MULTI-LEVEL MODELLING FOR MODEL-DRIVEN ENGINEERING

Juan de Lara

with contributions from
E. Guerra and J. Sánchez Cuadrado

Universidad Autónoma de Madrid

miso research group
Universidad Autónoma de Madrid (Spain)

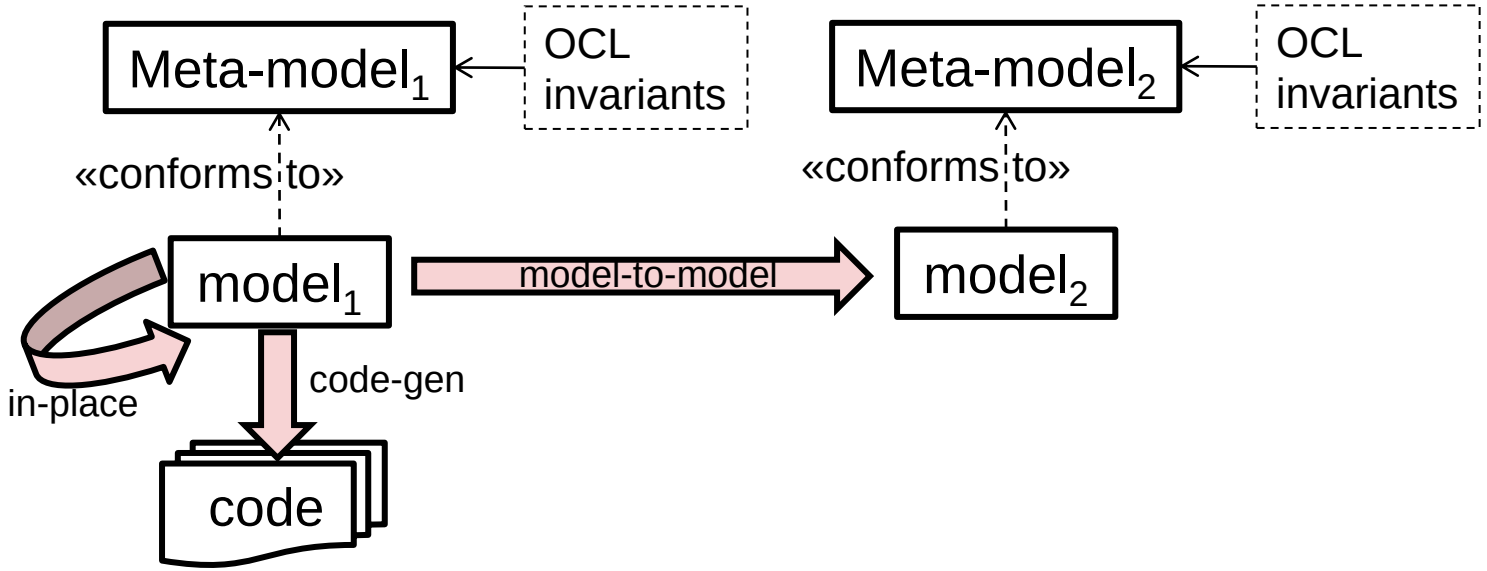
http://miso.es
@miso_uam

MULTI'2021

WHAT IS THE TALK ABOUT?

Model-Driven Engineering

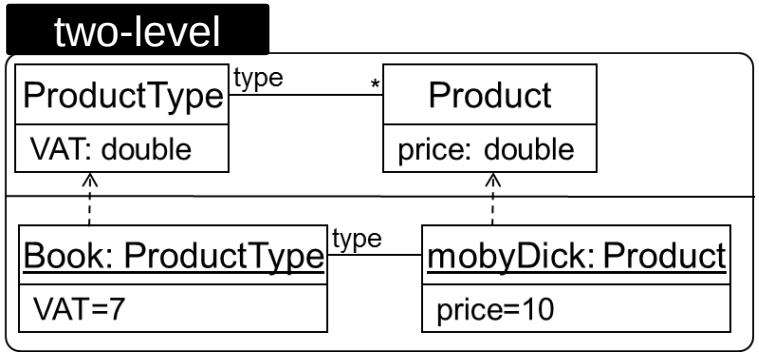
- Models are core assets of development
- Frequently built using DSLs, based on domain meta-models
- Transformed (in-place, out-place), refined into code, etc
- Mainstream modelling approach is two-level (meta-models/models)

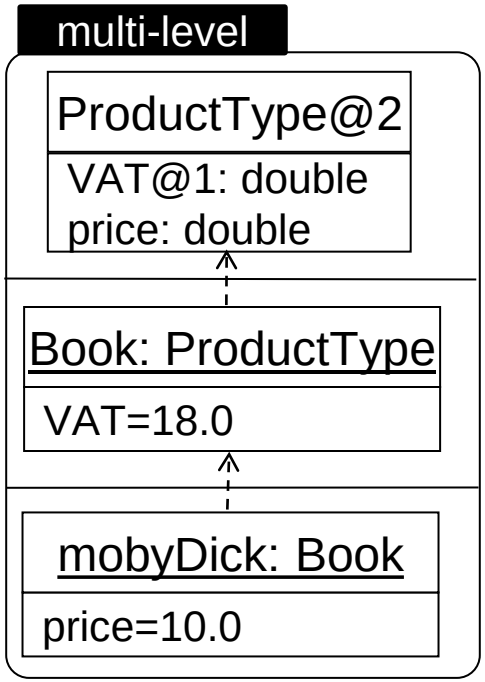


WHAT IS THE TALK ABOUT?

Multi-level modelling

- More than two meta-levels at a time
- Meta-levels can influence other levels beyond the immediate one
- Simplifies modelling in some scenarios





Can we use Multi-Level modelling for Model-driven Engineeering?

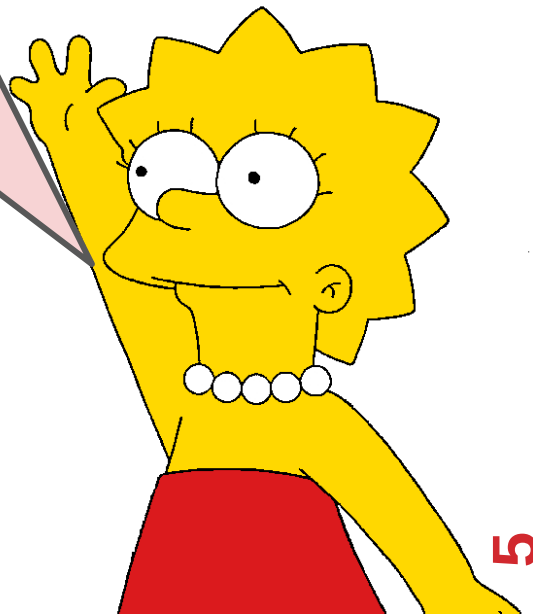
YES, BUT WE NEED...

Suitable scenarios for M(LM)DE

Multi-level:

- Constraints
- Transformations
- Code generators
- DSLs

Supporting tools



AGENDA

Basic concepts of (potency-based) multi-level modelling

- Motivation and real world applicability
- Clabjects, potency, levels, OCA
- Examples

Tool Support: MetaDepth

- Case study: game development

Multi-level model management

- Constraints, transformations, code generation
- Multi-level DSLs

Advanced concepts

MOTIVATION

Sometimes, one needs to explicitly model meta-modelling facilities, like:

- Instantiation (modelling both types and objects)
- Declaration and instantiation of attributes and references
- Inheritance

For example:

- Product types and instances [**e-commerce** applications]
- Task types and instances [**process modelling** languages]
- Component types and instances [**architectural** languages]
- Player types and instances [**gaming** applications]
- …

EXAMPLE

Model product types (books, CDs) and their instances, to be added on demand. Product types have a VAT, while instances have a price.

