# Metamodel matching for automatic generation of model transformation .... Or other purposes

Sattose

Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade et Clémentine Nebut

LIRMM, CNRS, Université Montpellier 2 et France Télécom

mai 2009

# Context

## Context

- Increasing use of MDE approaches ;
- Many tools for MDE ;
- Many models and metamodels.

# Context

## Context

- Increasing use of MDE approaches ;
- Many tools for MDE ;
- Many models and metamodels.

## Explosion of the metamodel number

Several metamodels for a same domain

# Several metamodels for a same domain

## Trace metamodels

- ETrace metamodel
- EML trace metamodel (Epsilon Merging Language)
- ATL trace metamodel
- ...

## Class metamodels

- Ecore metamodel
- UML metamodel
- Kermeta metamodel
- Minjava metamodel
- ...

# About metamodel compatibility

## Ideally:

- Tools should allow to use all the models conform with common metamodels of a given domain
- For instance: a graphical editor for class models with UML, Kermeta, or ECore

## But:

- Compatibility is ensured by hand-written transformations
- tedious, error-prone, cost

# Towards automatic generation of model transformations (for similar meta-model)

## Metamodel alignment

- Generation of mappings between metamodel elements $\longrightarrow$ metamodel alignment
- The alignment is used to produce the transformation code

## Using schema *matching* techniques

Coming from:

- Database domain *matchings* of database schemas
- Semantic web : ontology alignement, XML schema *matching*

# The proposed approach

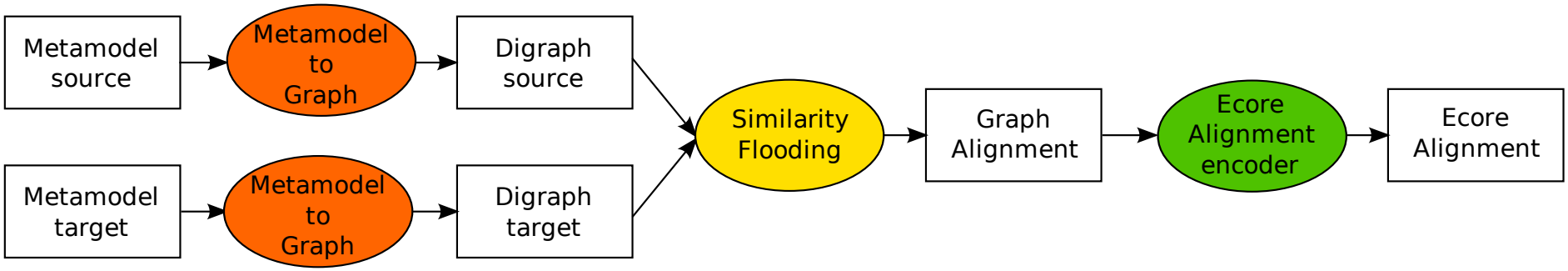
## Similarity Flooding [Melnik and al., 2002] for *matching*

- Similarity flooding works on labeled directed graphs
- Similarity flooding is easily tunable

## Using matching

- Testing several configurations for Similarity Flooding use
- Definition of a metamodel alignment
- Automatic construction of aligment models

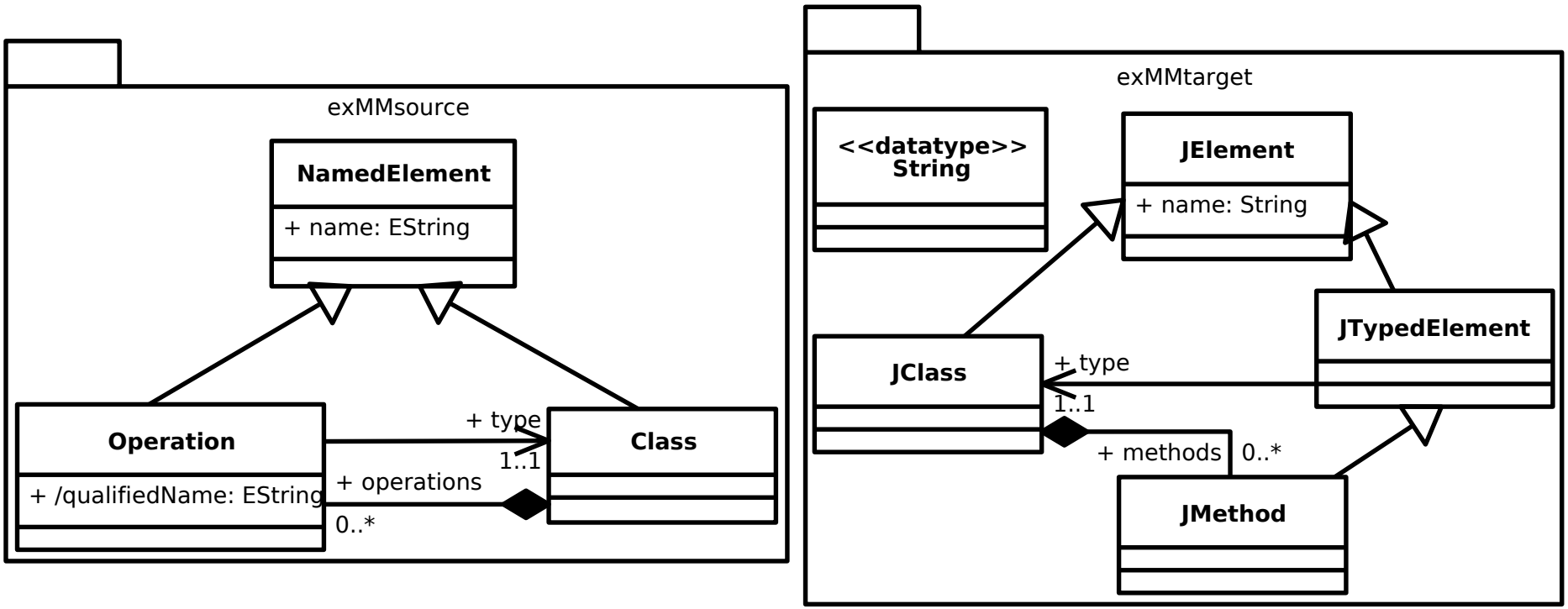
# Three steps



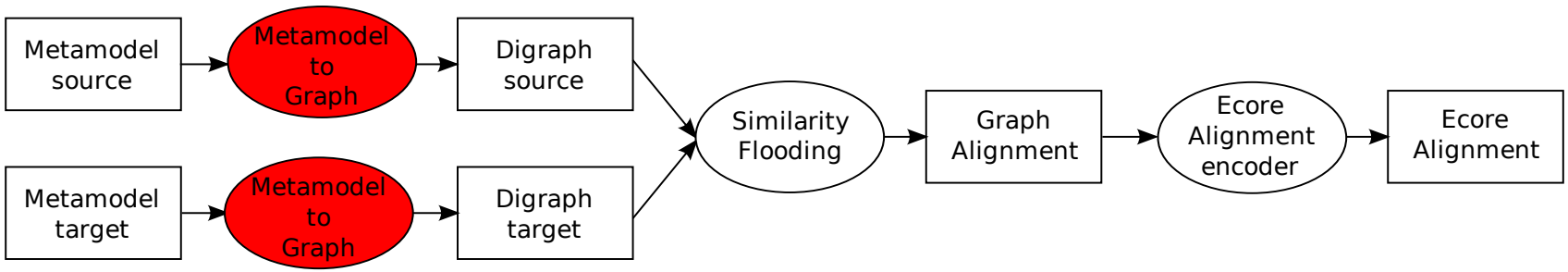

## The three steps

1. From metamodels to graphs
2. Application of Similarity Flooding
3. Construction of an alignement metamodel using the result of Similarity Flooding

# An example


exMMsource
NamedElement
+ name: EString
Operation
+ /qualifiedName: EString
Class
+ type
1..1
+ operations
0..*
exMMtarget
<<datatype>>
String
JElement
+ name: String
JTypedElement
JClass
+ type
1..1
+ methods
0..*
JMethod

# 1. From metamodels to graphs



# Transform a metamodel into a labelled directed graph

## Input

A metamodel

## Output

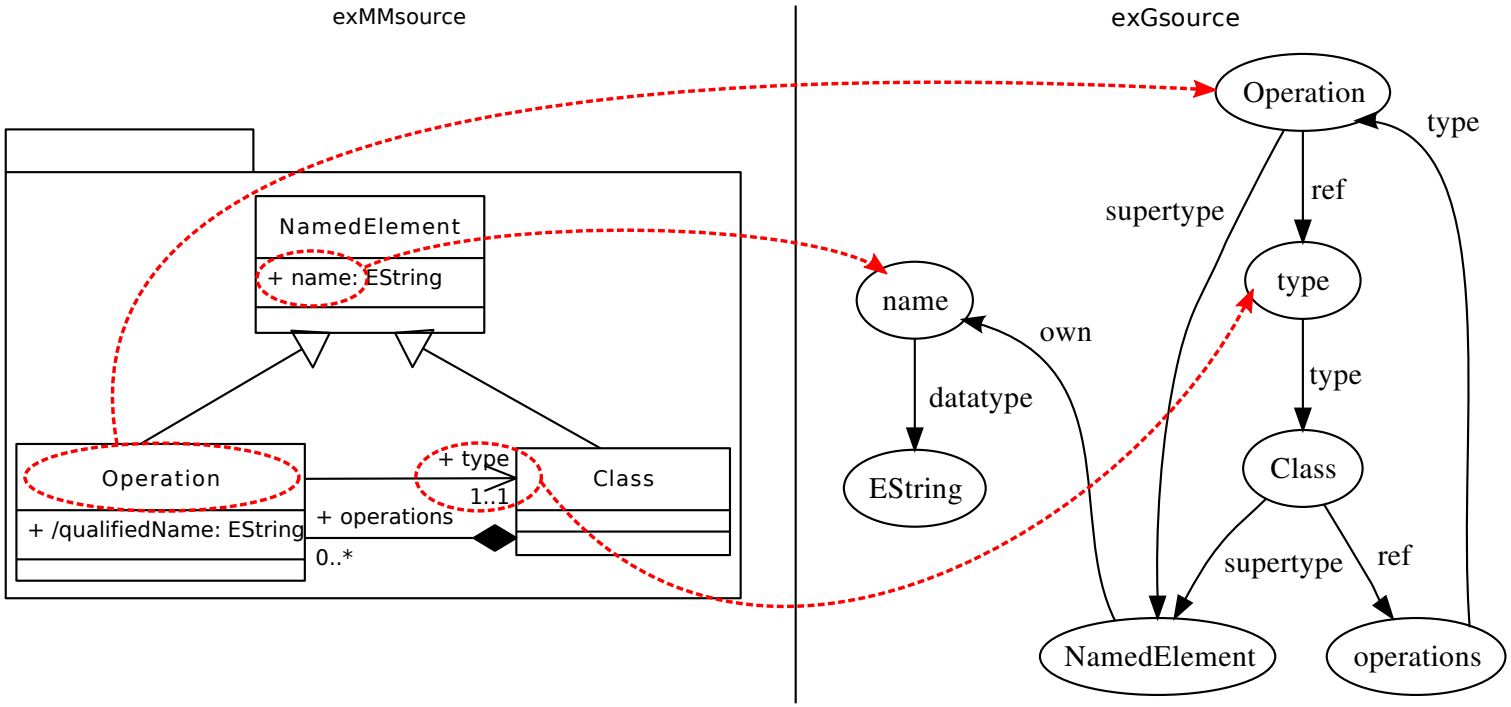
A directed labelled graph representing the model

