Tree Based Domain-Specific Mapping Languages

Elina Kalnina[1,2], Audris Kalnins[2], Agris Sostaks[2], Edgars Celms[2], Janis Iraids[2]

University of Latvia, IMCS

This work has been partially supported by
[1] the European Social Fund within the project ”Support for Doctoral Studies at University of Latvia”
[2] Latvian National Research Program Nr. 2 “Development of Innovative Multifunctional Materials, Signal Processing and Information Technologies for Competitive Science Intensive Products”

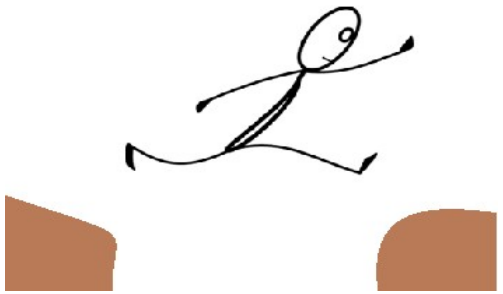
TOC

- Motivation
- Mapping Language for MDSD
  - Examples
  - Principles
  - Evaluation
- Mapping Language Family
  - Mapping Language Definition

Motivation

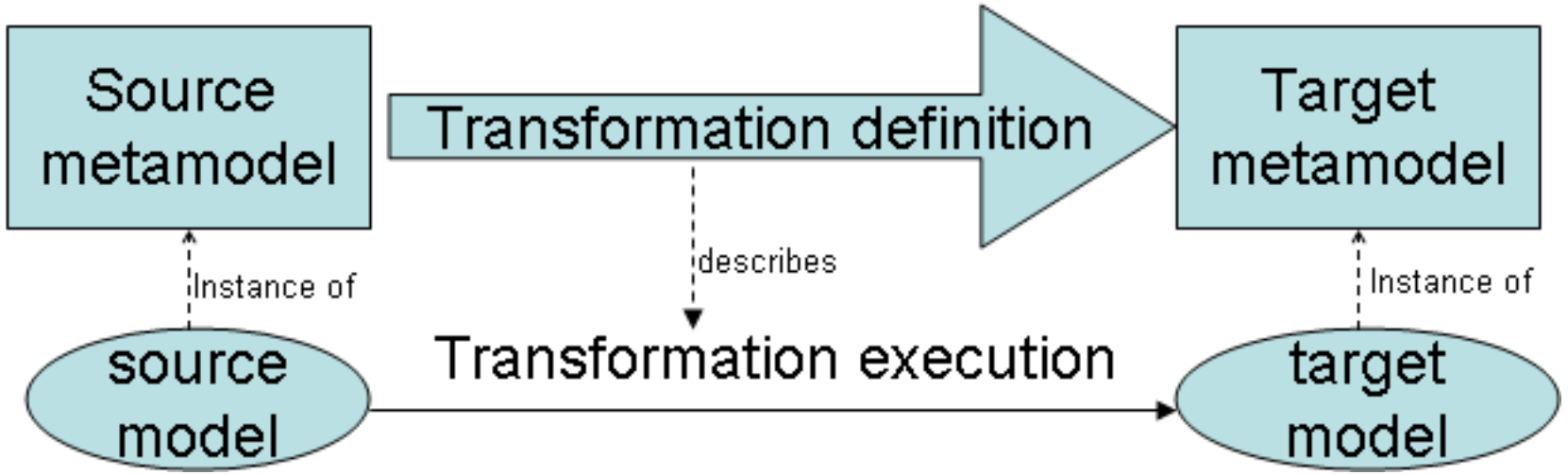
Problem

The industry has not yet widely accepted the **model driven approaches** – the gap between researchers and industry is still to be overcome.



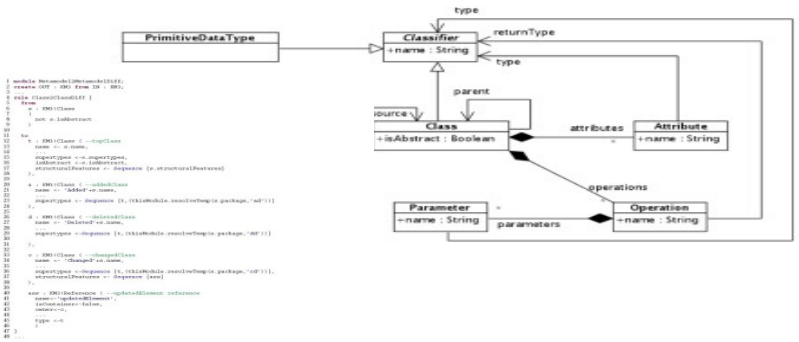
Model transformations are still one of the key feature to deal with models in any MD technology.

Model transformation



Problem

Defining model transformations **requires** deep knowledge of **metamodelling**.



Most of model **transformation languages** are actually **used by** their **authors** and small groups of researchers in the field.

Problem

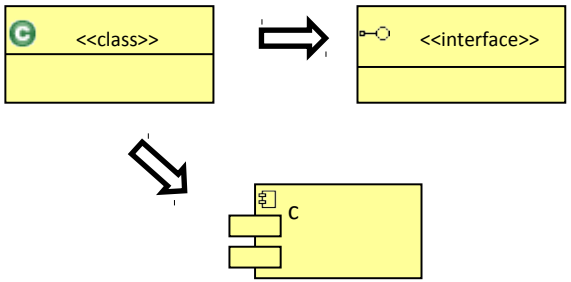
Who will <u>actually</u> write model transformations?

Transformation expert

Transformation user

Solution

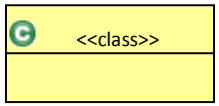
Domain-Specific Modelling (DSM) proposes to use modelling languages which use notation and concepts specific to the domain actually being modelled.

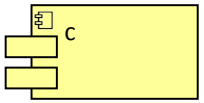


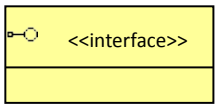
We propose to use domain-specific transformation languages which use elements specific to models being transformed.