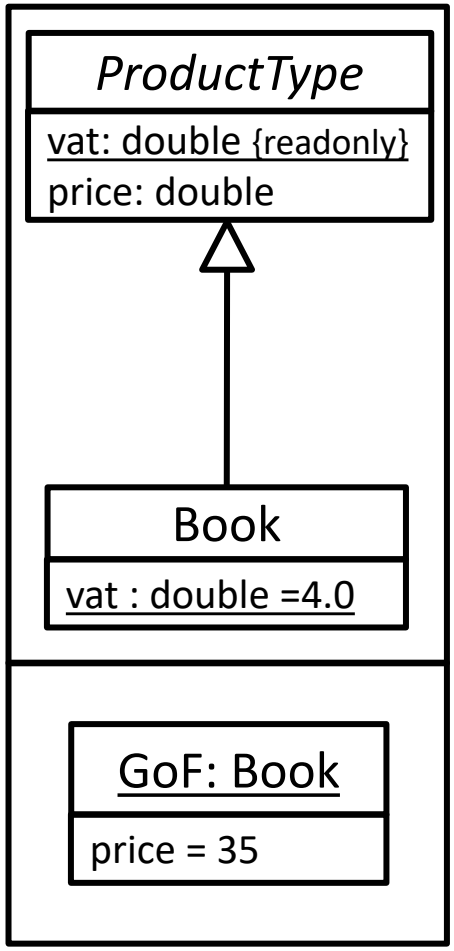
Explicit modelling of types and instances

- Types can be added dynamically
- Features of types are fixed and known a priori

Common modelling pattern, also known as: Type object [Martin et. al 97]**, item descriptor** [Coad 1992]**, metaobject** [Kiczales and Rivieres 1991]

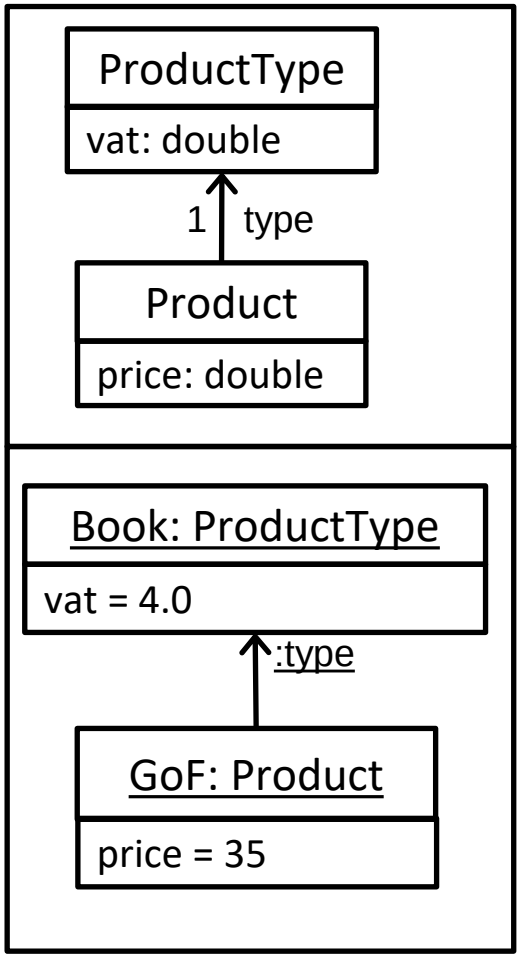
TWO-LEVEL SOLUTIONS

Static types



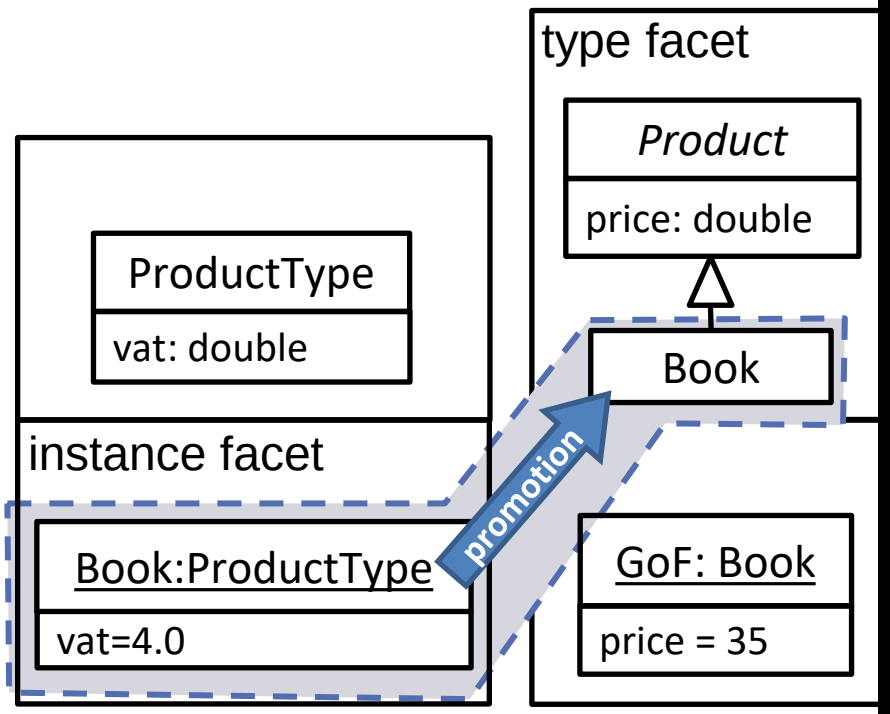
Requirements not satisfied

Explicit types



Complex meta-model

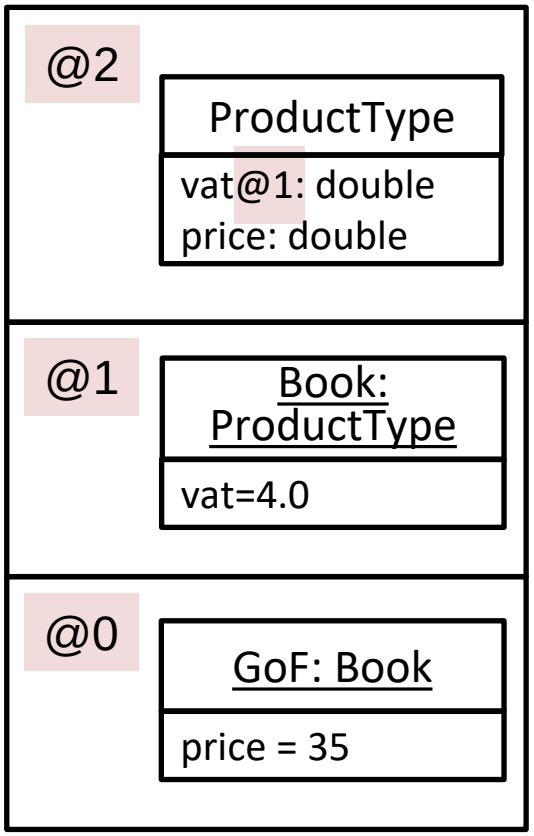
Promotion transformation



Requires a transformation
Hides information

MULTI-LEVEL SOLUTION

- Instantiation semantics is fixed
- Elements with instance and type facets at the same time
- Simplicity (less objects)
- ProductTypes can be extended with additional features, dynamically



wait!

But does this scenario occur often in practice?

MULTI-LEVEL PATTERNS

Patterns signalling that a multi-level solution could be beneficial

Emulate a meta-modelling facility within a meta-model:

- **Type-object**: typing and instantiation
- **Dynamic features**: feature definition and instantiation
- **Dynamic auxiliary domain concepts**: class definition and instantiation
- **Relation configurator**: Reference cardinality and its semantics
- **Element classification**: Inheritance

In multi-level these facilities are native at every meta-level

J. de Lara, E. Guerra, J. Sánchez Cuadrado: When and How to Use Multilevel Modelling. ACM Trans. Softw. Eng. Methodol. 24(2): 12:1-12:46 (2014)

PRACTICAL APPLICABILITY

How often do these patterns occur?