## Objective

- Study the impact on Similarity Flooding of the configuration choice
- Six tested configurations
- Comparison of the results

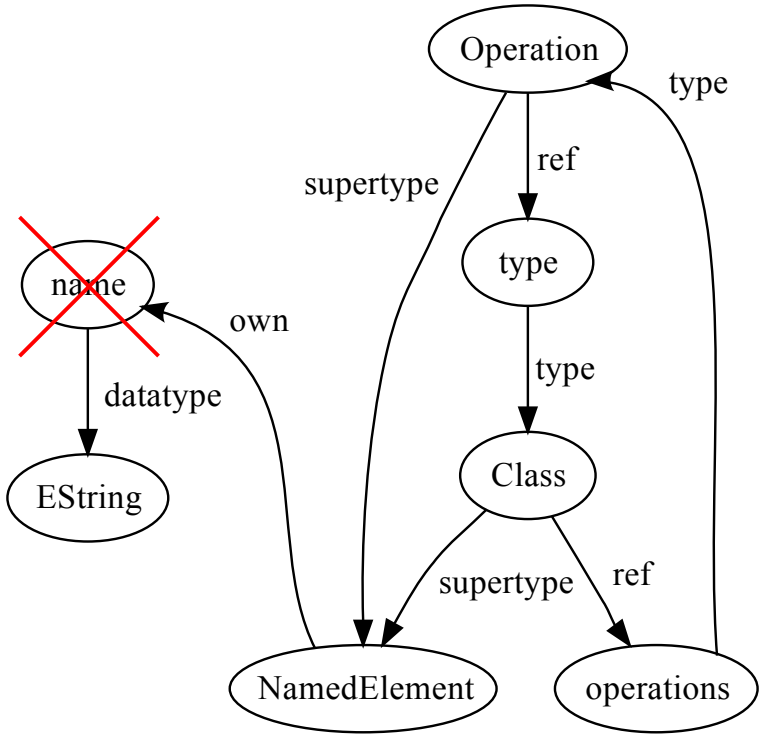
# Configuration *Minimal*

- Metamodel elements are converted into labelled nodes
- Relations are converted into labelled edges
- Derived attributes, references, operations and parameters are ignored



