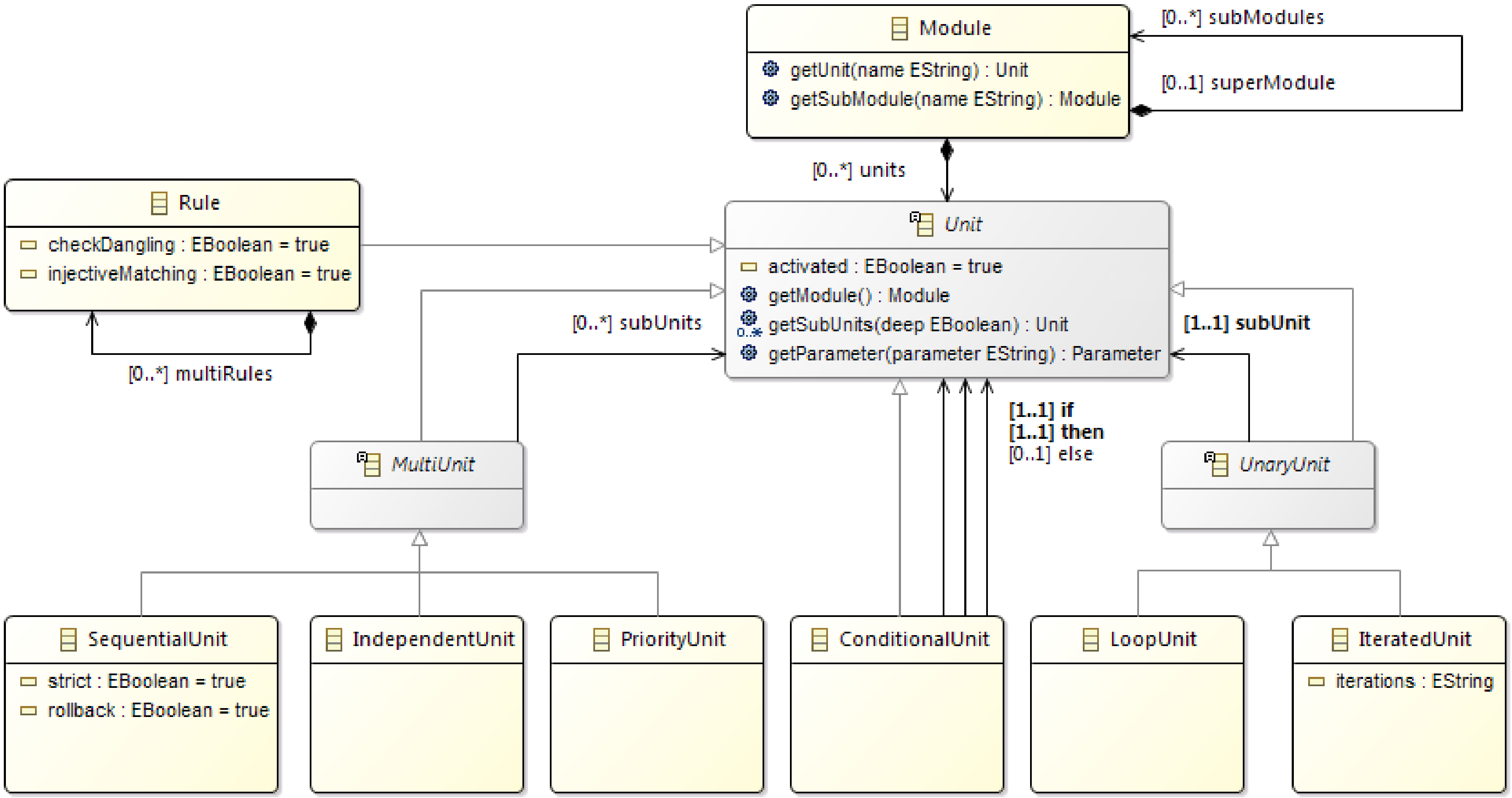
Control flow in transformations: units


Rule initGrid(out grid:Grid)
«create» grid:Grid
Rule startColumn(out next:Node) @Grid
«create» :Node
«create» hor
«create» :Node
«create» ver
«create» ver
«create» next:Node
«create» hor
«create» :Node
Rule extendColumn(inout next:Node) @Grid
«preserve» next->?:Node
«preserve» hor
«preserve» :Node
«create» ver
«create» ver
«create» next:Node
«create» hor
«create» :Node
SequentialUnit buildGrid(in width:EInt, in height:EInt, out grid:Grid)
initGrid(grid)
buildColumns(width, height)
IteratedUnit buildColumns(in width:EInt, in height:EInt) [width/2]
createColumn(height, ?)
SequentialUnit createColumn(inout height:EInt, inout next:Node)
startColumn(next)
expandColumn(height, next)
IteratedUnit expandColumn(in height:EInt, in next:Node) [height-2]
extendColumn(next)

Control flow in transformations: units


Rule initGrid(out grid:Grid)
«create» grid:Grid
Rule startColumn(out next:Node) @Grid
«create» :Node
«create» hor
«create» :Node
«create» ver
«create» ver
«create» next:Node
«create» hor
«create» :Node
Rule extendColumn(inout next:Node) @Grid
«preserve» next->?:Node
«preserve» hor
«preserve» :Node
«create» ver
«create» ver
«create» next:Node
«create» hor
«create» :Node
SequentialUnit buildGrid(in width:EInt, in height:EInt, out grid:Grid)
initGrid(grid)
buildColumns(width, height)
IteratedUnit buildColumns(in width:EInt, in height:EInt) [width/2]
createColumn(height, ?)
SequentialUnit createColumn(inout height:EInt, inout next:Node)
startColumn(next)
expandColumn(height, next)
IteratedUnit expandColumn(in height:EInt, in next:Node) [height-2]
extendColumn(next)

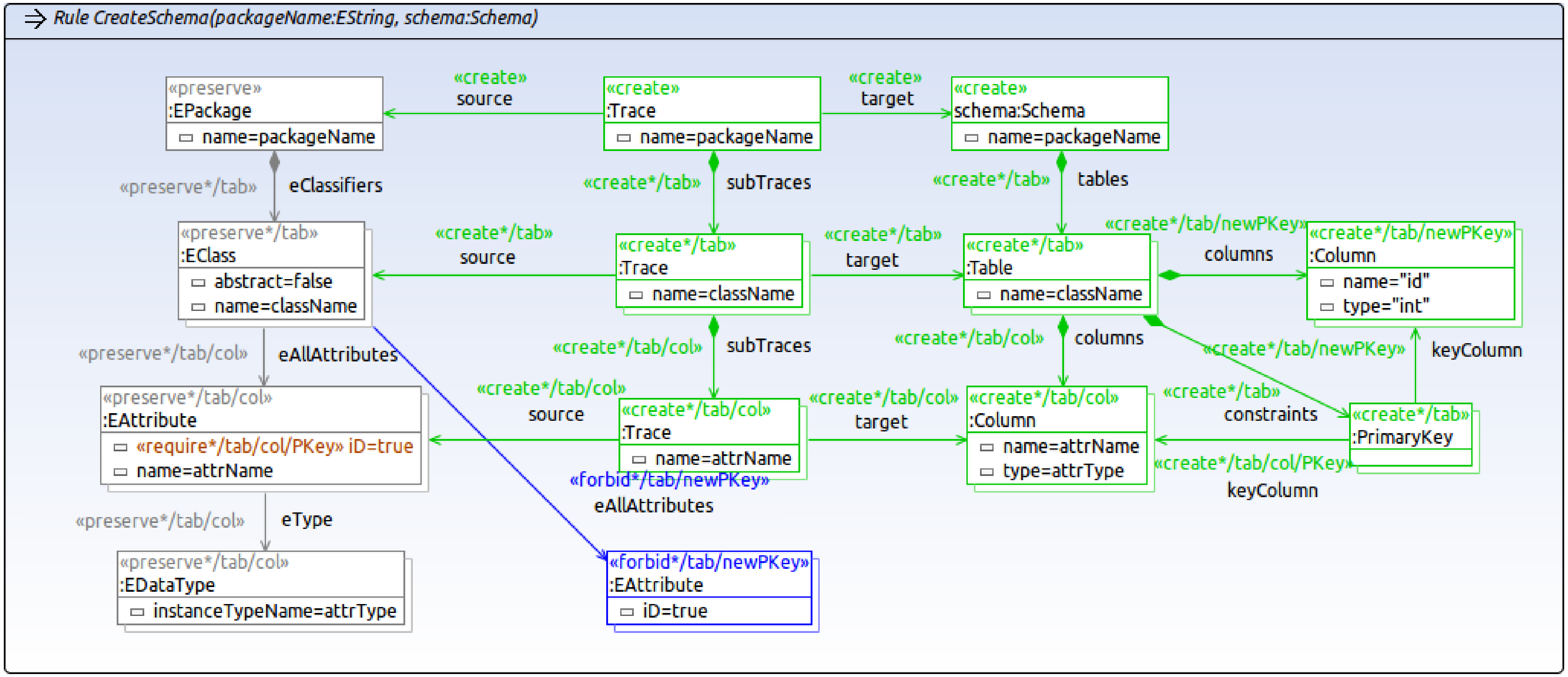
Meta-model excerpt: Units



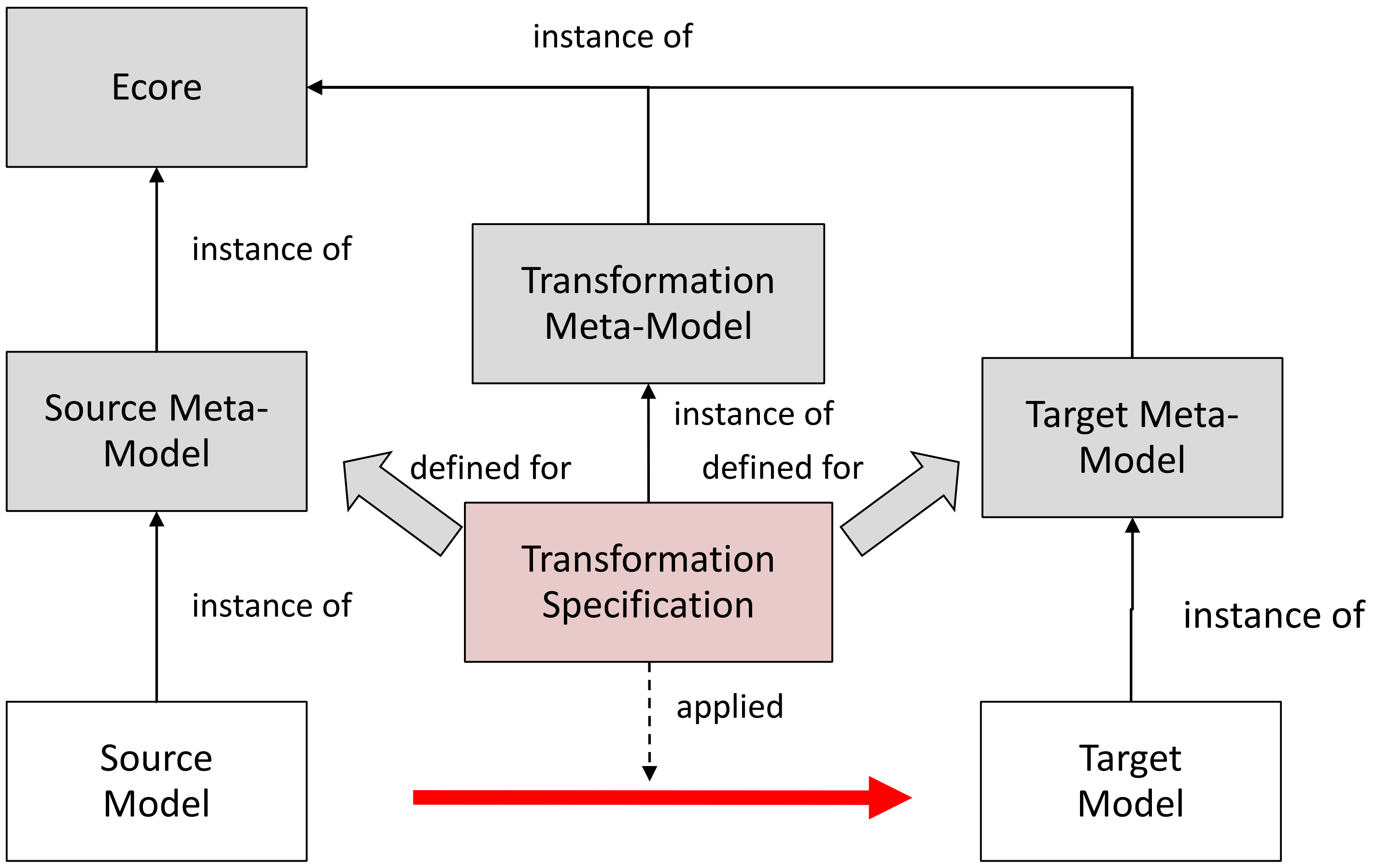
Exogenous transformations (model translation)

Metamodel → Relational database schema

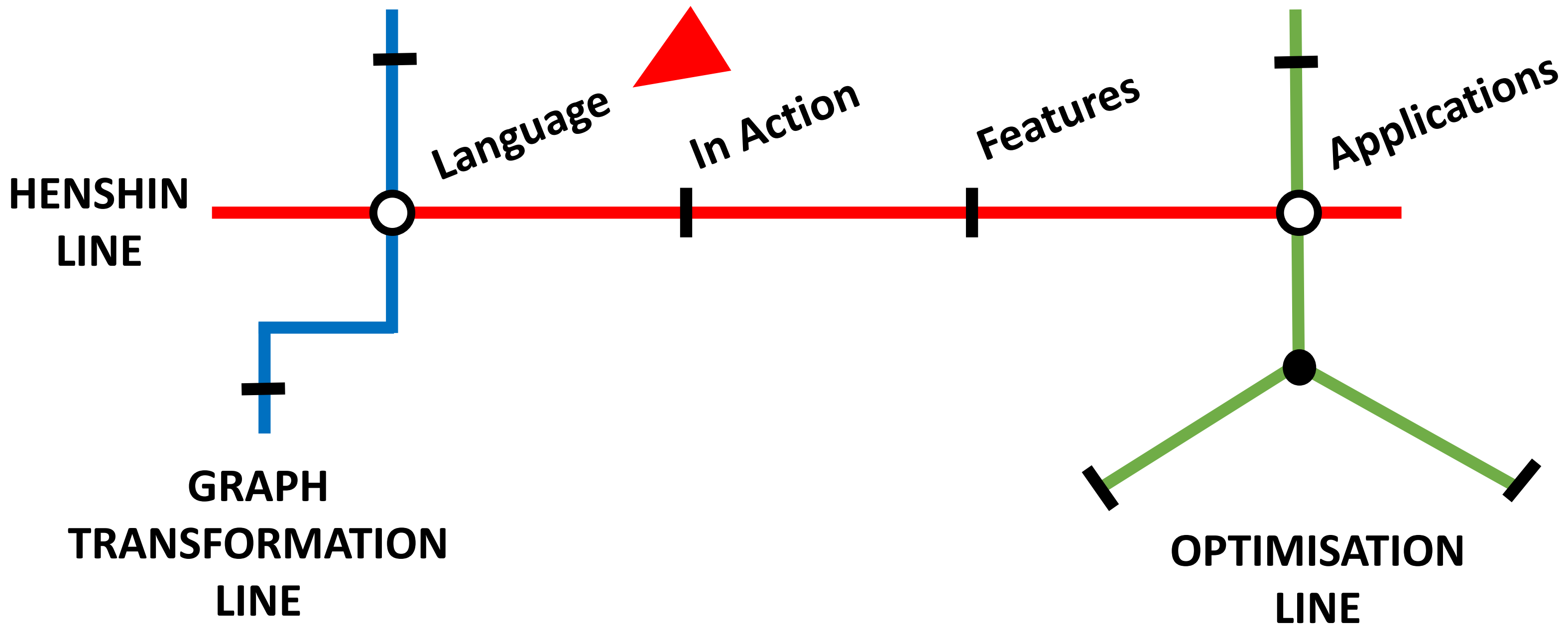
Henshin Trace meta-model
- Establishes traceability
- Supports containment of traces


Rule CreateSchema(packageName:EString, schema:Schema)
«preserve» :EPackage name=packageName
«create» source
«create» :Trace name=packageName
«create» target
«create» schema:Schema name=packageName
«preserve*/tab» eClassifiers
«create*/tab» subTraces
«create*/tab» tables
«preserve*/tab» :EClass abstract=false name=className
«create*/tab» source
«create*/tab» :Trace name=className
«create*/tab» target
«create*/tab» :Table name=className
«create*/tab/newPKey» columns
«create*/tab/newPKey» :Column name="id" type="int"
«preserve*/tab/col» eAllAttributes
«create*/tab/col» subTraces
«create*/tab/col» columns
«create*/tab/newPKey» keyColumn
«preserve*/tab/col» :EAttribute «require*/tab/col/PKey» iD=true name=attrName
«create*/tab/col» source
«create*/tab/col» :Trace name=attrName
«create*/tab/col» target
«create*/tab/col» :Column name=attrName type=attrType
«create*/tab» constraints
«create*/tab» :PrimaryKey
«create*/tab/col/PKey» keyColumn
«forbid*/tab/newPKey» eAllAttributes
«preserve*/tab/col» eType
«preserve*/tab/col» :EDataType instanceTypeName=attrType
«forbid*/tab/newPKey» :EAttribute iD=true

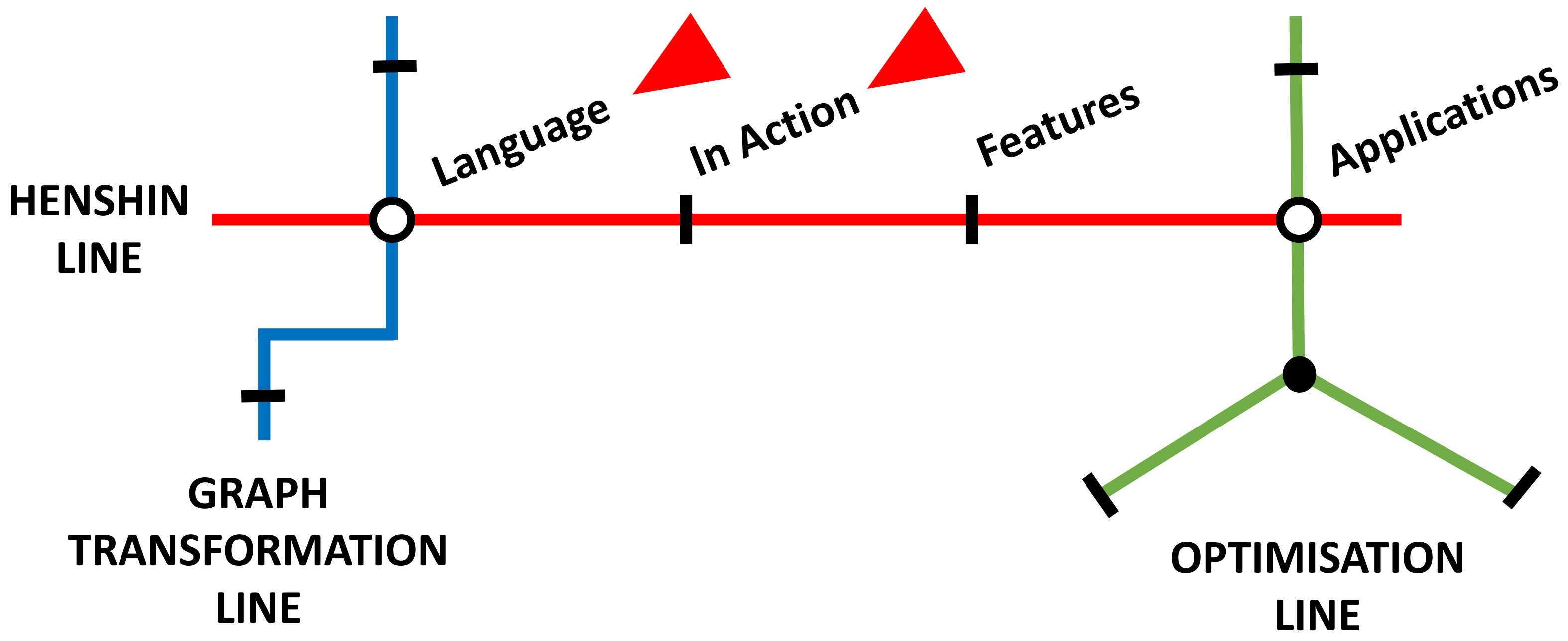
Big picture: Model transformations based on the Eclipse Modeling Framework (EMF)



Henshin: A Guided Tour



Henshin: A Guided Tour



Henshin in action

1. Import project
2. View rules
3. Execute rules with the Interpreter Wizard
4. Execute rules from Java, using interpreter API
5. Roll your own rule