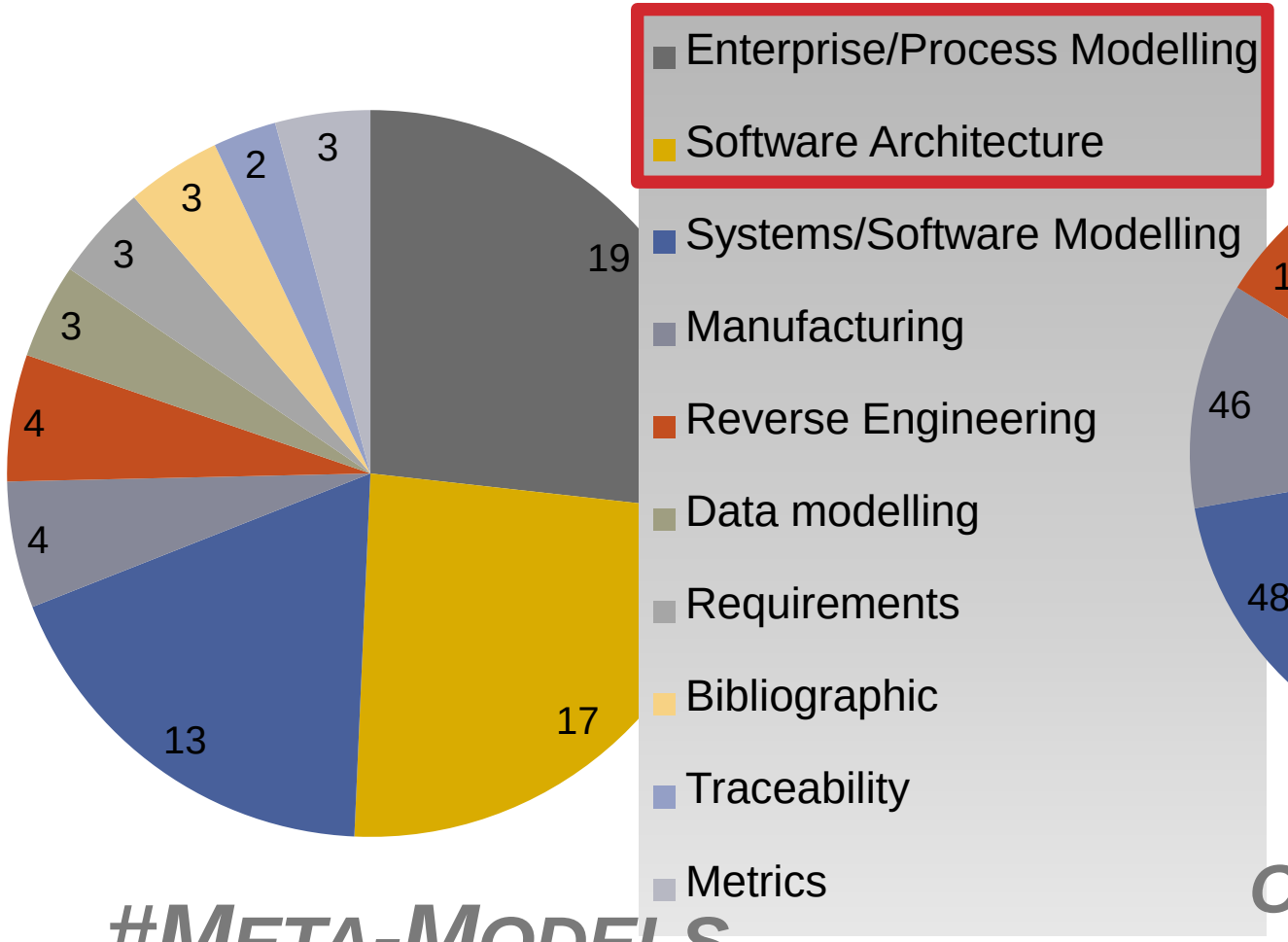
Analysis of >400 meta-models from:

- ATL meta-model zoo (305 meta-models) [8.5% occur.]
- OMG specifications (116 specs) [35%~38% occur.]
- ReMoDD (15 meta-models) [20% occur.]
- Articles in journals/conferences

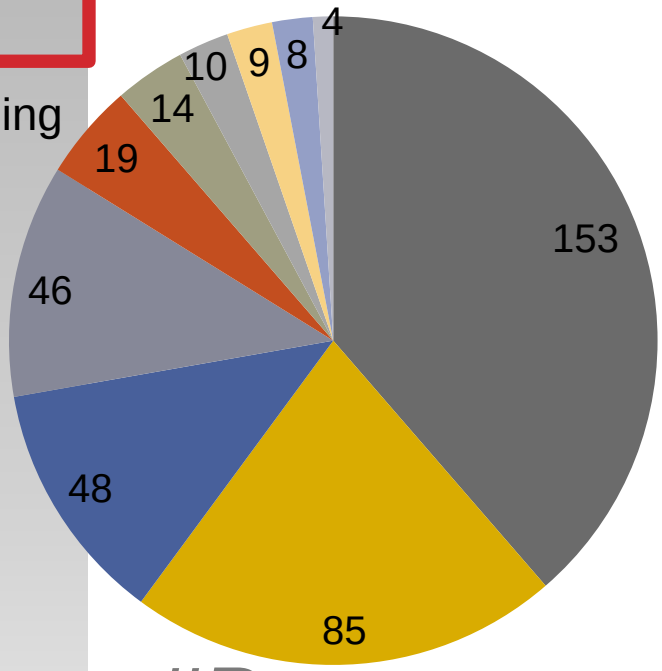
84 meta-models with 459 occurrences (avg 5.5)

- Most common pattern is the ***type-object***
- Most used solution is ***explicit types*** (~70%)

WHERE?



#META-MODELS

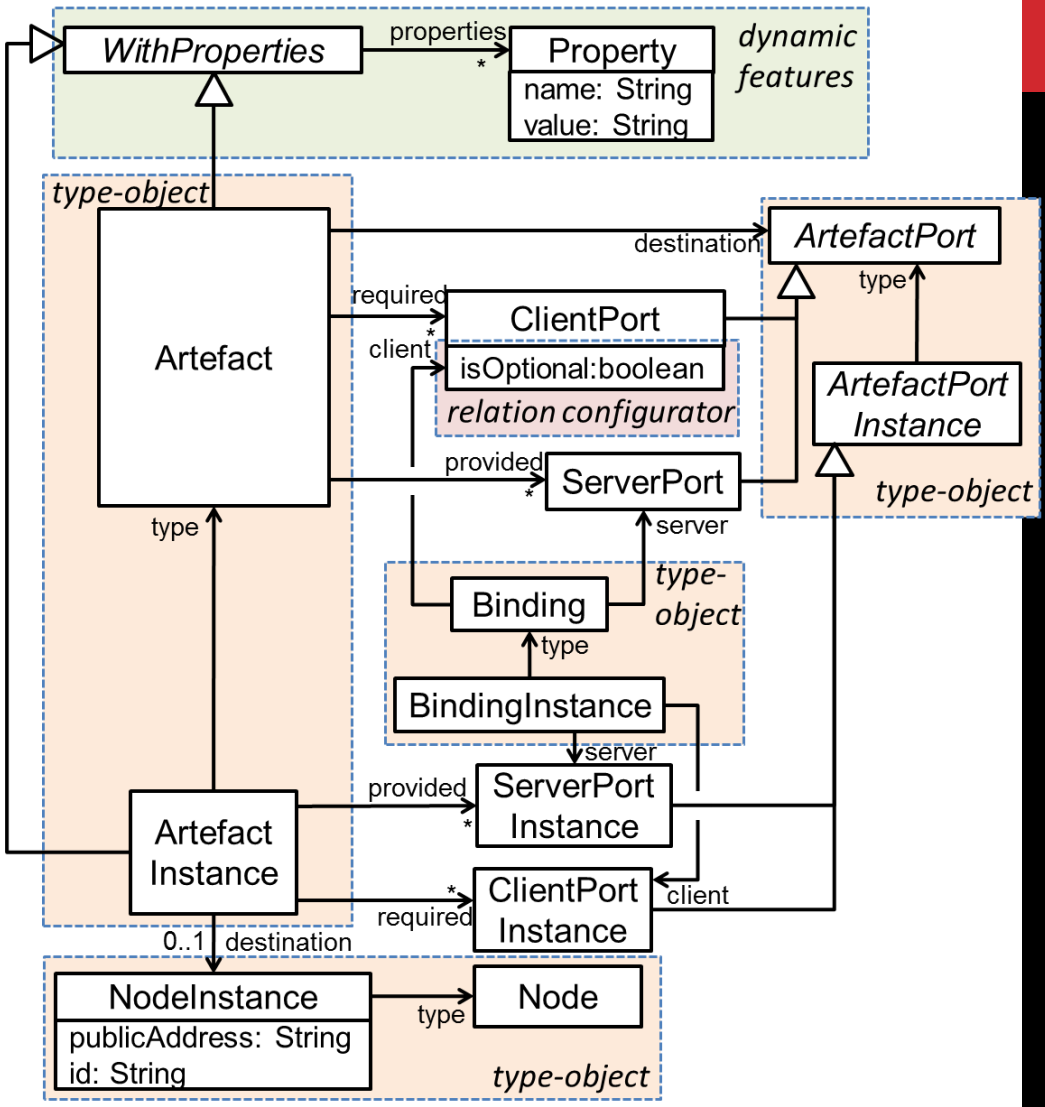


#PATTERN OCCURRENCES

WHERE?