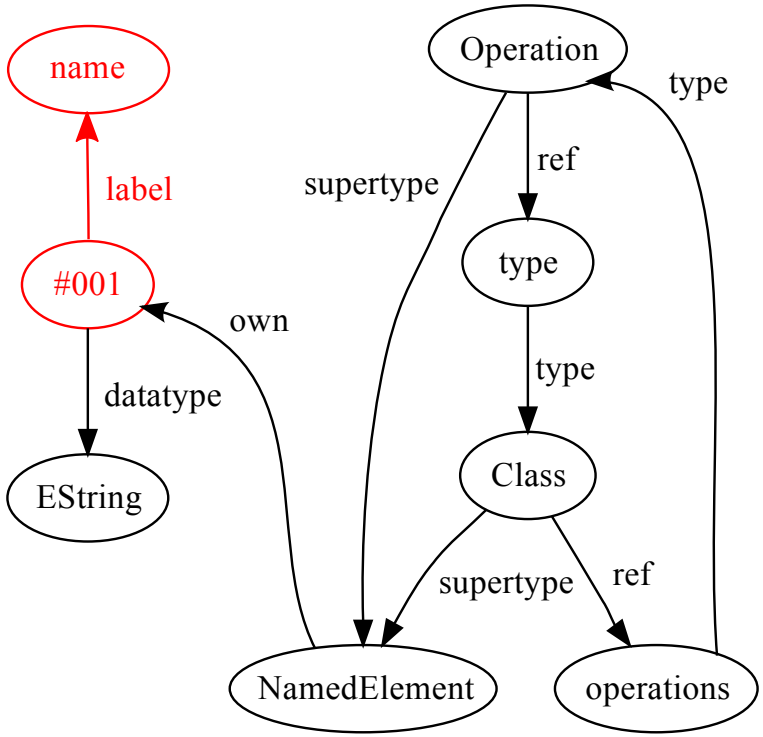
# Configuration *Basic*

Like *Minimal*, but elements and names are separated



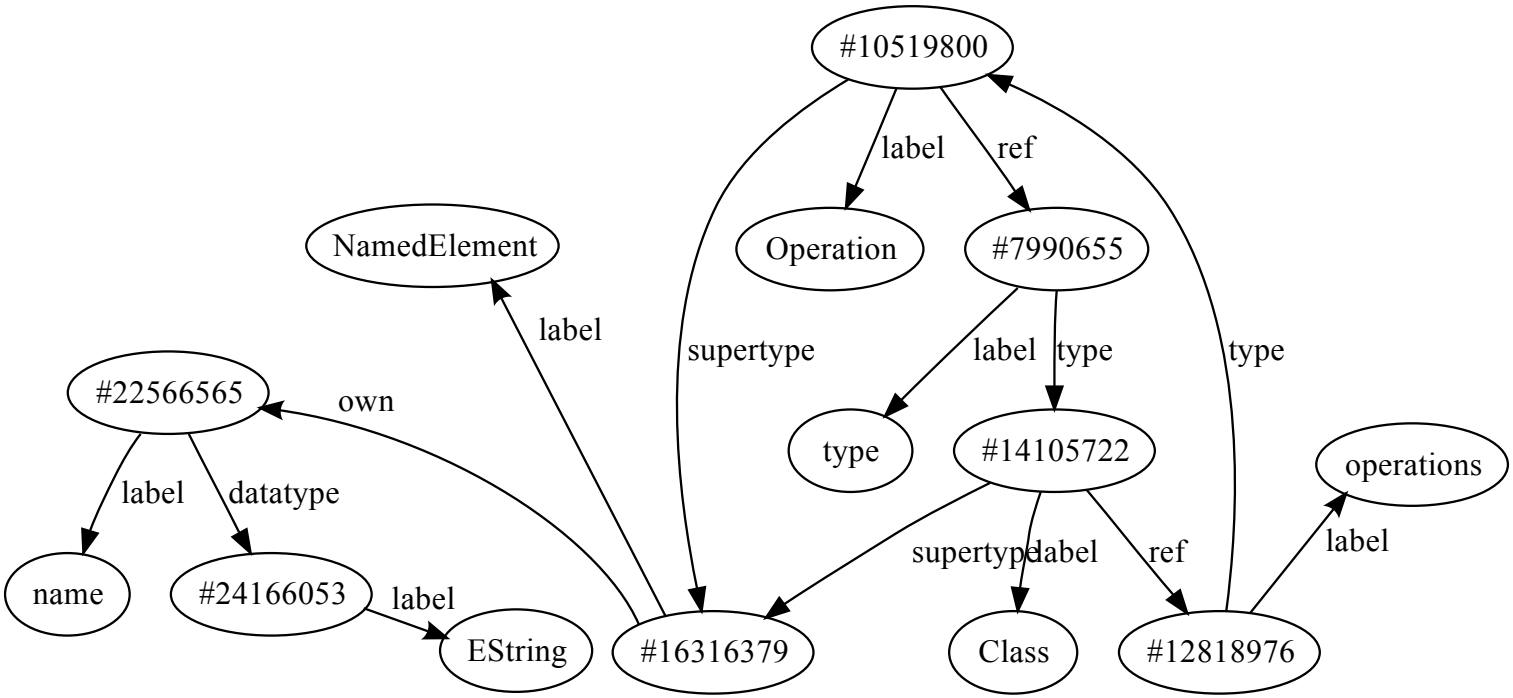
# Configuration *Basic*

Like *Minimal*, but elements and names are separated



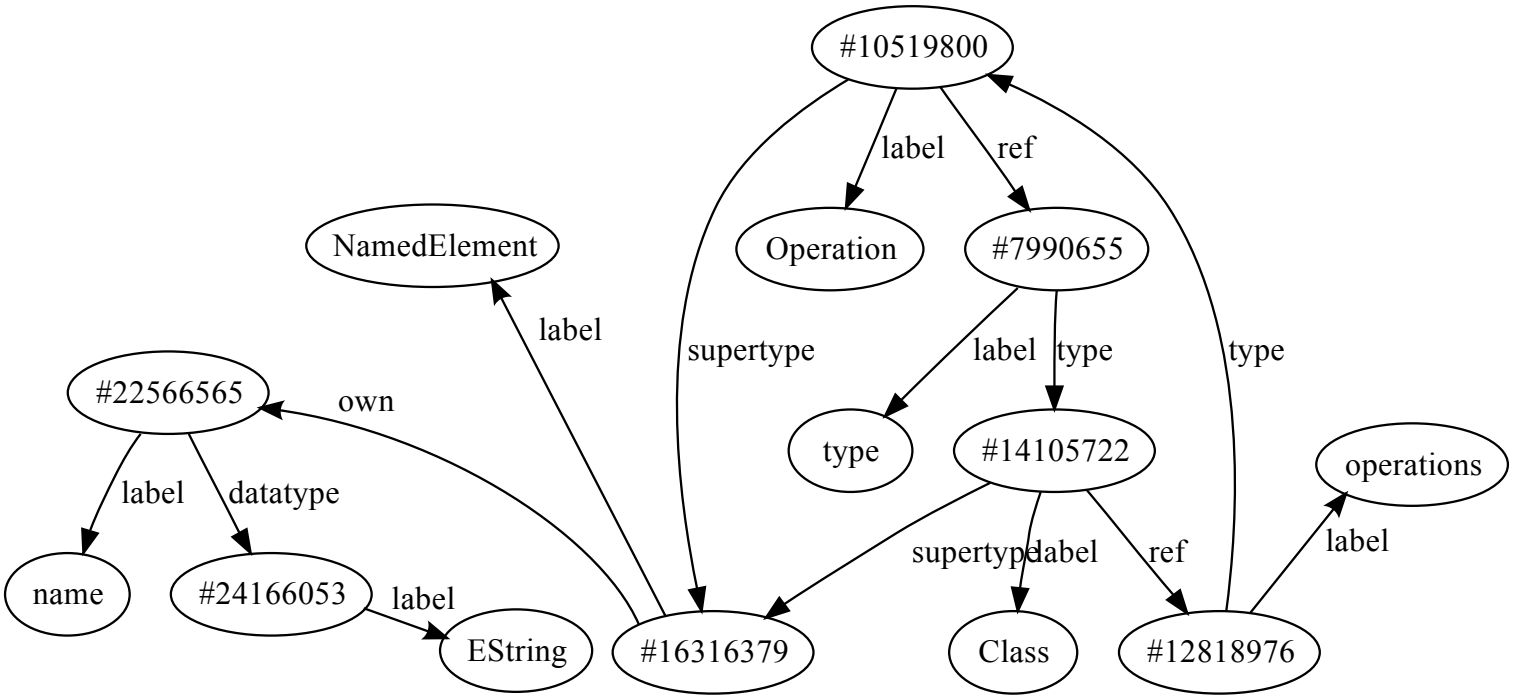
# Configuration *Basic*

Like *Minimal*, but elements and names are separated



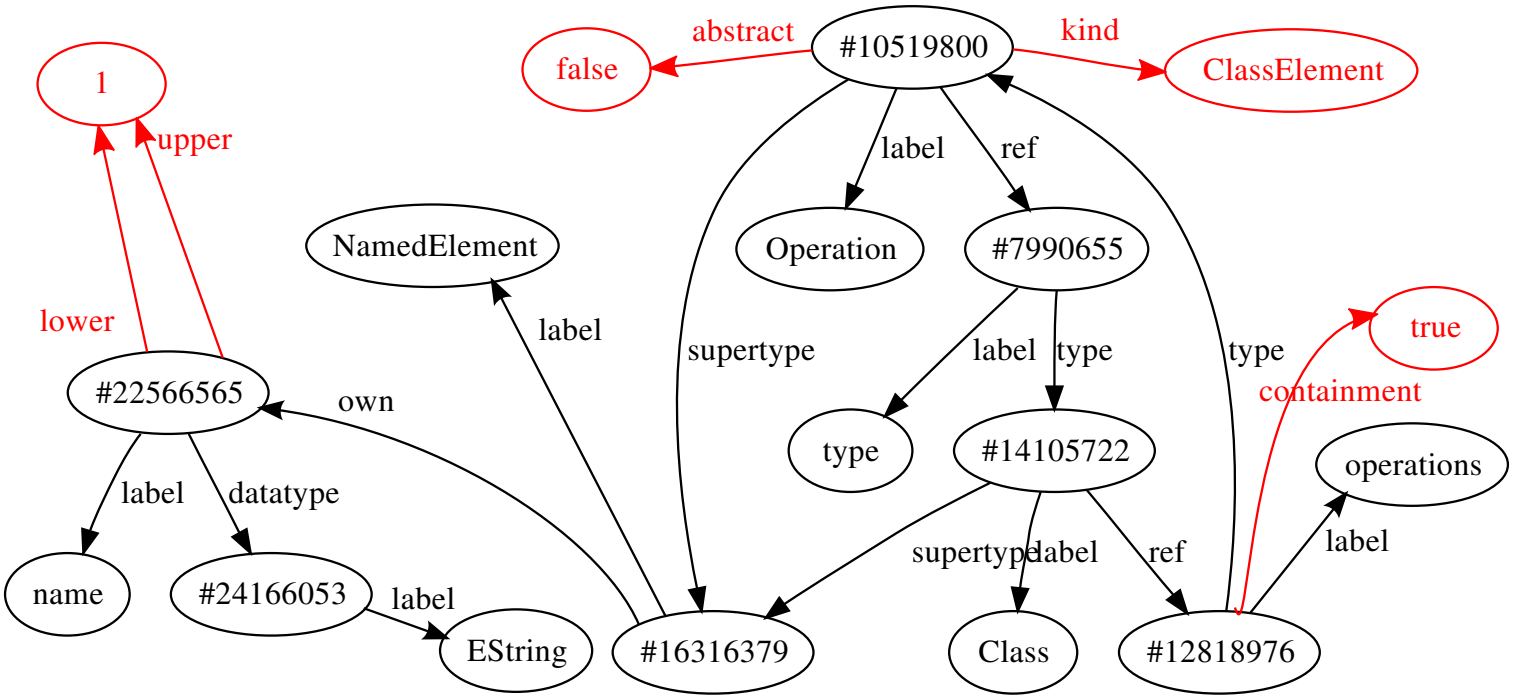
# Configuration *Standard*

Like *Basic*, but includes metaclasses, cardinality, abstract et containment



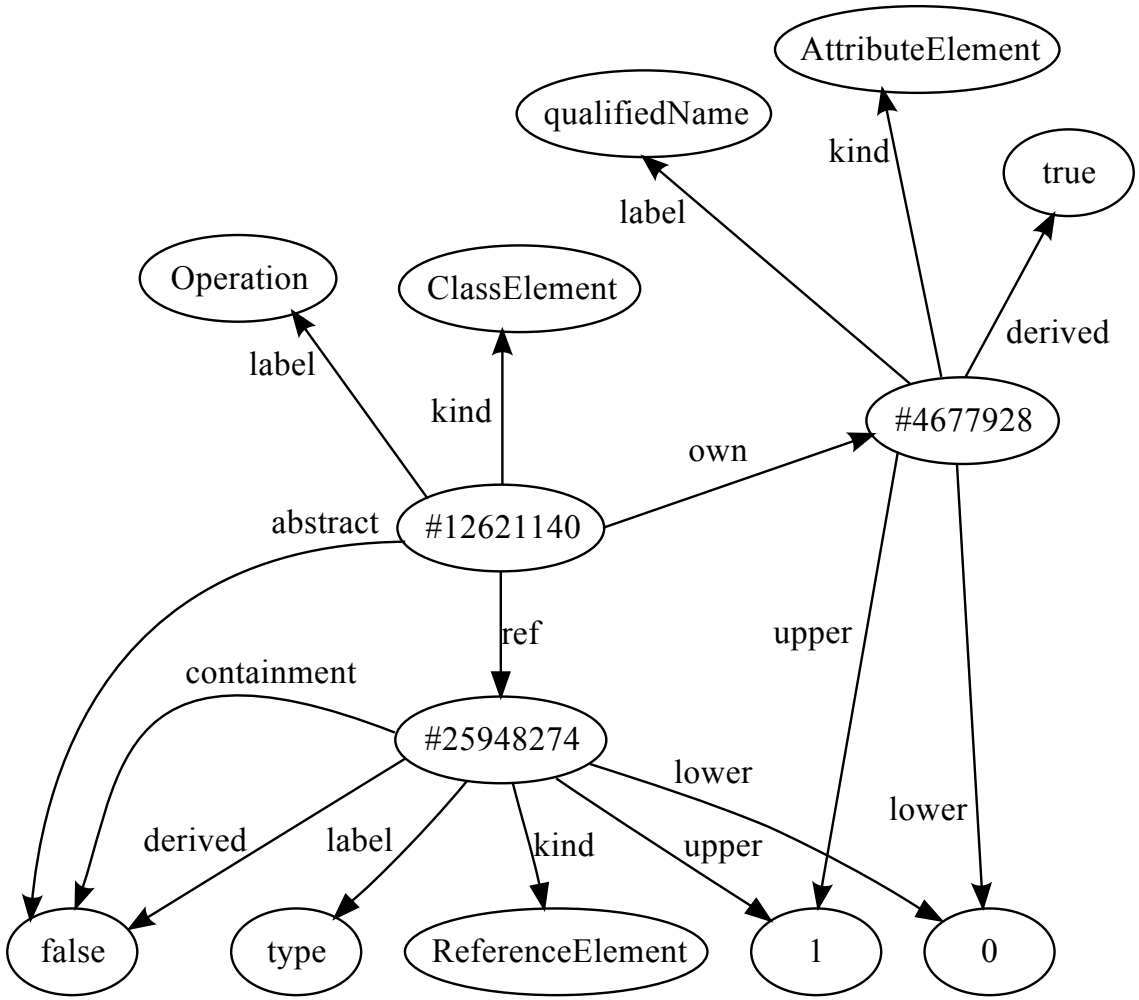
# Configuration *Standard*

Like *Basic*, but includes metaclasses, cardinality, abstract et containment



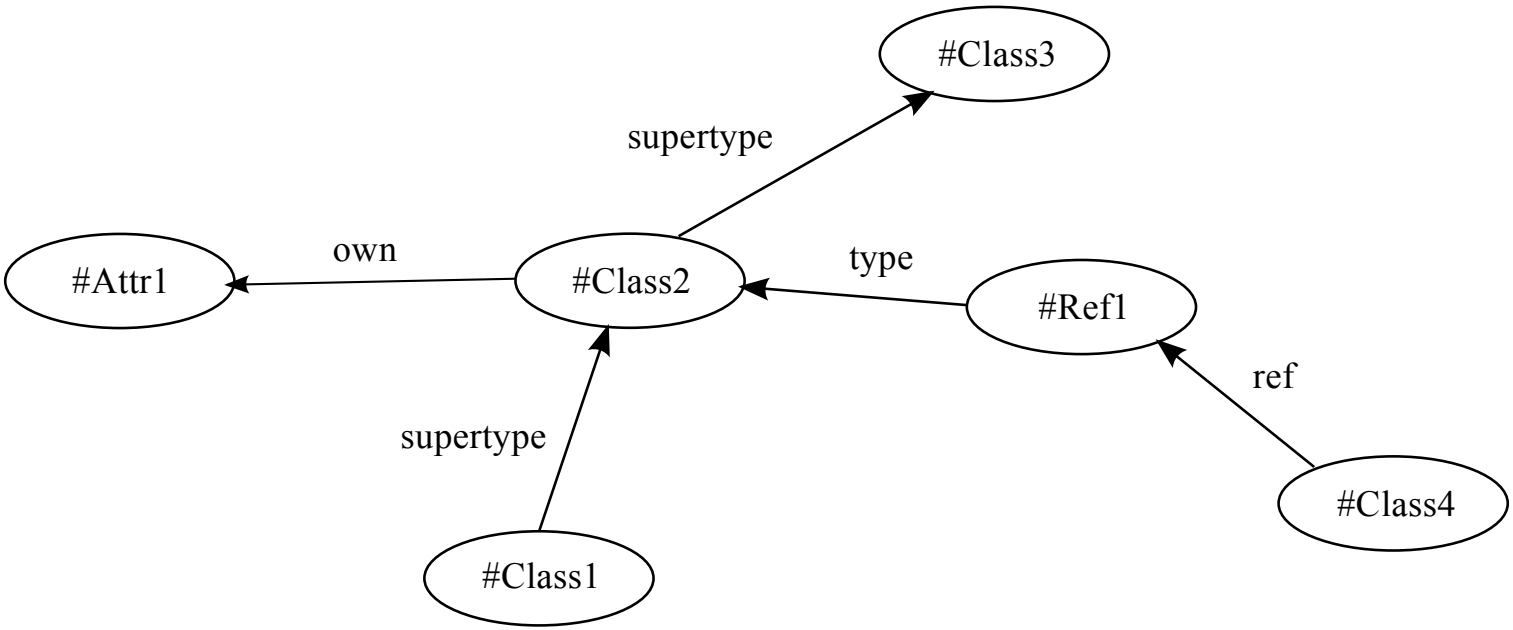