Solution

Use **familiar concepts** of modelling language

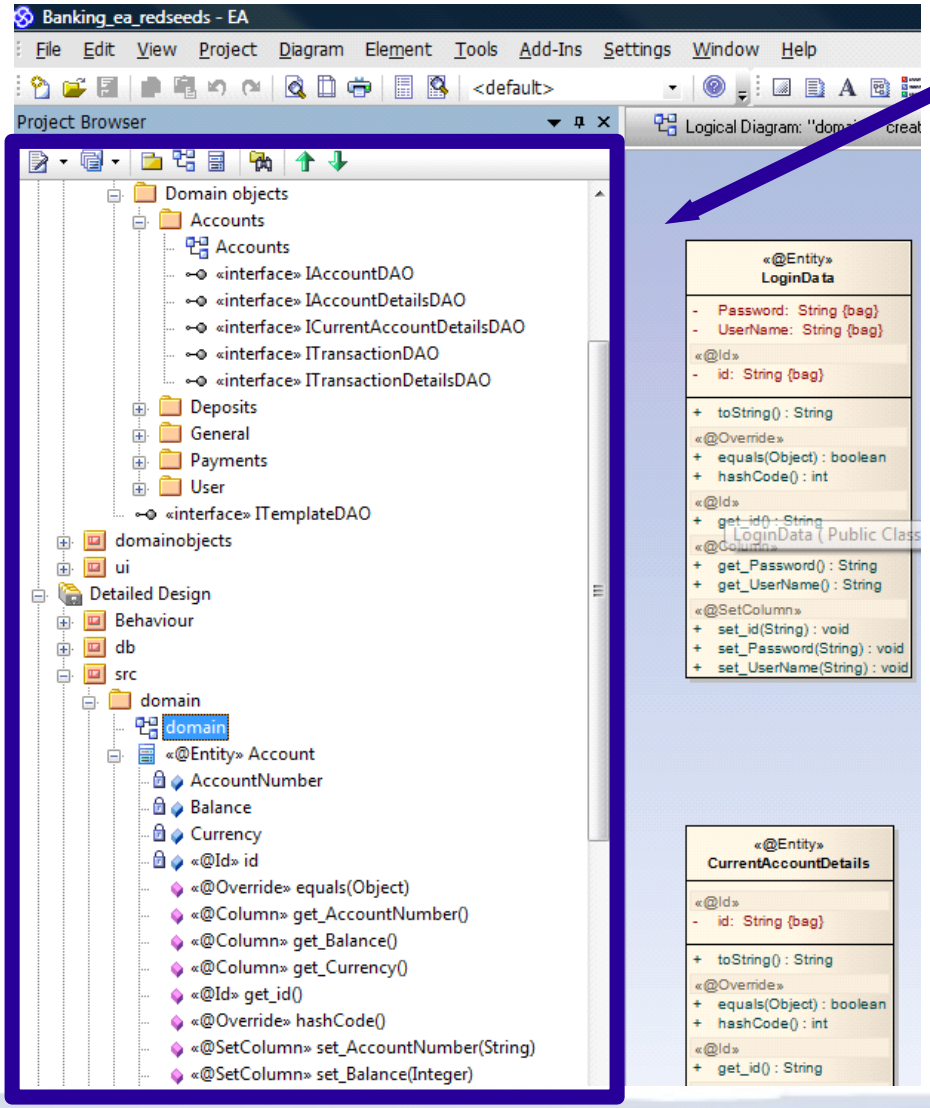






Use **concrete syntax** of modelling language

Solution



This is a model tree - the inspiration for syntax of domain specific model transformation language

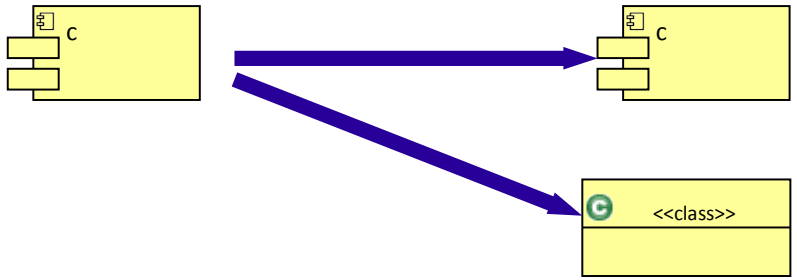
- **We can represent every model as a tree!**
- **We will try to use this representation in our model transformation language!**

Problem

How to represent relations between source and target?

Solution

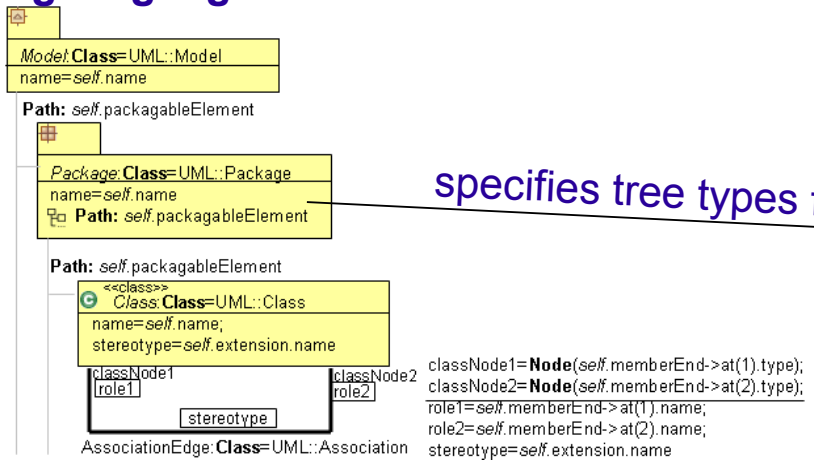
The most intuitive option to define model transformations is to use **mappings**.



Mappings permit to specify transformations in a simple way, frequently by very intuitive graphics.