Software architecture, components and services

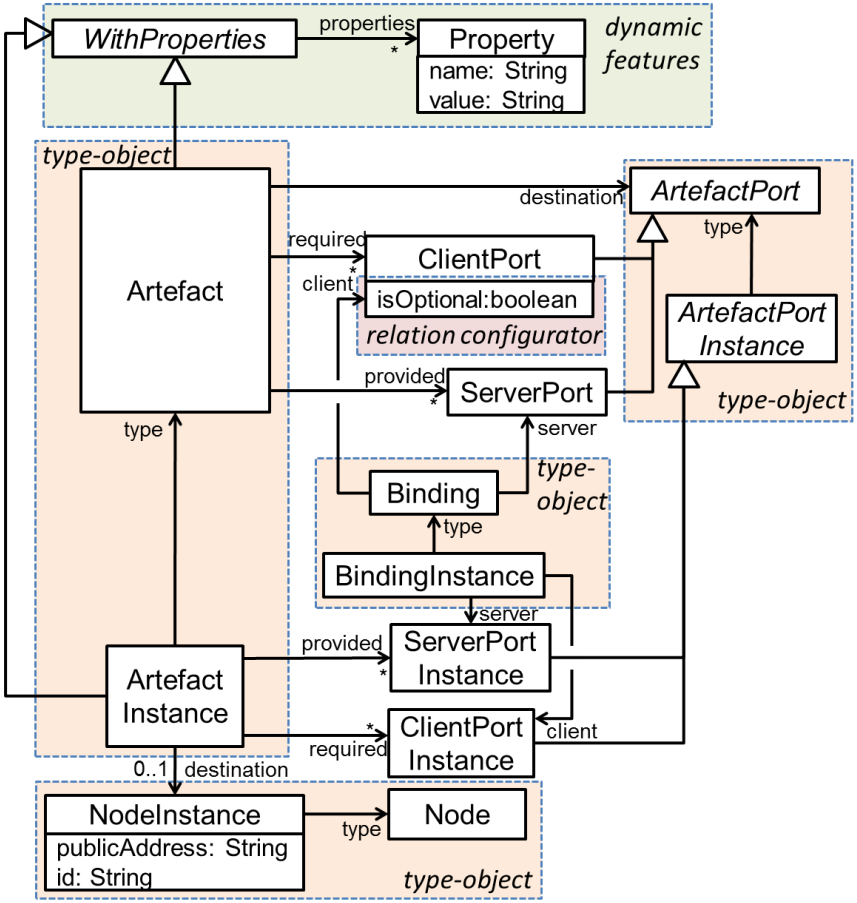
- Component types and instances, port types and instances
- Mostly explicit types solution



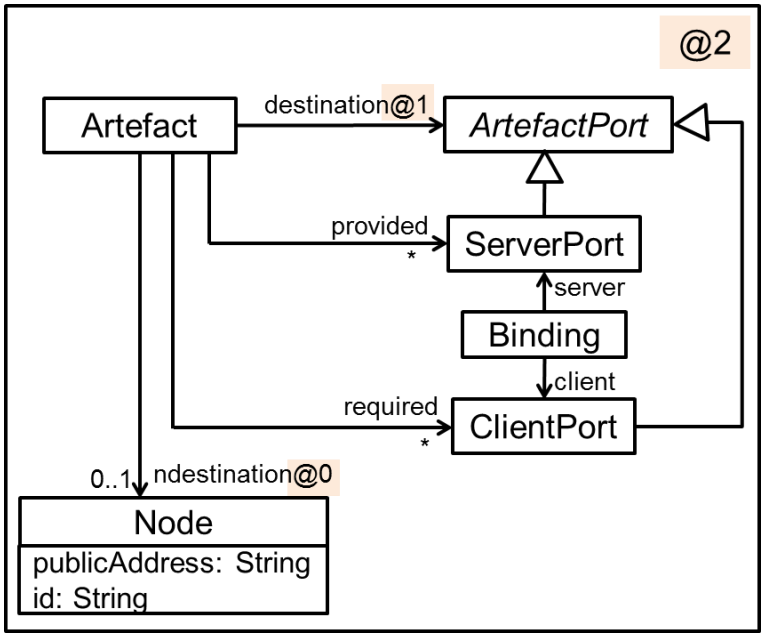
Example: CloudML

EXPLICIT TYPES TO MULTI-LEVEL

- CloudML (6 T-O, 1 DF)
- 14 classes



- Multi-level solution
- 6 classes



Multi-level modelling

the basics

CLABJECT = CLASS + OBJECT

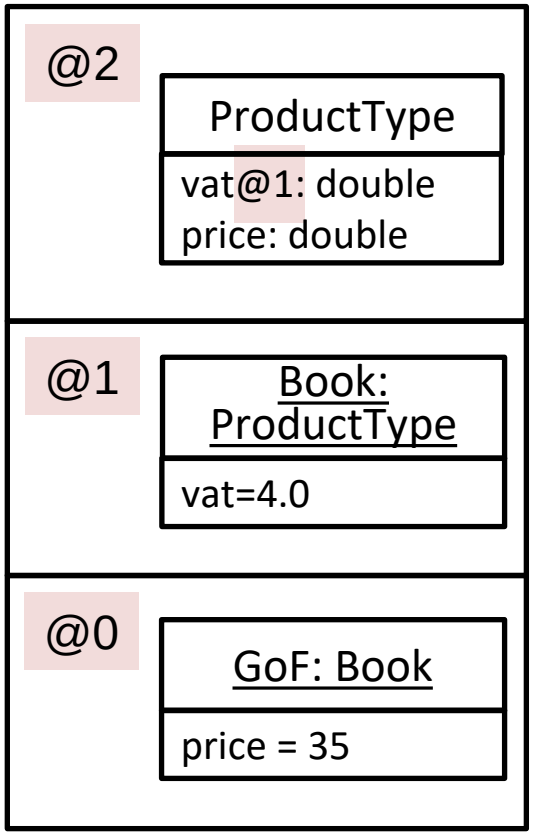
Elements have a combined type and instance facet

Book

- Instance of `ProductType`
  - Can provide a value for vat
- Type for `GoF`
  - Can declare new features
  - (We'll see how, using the OCA)

ProductType has type facet only

GoF has instance facet only



C. Atkinson and T. Kühne. 2001. The essence of multilevel metamodeling. *UML'01 (LNCS)*, Vol. 2185. Springer, 19–33.

POTENCY

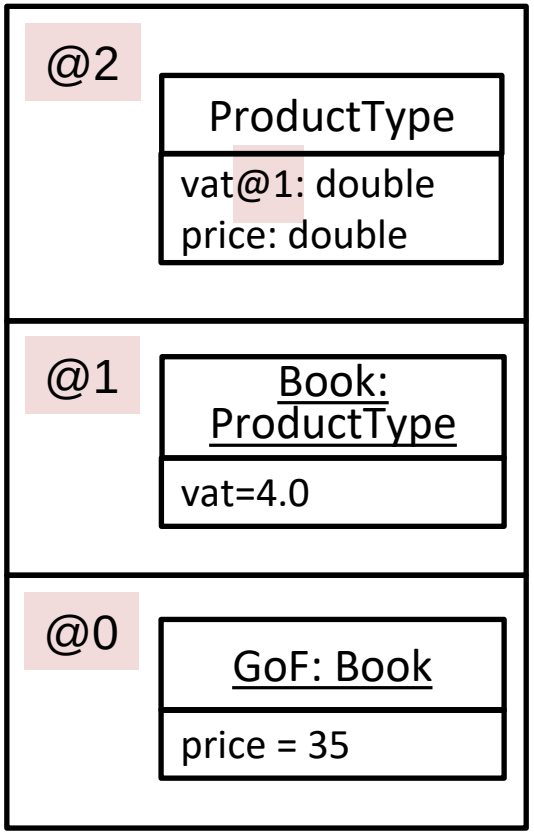
Used to characterize instances beyond the next meta-level

Models, clabjects and their features have a potency

- Natural number (or zero)
- Decreased at each lower meta-level
- Indicates at how many meta-levels the element can be instantiated