Henshin in action

1. **Import project**

2. View rules

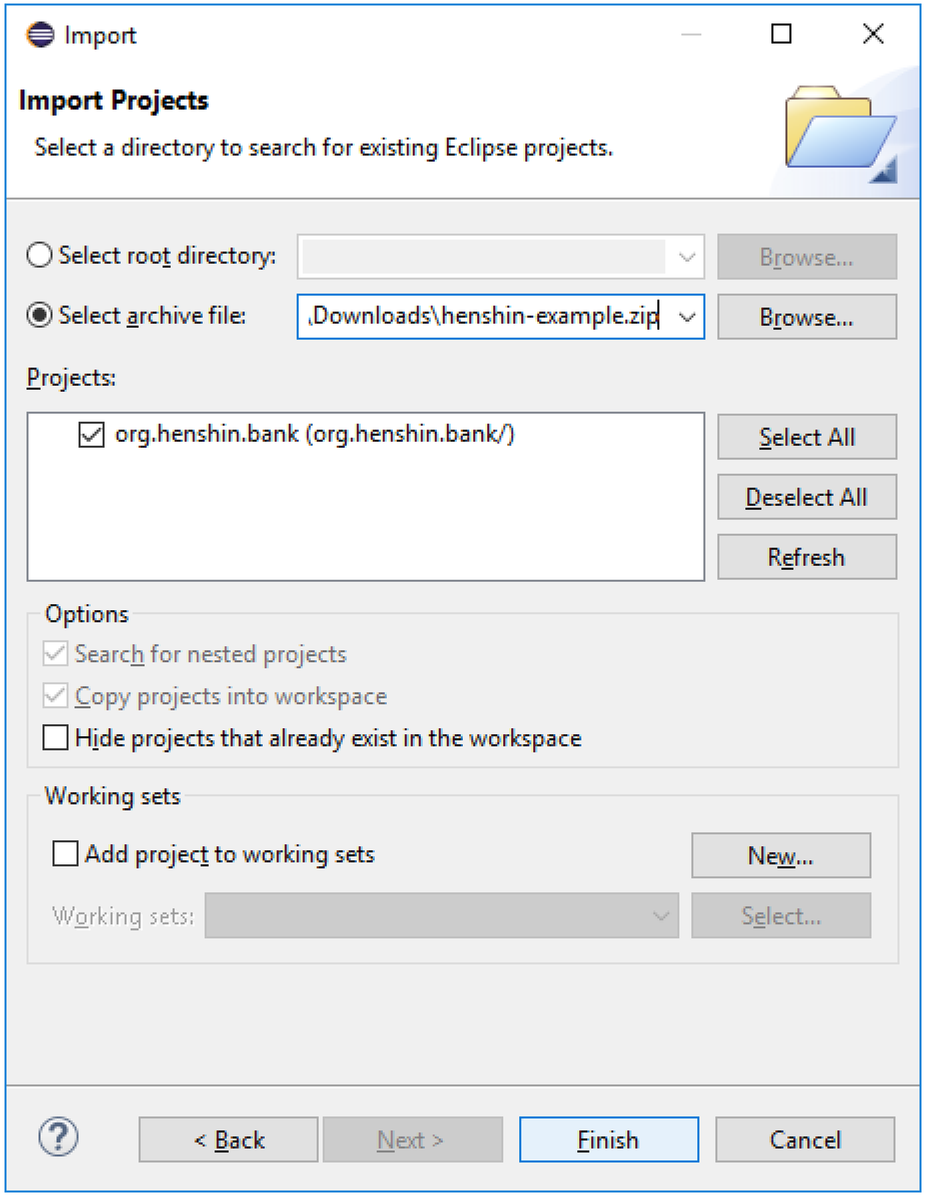
3. Execute rules with the Interpreter Wizard

4. Execute rules from Java, using interpreter API

5. Roll your own rule

Import project

- In Eclipse, do *File* → *Import…* → *General* → *Existing Projects Into Workspace* → *Next*
- Do *Select Archive File* → Choose **henshin-example.zip**
- The dialog should now look like the image to the right
- Click *Finish*
- The project *org.henshin.bank* should appear in the Package Explorer


Import

Import Projects

Select a directory to search for existing Eclipse projects.

( ) Select root directory: [ ] Browse...

(●) Select archive file: .Downloads\henshin-example.zip Browse...

Projects:

☑ org.henshin.bank (org.henshin.bank/)

Select All | Deselect All | Refresh

Options

☑ Search for nested projects

☑ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets New...

Working sets: [ ] Select...

< Back | Next > | Finish | Cancel

Henshin in action

1. Import project

2. View rules

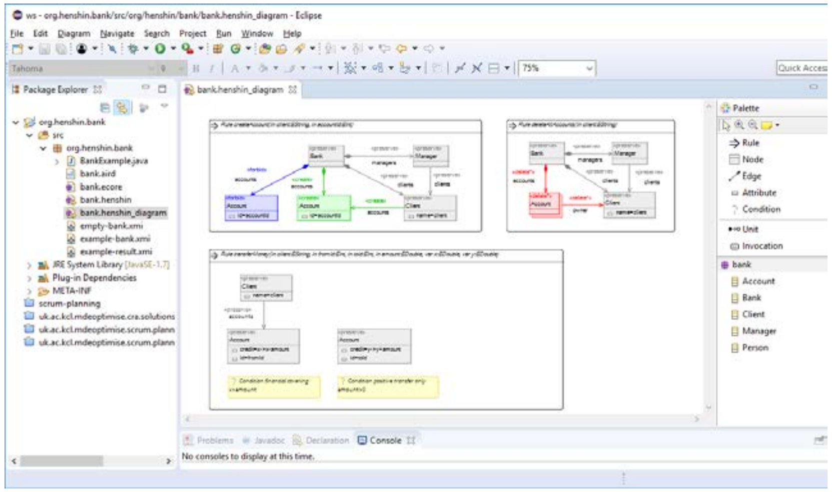
3. Execute rules with the Interpreter Wizard

4. Execute rules from Java, using interpreter API

5. Roll your own rule

View rules (and related files)

- In the Package Explorer, inspect the imported project: Navigate tofolder **src/org.henshin.bank**.
- Have a look at the files, including the meta-model **bank.ecore,** its visualization **bank.aird** and example models like **example-bank.xmi** (without visualization).
- Open **bank.henshin_diagram**. The example rules are now shown in Henshin‘s graphical editor. You can use this editor to modify and edit rules.



No consoles to display at this time.

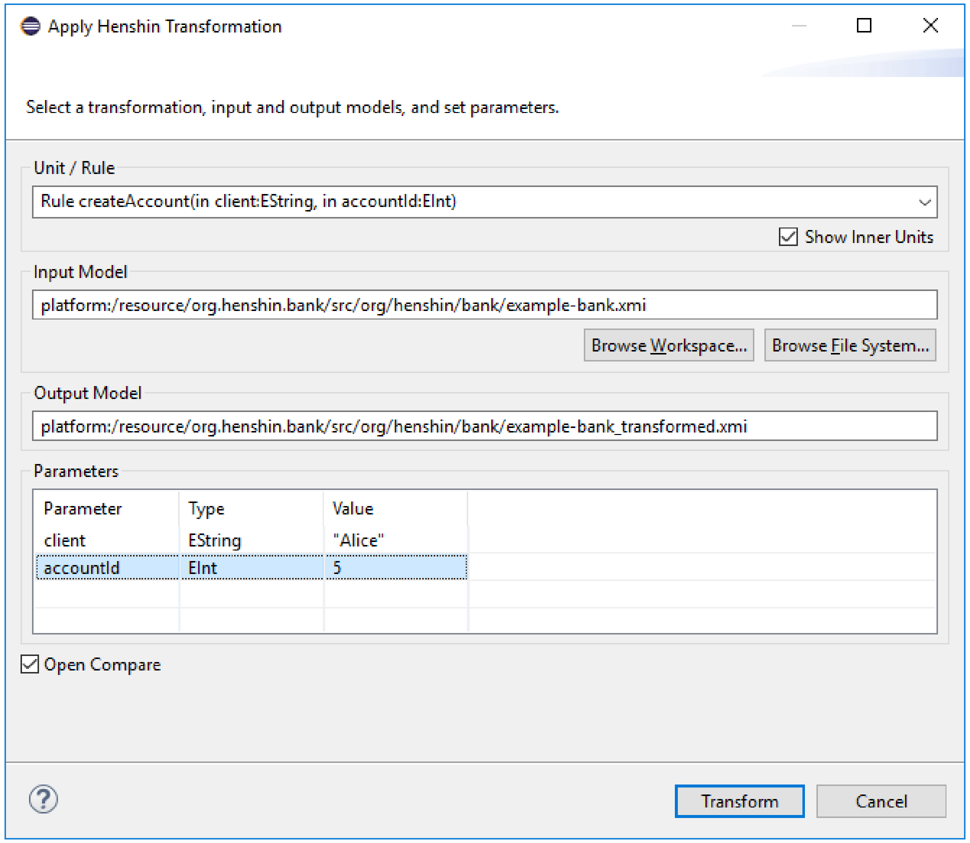
Henshin in action

1. Import project
2. View rules
3. **Execute rules with the Interpreter Wizard**
4. Execute rules from Java, using interpreter API
5. Roll your own rule

Apply rules using the Interpreter Wizard

To apply the rule **createAccount** to the model **example-bank.xmi**:

- In Package Explorer, right-click on *bank.henshin -> Henshin -> Apply transformation*
- In the dialog, use *Browse Workspace…* to select *example-bank.xmi*
- Use the suggested output model, and enter parameter values *"Alice"* and *5* (see figure)
- Click on *Transform*



Select a transformation, input and output models, and set parameters.

Unit / Rule: Rule createAccount(in client:EString, in accountId:EInt)

☑ Show Inner Units

Input Model: platform:/resource/org.henshin.bank/src/org/henshin/bank/example-bank.xmi

Output Model: platform:/resource/org.henshin.bank/src/org/henshin/bank/example-bank_transformed.xmi

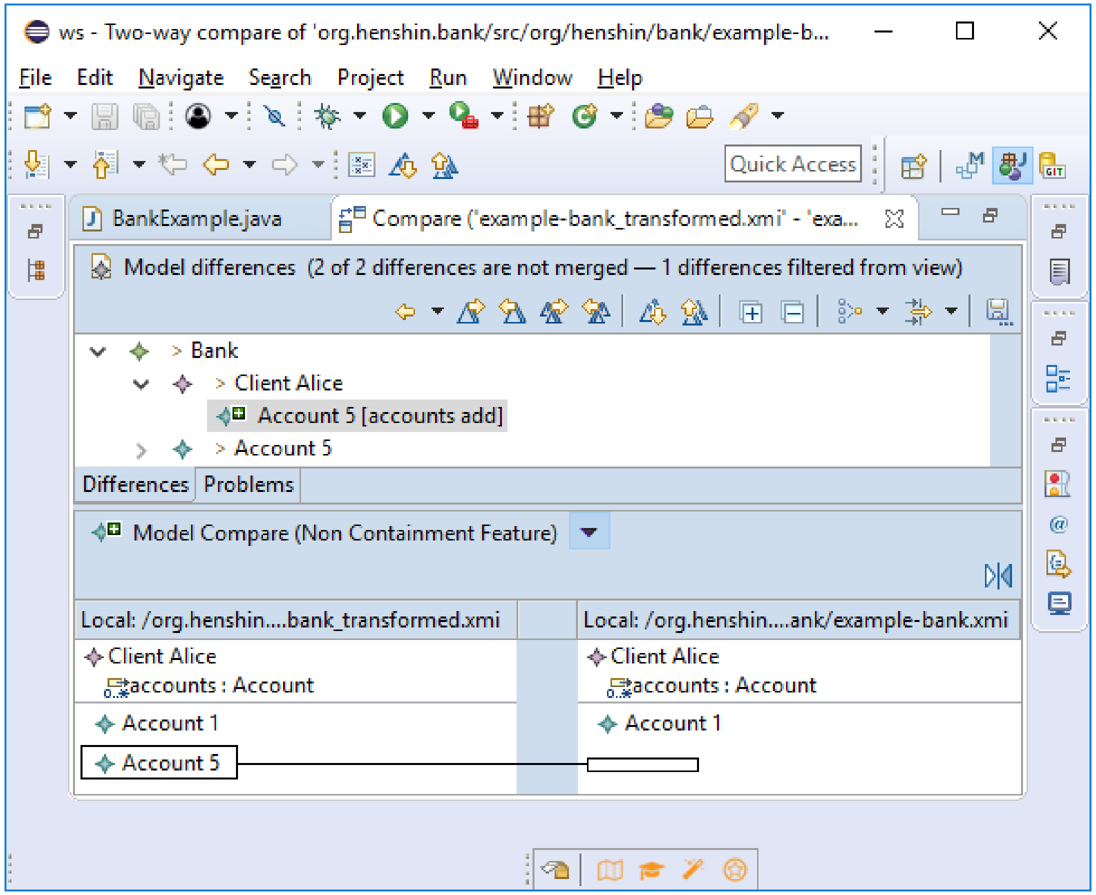
Parameters

Parameter	Type	Value
client	EString	"Alice"
accountId	EInt	5

☑ Open Compare

Apply rules using the Interpreter Wizard

- The result is saved to example-bank_transformed.xmi
- A Compare viewer opens automatically, allowing us to see the changes performed to the model.



ws - Two-way compare of 'org.henshin.bank/src/org/henshin/bank/example-b...

Compare ('example-bank_transformed.xmi' - 'exa...

Model differences (2 of 2 differences are not merged — 1 differences filtered from view)

```
> Bank
    > Client Alice
        Account 5 [accounts add]
    > Account 5
```

Model Compare (Non Containment Feature)

Local: /org.henshin....bank_transformed.xmi	Local: /org.henshin....ank/example-bank.xmi
Client Alice	Client Alice
accounts : Account	accounts : Account
Account 1	Account 1
Account 5	

Henshin in action

1. Import project

2. View rules

3. Execute rules with the Interpreter Wizard

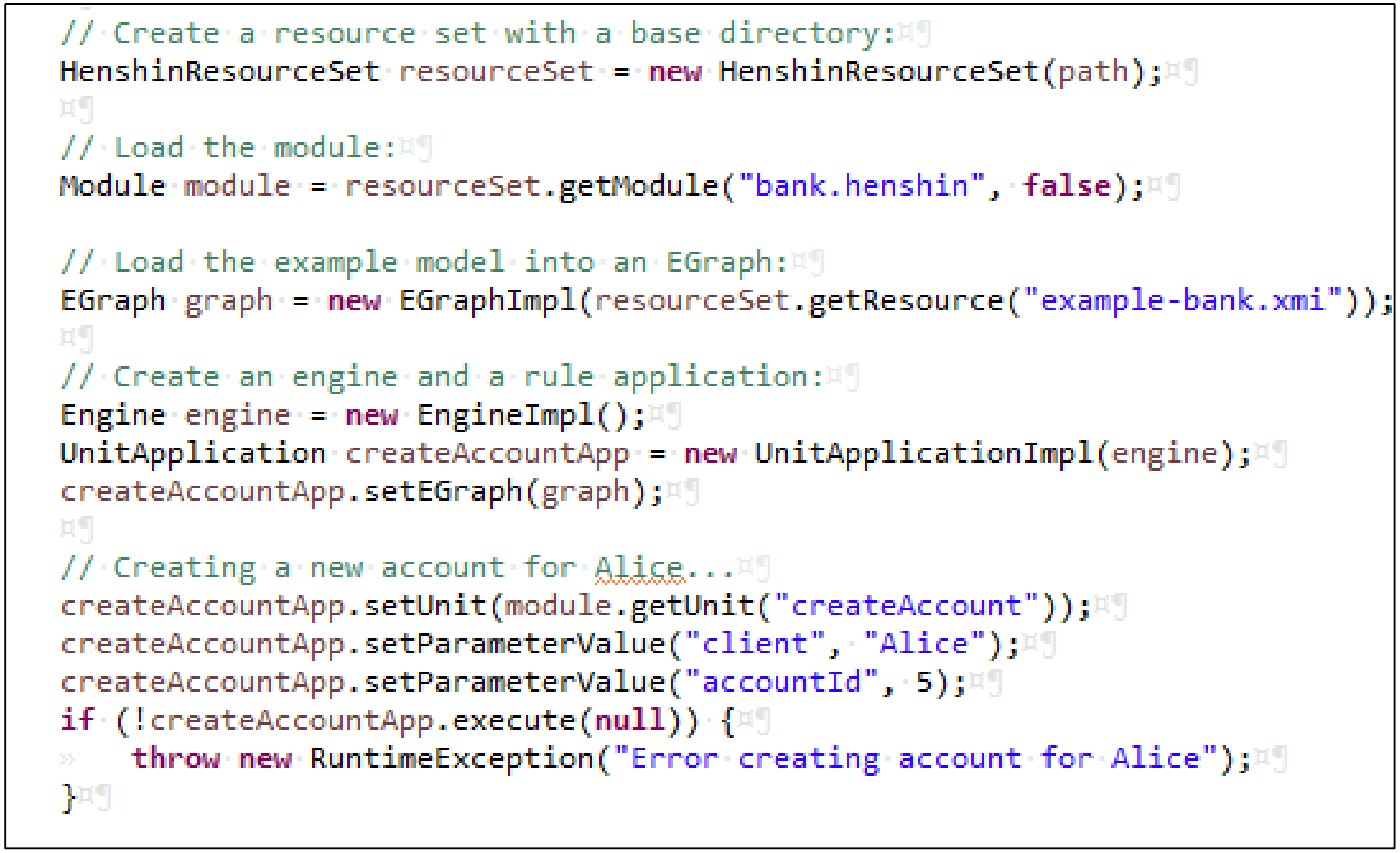
4. Execute rules from Java, using interpreter API

5. Roll your own rule

Execute rules from Java, using Interpreter API

Problem: Want to automate the application of rules - for example, when developing some refactoring tool on top of Henshin

Solution: The Interpreter API. Usage example in *BankExample.java*:



```java
// Create a resource set with a base directory:
HenshinResourceSet resourceSet = new HenshinResourceSet(path);

// Load the module:
Module module = resourceSet.getModule("bank.henshin", false);

// Load the example model into an EGraph:
EGraph graph = new EGraphImpl(resourceSet.getResource("example-bank.xmi"));

// Create an engine and a rule application:
Engine engine = new EngineImpl();
UnitApplication createAccountApp = new UnitApplicationImpl(engine);
createAccountApp.setEGraph(graph);

// Creating a new account for Alice...
createAccountApp.setUnit(module.getUnit("createAccount"));
createAccountApp.setParameterValue("client", "Alice");
createAccountApp.setParameterValue("accountId", 5);
if (!createAccountApp.execute(null)) {
    throw new RuntimeException("Error creating account for Alice");
}
```

Henshin in action

1. Import project
2. View rules
3. Execute rules with the Interpreter Wizard
4. Execute rules from Java, using interpreter API
5. **Roll your own rule**

Roll your own rule

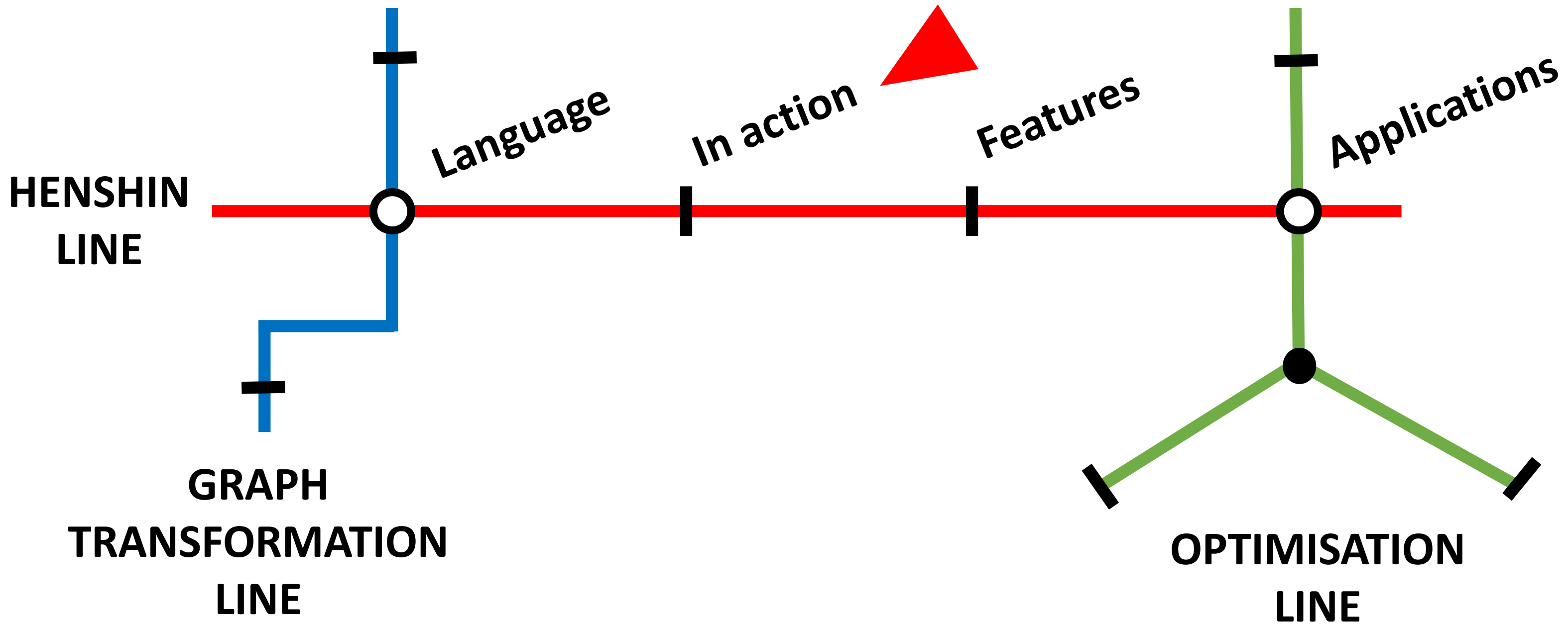
Task 1: *PayLongtimeBonus*: Add 10$ to an account whose ID is lower than 5