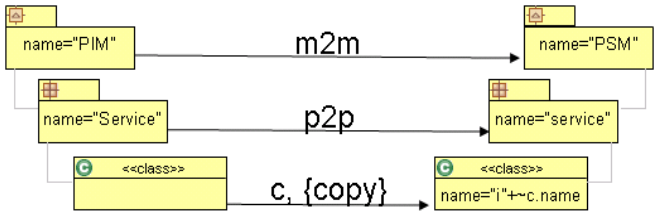
Shift of roles

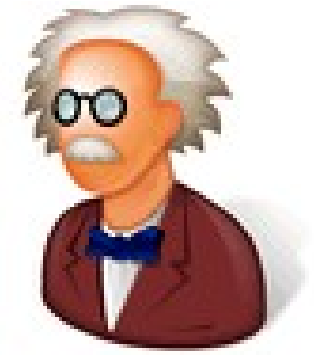
Mapping language definition



specifies tree types for

Mapping language



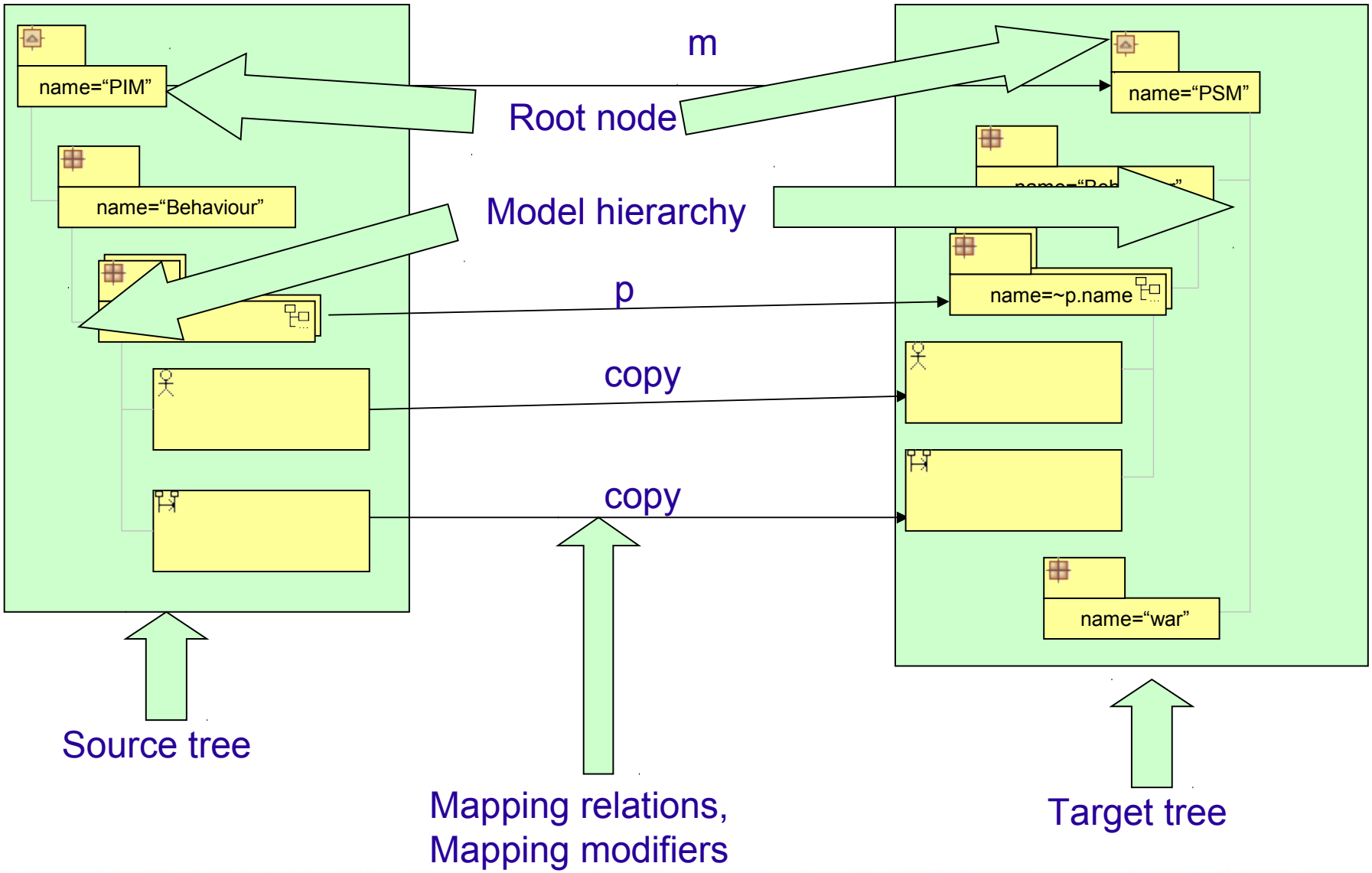


Transformation expert,
Language developer

Transformation user

MALA4MDSD

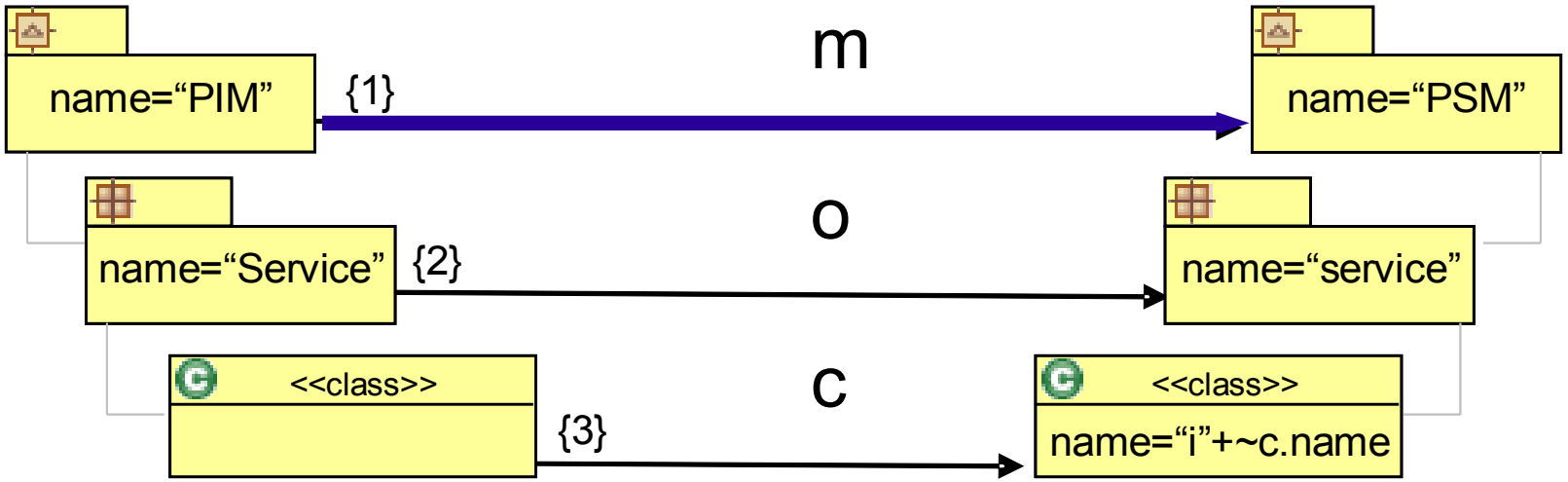
Mapping language



Mapping relations

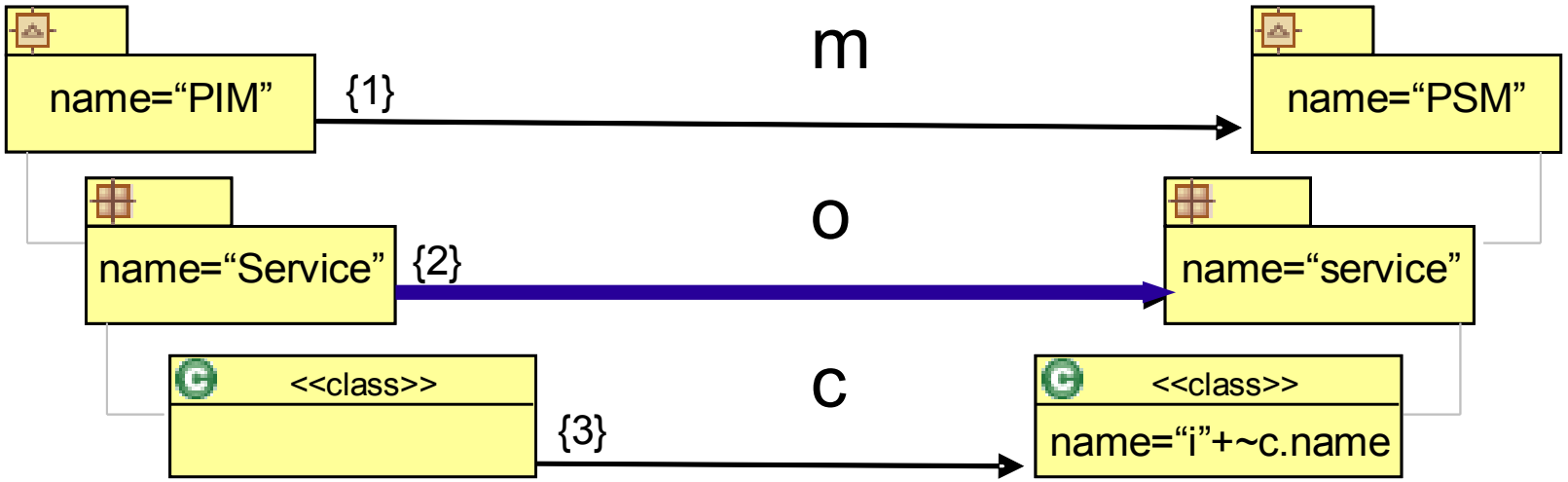
- If an **instance corresponding** to the **source node** is **found** in the source model then an **appropriate instance** should be **created** in the **target** model.
- **For each valid instance** of a **source** node the outgoing **mappings** are **executed** (validity checked using the containment relationship to parent and filter conditions).
- The general "**create if not exists**" semantics is used for mappings – a duplicate target instance defined by the same type of mapping is never created (for it *trace edge* is used) .