We use the "@potency" notation

By default elements take the potency of their containers

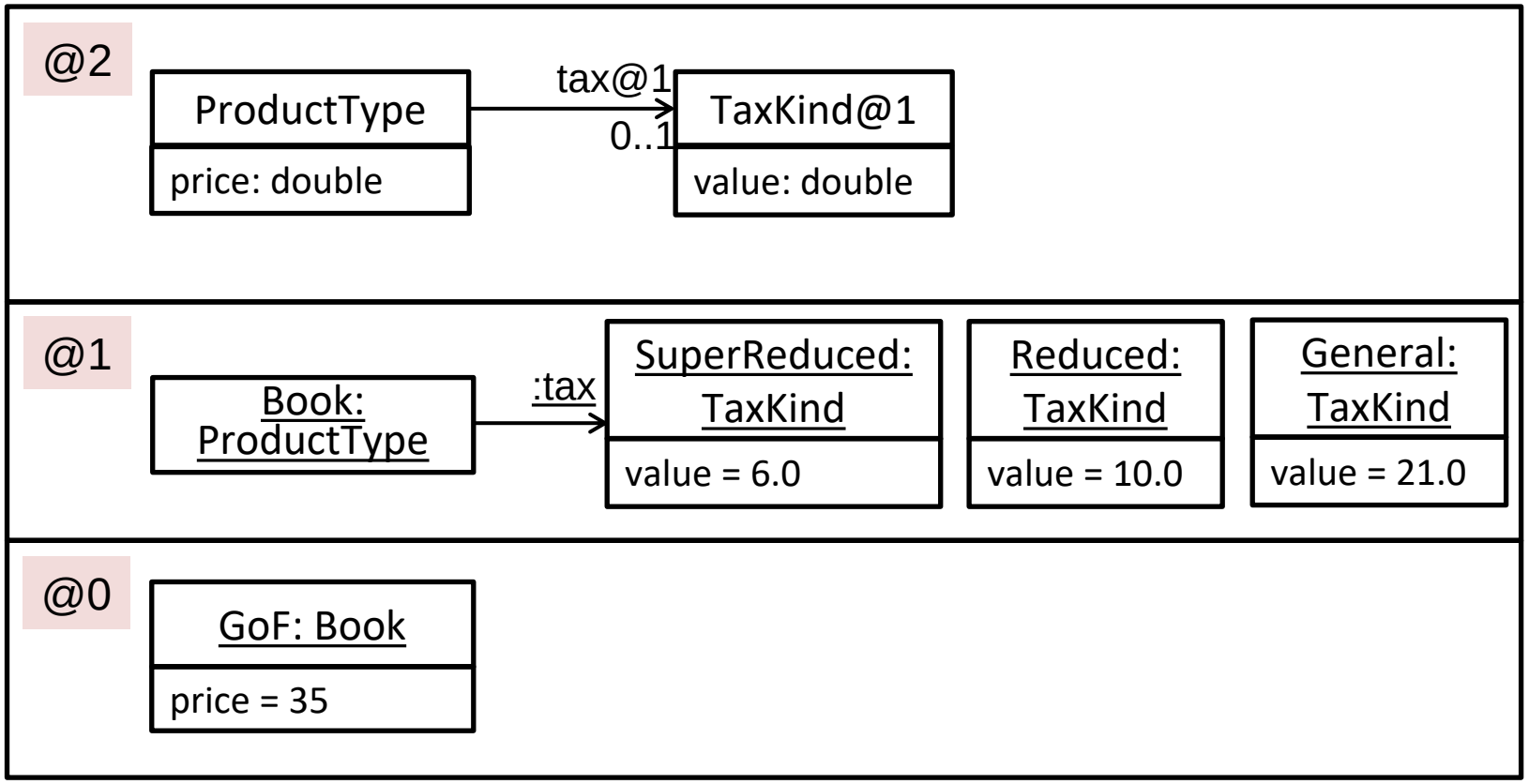


C. Atkinson and T. Kühne. 2001. The essence of multilevel metamodeling. *UML'01 (LNCS)*, Vol. 2185. Springer, 19–33.

LEVEL

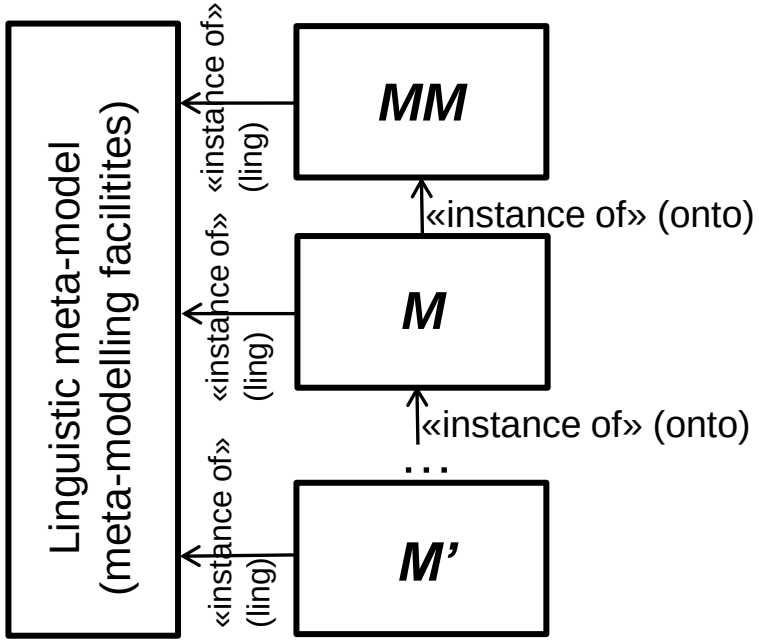
Refers to the potency of the model

A model with level L can hold elements with potency ≤ L



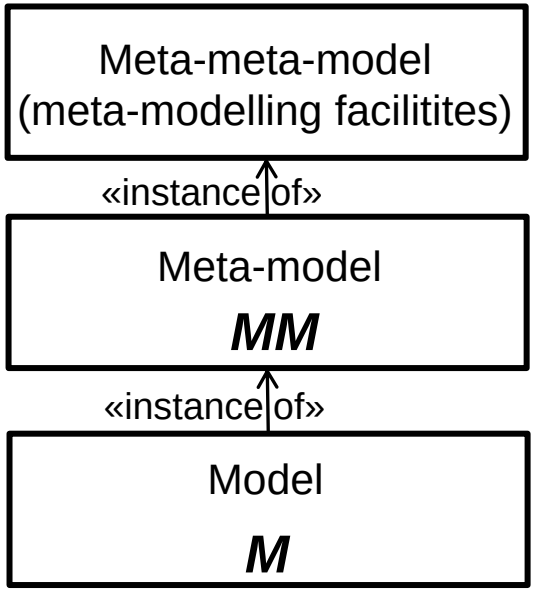
ORTHOGONAL CLASSIFICATION (OCA)

Multi-level



- Dual typing (ontological, linguistic)
- Make meta-modelling facilities available at every meta-level

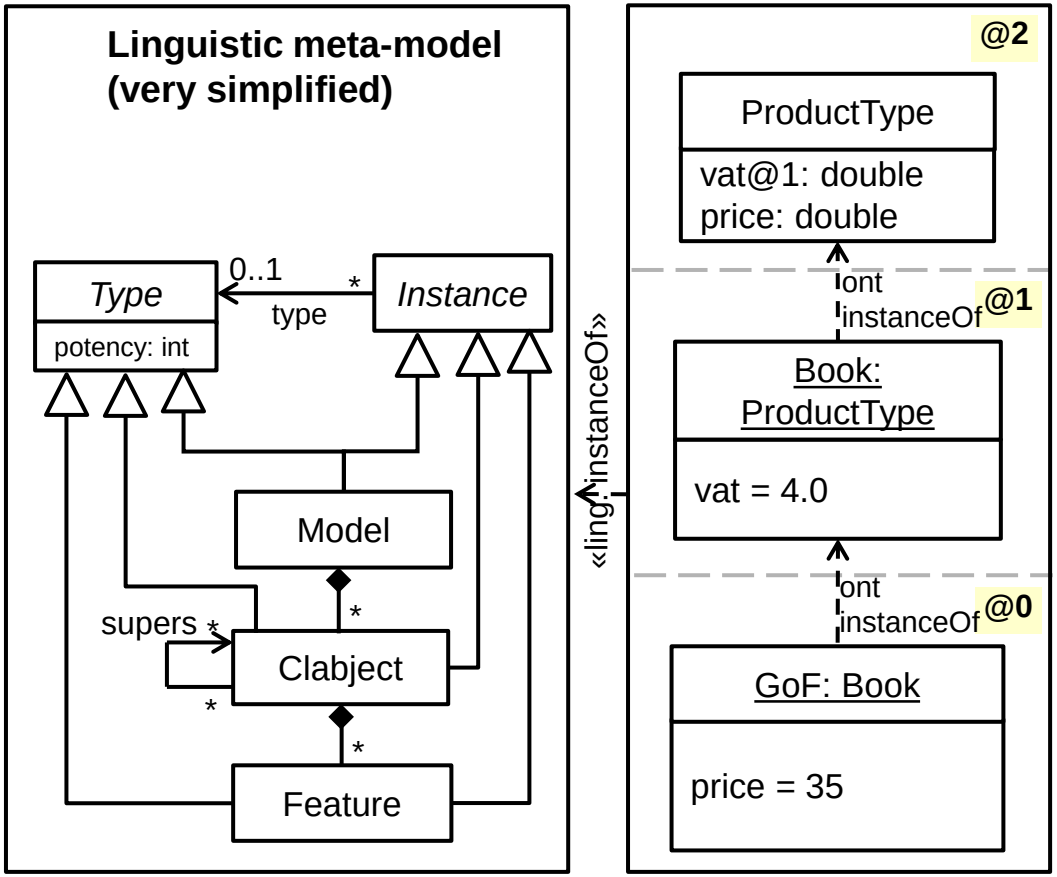
Two-Level (eg., EMF)