Task 2: *FireUnproductiveManager*: Delete from a given bank a manager who is not assigned to any customers

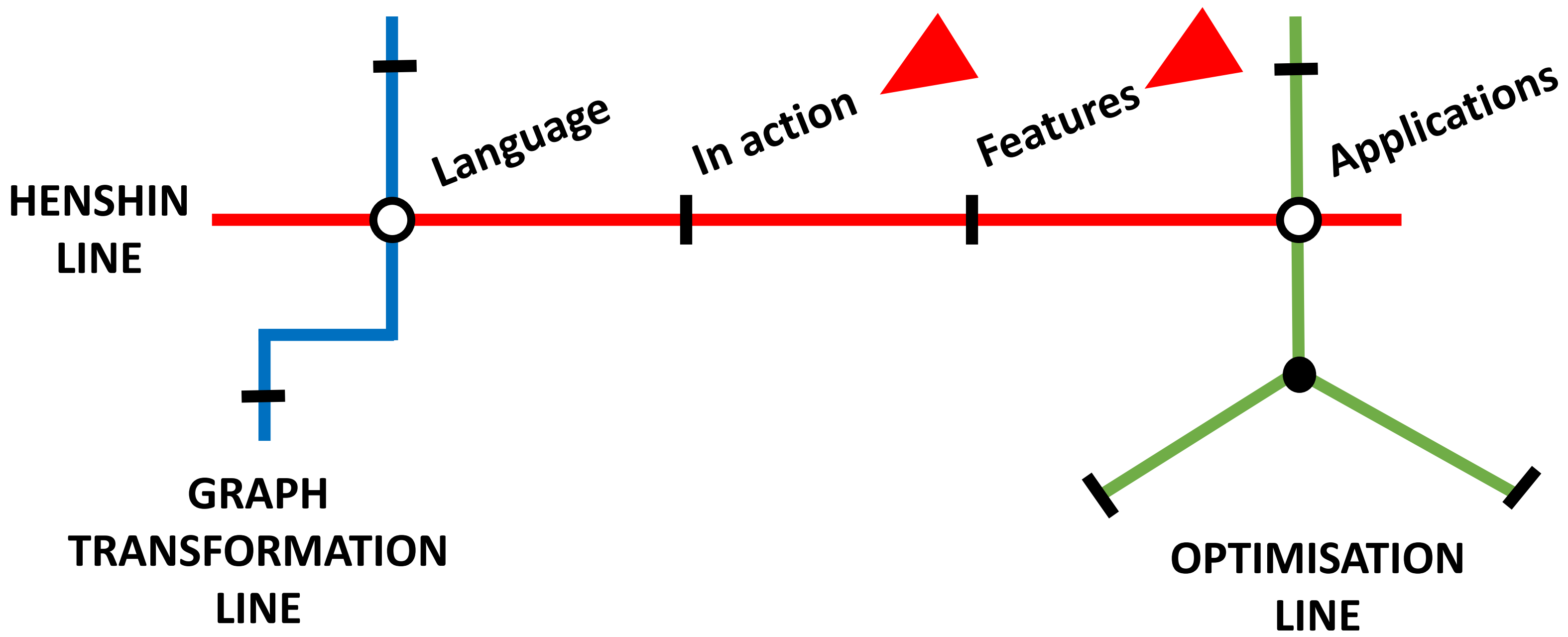
Hints:

- To **create a parameter or variable** in a rule, double-click the rule‘s title bar and change the list after the rule name (in round brackets)
- To **change the action of an element** (e.g. from preserve to delete), double-click on the action in the graphical editor, and type in the new action

Henshin: A Guided Tour

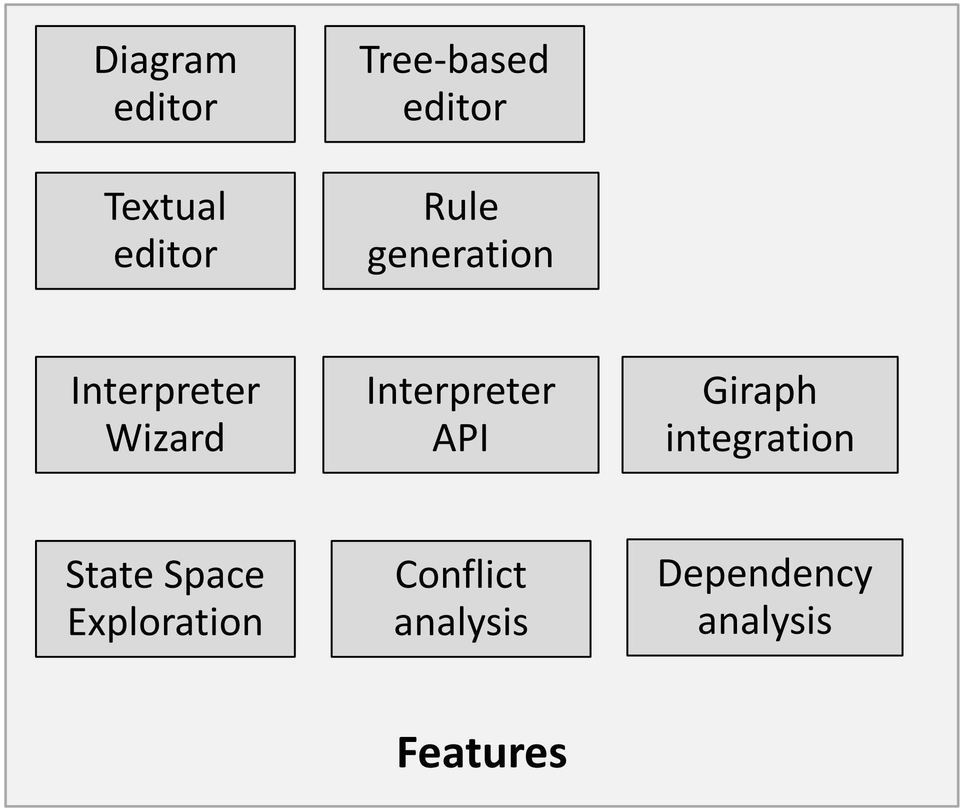


Henshin: A Guided Tour



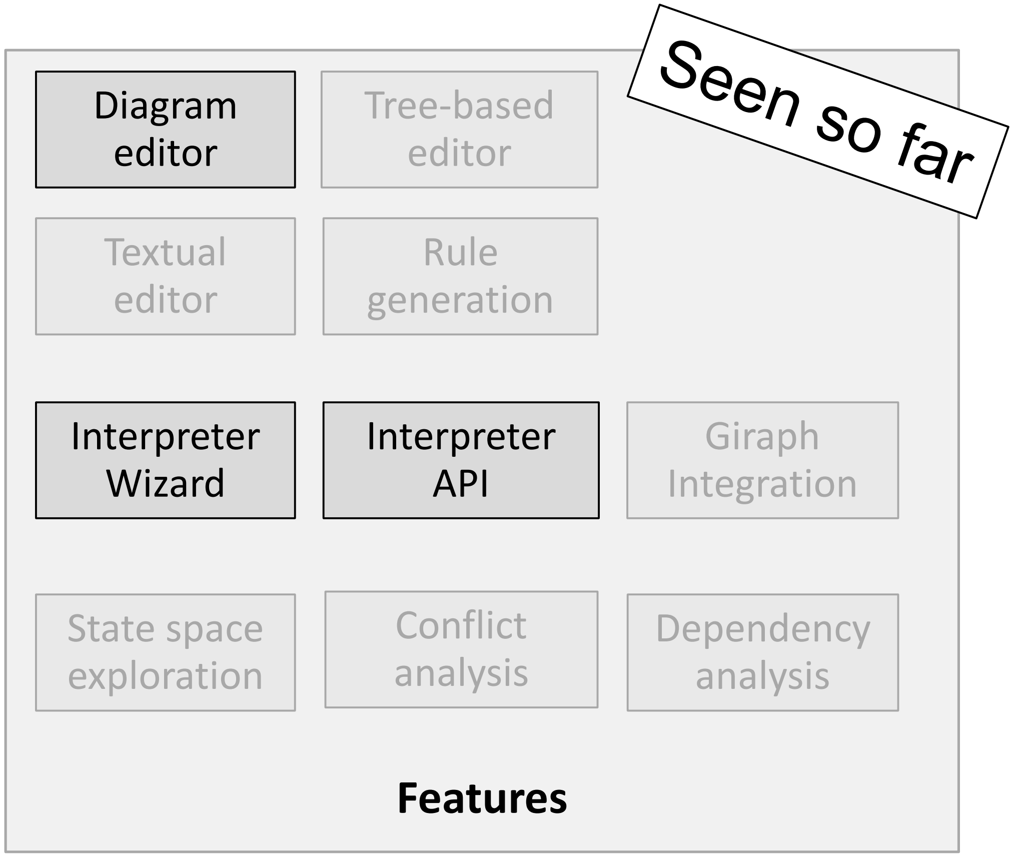
Features: What would you like to do today?

- Define a transformation
- Execute a transformation
- Analyse a transformation



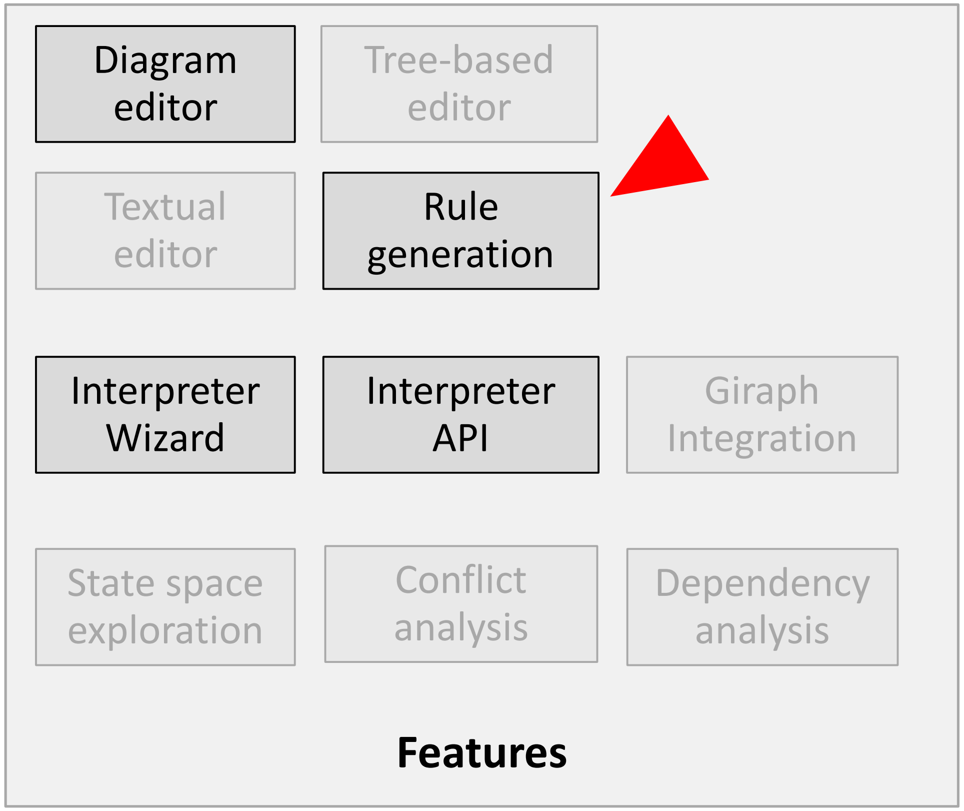
Features: What would you like to do today?

- Define a transformation
- Execute a transformation
- Analyse a transformation



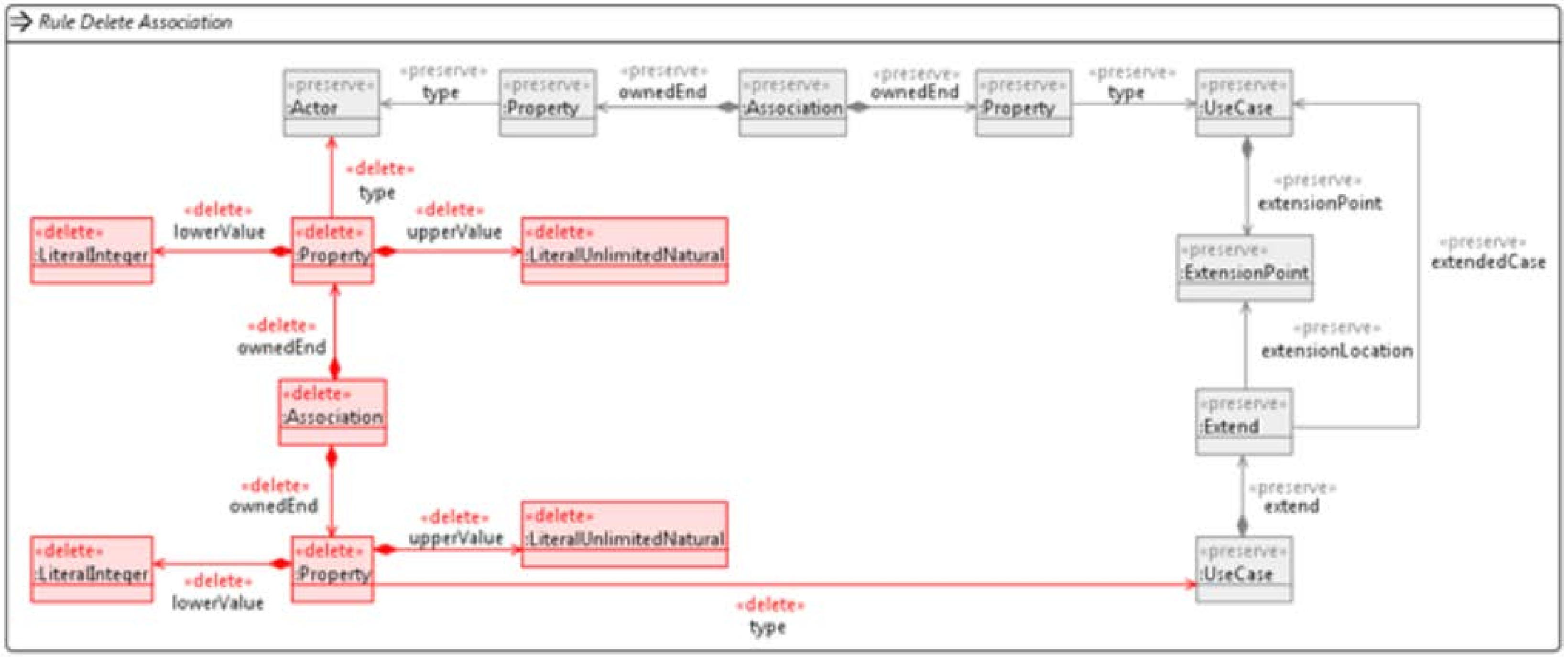
Features: What would you like to do today?

- Define a transformation
- Execute a transformation
- Analyse a transformation

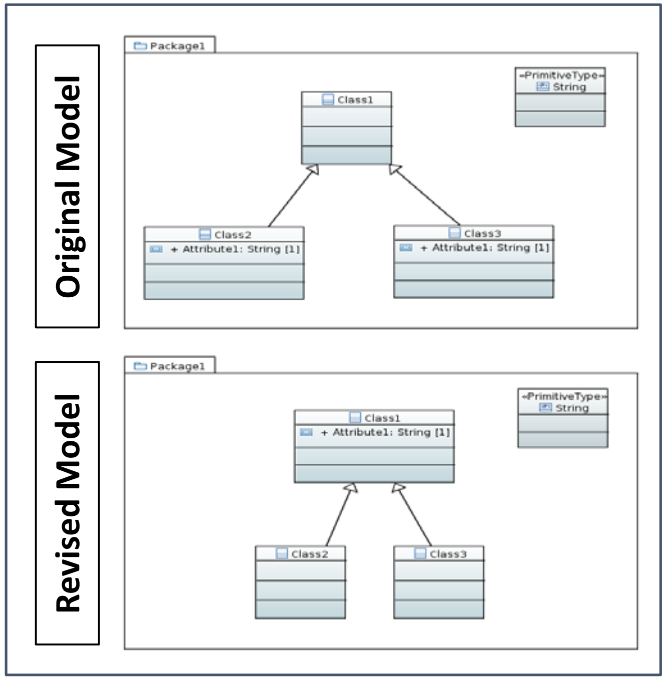


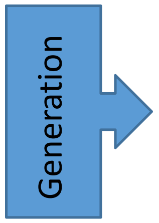
Problem: defining complex rules takes effort

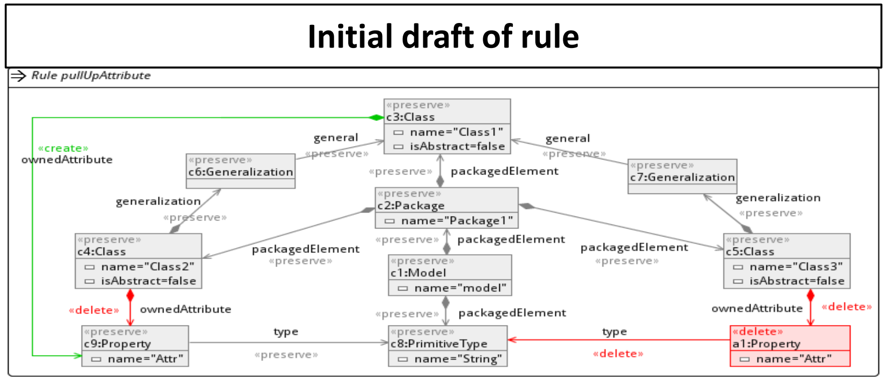
Deleting an association in a UML model



Solution: generate rules from examples



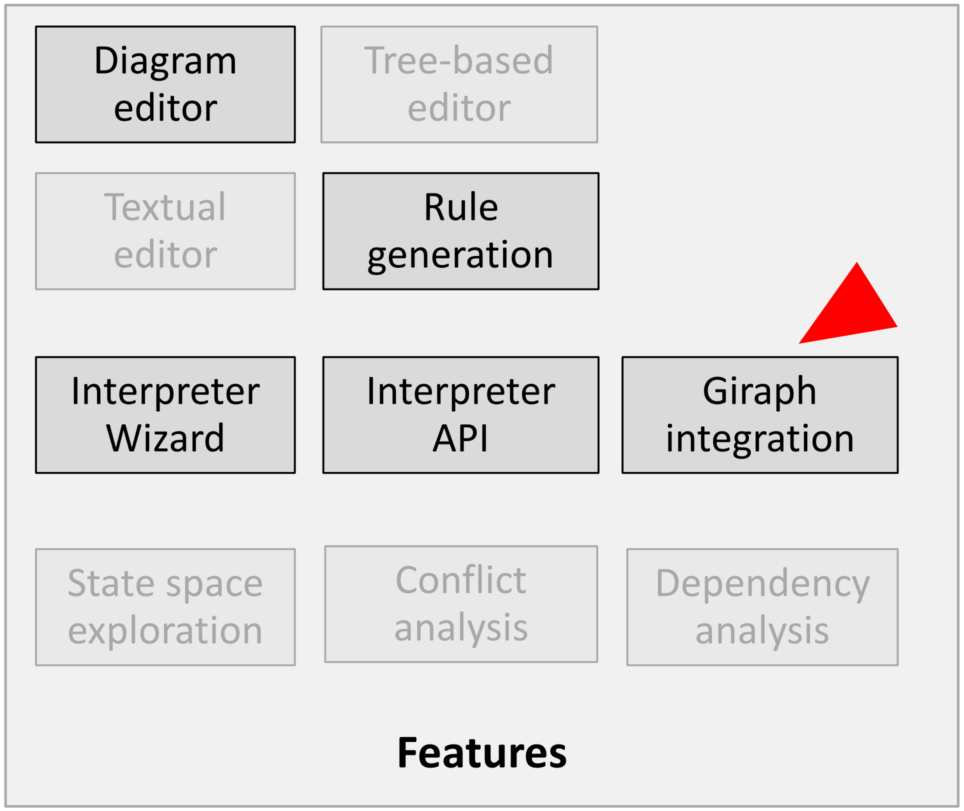




- Use familiar graphical editors to define model pair: original-revised
- Uses model comparison to identify identical elements
- First draft of rule: may need to add parameters, NACs etc.