MALA4MDSD, Example



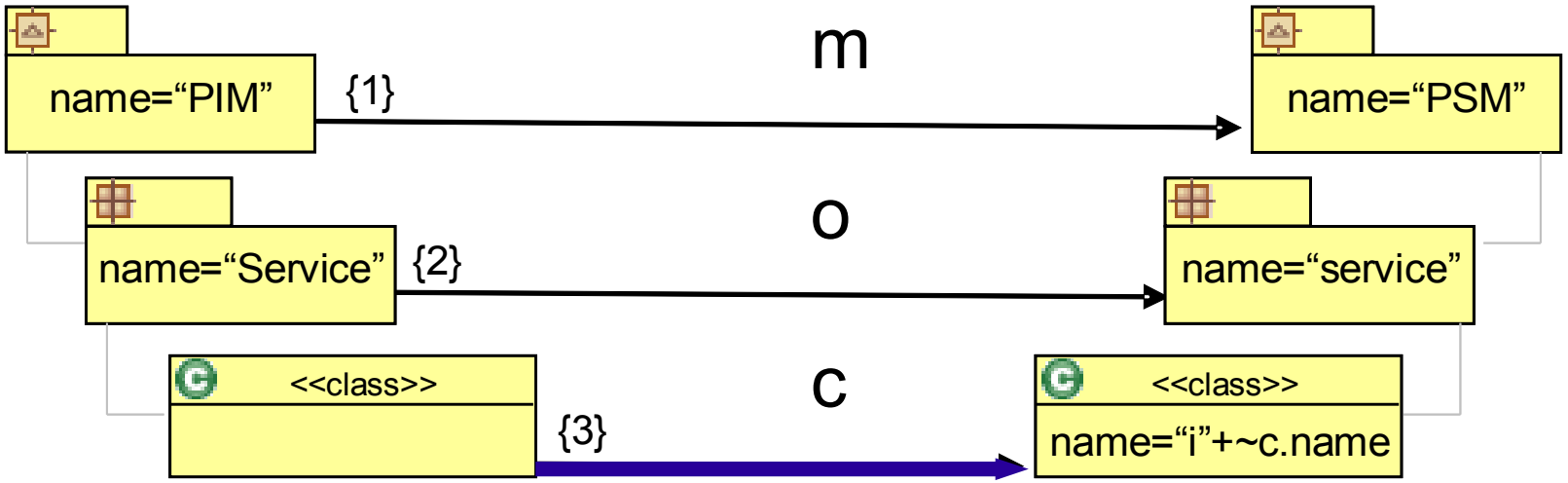
In the source a UML model named "**PIM**" is **sought for**. For each such model a UML model named "**PSM**" is **created** in the target .

MALA4MDSD, Example



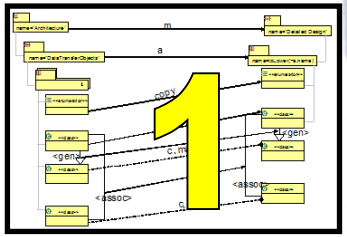
Then **packages** named "**Service**" **in** the UML model "**PIM**" are **sought for**. For each such package the corresponding **package** named "**service**" **in** the target UML model "**PSM**" is **created**.

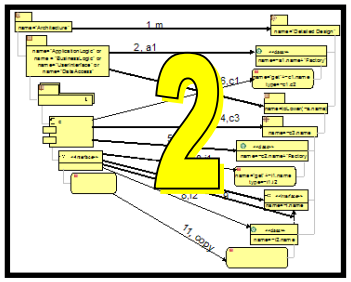
MALA4MDSD, Example

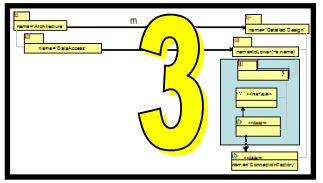


The third mapping relation processes **all classes in** the source model package "**Service**" and **creates** the corresponding **classes in** the target model package "**service**". The prefix "i" is added to the source model class name .