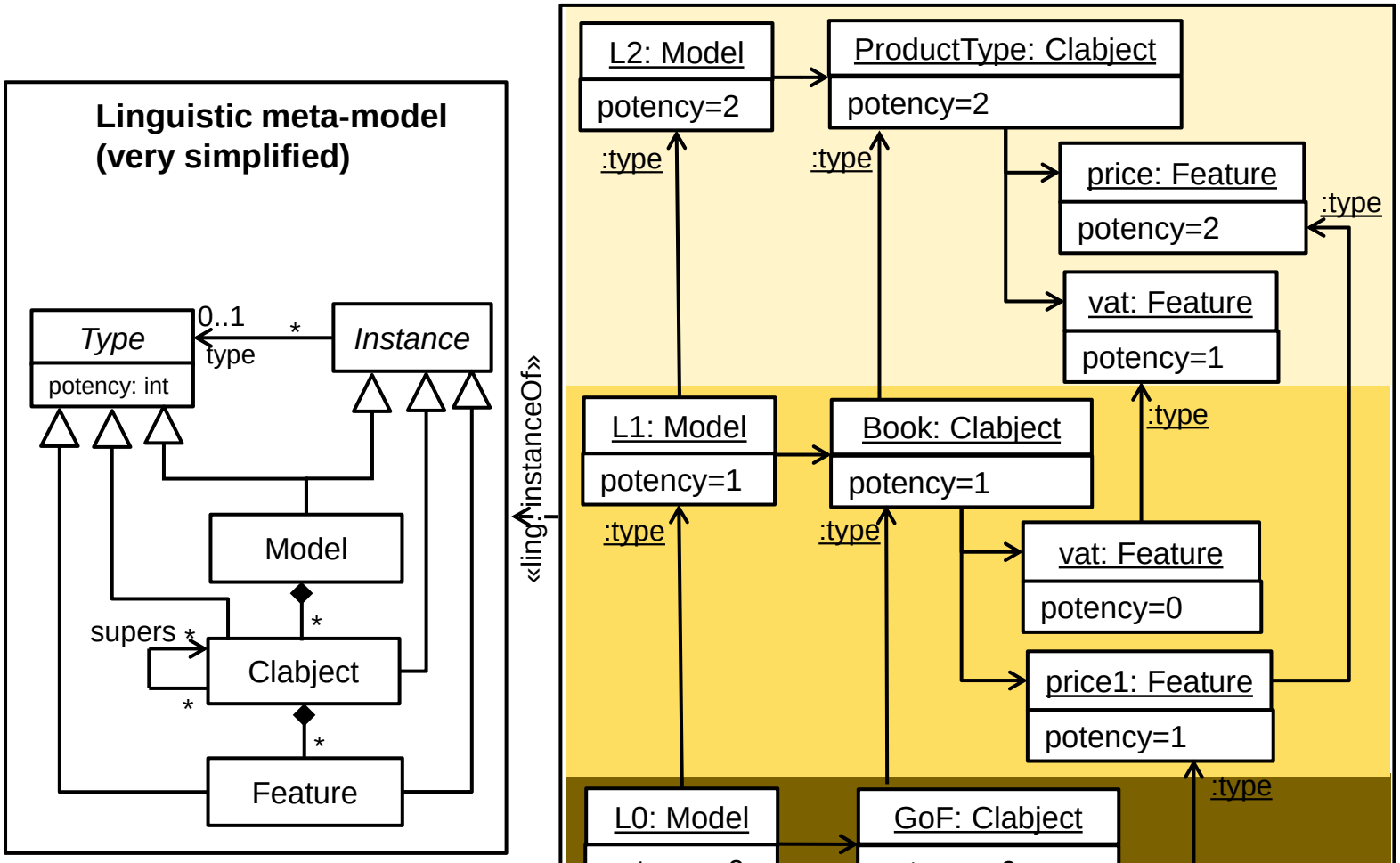
- Types and meta-modelling facilities only at the meta-model level

C. Atkinson, T. Kühne. 2002. Rearchitecting the UML infrastructure. ACM Trans. Model. Comput. Simul. 12, 4 (2002), 290–321.

ORTHOGONAL CLASSIFICATION (OCA)