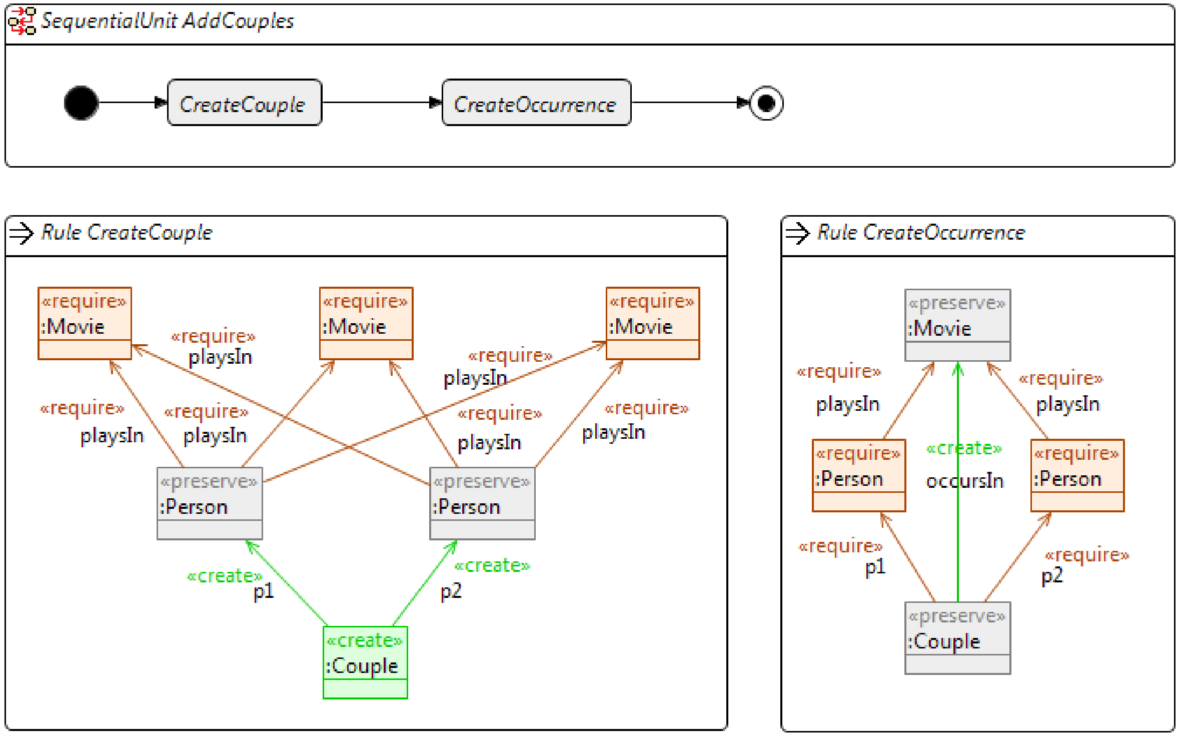
Features: What would you like to do today?

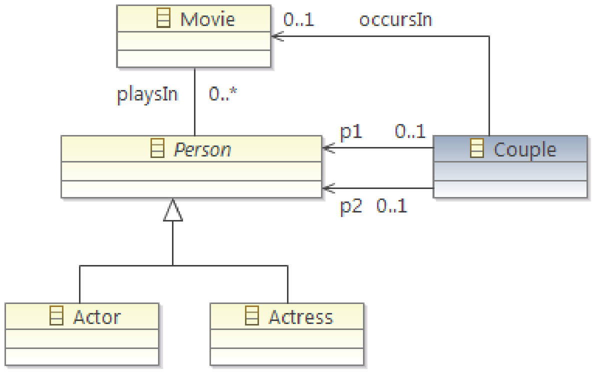
- Define a transformation
- Execute a transformation
- Analyse a transformation



Problem: EMF does not scale to large models

Solution: Massive parallel model transformation with Giraph



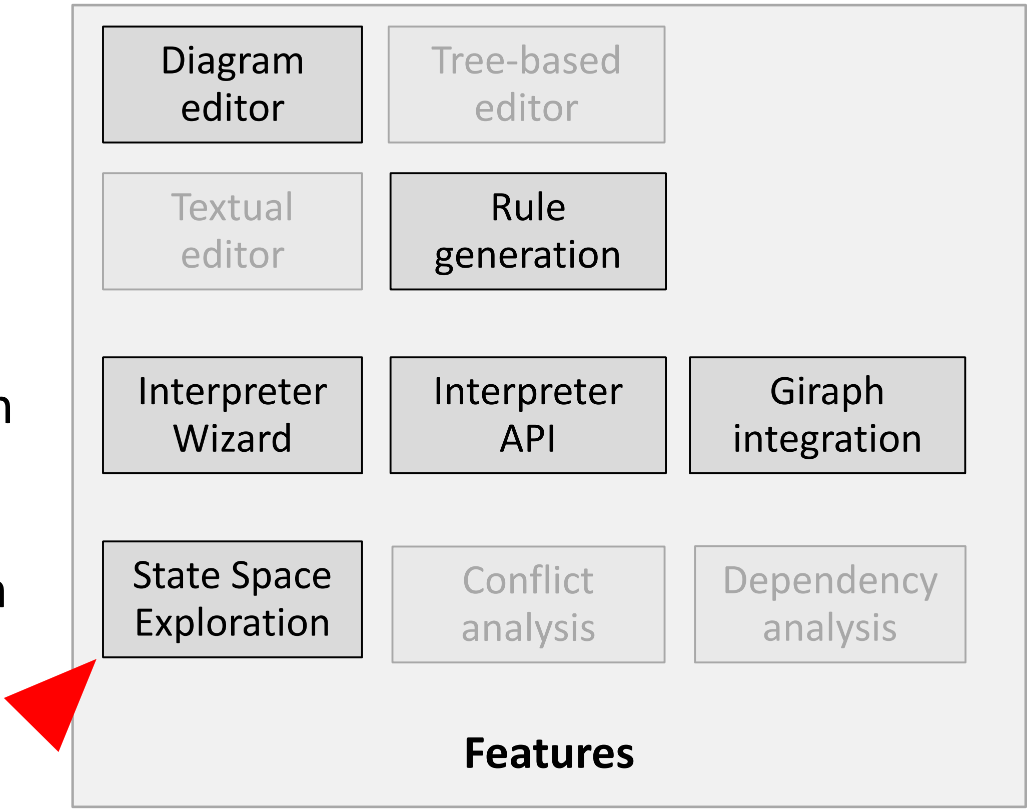


Scales to IMDB data
924054 movies
1777656 male actors
980396 actresses

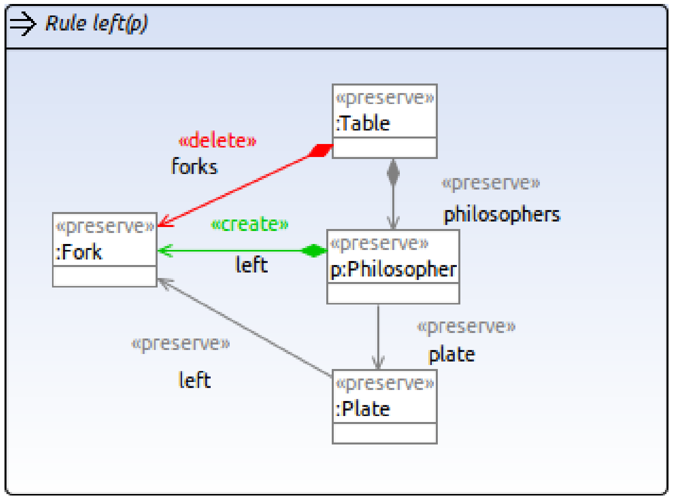
- Code generation for Apache Giraph
- Massive parallel execution
- Scales to millions of nodes and edges

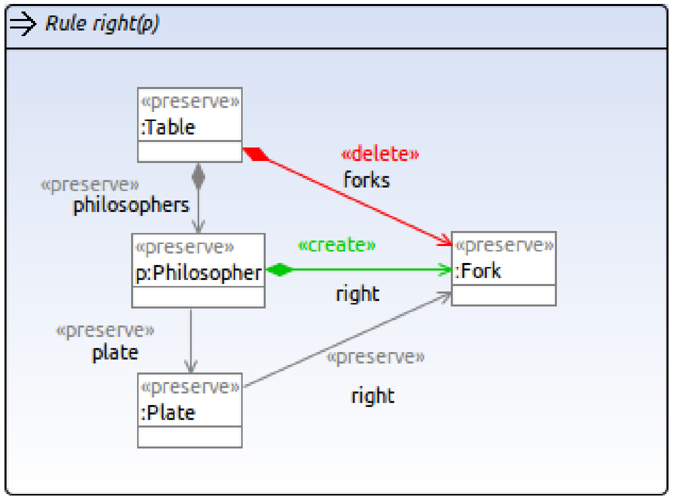
Features: What would you like to do today?

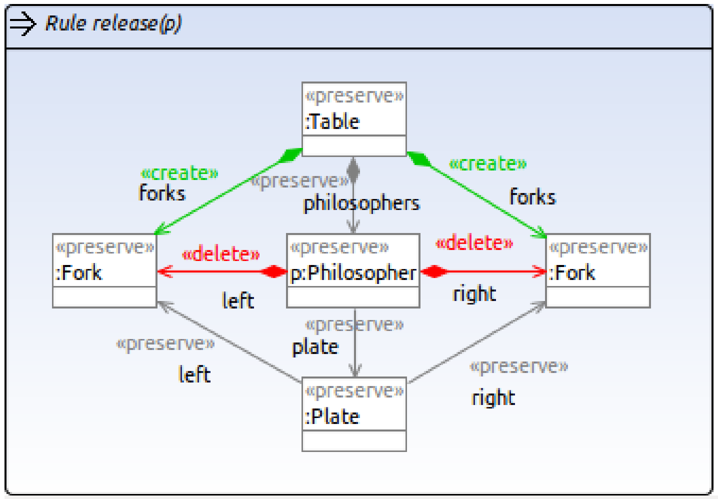
- Define a transformation
- Execute a transformation
- Analyse a transformation

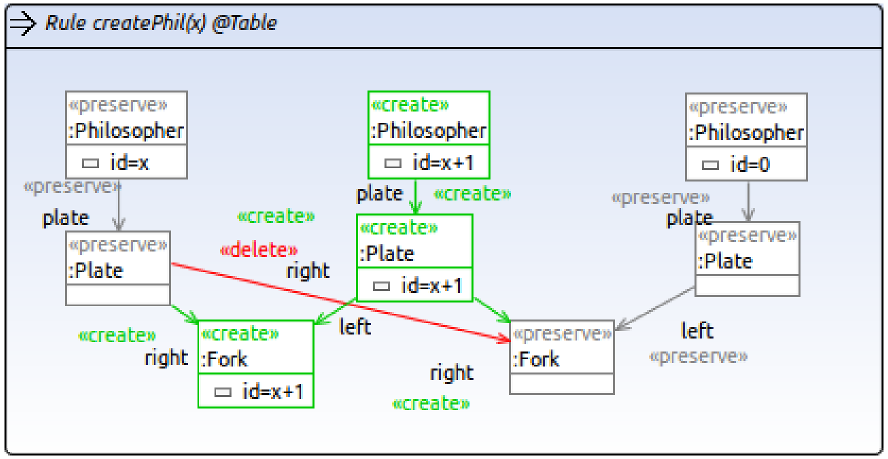


Example: Dining Philosophers



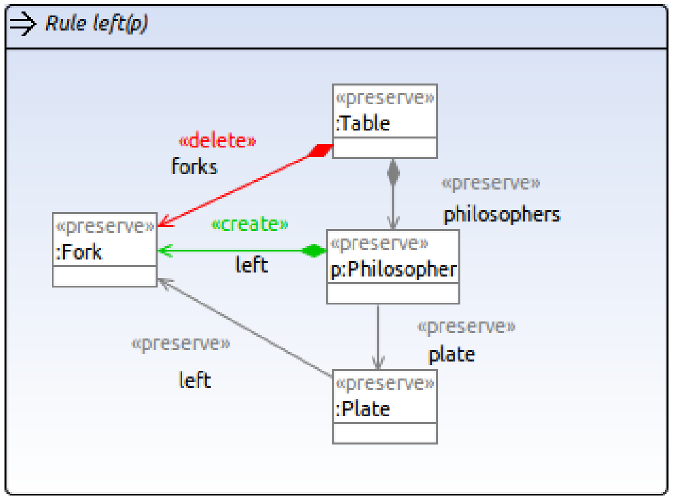


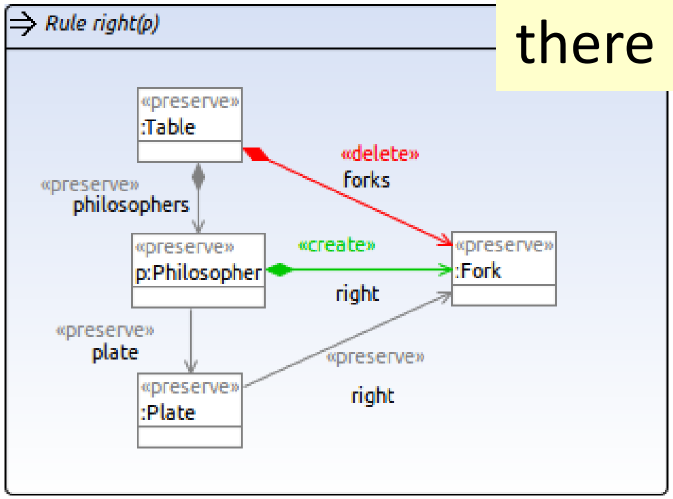


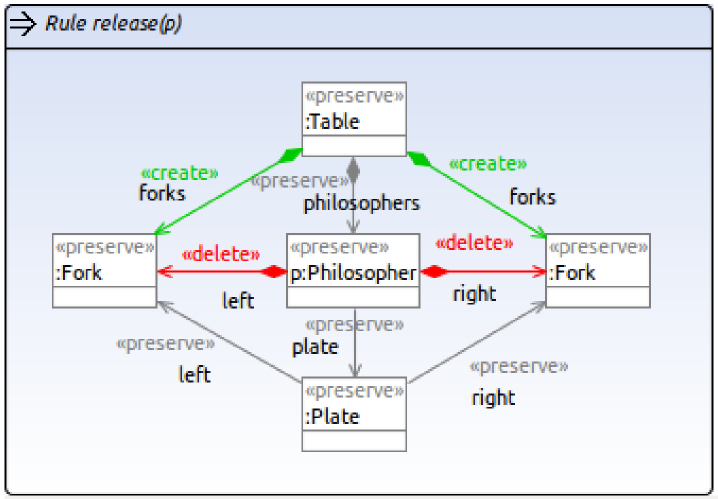


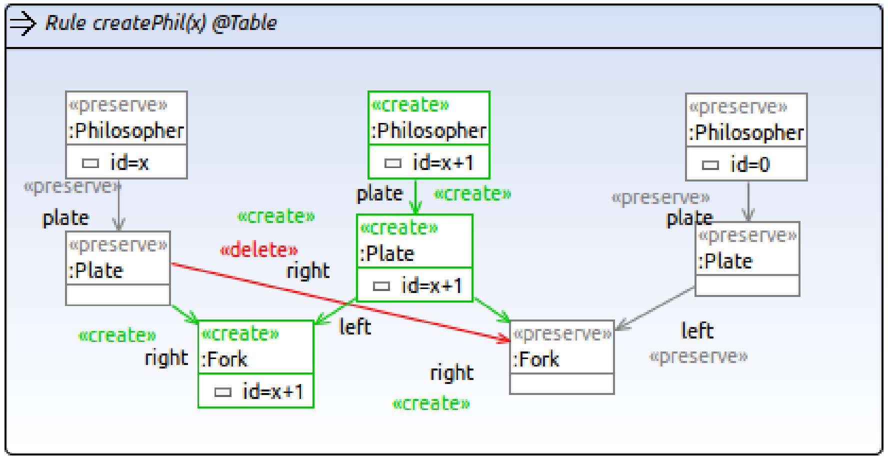
Example: Dining Philosophers

Question: Is there a deadlock?





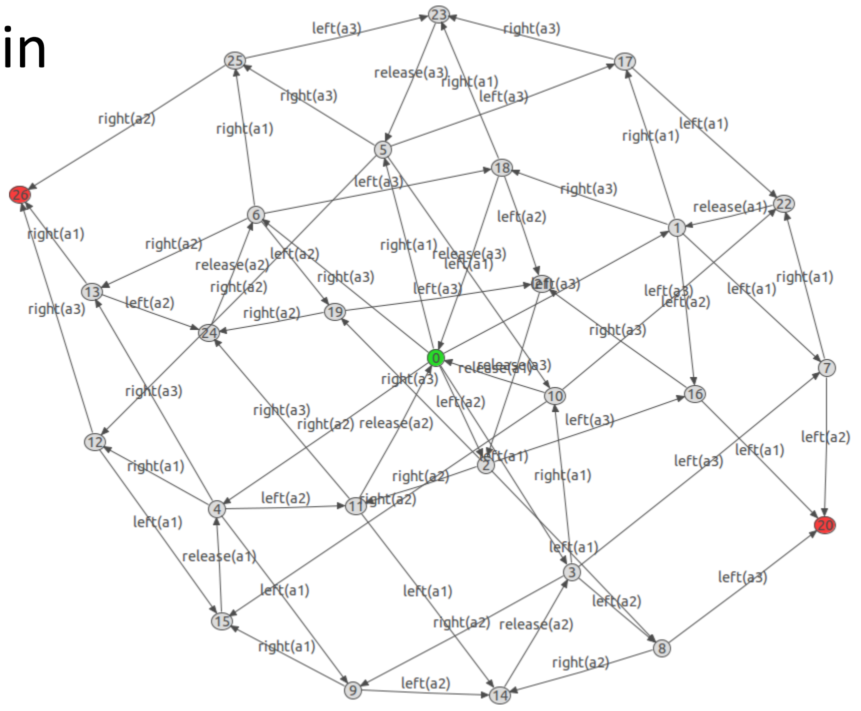




Computing the state space for verification

- Full state space is computed
- Abstracts from order and a certain attributes
- State invariants, qualitative and probabilistic model checking

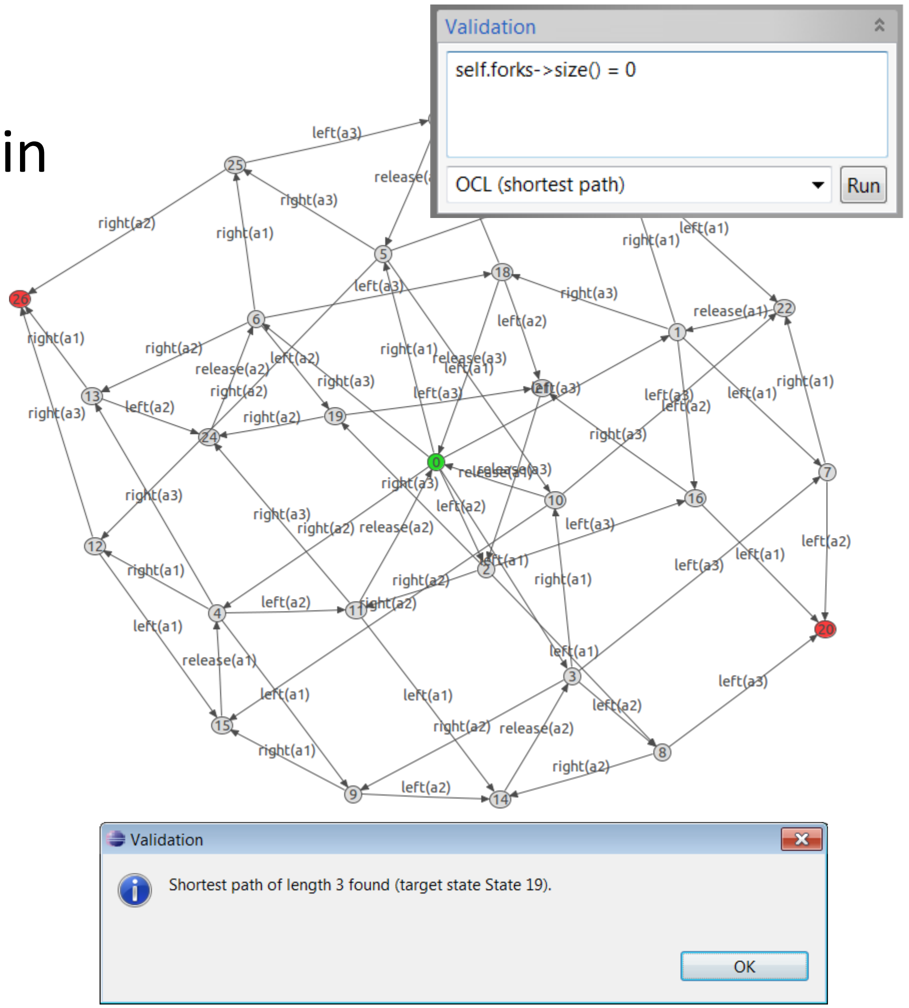
Philosophers	States (= 3^p)	Transitions	Time
3	27	63	56ms
4	81	252	69ms
5	243	945	224ms
6	729	3,402	616ms
7	2,187	11,907	1.3s
8	6,561	40,824	5.0s
9	19,683	137,781	19.8s
10	59,049	459,270	80.5s
11	177,147	1,515,591	6min
12	531,441	4,960,116	61min
13	1,594,323	16,120,377	593min