# Configuration *Full*

- Based on *Standard* configuration,
- with nodes for derived attributes and references



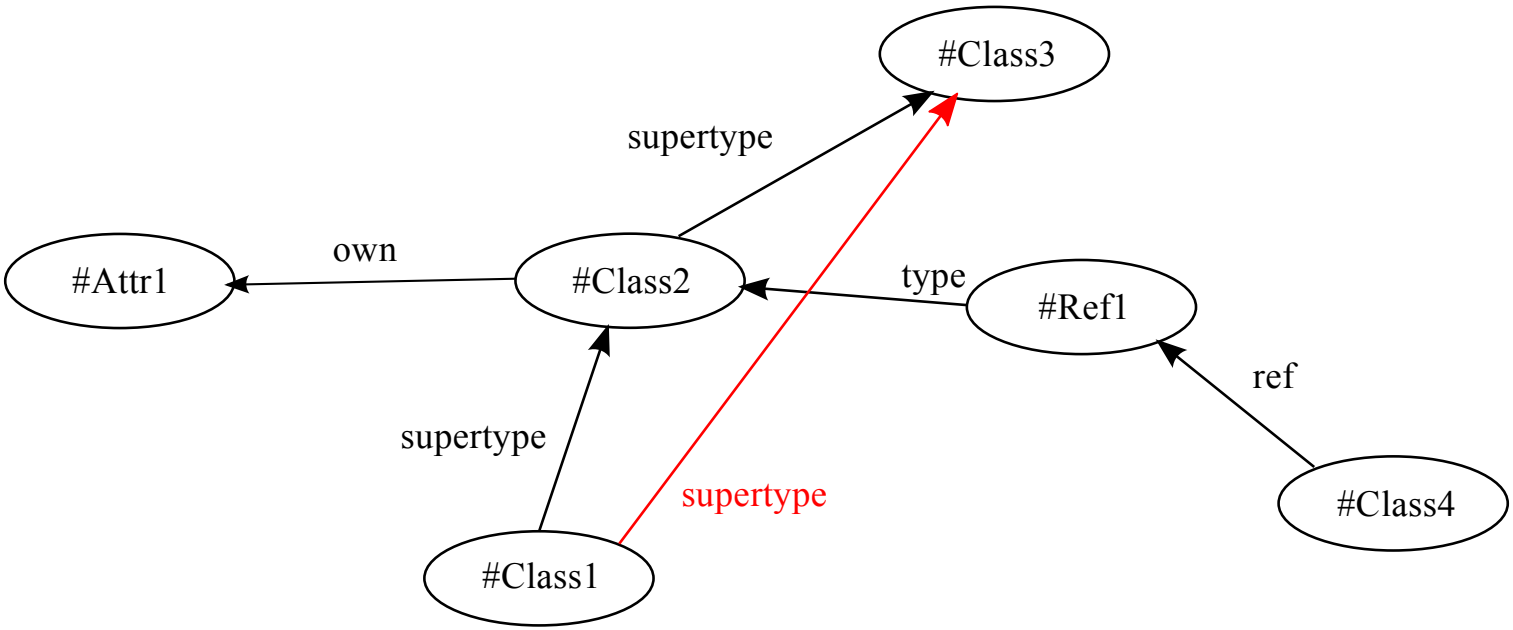
# Configuration *Saturated*

- Based on *Standard*.
- *supertype* is closed by transitivity
- For a class node, we add *own* and *ref* towards inherited attributes and references
- For a reference node, we add *type* towards all superclasses



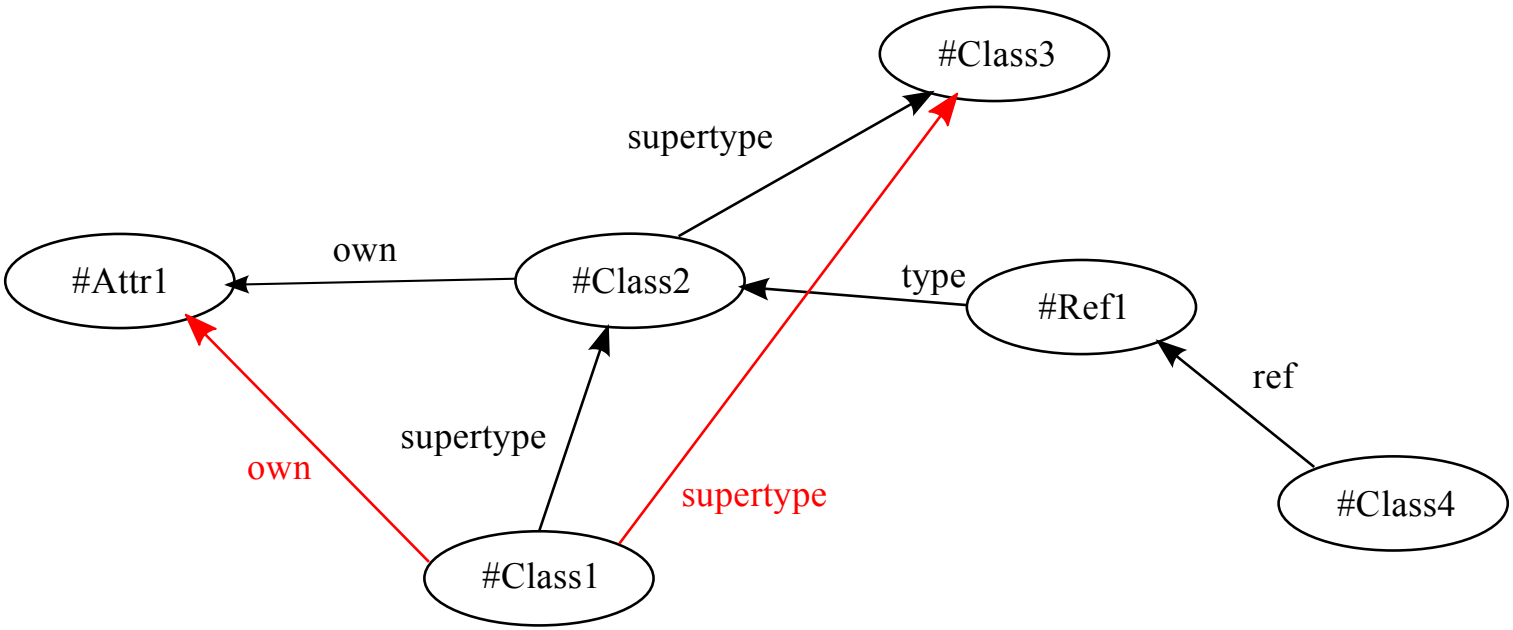
# Configuration *Saturated*

- Based on *Standard*.
- *supertype* is closed by transitivity
- For a class node, we add *own* and *ref* towards inherited attributes and references
- For a reference node, we add *type* towards all superclasses