Mala4MDSD, ordering

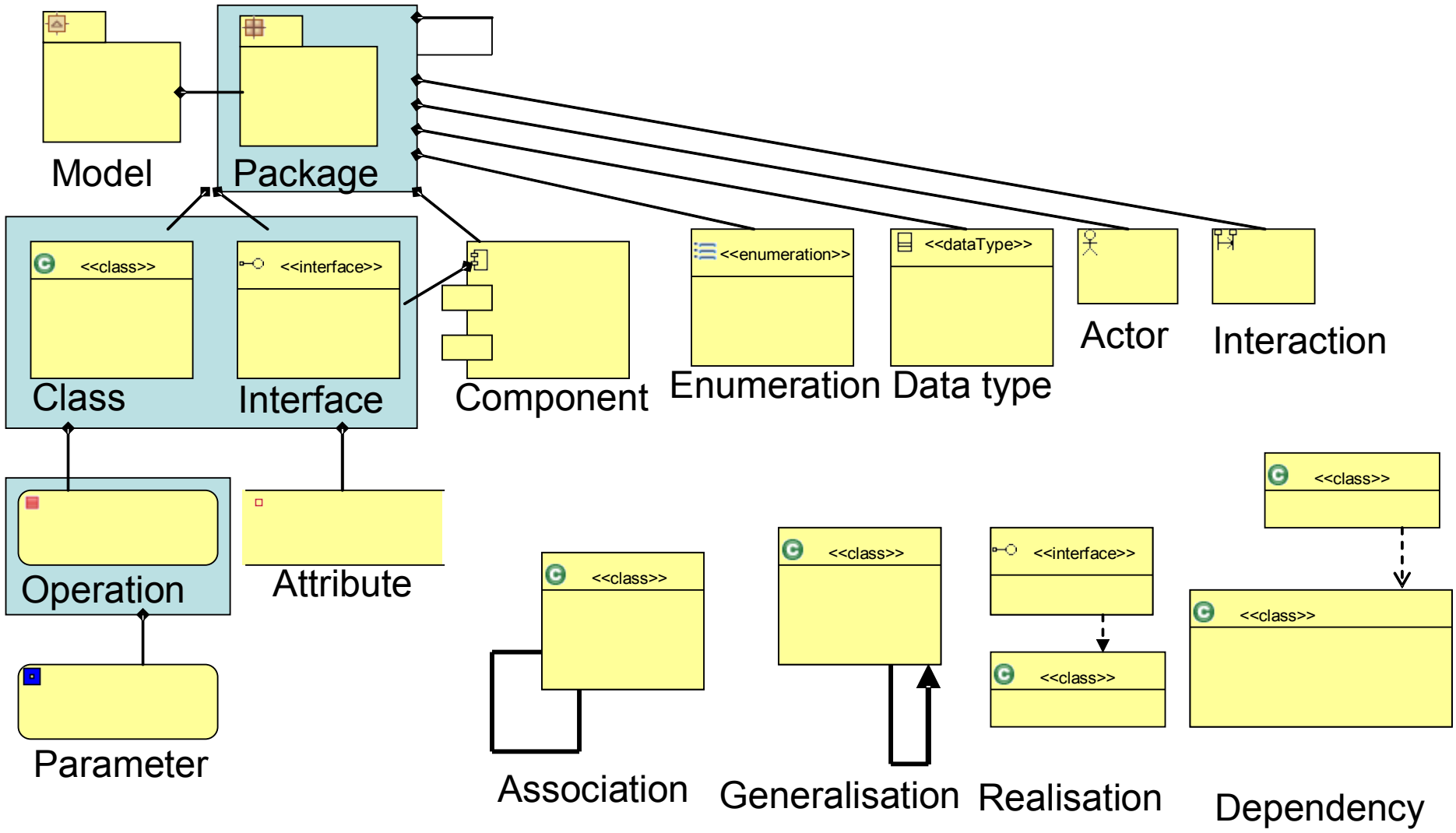






- One mapping program (transformation) consists of **several** ordered **mapping diagrams**. They are executed separately in the **given order**.
- **Mapping relations** in a mapping diagram are **ordered top-down** – according to their start position in the source tree. Mappings are executed according to their ordering.

Language elements, containment



Attributes

- **Model**: name
- **Package**: name
- **Class**: name, stereotype
- **Interface**: name
- **Component**: name
- **Interaction**: name
- **Data Type**: name
- **Operation**: name, type, stereotype
- **Attribute**: name, stereotype, type
- **Parameter**: name, direction, type
- **Association**: stereotype

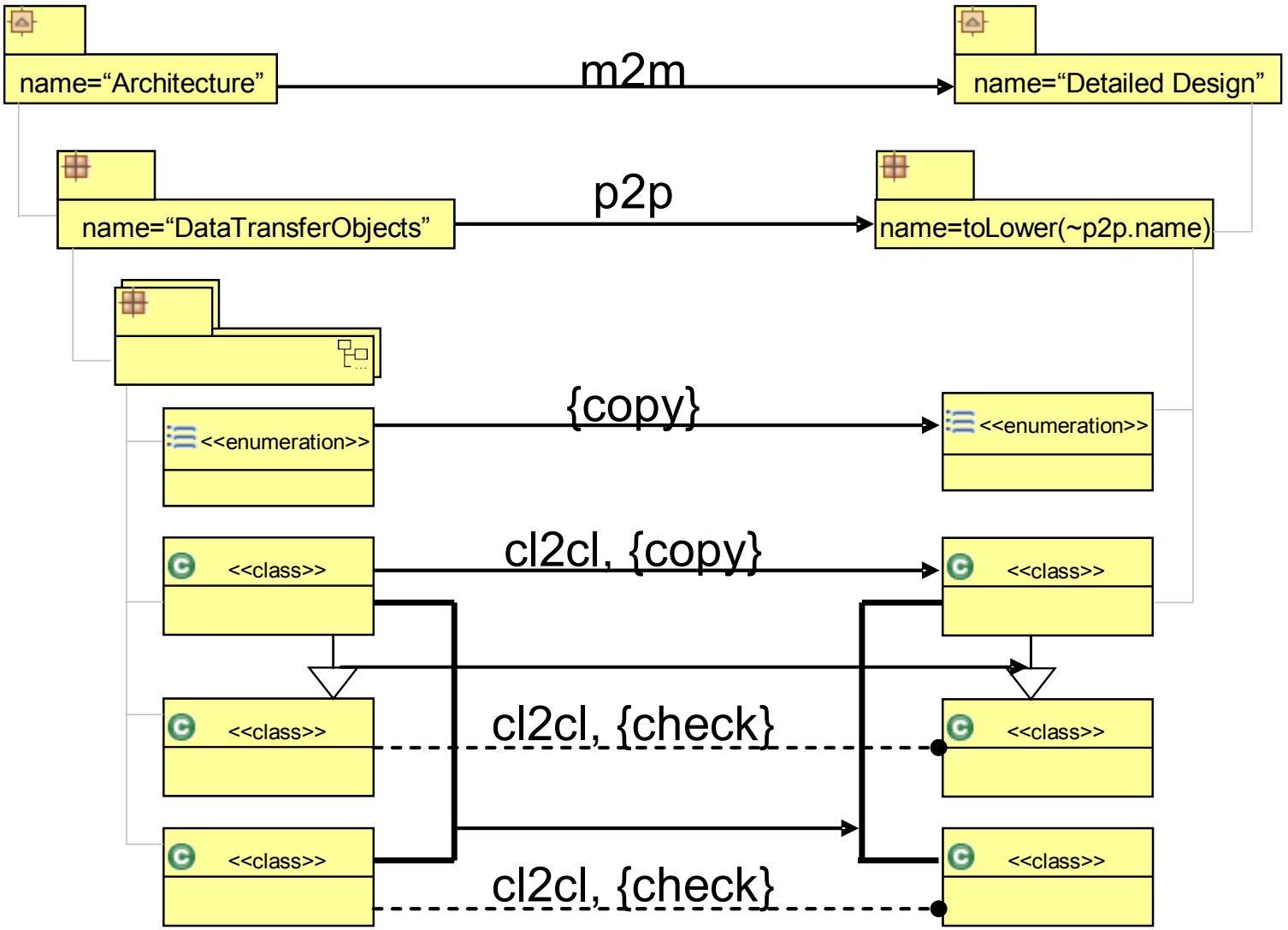
Mapping modifiers

- **copyAttributes** - in the target node for each attribute an implicit assignment is performed setting it to the value corresponding to that attribute value in the source node.
- **copy** -implicit mappings for all children types of the node (at any depth) with the *copyAttributes* modifier are performed. Used for copying tree fragments.
- **check -** nothing is created in the target tree only the relevant node is found using the traces between source and target. Used e.g., for locating edge endpoints.