Computing the state space for verification

- Full state space is computed
- Abstracts from order and a certain attributes
- State invariants, qualitative and probabilistic model checking

Philosophers	States (= 3^p)	Transitions	Time
3	27	63	56ms
4	81	252	69ms
5	243	945	224ms
6	729	3,402	616ms
7	2,187	11,907	1.3s
8	6,561	40,824	5.0s
9	19,683	137,781	19.8s
10	59,049	459,270	80.5s
11	177,147	1,515,591	6min
12	531,441	4,960,116	61min
13	1,594,323	16,120,377	593min



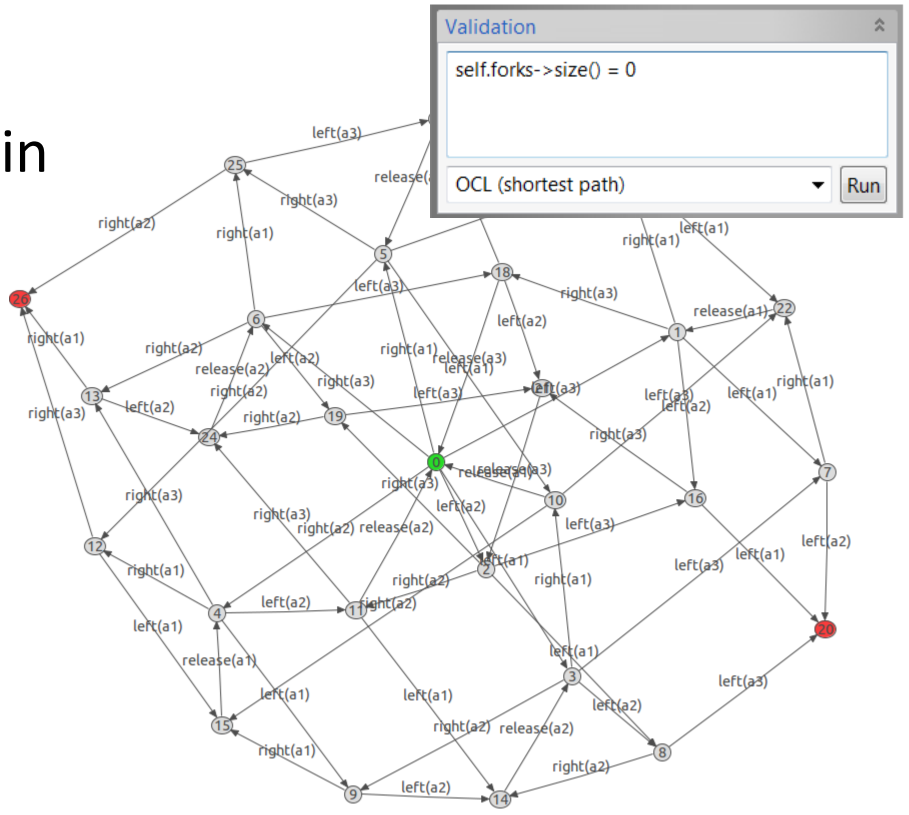
self.forks->size() = 0

Shortest path of length 3 found (target state State 19).

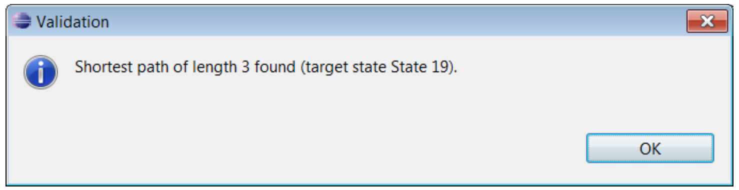
Computing the state space for verification

- Full state space is computed
- Abstracts from order and a certain attributes
- State invariants, qualitative and probabilistic model checking

Philosophers	States (= 3^p)	Transitions	Time
3	27	63	56ms
4	81	252	69ms
5	243	945	224ms
6	729	3,402	616ms
7	2,187	11,907	1.3s
8	6,561	40,824	5.0s
9	19,683	137,781	19.8s
10	59,049	459,270	80.5s
11	177,147	1,515,591	6min
12	531,441	4,960,116	61min
13	1,594,323	16,120,377	593min



```
self.forks->size() = 0
```

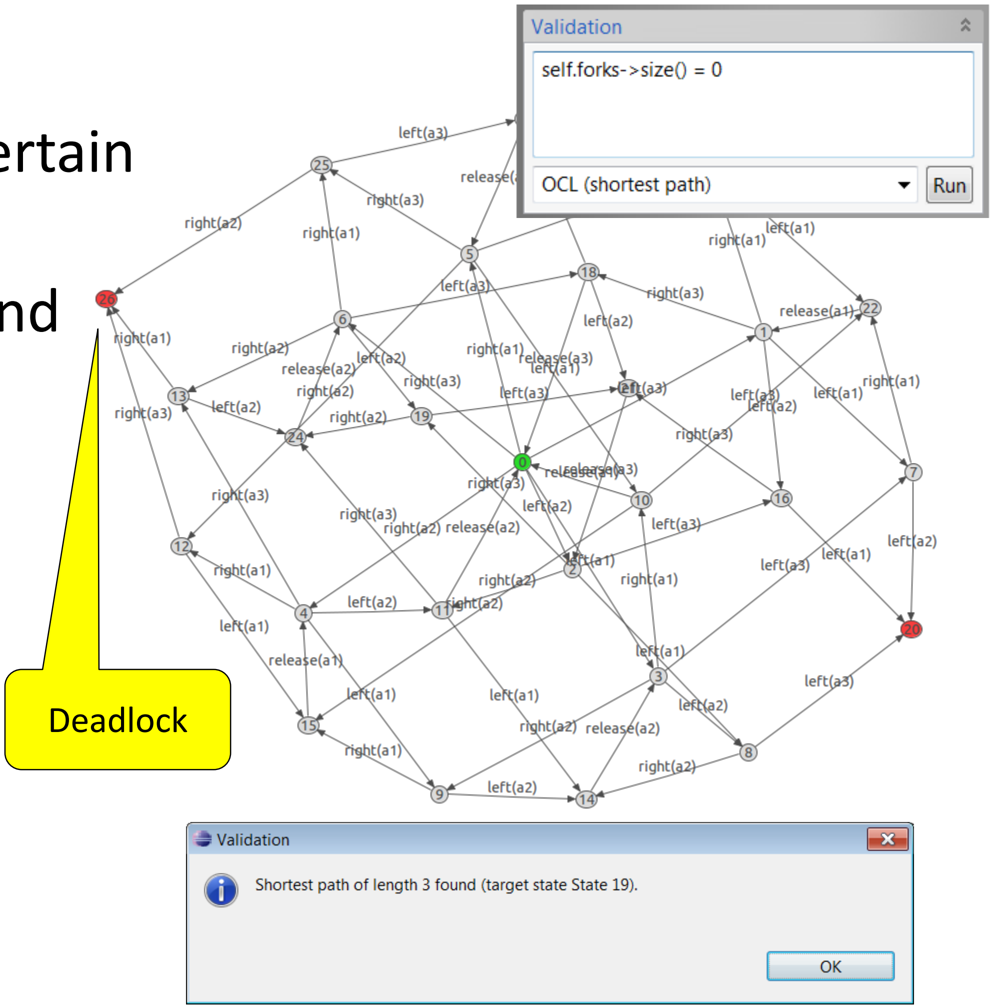


Shortest path of length 3 found (target state State 19).

Computing the state space for verification

- Full state space is computed
- Abstracts from order and a certain attributes
- State invariants, qualitative and probabilistic model checking

Philosophers	States (= 3^p)	Transitions	Time
3	27	63	56ms
4	81	252	69ms
5	243	945	224ms
6	729	3,402	616ms
7	2,187	11,907	1.3s
8	6,561	40,824	5.0s
9	19,683	137,781	19.8s
10	59,049	459,270	80.5s
11	177,147	1,515,591	6min
12	531,441	4,960,116	61min
13	1,594,323	16,120,377	593min



Validation

self.forks->size() = 0

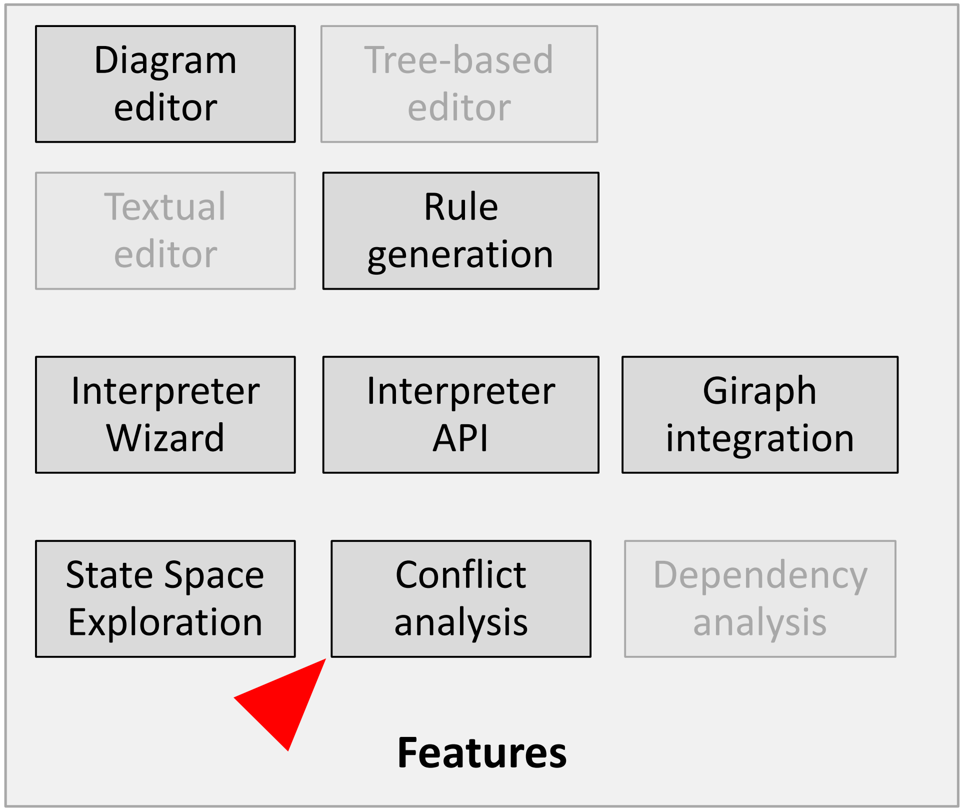
OCL (shortest path)

Validation

Shortest path of length 3 found (target state State 19).

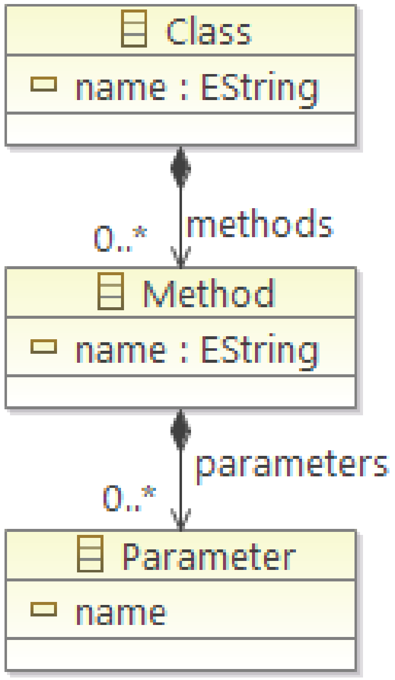
Features: What would you like to do today?

- Define a transformation
- Execute a transformation
- Analyse a transformation

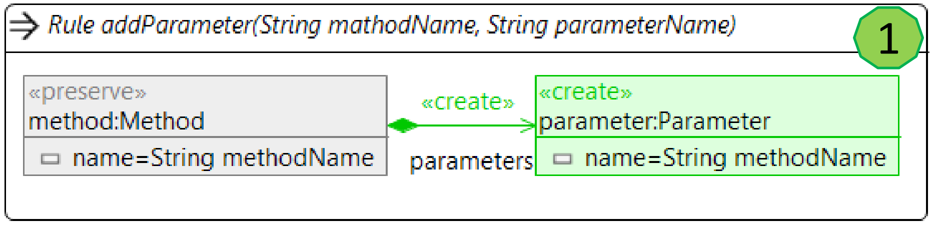


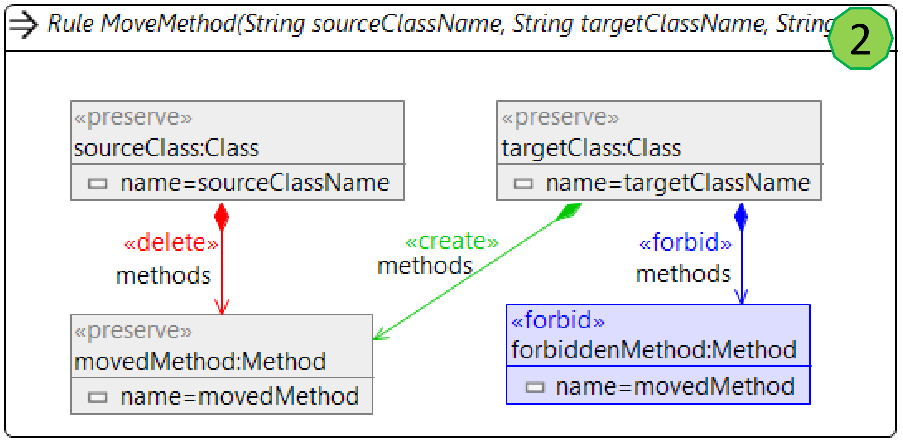
Example: conflicts in model refactorings

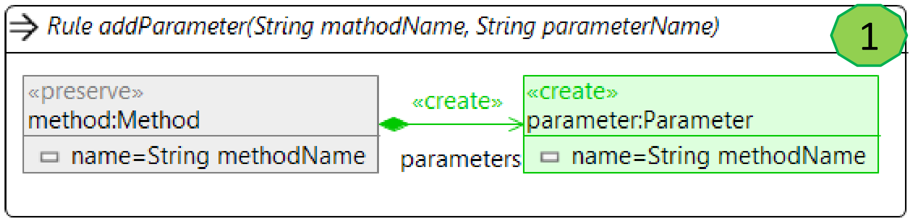
Meta-model

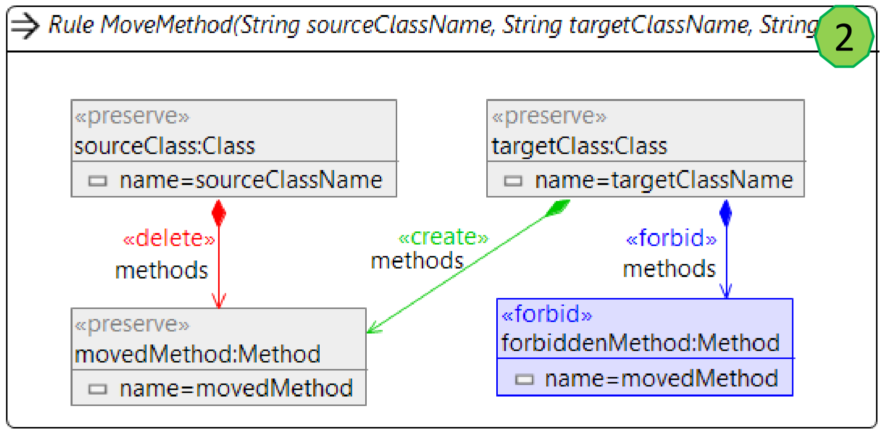


Rules

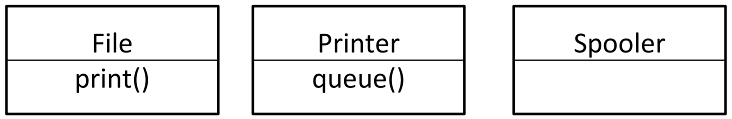


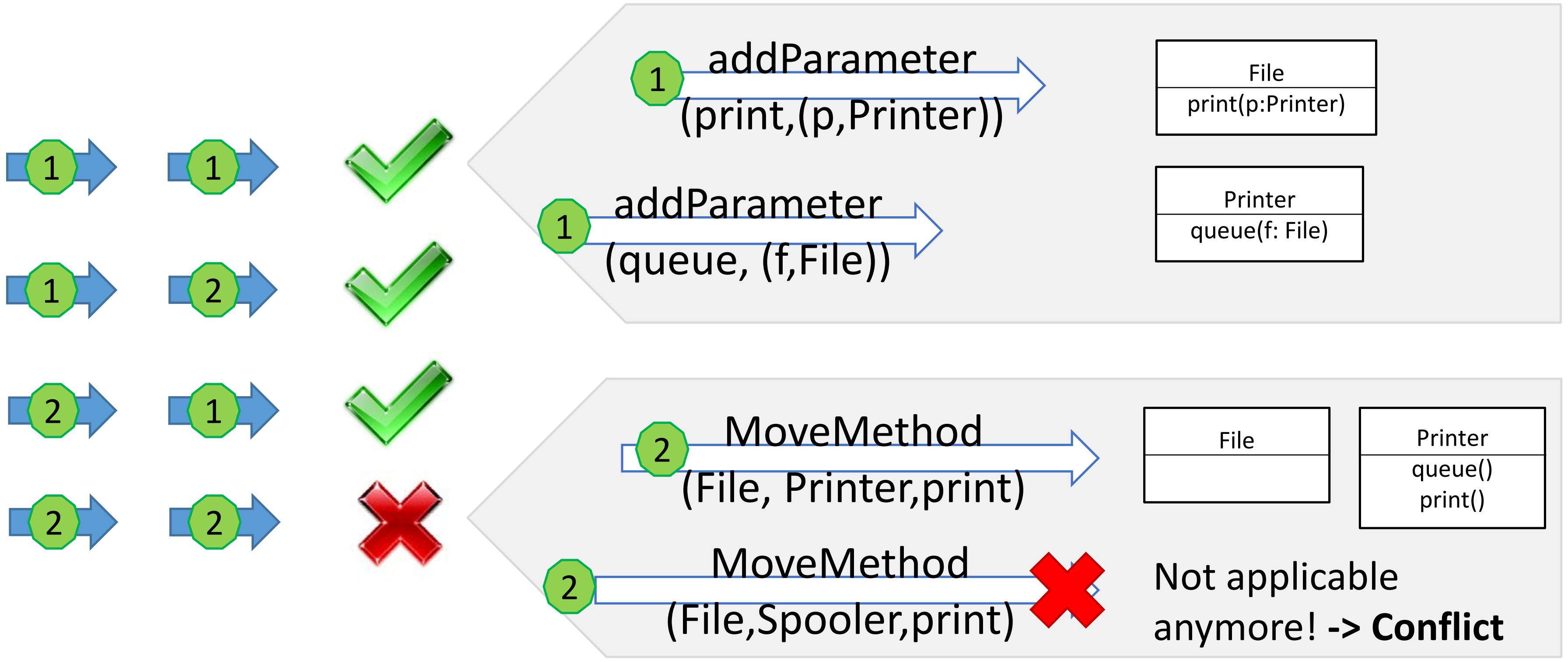



Rule addParameter(String mathodName, String parameterName)
1
«preserve»
method:Method
name=String methodName
«create»
parameters
«create»
parameter:Parameter
name=String methodName


Rule MoveMethod(String sourceClassName, String targetClassName, String
2
«preserve»
sourceClass:Class
name=sourceClassName
«preserve»
targetClass:Class
name=targetClassName
«delete»
methods
«create»
methods
«forbid»
methods
«preserve»
movedMethod:Method
name=movedMethod
«forbid»
forbiddenMethod:Method
name=movedMethod

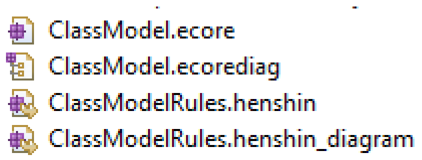
Input model


File
print()
Printer
queue()
Spooler

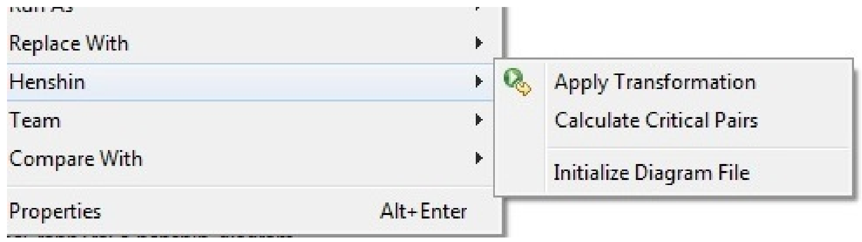

1
1
1
2
2
1
2
2
1
addParameter
(print,(p,Printer))
File
print(p:Printer)
1
addParameter
(queue, (f,File))
Printer
queue(f: File)
2
MoveMethod
(File, Printer,print)
File
Printer
queue()
print()
2
MoveMethod
(File,Spooler,print)
Not applicable
anymore! -> Conflict