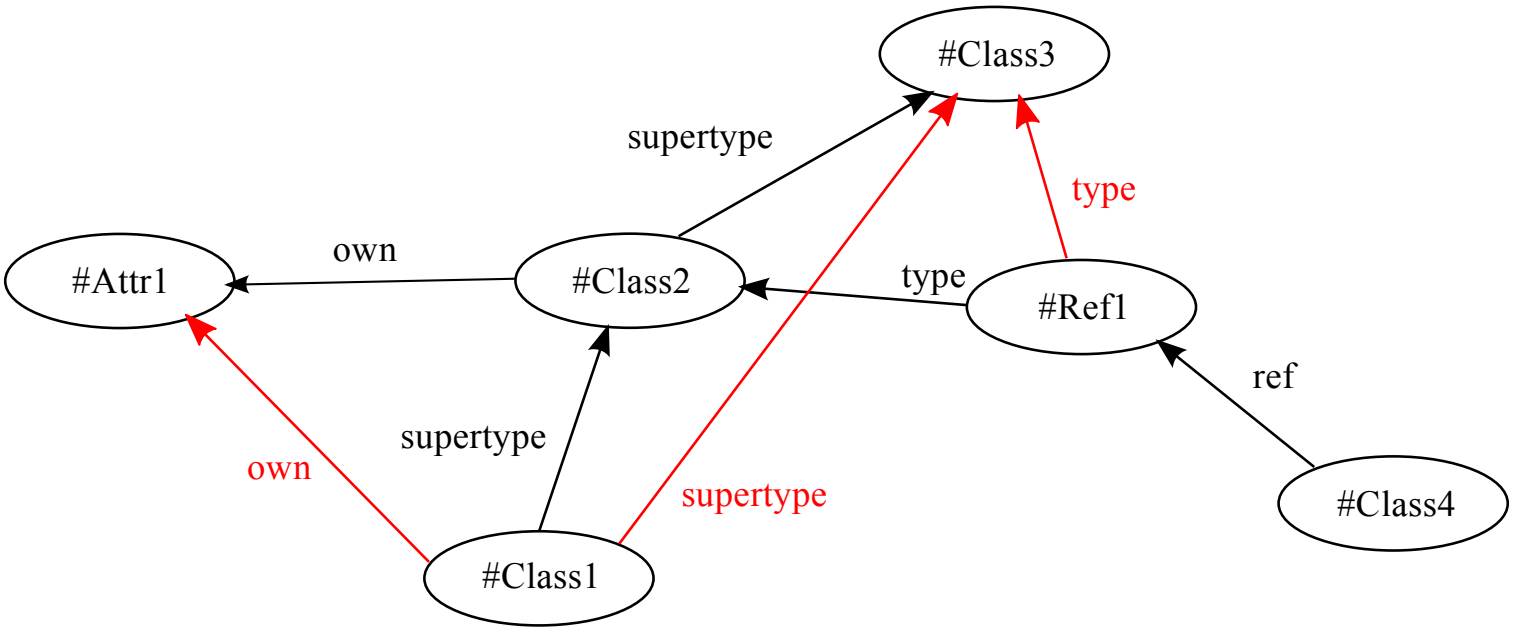
# Configuration *Saturated*

- Based on *Standard*.
- *supertype* is closed by transitivity
- For a class node, we add *own* and *ref* towards inherited attributes and references
- For a reference node, we add *type* towards all superclasses



# Configuration *Saturated*

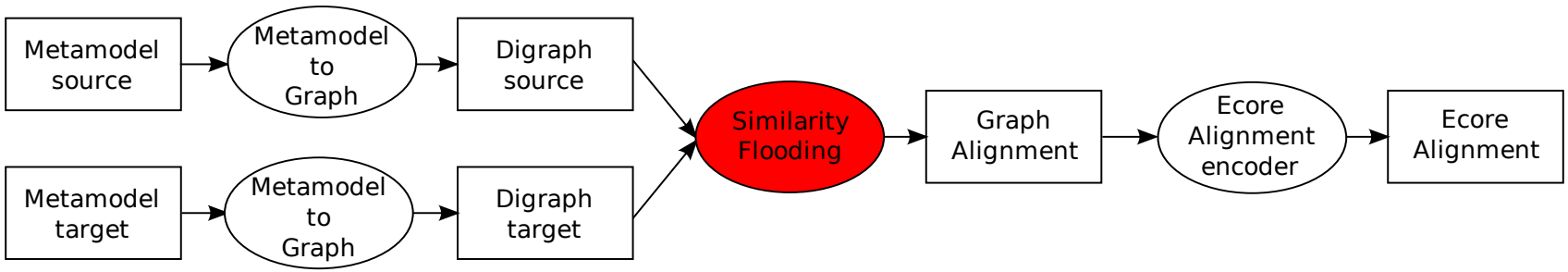
- Based on *Standard*.
- *supertype* is closed by transitivity
- For a class node, we add *own* and *ref* towards inherited attributes and references
- For a reference node, we add *type* towards all superclasses



# Configuration *Flattened*

- Based *Standard*.
- Abstract class nodes and *supertype* edges are removed,
- references typed by an abstract class are replaced by a set of references typed by concrete classes
- Links *own* and *ref* are added towards attributes and references of superclasses when a *supertype* edge is removed

# 2. Similarity Flooding



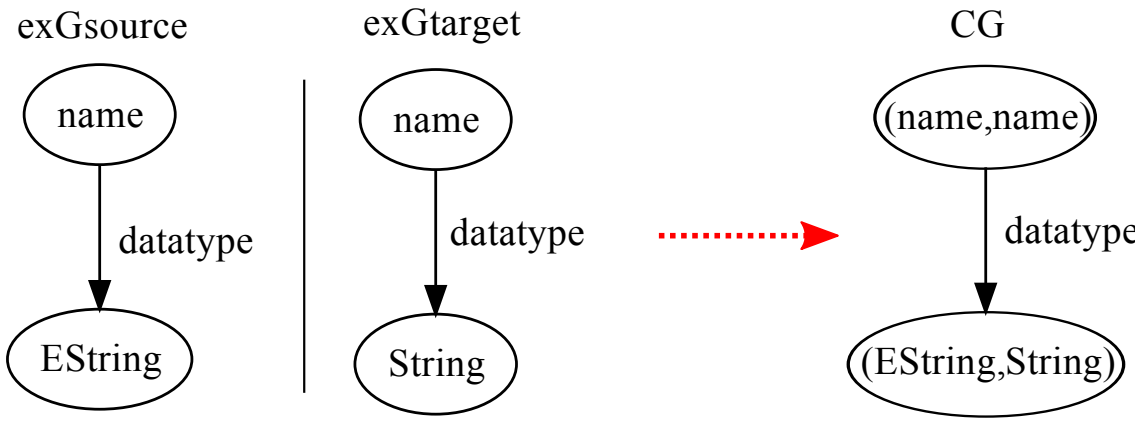
# Similarity Flooding

- Computes similarities between the nodes of two labelled directed graphs
- Input: two graphs $G_{source}$ and $G_{target}$.
- Output: an association set between one node of $G_{source}$ and one node of $G_{target}$.
- A 5-steps algorithm.
- Two nodes are similar when their neighbors are

# First step: The compatibility graph

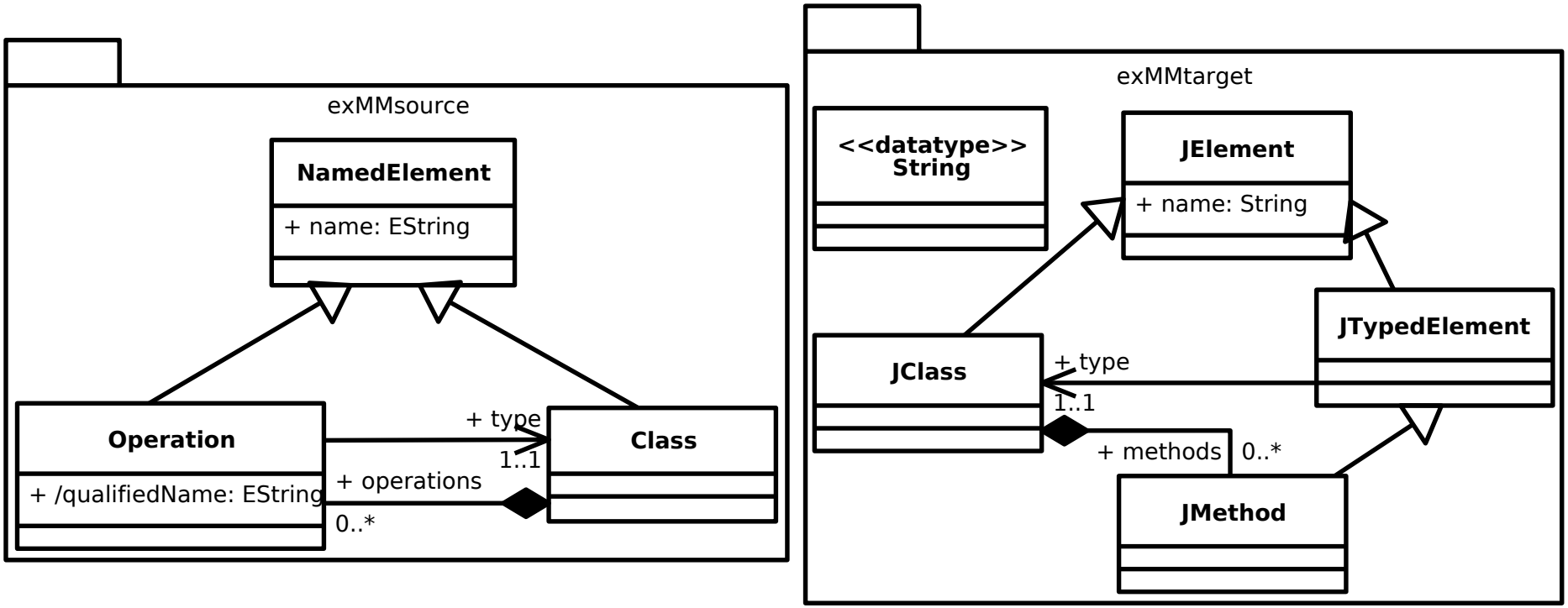
## Principe

- $G_{source}$ and $G_{target}$ are composed of triplets $(x, r, y)$,
- $CG$ is the compatibility graph,
- $((x_1, x_2), r, (y_1, y_2)) \in CG$ iff $(x_1, r, y_1) \in G_{source}$ et $(x_2, r, y_2) \in G_{target}$