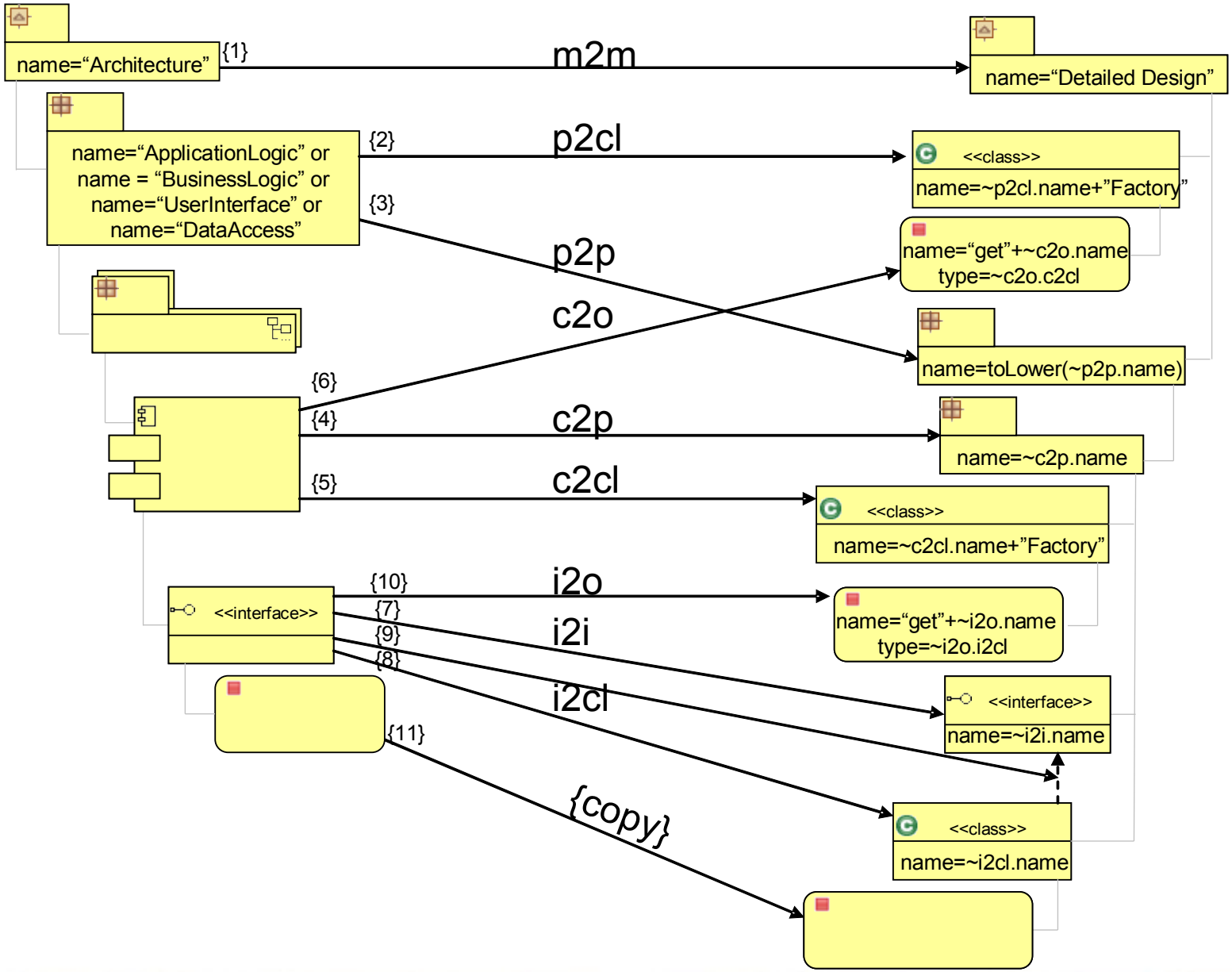
Other features

- Processing of package hierarchy with arbitrary depth;
- Edge processing;
- Source pattern;
- Conditional mappings;
- Conditional expressions;
- Support for calling Custom transformations.

Example (1)


name="Architecture"
m2m
name="Detailed Design"
name="DataTransferObjects"
p2p
name=toLower(~p2p.name)
{copy}
<<enumeration>>
<<enumeration>>
cl2cl, {copy}
<<class>>
<<class>>
<<class>>
cl2cl, {check}
<<class>>
<<class>>
cl2cl, {check}
<<class>>
<<class>>

Example (2)


name="Architecture"
{1}
m2m
name="Detailed Design"
name="ApplicationLogic" or
name = "BusinessLogic" or
name="UserInterface" or
name="DataAccess"
{2}
p2cl
<<class>>
name=~p2cl.name+"Factory"
{3}
p2p
name="get"+~c2o.name
type=~c2o.c2cl
c2o
name=toLower(~p2p.name)
{6}
{4}
c2p
name=~c2p.name
{5}
c2cl
<<class>>
name=~c2cl.name+"Factory"
<<interface>>
{10}
i2o
name="get"+~i2o.name
type=~i2o.i2cl
{7}
i2i
{9}
{8}
i2cl
<<interface>>
name=~i2i.name
{11}
{copy}
<<class>>
name=~i2cl.name