Conflict and dependency analysis

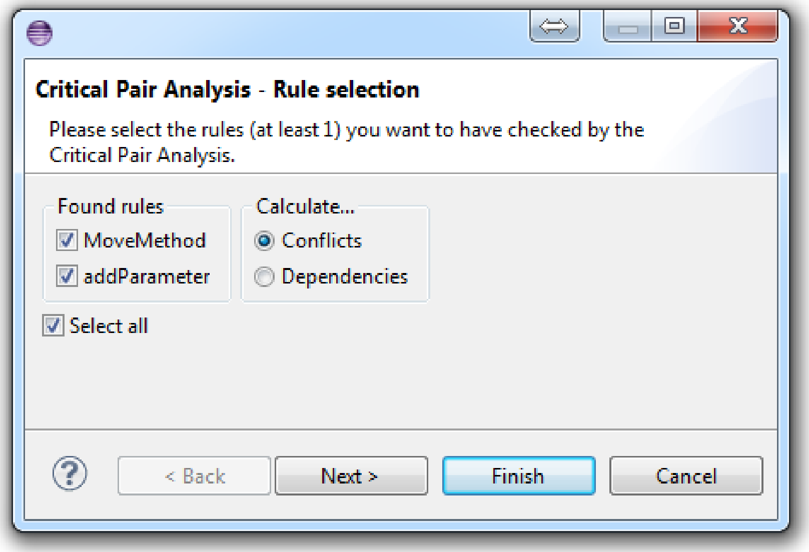
1. Input: meta-model + rules


ClassModel.ecore
ClassModel.ecorediag
ClassModelRules.henshin
ClassModelRules.henshin_diagram

2. Context menu ->
 Calculate Critical Pairs



3. Rule selection + options

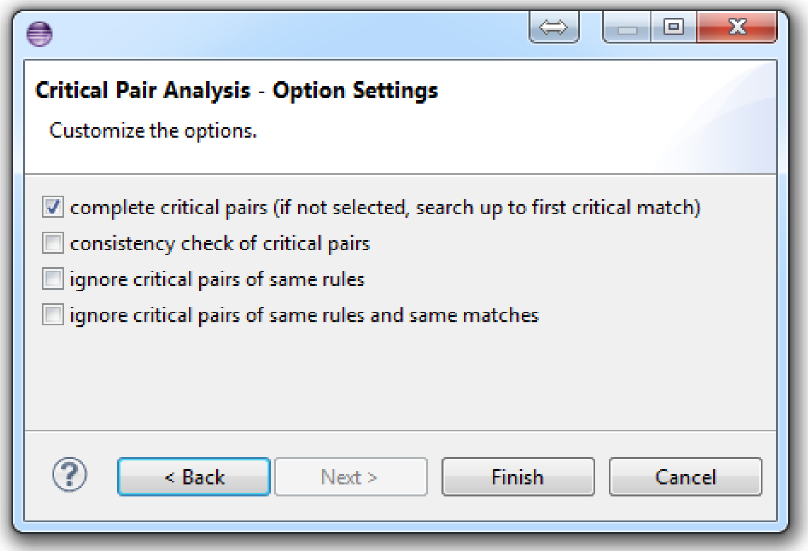

Critical Pair Analysis - Rule selection

Please select the rules (at least 1) you want to have checked by the Critical Pair Analysis.

Found rules
- [✓] MoveMethod
- [✓] addParameter

Calculate...
- [✓ Conflicts]
- Dependencies

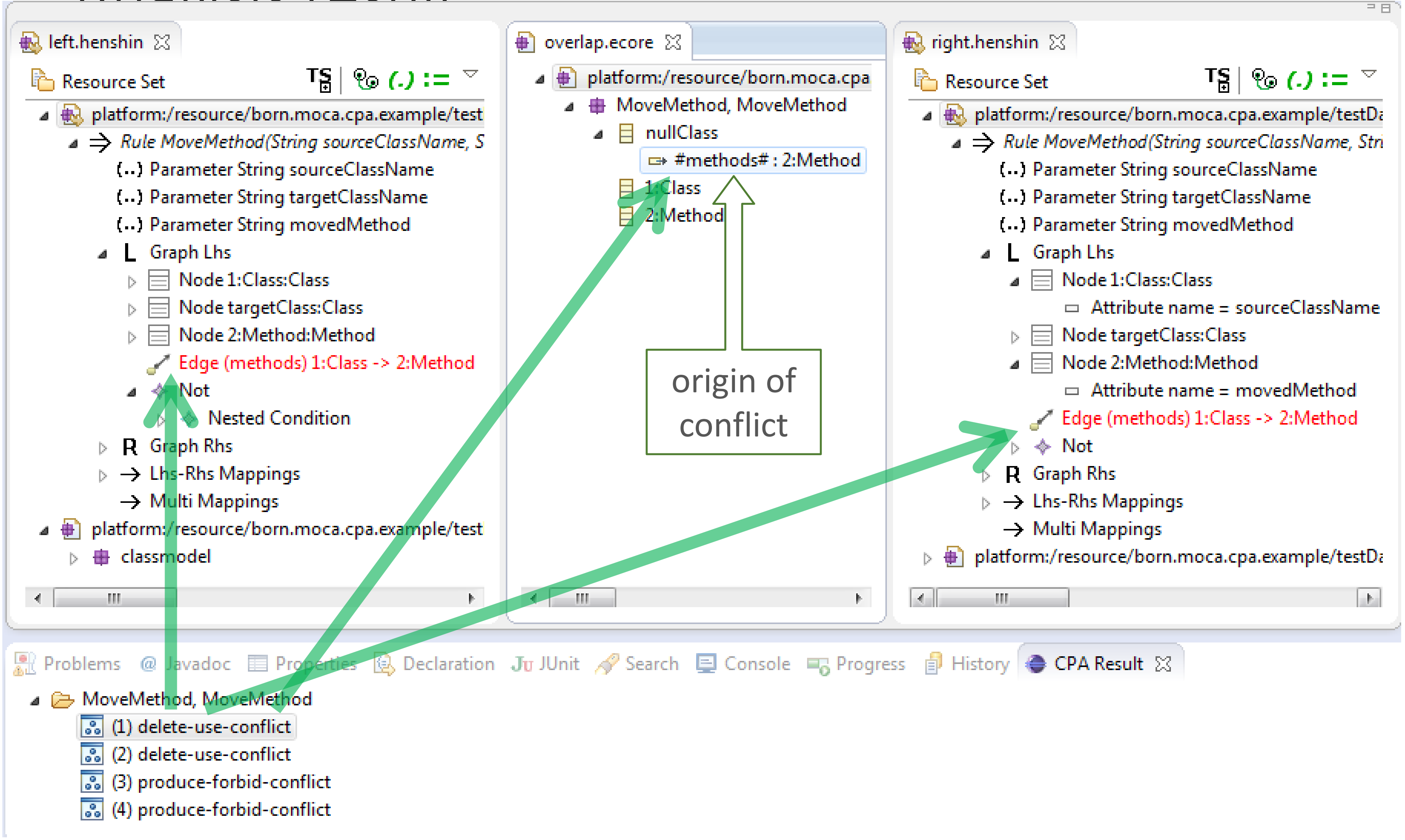
[✓] Select all


Critical Pair Analysis - Option Settings

Customize the options.

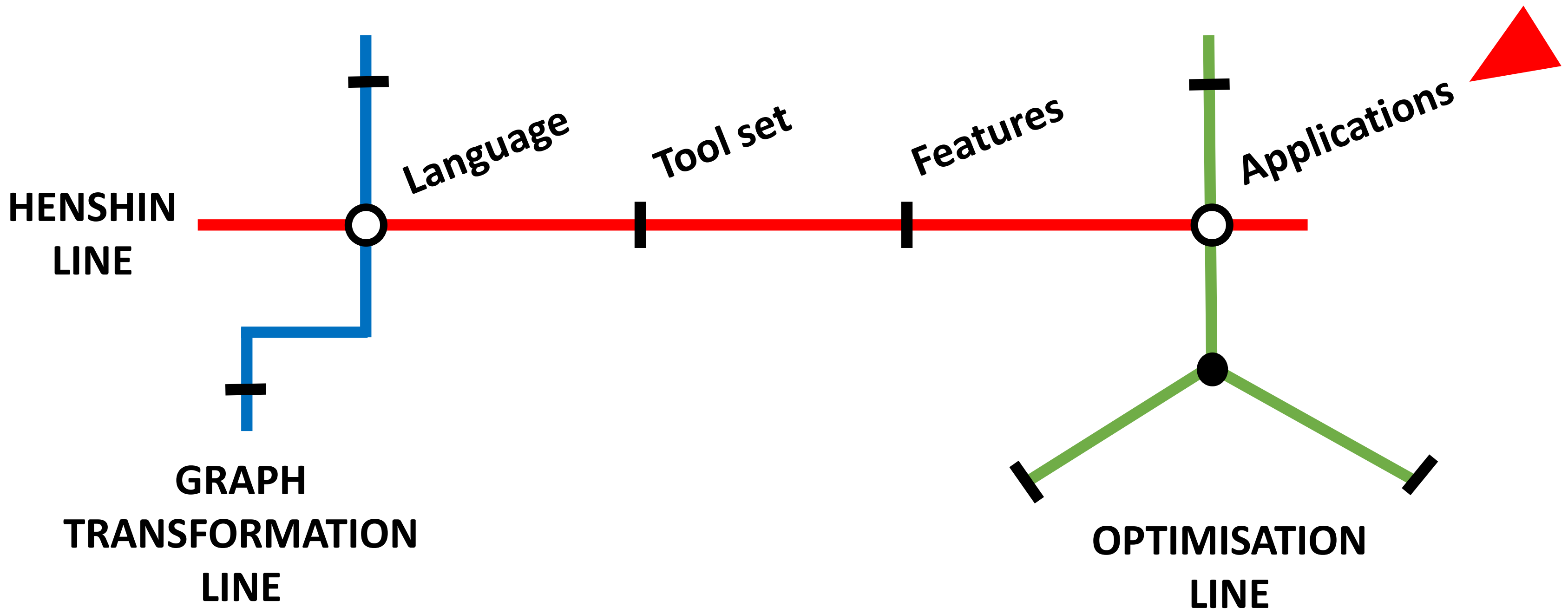
- [✓] complete critical pairs (if not selected, search up to first critical match)
- [ ] consistency check of critical pairs
- [ ] ignore critical pairs of same rules
- [ ] ignore critical pairs of same rules and same matches

Analysis result



left.henshin
Resource Set
platform:/resource/born.moca.cpa.example/test
Rule MoveMethod(String sourceClassName, S
(..) Parameter String sourceClassName
(..) Parameter String targetClassName
(..) Parameter String movedMethod
L Graph Lhs
Node 1:Class:Class
Node targetClass:Class
Node 2:Method:Method
Edge (methods) 1:Class -> 2:Method
Not
Nested Condition
R Graph Rhs
Lhs-Rhs Mappings
Multi Mappings
platform:/resource/born.moca.cpa.example/test
classmodel
overlap.ecore
platform:/resource/born.moca.cpa
MoveMethod, MoveMethod
nullClass
#methods# : 2:Method
1:Class
2:Method
origin of conflict
right.henshin
Resource Set
platform:/resource/born.moca.cpa.example/testDa
Rule MoveMethod(String sourceClassName, Stri
(..) Parameter String sourceClassName
(..) Parameter String targetClassName
(..) Parameter String movedMethod
L Graph Lhs
Node 1:Class:Class
Attribute name = sourceClassName
Node targetClass:Class
Node 2:Method:Method
Attribute name = movedMethod
Edge (methods) 1:Class -> 2:Method
Not
R Graph Rhs
Lhs-Rhs Mappings
Multi Mappings
platform:/resource/born.moca.cpa.example/testDa
Problems
Javadoc
Properties
Declaration
JUnit
Search
Console
Progress
History
CPA Result
MoveMethod, MoveMethod
(1) delete-use-conflict
(2) delete-use-conflict
(3) produce-forbid-conflict
(4) produce-forbid-conflict

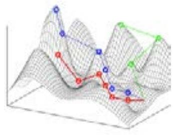
Henshin: A Guided Tour



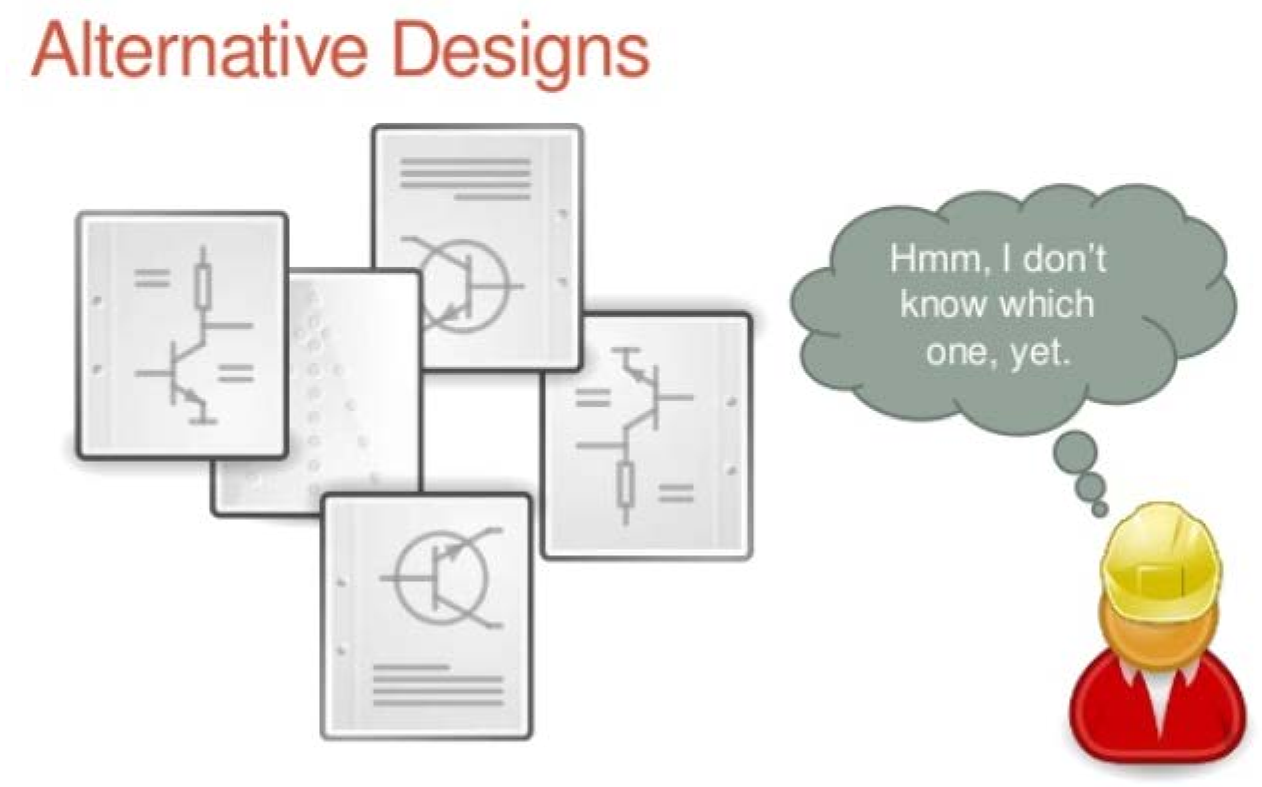
Applications

Can do many things with Henshin

- Model uncertainty
- Model-based security
- Model versioning
- Search-based model optimisation
- …

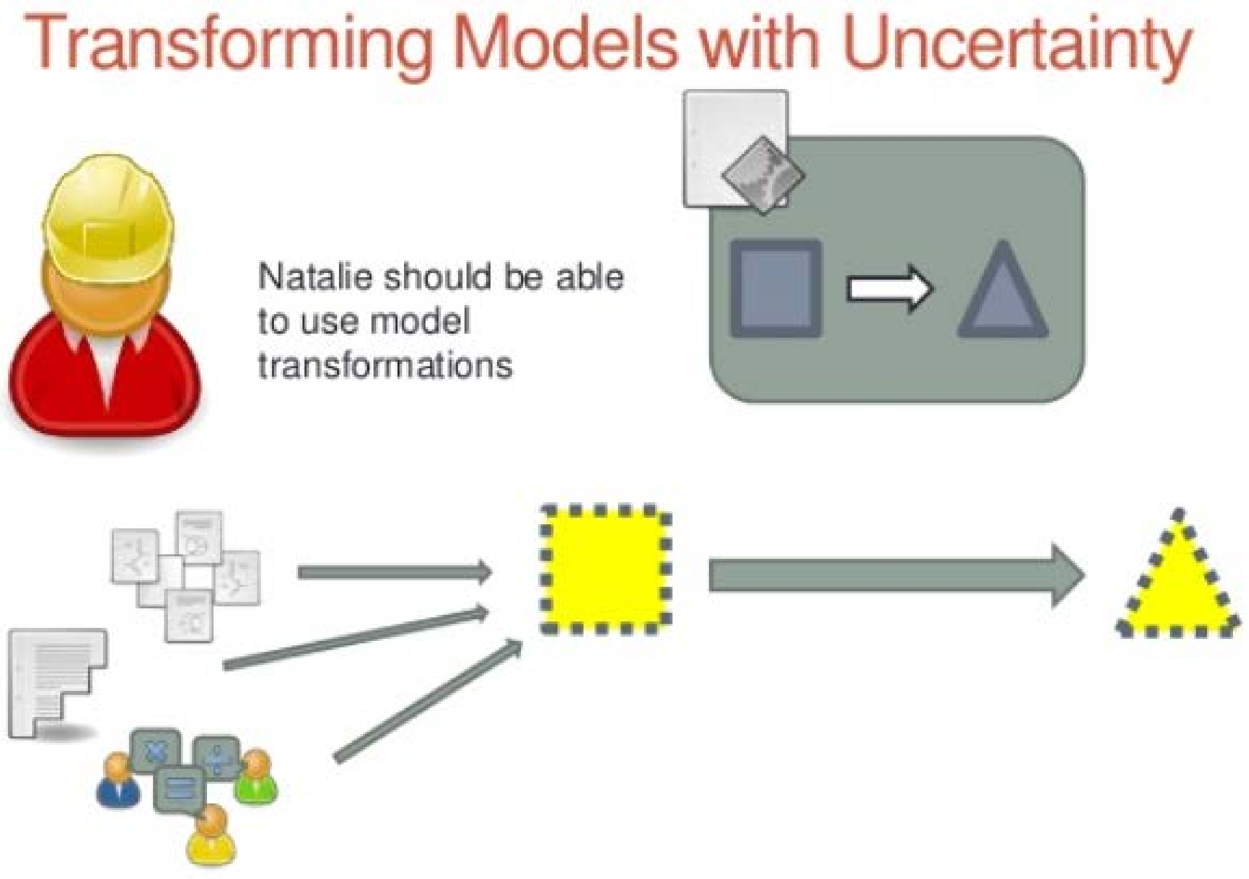


Models with uncertainty and variability



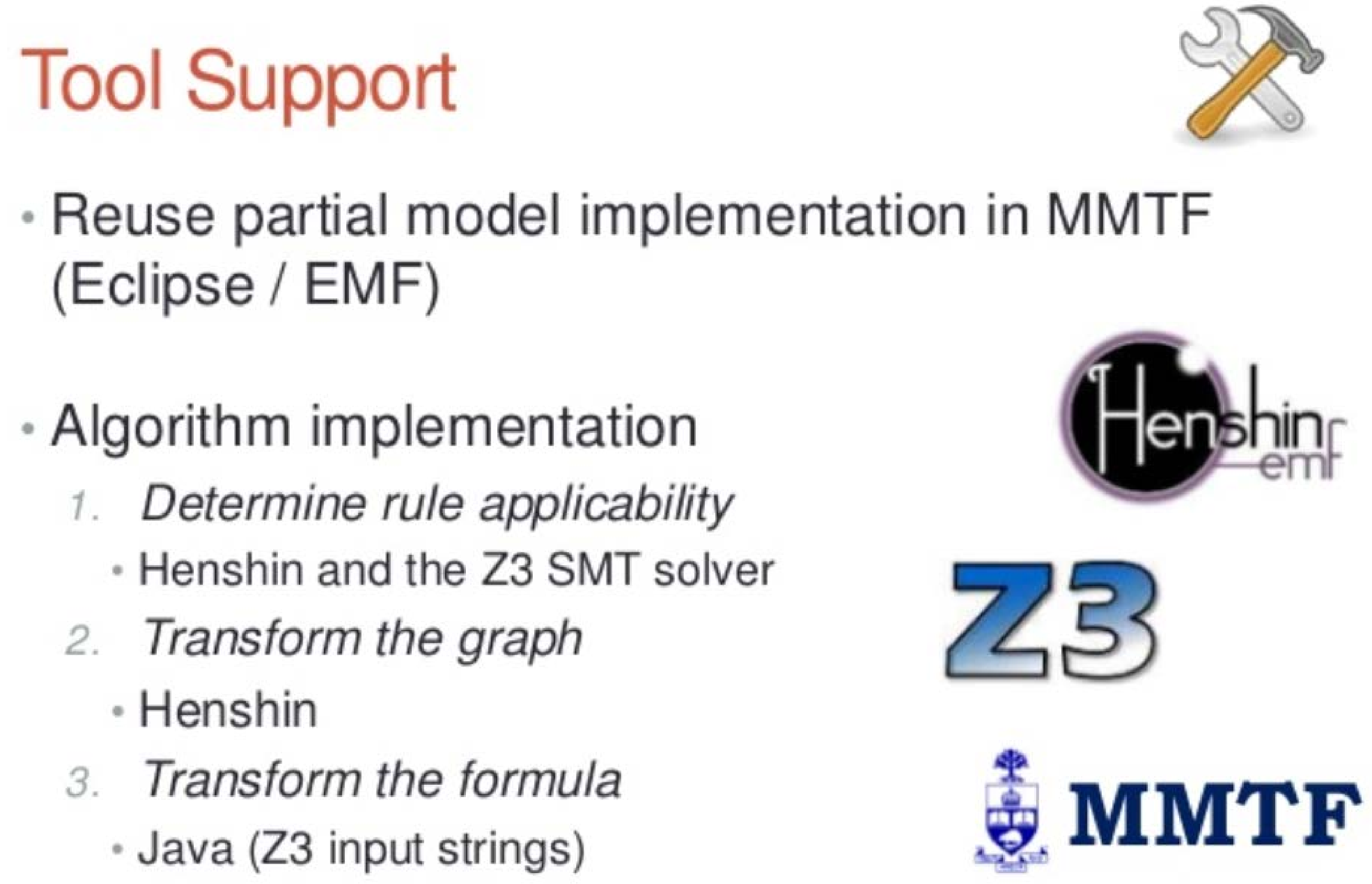
Courtesy of Famelis et al. [MODELS 2013]