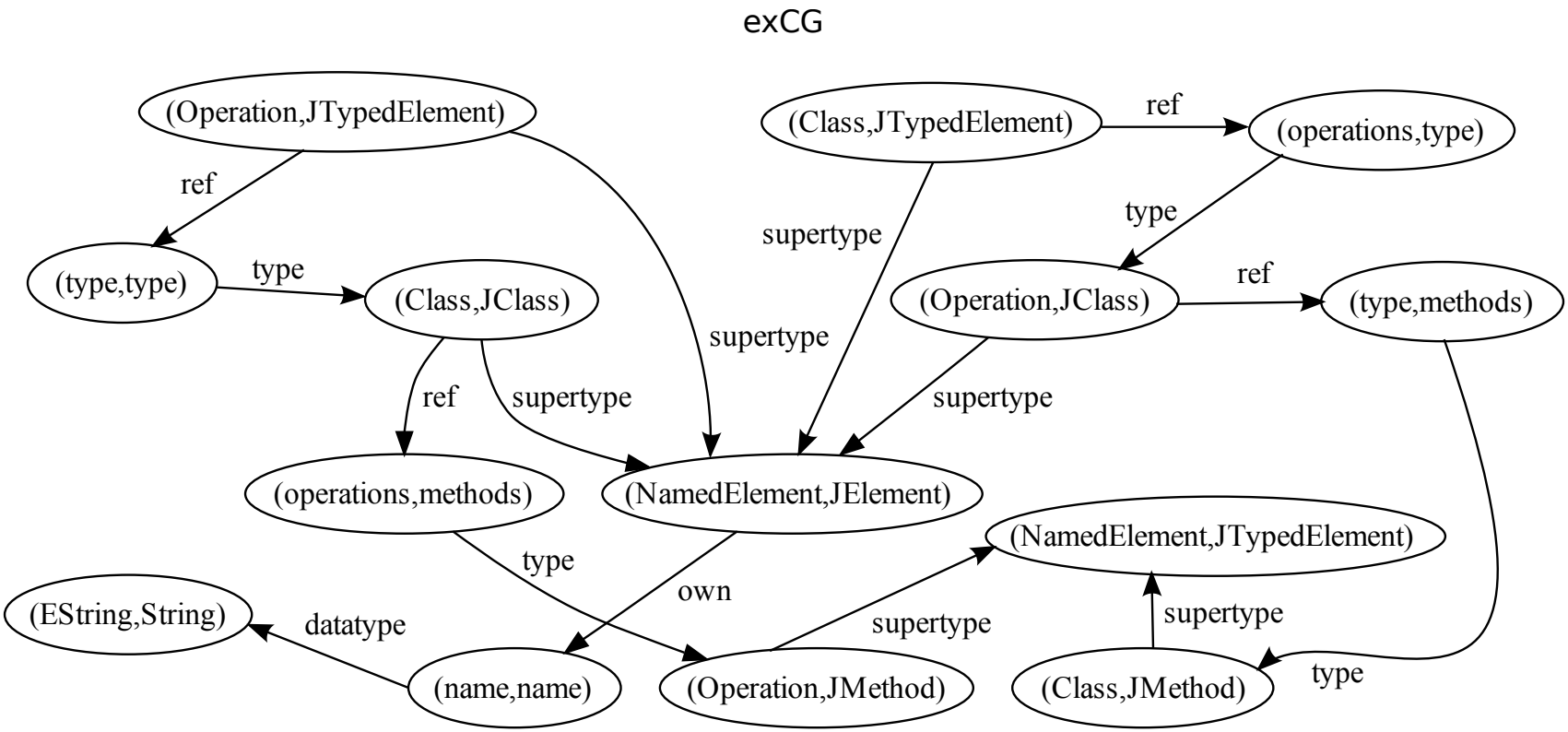
# Exemple

Example, with the configuration *Minimal*



# Exemple

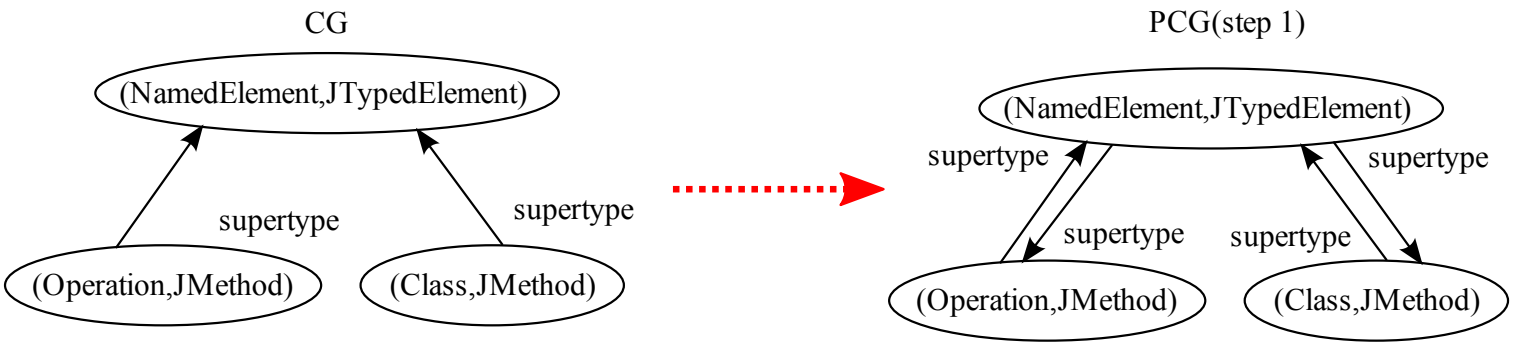
Example, with the configuration *Minimal*



# Second step: propagation graph

## Principle

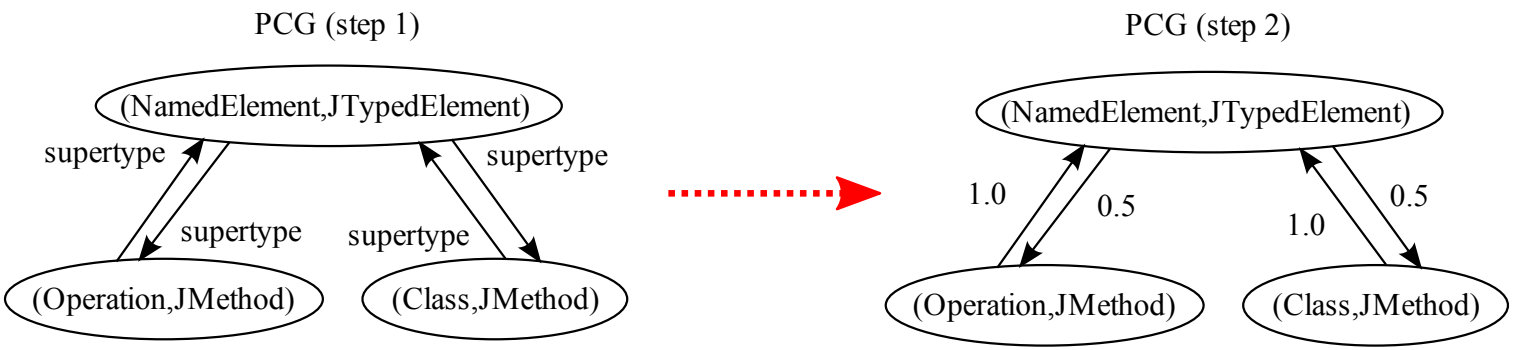
- For each $(s, r, t)$ in the compatibility graph, we add the reverse triplet $(t, r, s)$.
- $O_l^n = \#$edges with label $l$ and leaving $n$.
- Triplets $(s, r, t)$ are replaced by $(s, \frac{1}{O_r^s}, t)$