LINGUISTIC VIEW



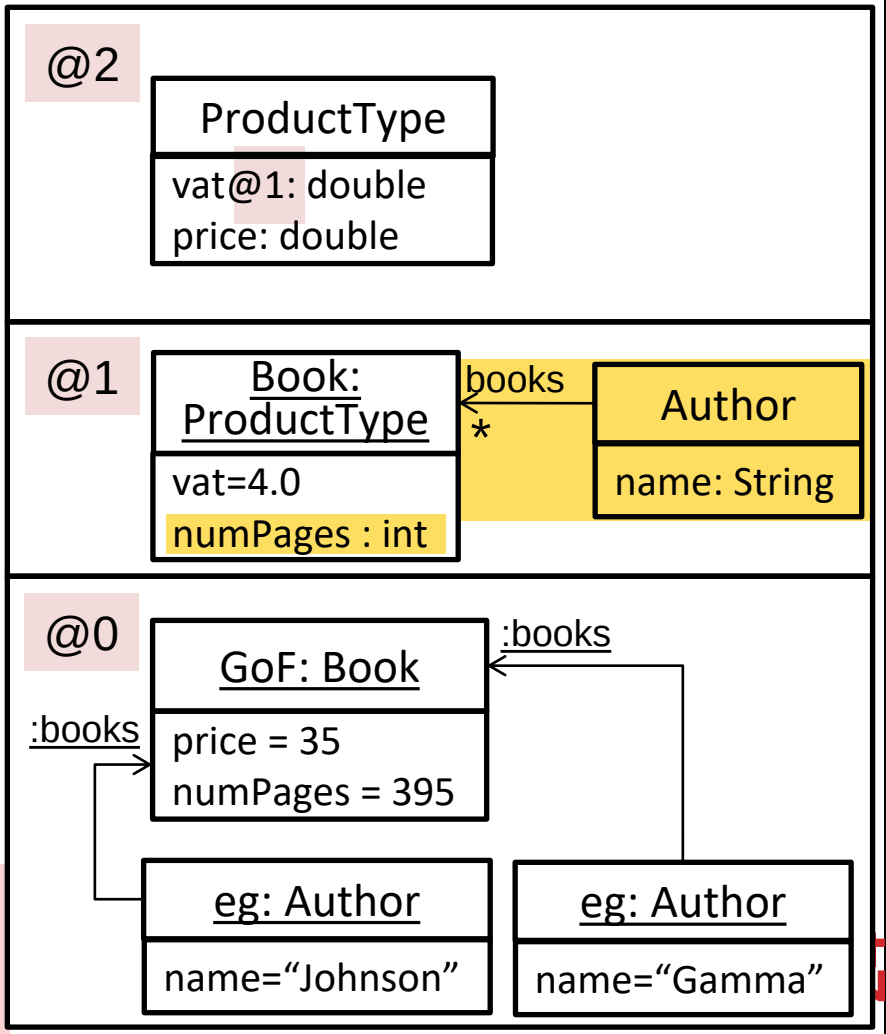
LINGUISTIC EXTENSIONS

Elements with no ontological type

- Ontological typing is optional
- Linguistic typing is mandatory

New clabjects or features

Not everything can be anticipated at the top-most level

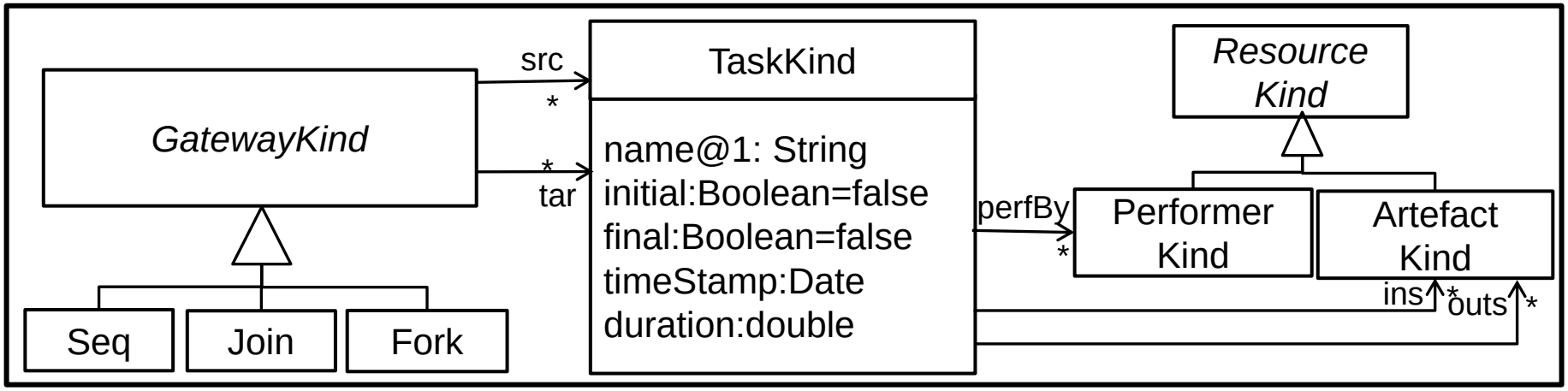


de Lara, Guerra. 2010. Deep meta-modelling with metaDepth. In TOOLS'10 (LNCS), Vol. 6141. Springer, 1–20.