Models with uncertainty and variability


Transforming Models with Uncertainty
Natalie should be able to use model transformations

Courtesy of Famelis et al. [MODELS 2013]

Models with uncertainty and variability

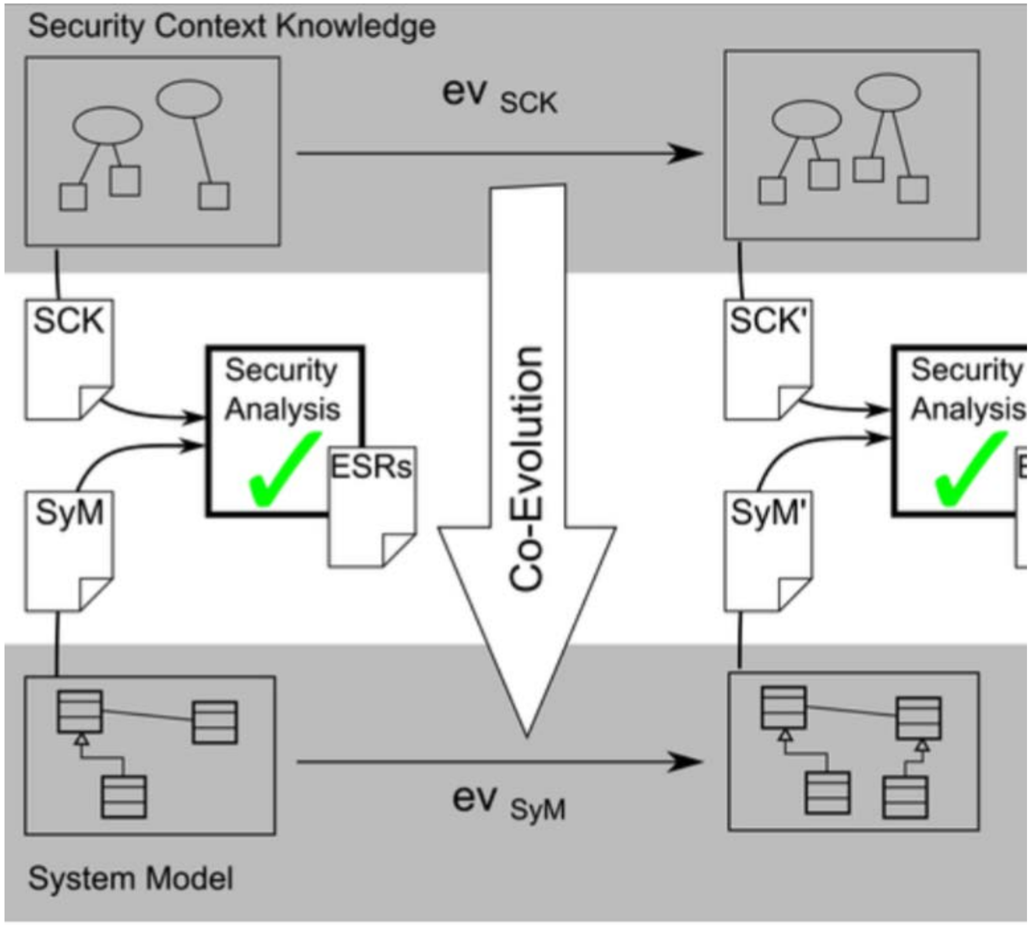


Tool Support

- Reuse partial model implementation in MMTF (Eclipse / EMF)
- Algorithm implementation
  1. *Determine rule applicability*
     - Henshin and the Z3 SMT solver
  2. *Transform the graph*
     - Henshin
  3. *Transform the formula*
     - Java (Z3 input strings)

Courtesy of Famelis et al. [MODELS 2013]

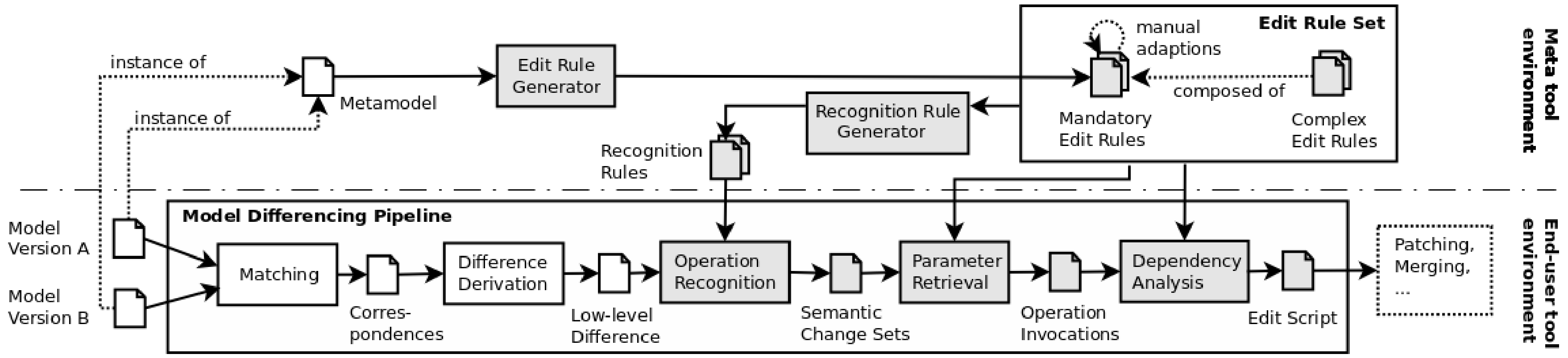
Security Engineering: keep system design aligned with security knowledge



- Security knowledge maintained in a **security ontology**
- Ontology evolution triggers corresponding **design-model co-evolution rules**

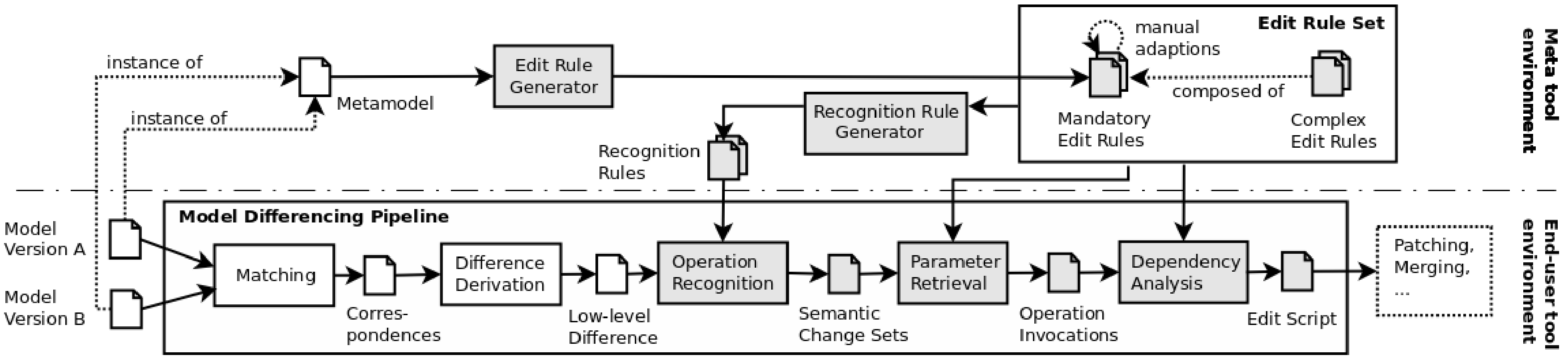
Courtesy of Bürger et al. [JSS 2018]

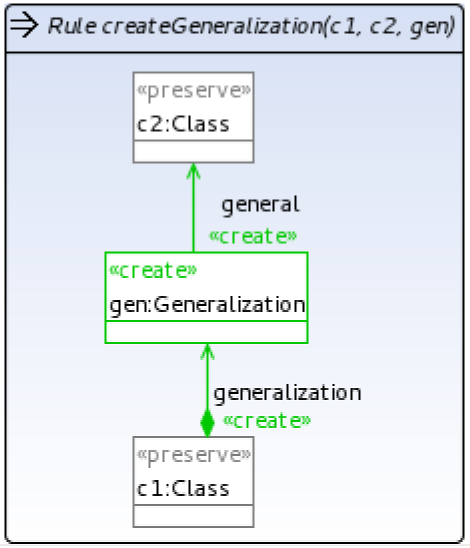
Model versioning: Recognizing executed edit operations

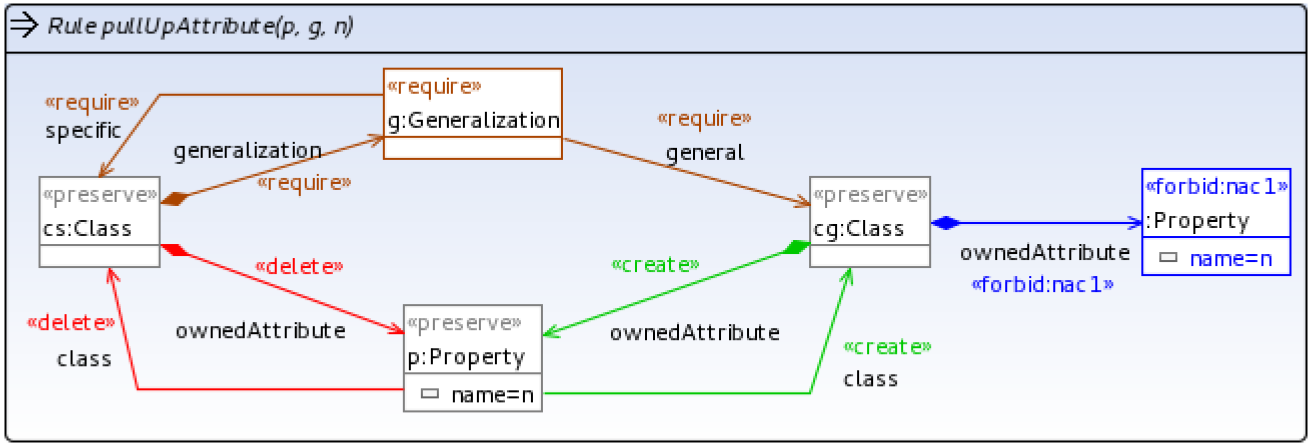


Courtesy of Kehrer et al. [ASE 2013]

Model versioning: Recognizing executed edit operations

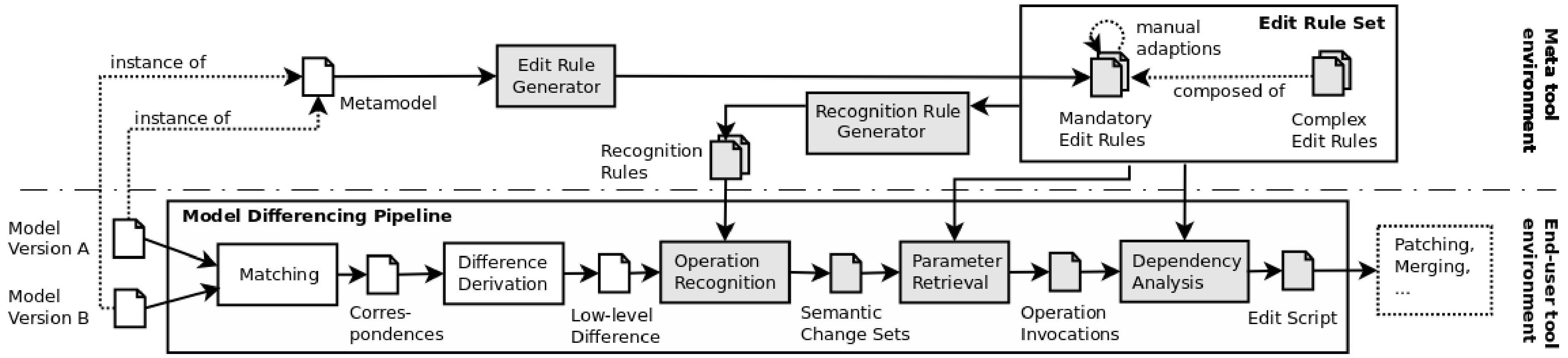


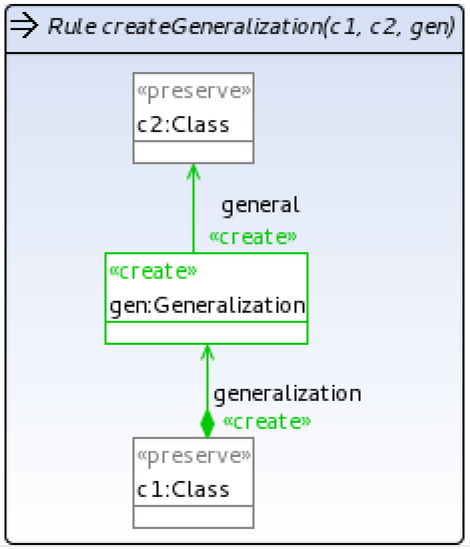


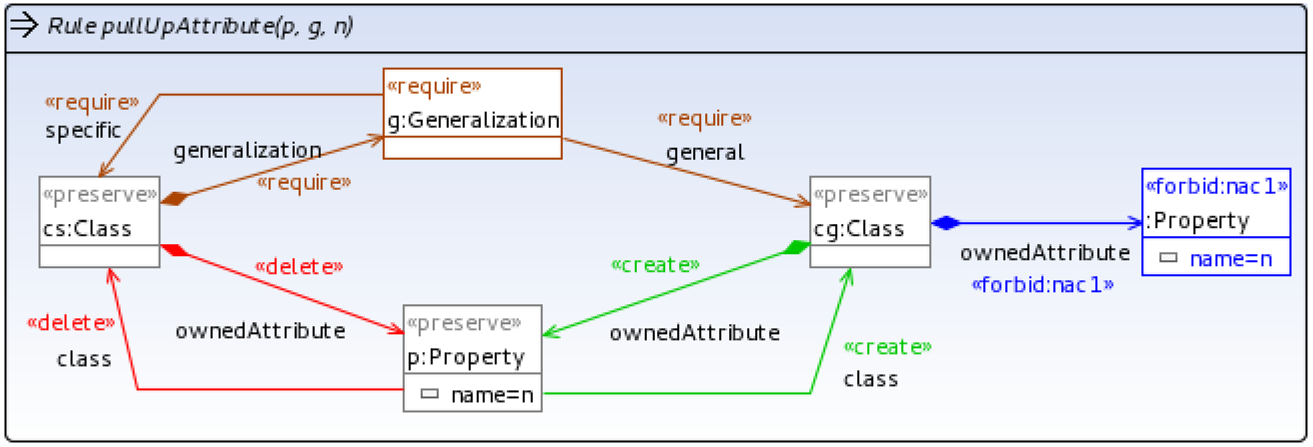


Courtesy of Kehrer et al. [ASE 2013]

Model versioning: Recognizing executed edit operations

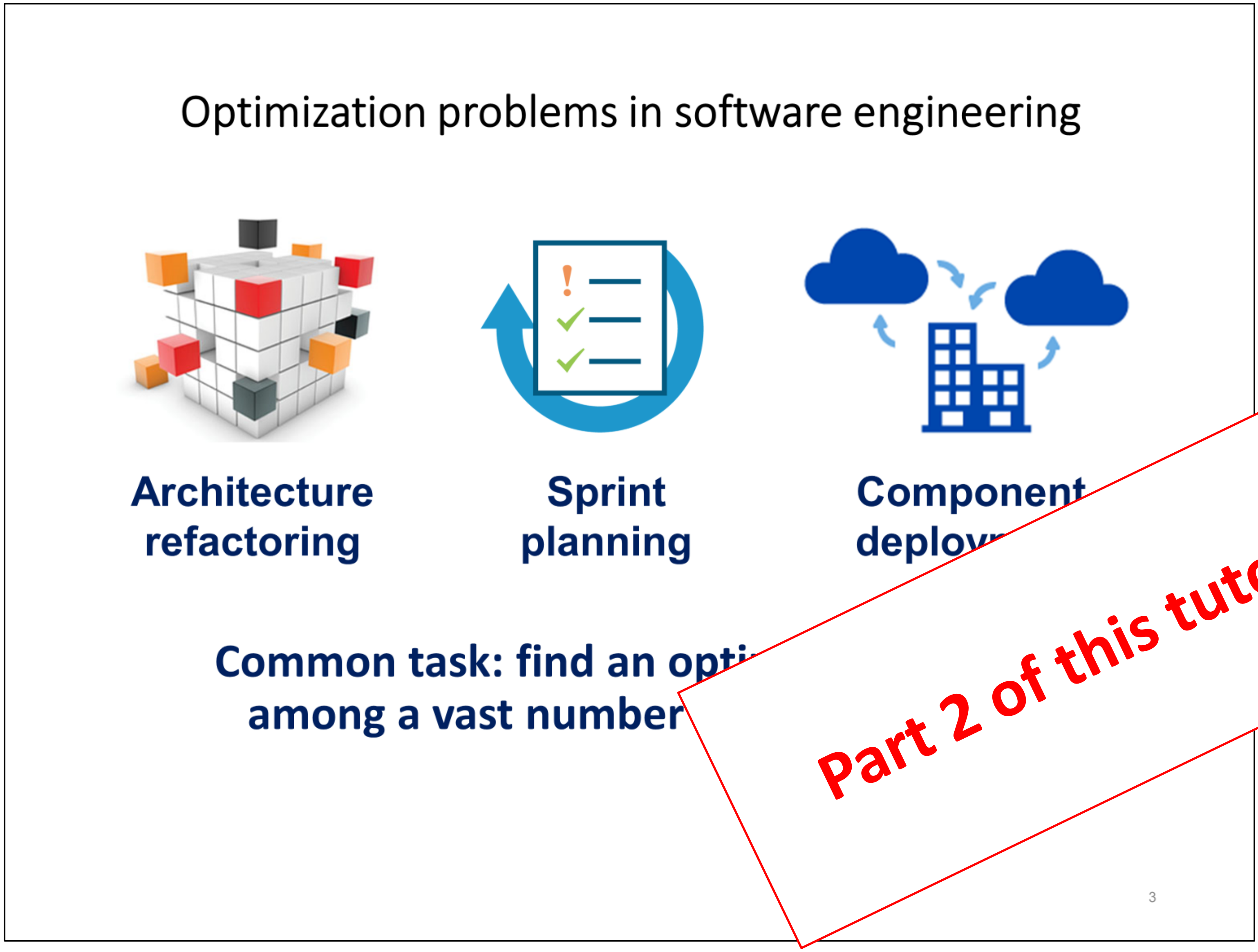






Courtesy of Kehrer et al. [ASE 2013]

Search-based model optimisation



Optimization problems in software engineering

Architecture refactoring

Sprint planning

Component deploy

Common task: find an opti
among a vast number

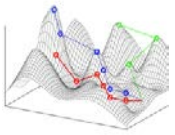
3

Part 2 of this tutorial

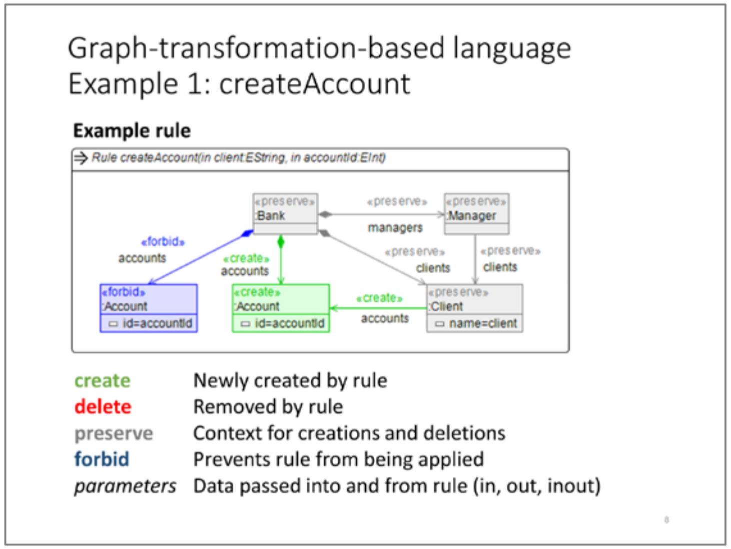
Applications

Can do many things with Henshin

- Model uncertainty
- Model-based security
- Model versioning
- Search-based model optimisation
- …