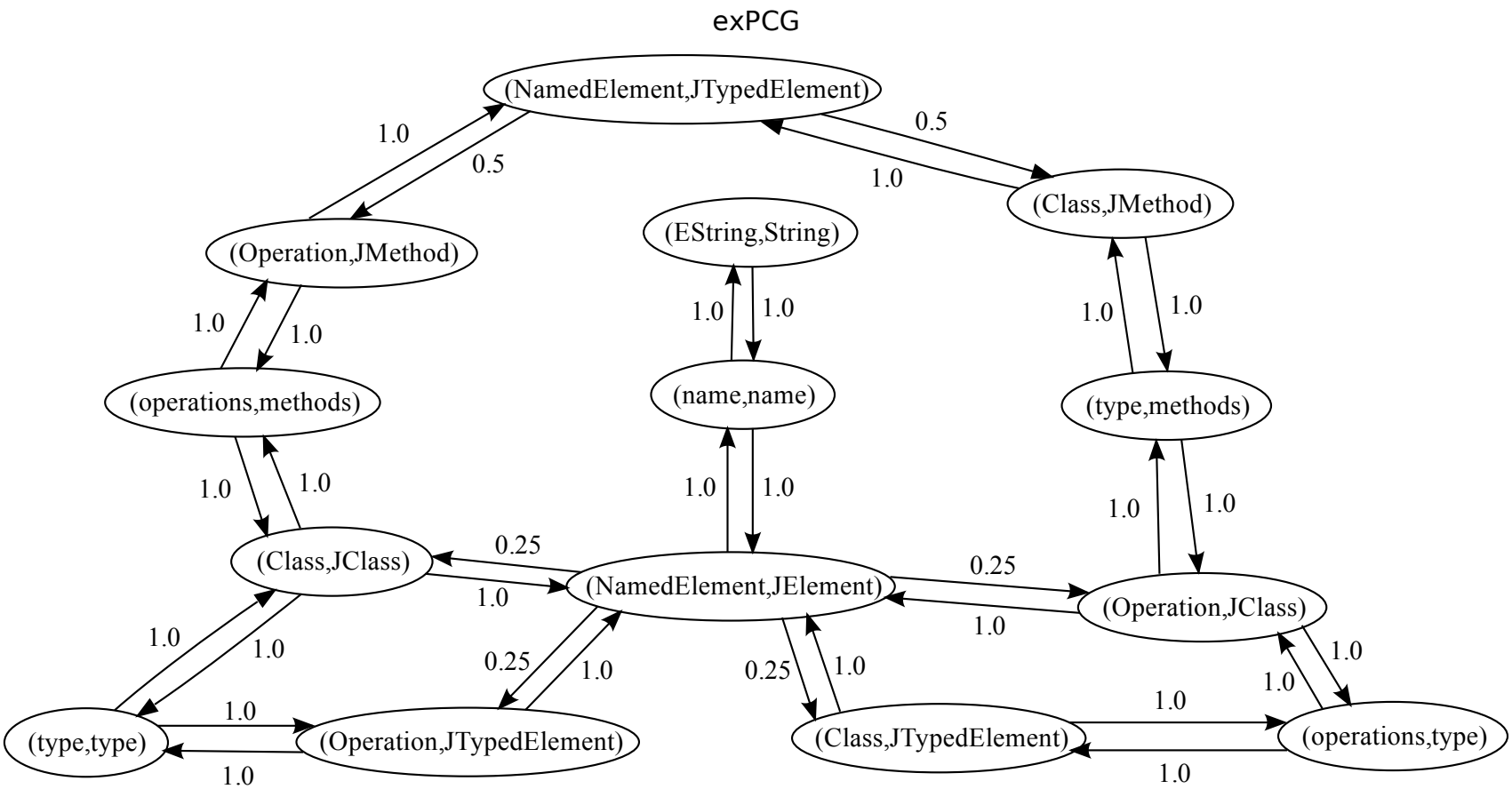
# Second step: propagation graph

## Principle

- For each $(s, r, t)$ in the compatibility graph, we add the reverse triplet $(t, r, s)$.
- $O_l^n = \#$edges with label $l$ and leaving $n$.
- Triplets $(s, r, t)$ are replaced by $(s, \frac{1}{O_r^s}, t)$



# Example


exPCG
(NamedElement,JTypedElement)
(Operation,JMethod)
(EString,String)
(Class,JMethod)
(operations,methods)
(name,name)
(type,methods)
(Class,JClass)
(NamedElement,JElement)
(Operation,JClass)
(type,type)
(Operation,JTypedElement)
(Class,JTypedElement)
(operations,type)
1.0
0.5
0.25

# Third step: assigning initial similarity values

## Principe

For a compatibility node $n = (x, y)$, initial value $s_n^0$ is given by:

- $O$ if $x$ or $y$ is an identifier (label begins with #)
- $1 - lev(x, y)/max(len(x), len(y))$ otherwise

With:

- $lev(x, y)$ Levenshtein distance [Levenshtein, 1966] between label of $x$ and $y$ (editing distance between two strings)
- $len(x)$ the length of $x$

# Example

- $O$ if $x$ or $y$ is an identifier (label begins with #)
- $1 - lev(x, y)/max(len(x), len(y))$ otherwise

| **Compatibility node** | **Initial similarity value** |
|---|---|
| (NamedElement,JElement) | 0.5833334 |
| (name,name) | 1.0 |
| (EString,String) | 0.85714287 |
| (NamedElement,JTypedElement) | 0.6923077 |
| (Operation,JTypedElement) | 0.23076922 |

# Fourth step: propagation and fix point calculus

## Principe

- Propagation of similarity values in the propagation graph, until finding a fix point
- Propagation formulae : at step $i$,

$$s_n^{i+1} = s_n^i + s_n^0 + \sum_{m \in I^n} w(m,n) \times (s_m^0 + s_m^i)$$

- Fixpoint: when similarity values differences is less than $\epsilon$ during two successive steps.

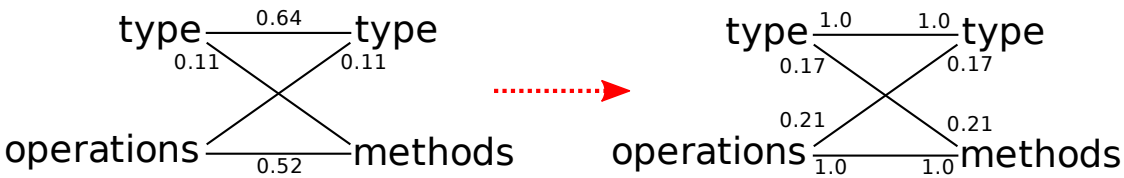
# Example

| Compatibility node | Final similarity value |
|---|---|
| (NamedElement,JElement) | 1.0 |
| (name,name) | 0.7771561 |
| (EString,String) | 0.5186626 |
| (NamedElement,JTypedElement) | 0.2985012 |
| (Operation,JTypedElement) | 0.44552472 |

# Fifth step: filtering

## Principe

- To keep best matches.
- A node of $G_{source}$ can match with several nodes of $G_{target}$.
- A relative similarity value is computed for each node
- Pairs with a similarity under a threshold are eliminated



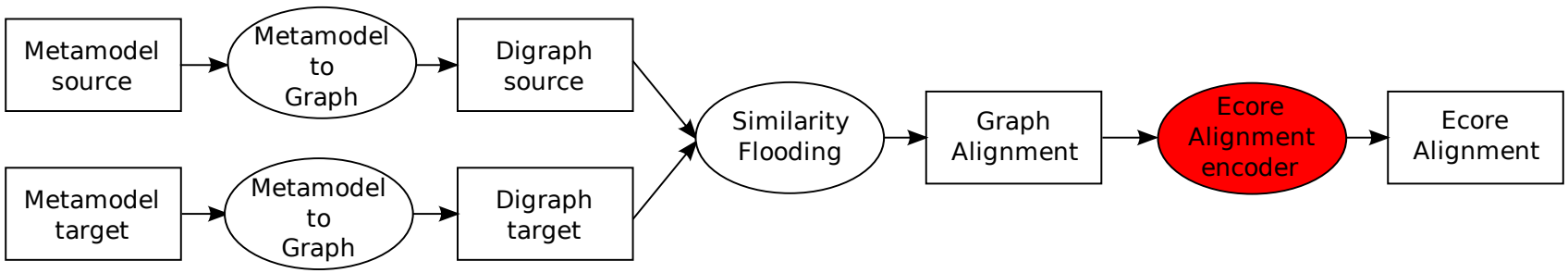
# Example

| Compatibility node | Final similarity value |
|---|---|
| (NamedElement,JElement) | 1.0 |
| (name,name) | 0.7771561 |
| (EString,String) | 0.5186626 |
| (NamedElement,JTypedElement) | 0.2985012 |
| (Operation,JTypedElement) | 0.44552472 |

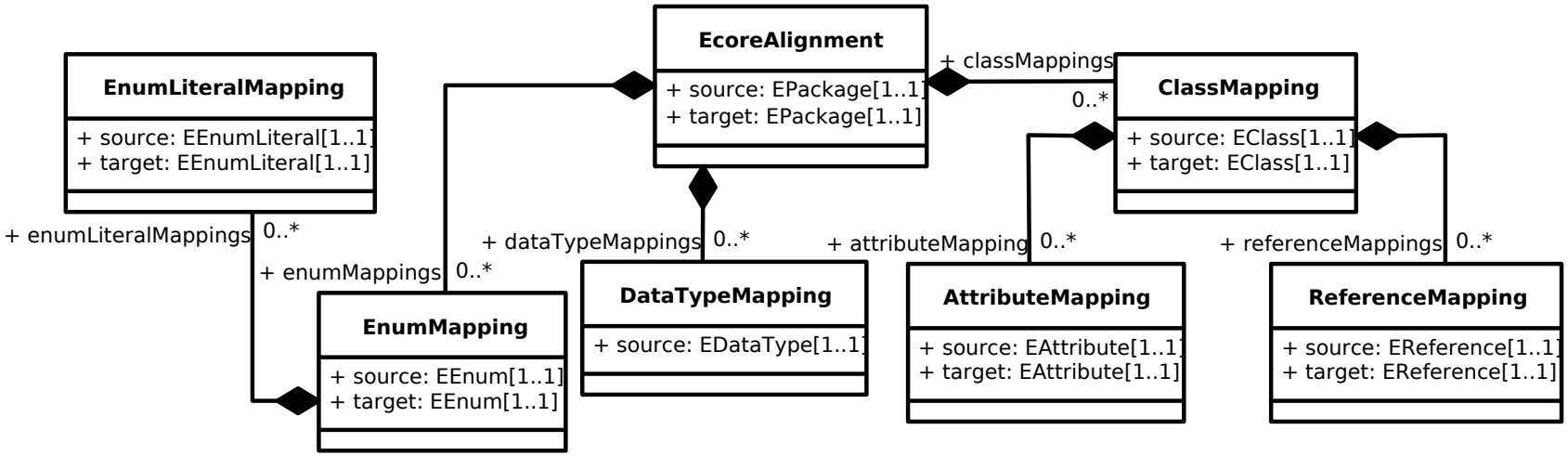
# Example

| Compatibility node | Final similarity value |
|---|---|
| (NamedElement,JElement) | 1.0 |
| (name,name) | 0.7771561 |
| (EString,String) | 0.5186626 |
| (Operation,JTypedElement) | 0.44552472 |

# 3. Encoding into an Ecore alignment: reorganizing the alignment graph



# The alignment metamodel in ECore




## Constraints

- Two classes that match are not abstract.
- Two attributes or references that match are not derived.