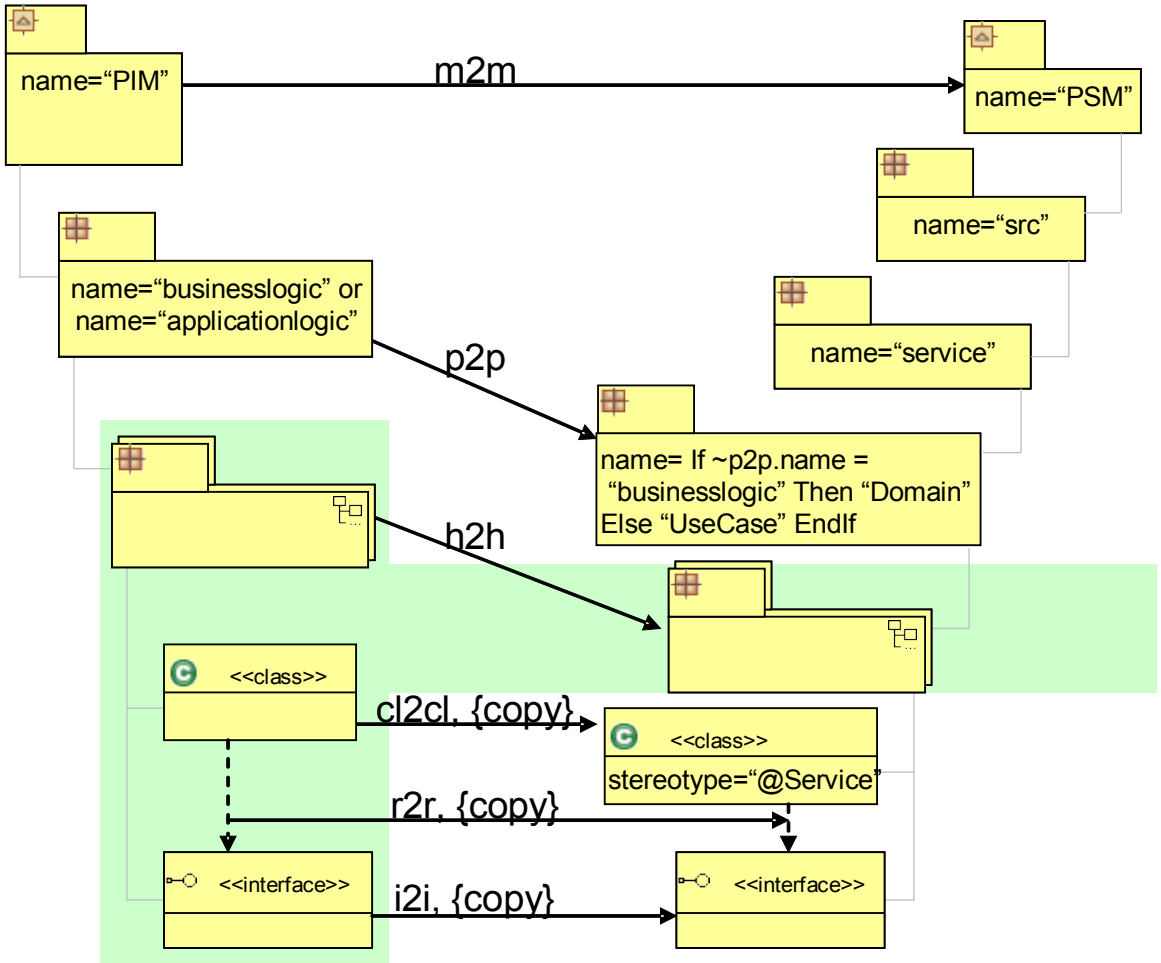
Evaluation

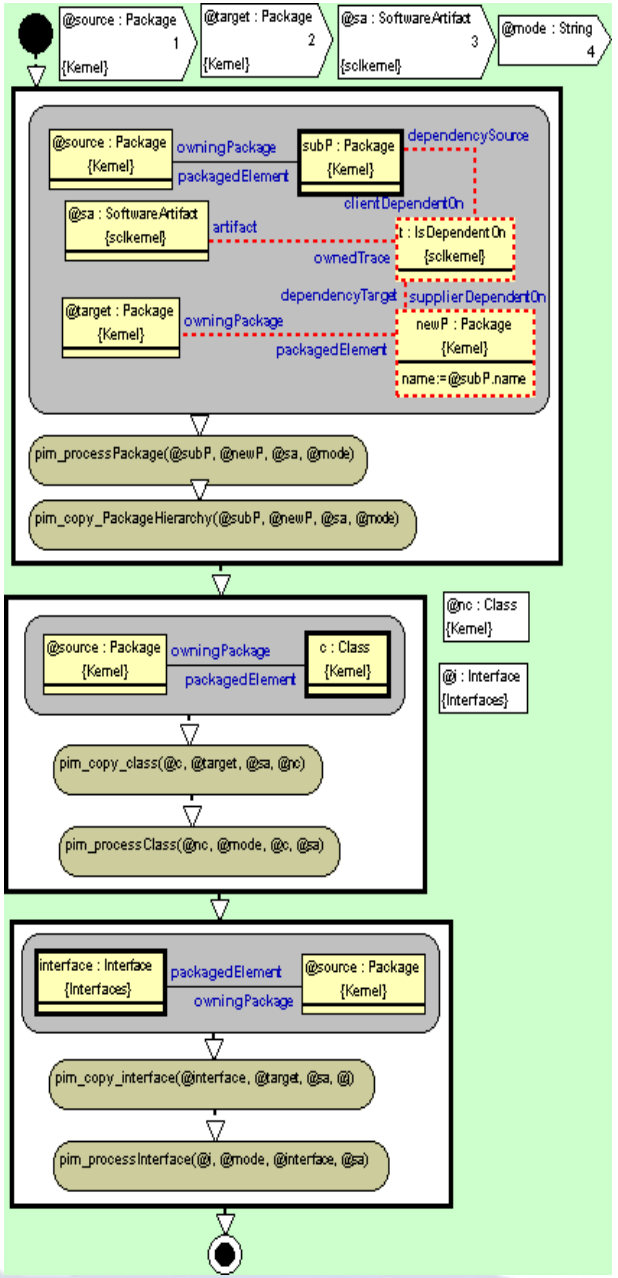
Basic Style	MOLA procedures	19	3	MALA4MDSD diagrams
	MOLA rules	84	19	MALA4MDSD mappings
	MOLA class elements	265	29 (source:11; target:18)	MALA4MDSD nodes
Keyword-based Style	MOLA procedures	51	8	MALA4MDSD diagrams
	MOLA rules	137	41	MALA4MDSD mappings
	MOLA class elements	418	66 (source:27; target:39)	MALA4MDSD nodes

In IST 6th framework project ReDSeeDS a model driven path from requirements to code is investigated.
These transformations have been developed in the model transformation language MOLA.
We have rewritten these transformations in the language MALA4MDSD.

Transformation development in MALA4MDSD was at least 3 times faster.