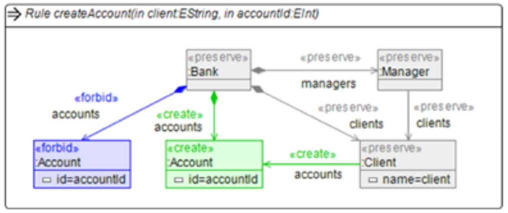


Summary of Part 1

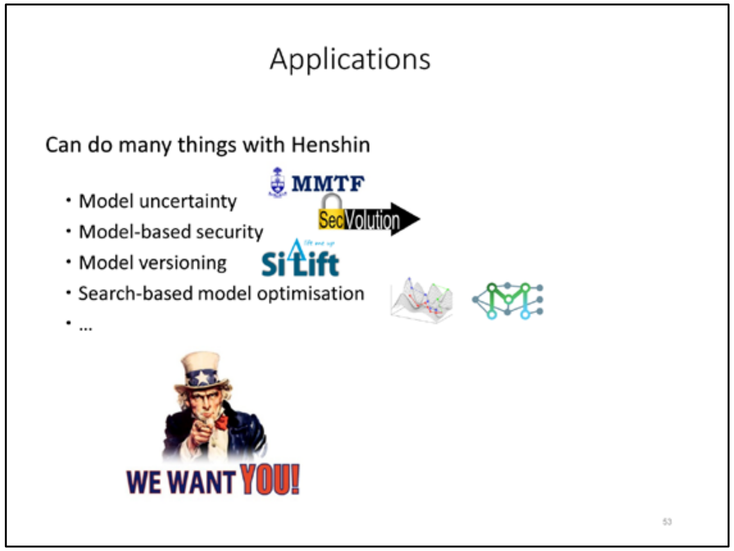


Graph-transformation-based language Example 1: createAccount

Example rule



create	Newly created by rule
delete	Removed by rule
preserve	Context for creations and deletions
forbid	Prevents rule from being applied
parameters	Data passed into and from rule (in, out, inout)

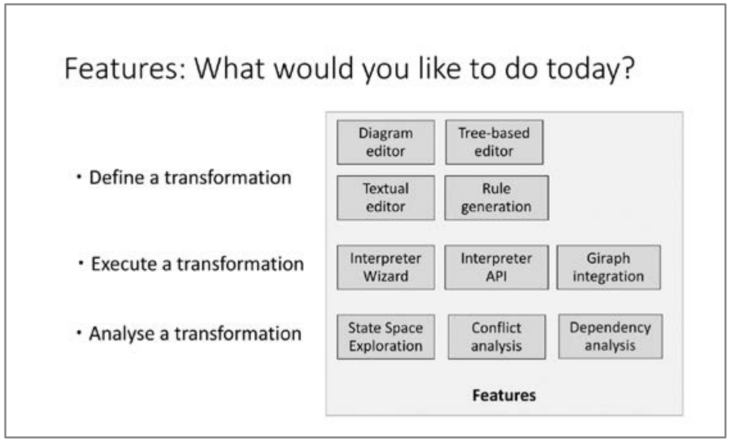


Applications

Can do many things with Henshin

- Model uncertainty
- Model-based security
- Model versioning
- Search-based model optimisation
- …

MMTF
SecVolution
SiLift
WE WANT YOU!



Features: What would you like to do today?

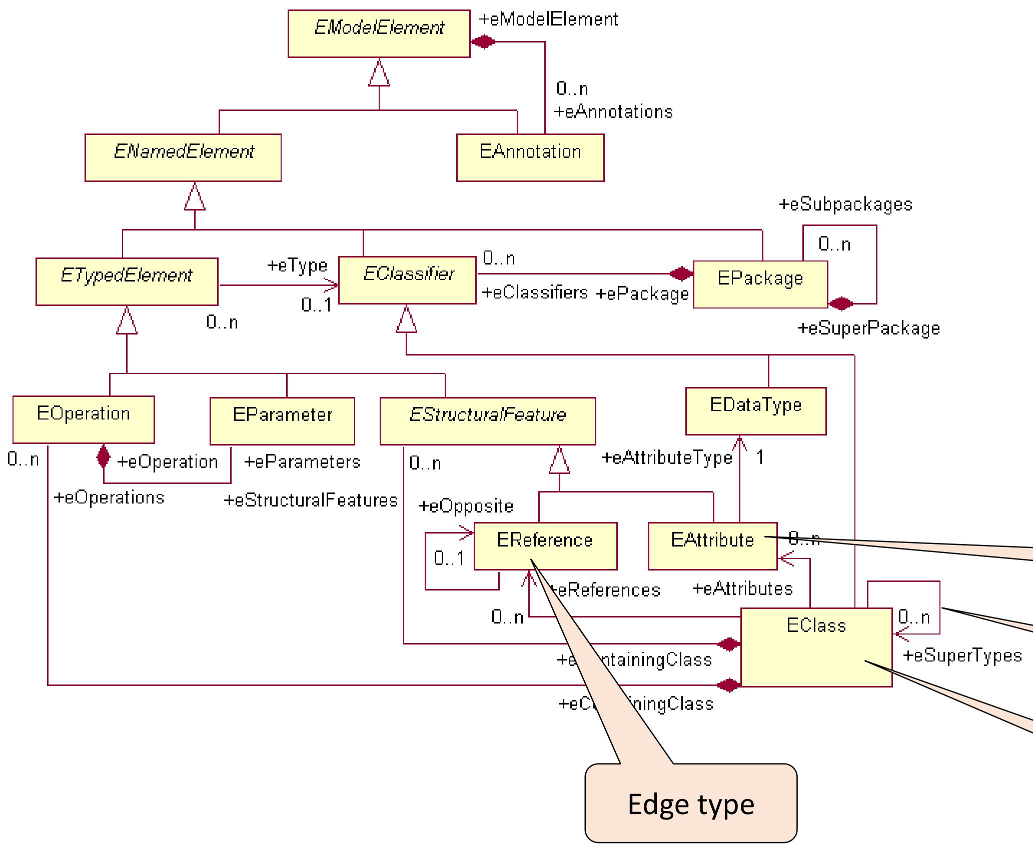
- Define a transformation
- Execute a transformation
- Analyse a transformation

Diagram editor
Tree-based editor
Textual editor
Rule generation
Interpreter Wizard
Interpreter API
Giraph integration
State Space Exploration
Conflict analysis
Dependency analysis
Features

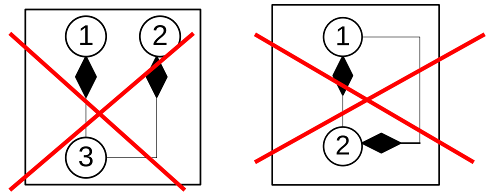
Further information:
www.eclipse.org/henshin

Backup material

Model instances as graphs

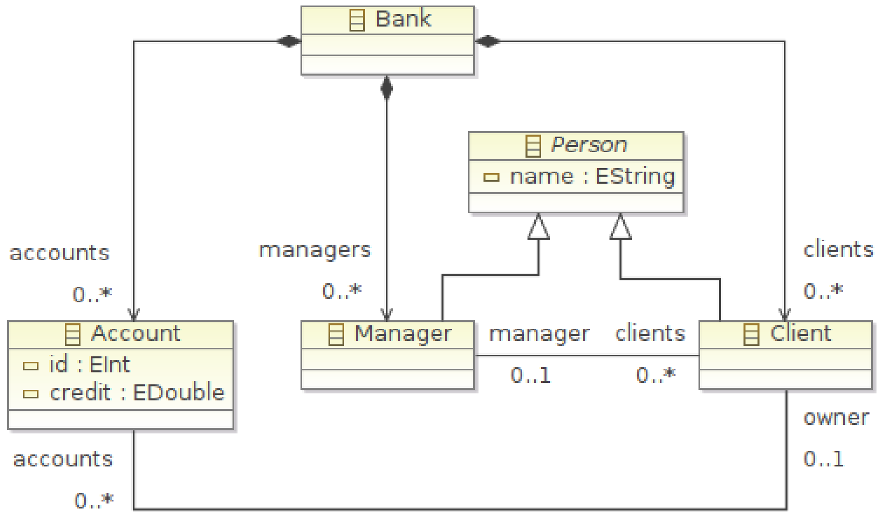


- typed attributed graphs
  - with node type inheritance
  - Containment constraints



Transformation rules need to comply with containment constraints

Henshin in action 1: EMF meta-models and models

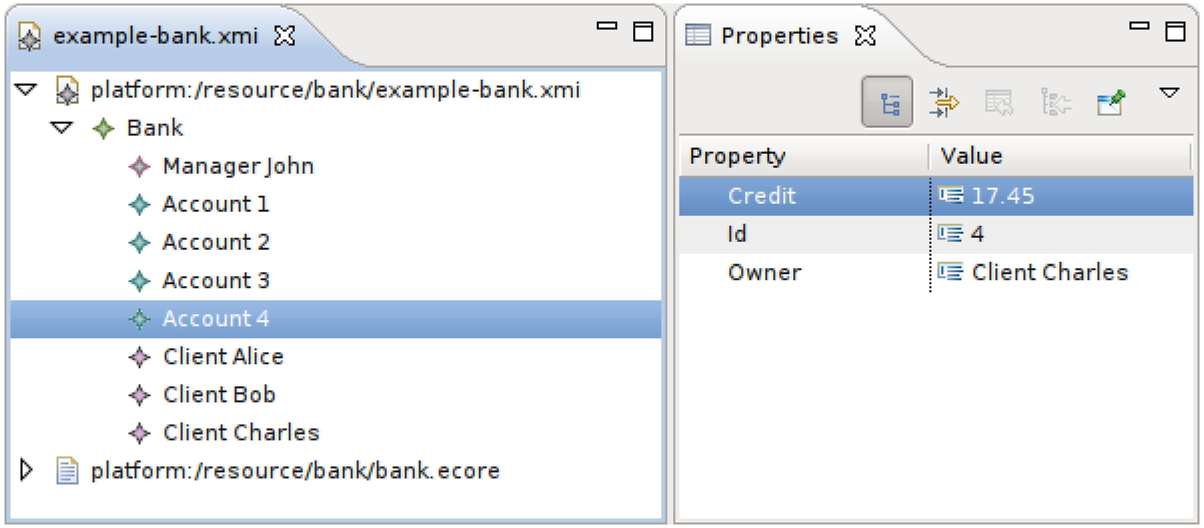


Eclipse Modeling Framework:

- base technology for modeling in Eclipse
- supports various technologies
    - graphical editors
    - model query, comparison, transformation etc.

Structured data models

- Classes with references (instead of associations)
- Containment
- Resource Sets

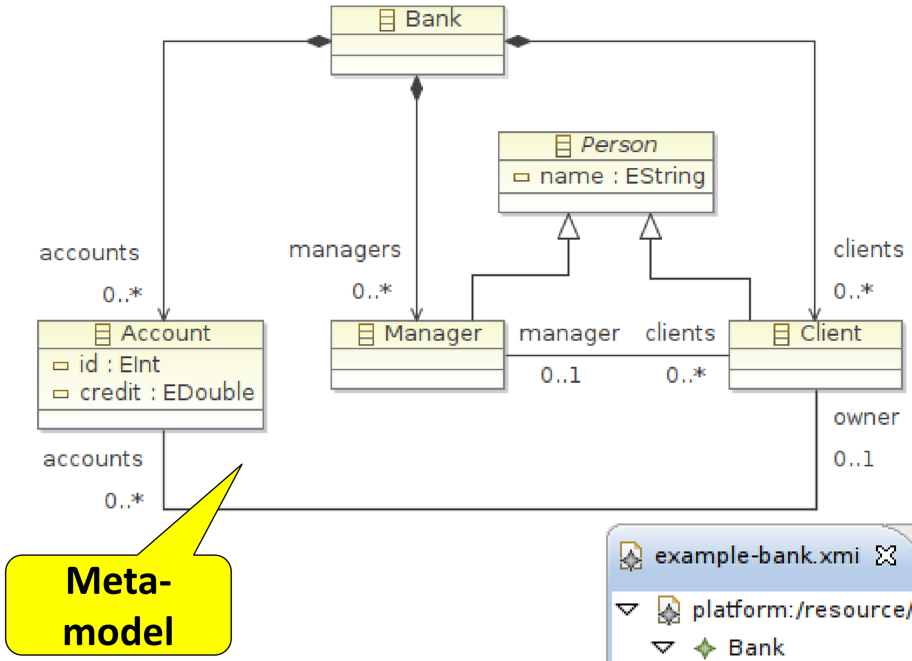


platform:/resource/bank/example-bank.xmi
- Bank
    - Manager John
    - Account 1
    - Account 2
    - Account 3
    - Account 4
    - Client Alice
    - Client Bob
    - Client Charles

platform:/resource/bank/bank.ecore

Property	Value
Credit	17.45
Id	4
Owner	Client Charles

Henshin in action 1: EMF meta-models and models

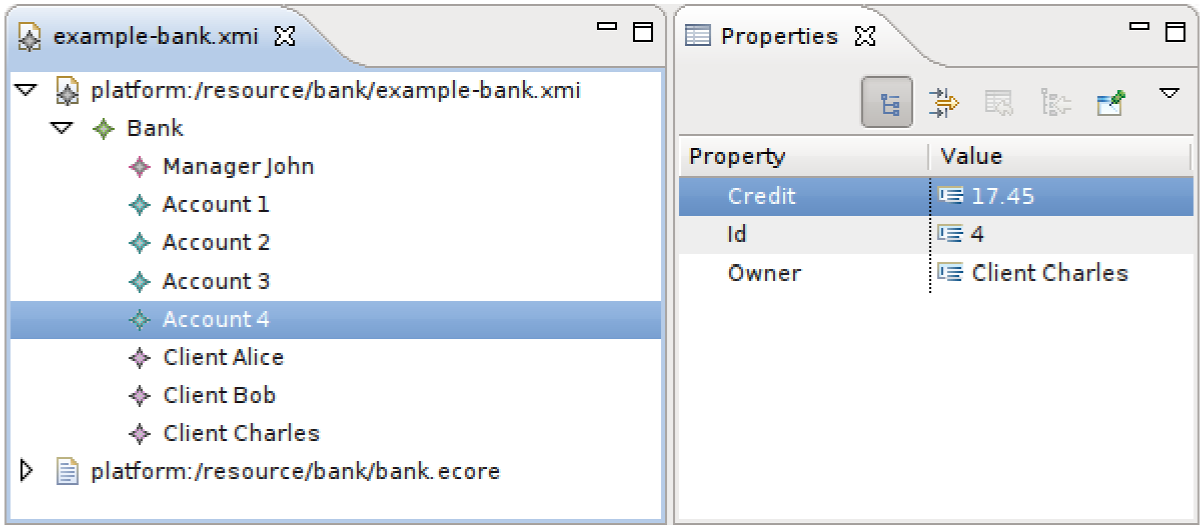


Eclipse Modeling Framework:

- base technology for modeling in Eclipse
- supports various technologies
  - graphical editors
  - model query, comparison, transformation etc.

Structured data models

- Classes with references (instead of associations)
- Containment
- Resource Sets

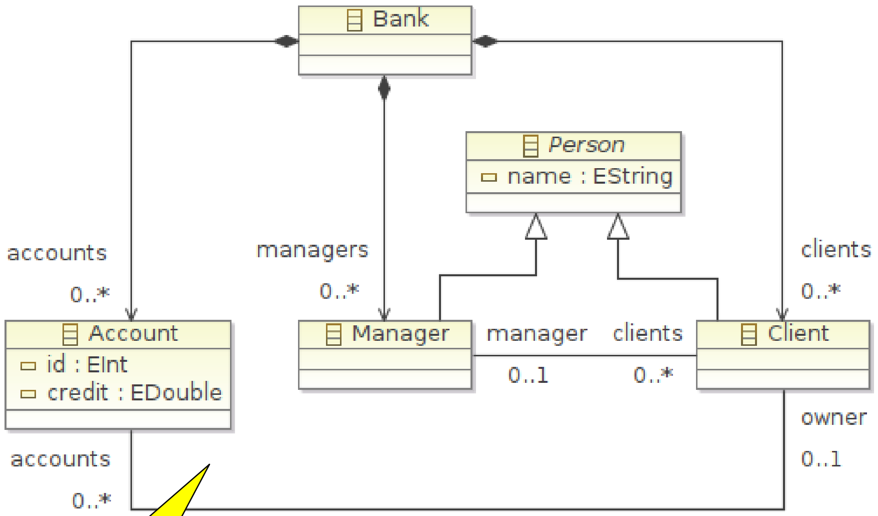


platform:/resource/bank/example-bank.xmi
- Bank
  - Manager John
  - Account 1
  - Account 2
  - Account 3
  - Account 4
  - Client Alice
  - Client Bob
  - Client Charles

platform:/resource/bank/bank.ecore

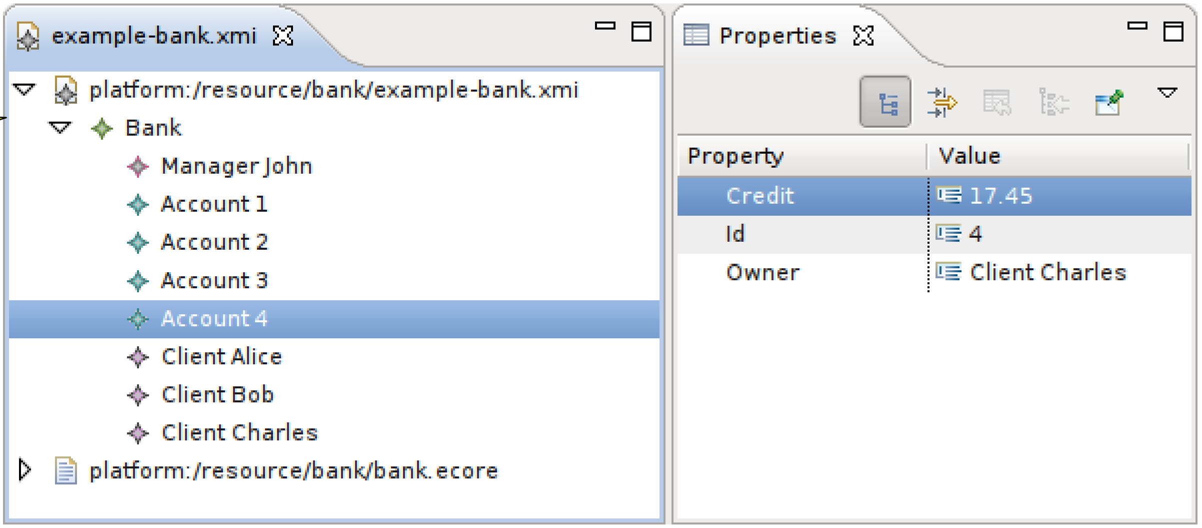
Property	Value
Credit	17.45
Id	4
Owner	Client Charles

Henshin in action 1: EMF meta-models and models



Eclipse Modeling Framework:

- base technology for modeling in Eclipse
- supports various technologies
  - graphical editors
  - model query, comparison, transformation etc.



example-bank.xmi

- platform:/resource/bank/example-bank.xmi
  - Bank
    - Manager John
    - Account 1
    - Account 2
    - Account 3
    - Account 4
    - Client Alice
    - Client Bob
    - Client Charles
- platform:/resource/bank/bank.ecore

Properties

Property	Value
Credit	17.45
Id	4
Owner	Client Charles

Structured data models

- Classes with references (instead of associations)
- Containment
- Resource Sets

Language definition: Henshin meta-model

Background: Rule Applications are a Double Pushout

Left-hand side (LHS) of rule:
Deleted + Preserved elements

Mappings

Right-hand side (RHS) of rule:
Created + Preserved elements

$$\begin{tikzcd}
L \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Match

Input
graph

Modified
graph

Language definition: Henshin meta-model

Background: Rule Applications are a Double Pushout

Left-hand side (LHS) of rule:
Deleted + Preserved elements

Mappings

Right-hand side (RHS) of rule:
Created + Preserved elements

Negative and positive application conditions

Match

$$\begin{tikzcd}
L \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Input graph

Modified graph

Language definition: Henshin meta-model

Background: Rule Applications are a Double Pushout

Left-hand side (LHS) of rule:
Deleted + Preserved elements

Mappings

Right-hand side (RHS) of rule:
Created + Preserved elements

$$\begin{tikzcd}
L \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Negative and positive application conditions

Match

Input graph

Modified graph