# Étude de cas

# Case study

## Objectif

Test the six configurations

## Data

- exMMSource $\longrightarrow$ exMMTarget
- Ecore $\longrightarrow$ Minjava
- Ecore $\longrightarrow$ Kermeta
- Ecore $\longrightarrow$ UML

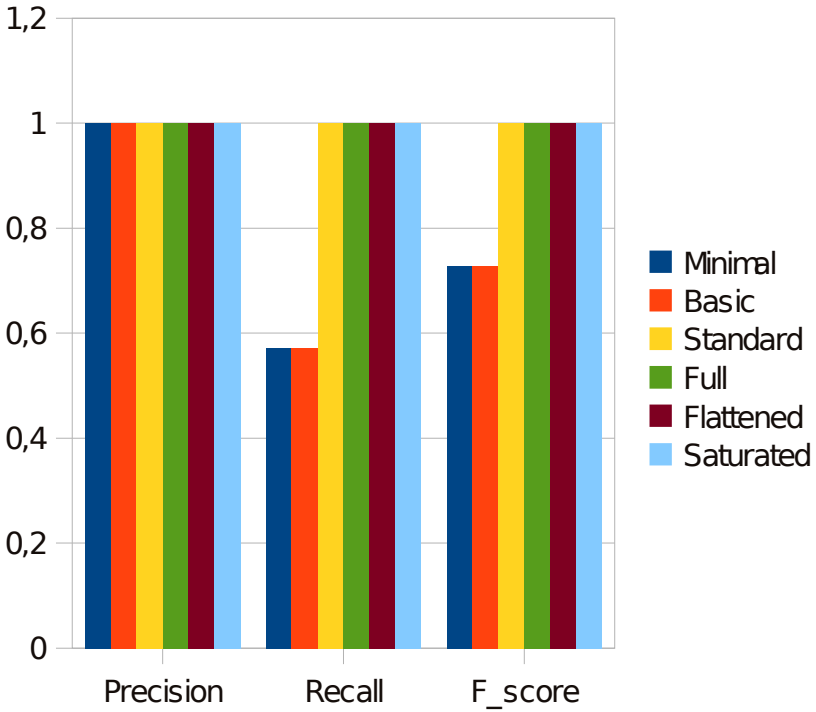
# Metrics

*precision*, *recall* et $f\_score$:

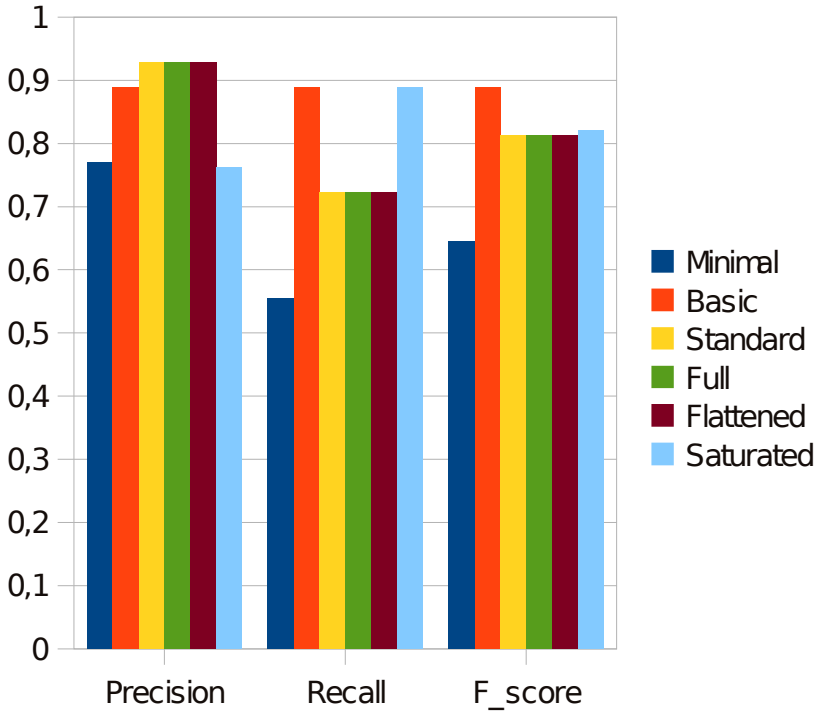
- $precision = \frac{Number\_of\_Correct\_Found\_Mappings}{Number\_of\_Total\_Found\_Mappings}$
- $recall = \frac{Number\_of\_Correct\_Found\_Mappings}{Number\_of\_Total\_Existing\_Mappings}$
- $f\_score = \frac{2 \times recall \times precision}{recall + precision}$

# Results

exMMSource → exMMTarget

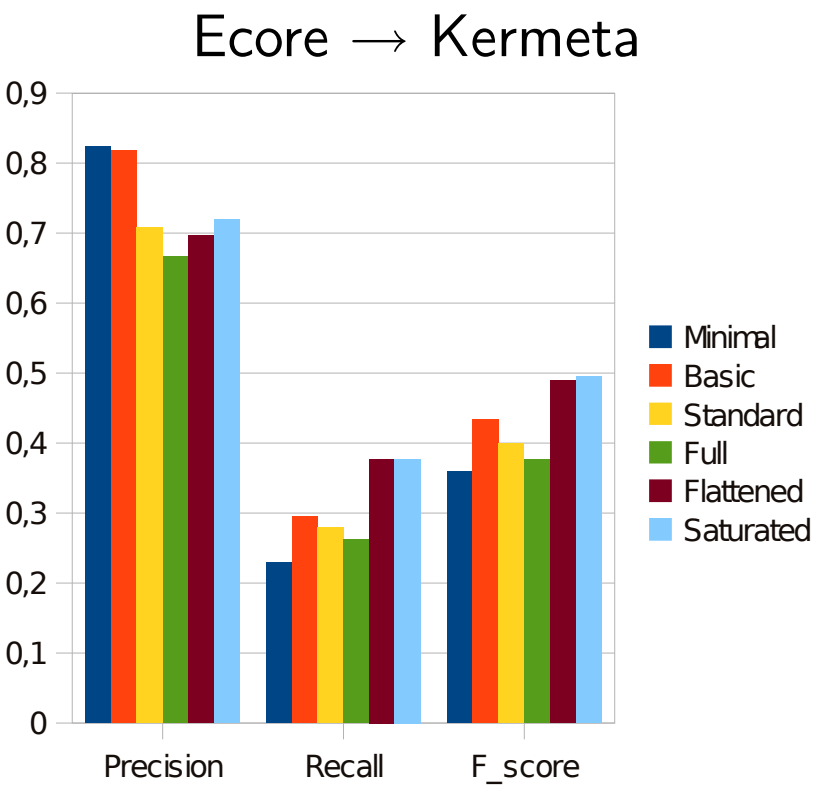


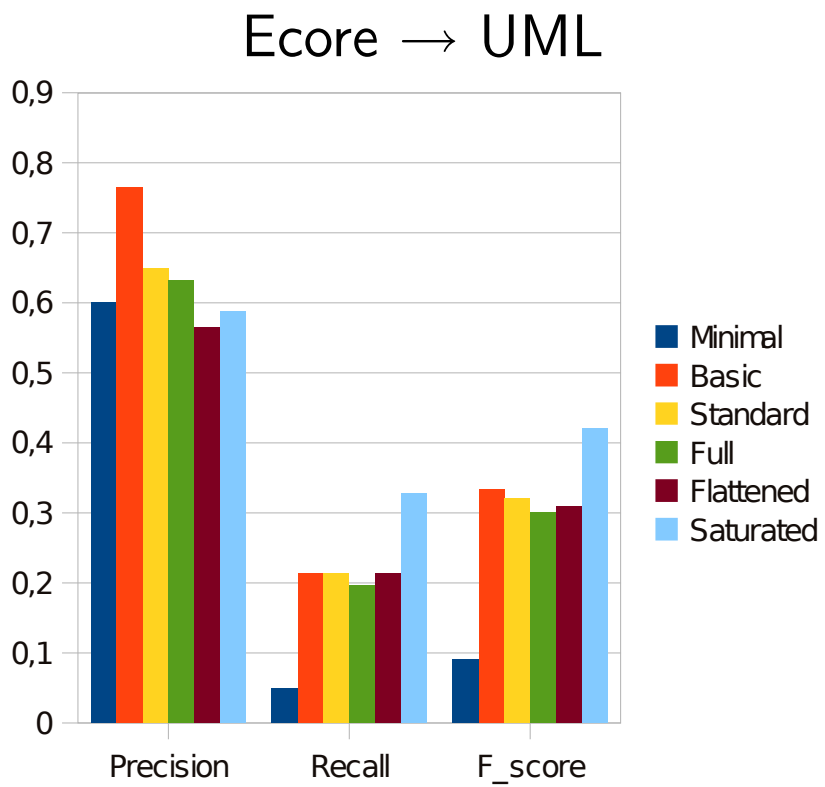

Ecore → Minjava




- Good results
- Better results for similar metamodel size
- Configurations Saturated et Basic give good results

# Results




- Good results
- Better results for similar metamodel size
- Configurations Saturated et Basic give good results

# Conclusion

- A tool that automatically aligns two metamodels
- Assessment of different configurations
- alignments can be used for code generation

# Future work

- Code generation
- more complex alignement
- Testing other tunings
- Generic tool

# Bibliographie


Sergey Melnik, Erhard Rahm and Philip A. Bernstein
Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching
International Conference on Data Engineering, 2002

Vladimir I. Levenshtein
Binary codes with correction of deletions, insertions and substitution of symbols
Soviet Physics Doklady, 1966

# Links

## GUM

Gumm (Generic and Useful Model Matcher)
http://code.google.com/p/gumm-project

## J.R. Falleri

http://www.lirmm.fr/ falleri