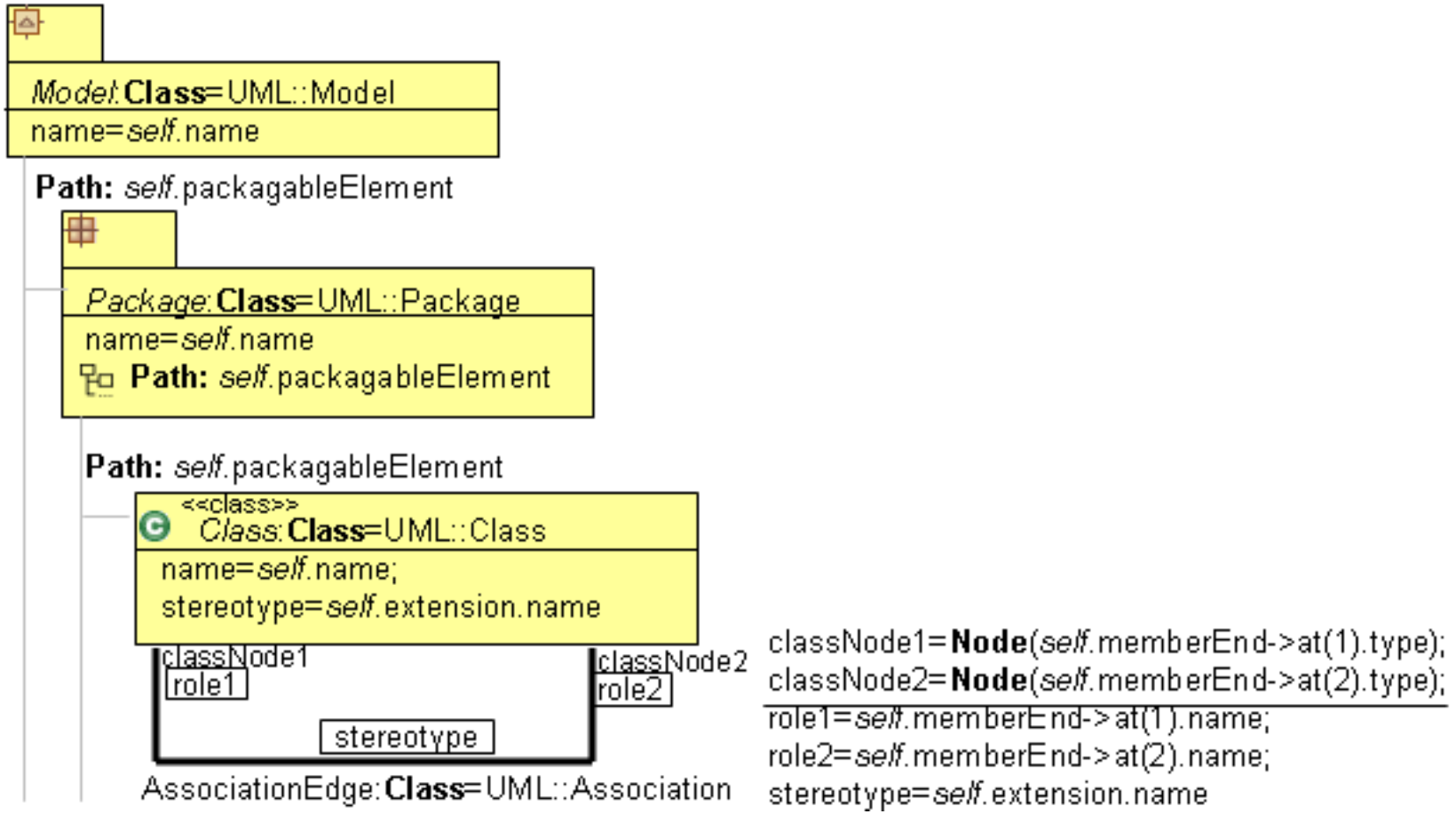
Evaluation (2)



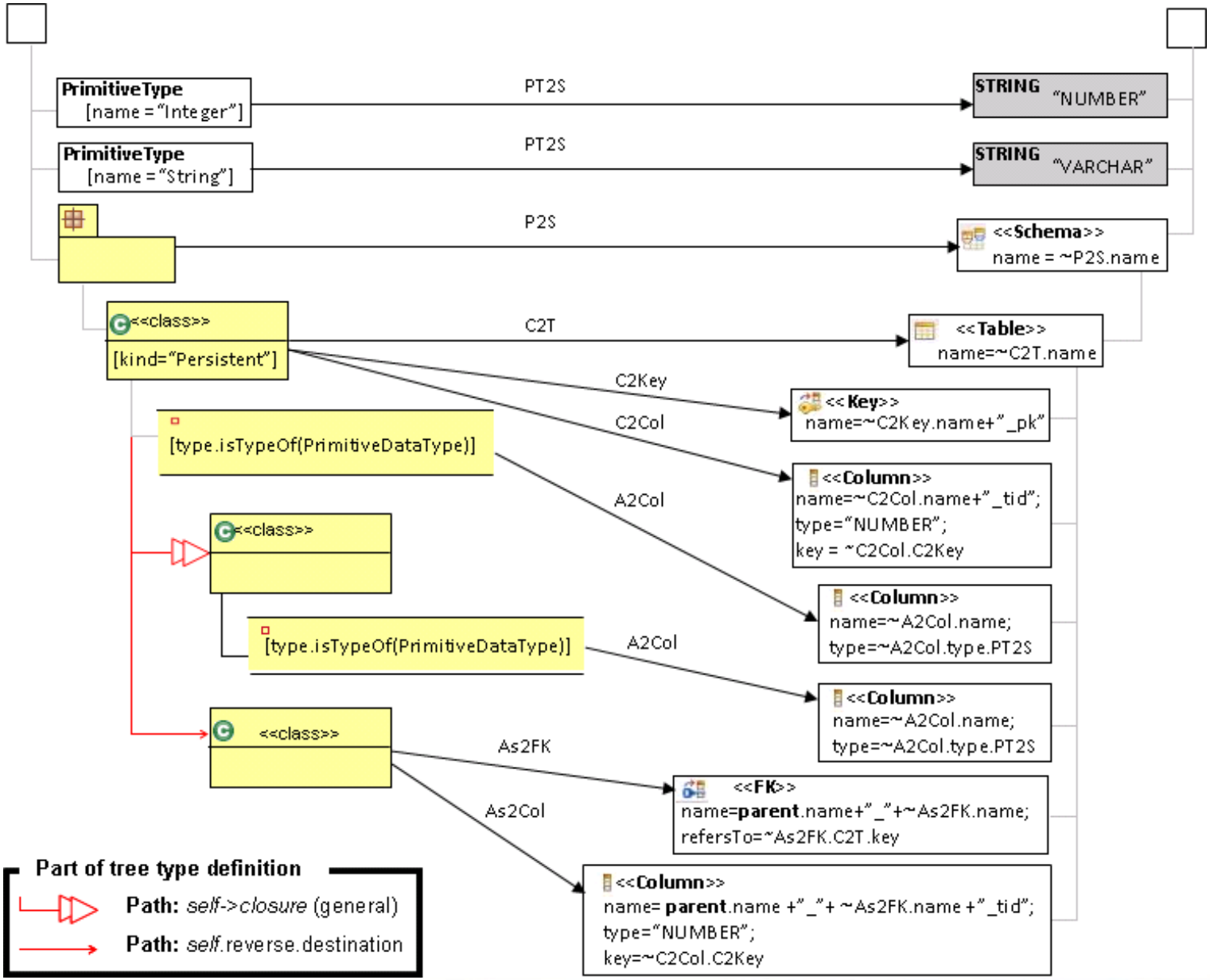


Mapping language family

Mapping language definition



Another mapping language



Conclusions

- It is proposed to define model transformations using simple mapping relations and tree syntax of source and target models.
- One specific mapping language – MALA4MDSD for transformations from a UML subset to another UML subset is proposed.
- A generic approach for the creation of domain-specific mapping languages is proposed.
- Mapping language family is being implemented.

Thank You!