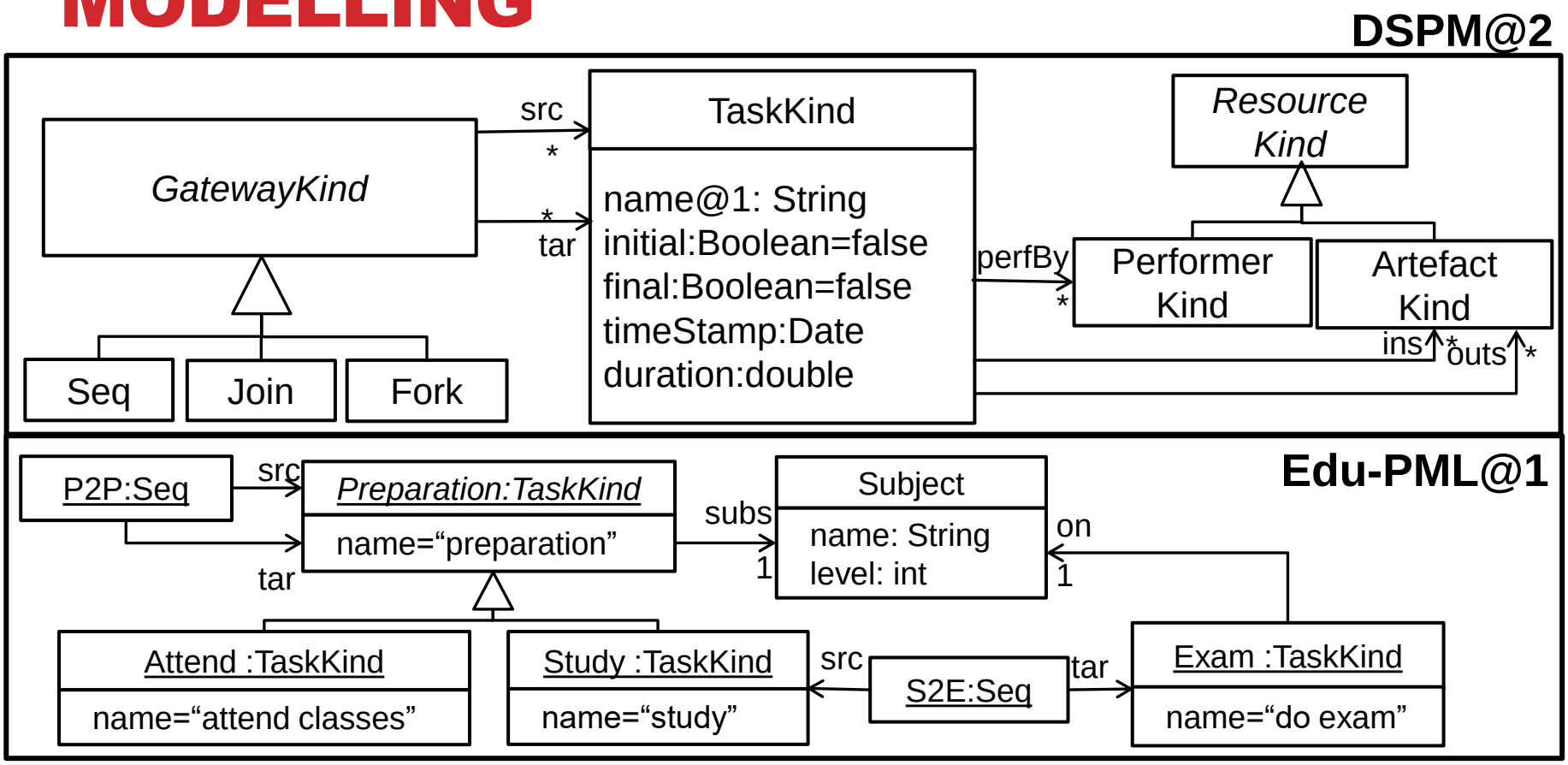
Examples

DOMAIN SPECIFIC PROCESS MODELLING

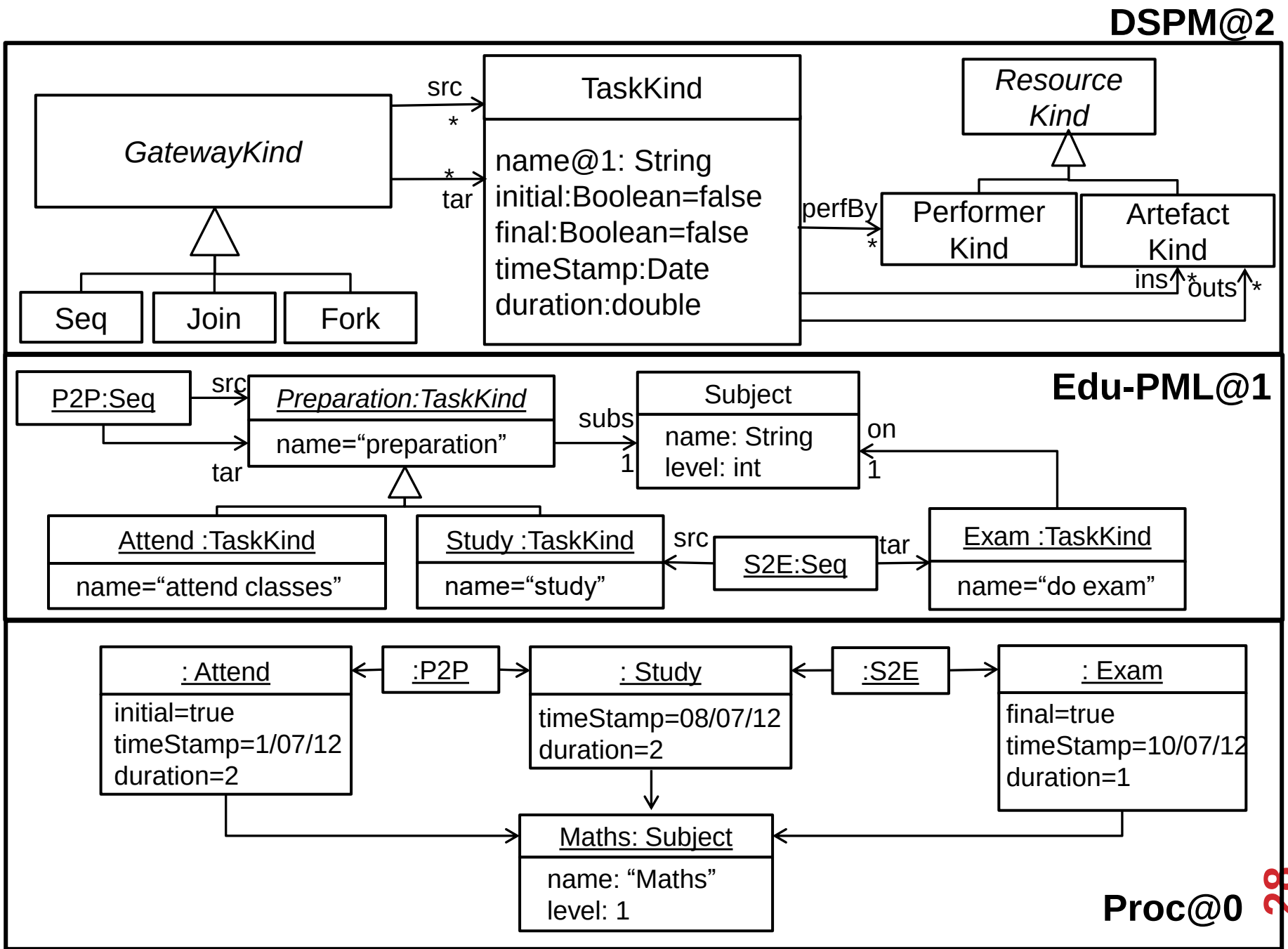
DSPM@2



DOMAIN SPECIFIC PROCESS MODELLING

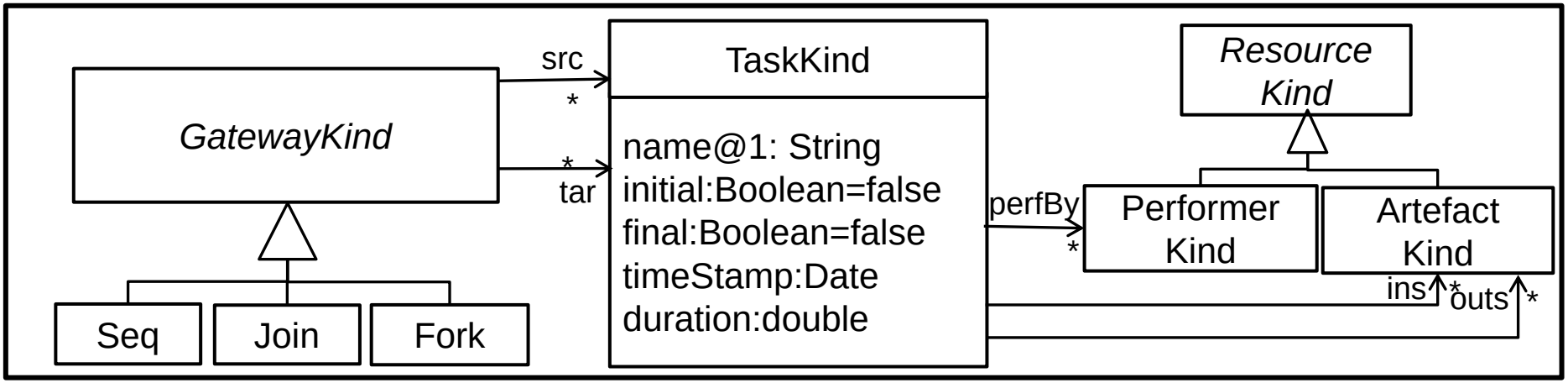


de Lara, Guerra, Sánchez Cuadrado. *Model-driven engineering with domain-specific meta-modelling languages*. Software and System Modeling 14(1): 429-459 (2015)

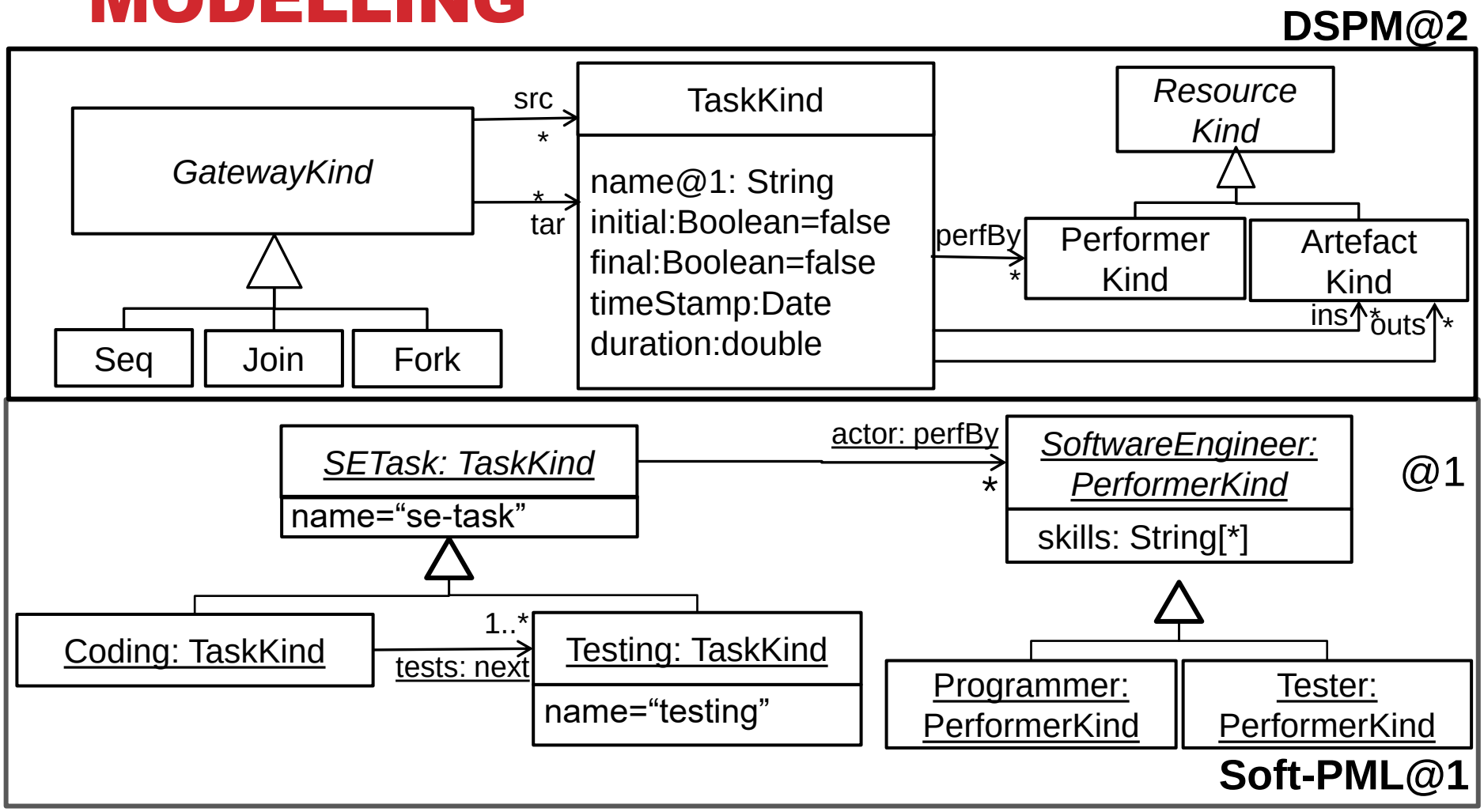

DSPM@2
GatewayKind
src
*
*
tar
TaskKind
name@1: String
initial:Boolean=false
final:Boolean=false
timeStamp:Date
duration:double
Resource Kind
perfBy
*
Performer Kind
Artefact Kind
ins
*
outs
*
Seq
Join
Fork
Edu-PML@1
P2P:Seq
src
tar
Preparation:TaskKind
name="preparation"
subs
1
Subject
name: String
level: int
on
1
Attend :TaskKind
name="attend classes"
Study :TaskKind
name="study"
src
S2E:Seq
tar
Exam :TaskKind
name="do exam"
: Attend
initial=true
timeStamp=1/07/12
duration=2
:P2P
: Study
timeStamp=08/07/12
duration=2
:S2E
: Exam
final=true
timeStamp=10/07/12
duration=1
Maths: Subject
name: "Maths"
level: 1
Proc@0

DOMAIN SPECIFIC PROCESS MODELLING

DSPM@2



DOMAIN SPECIFIC PROCESS MODELLING



ADVANTAGES

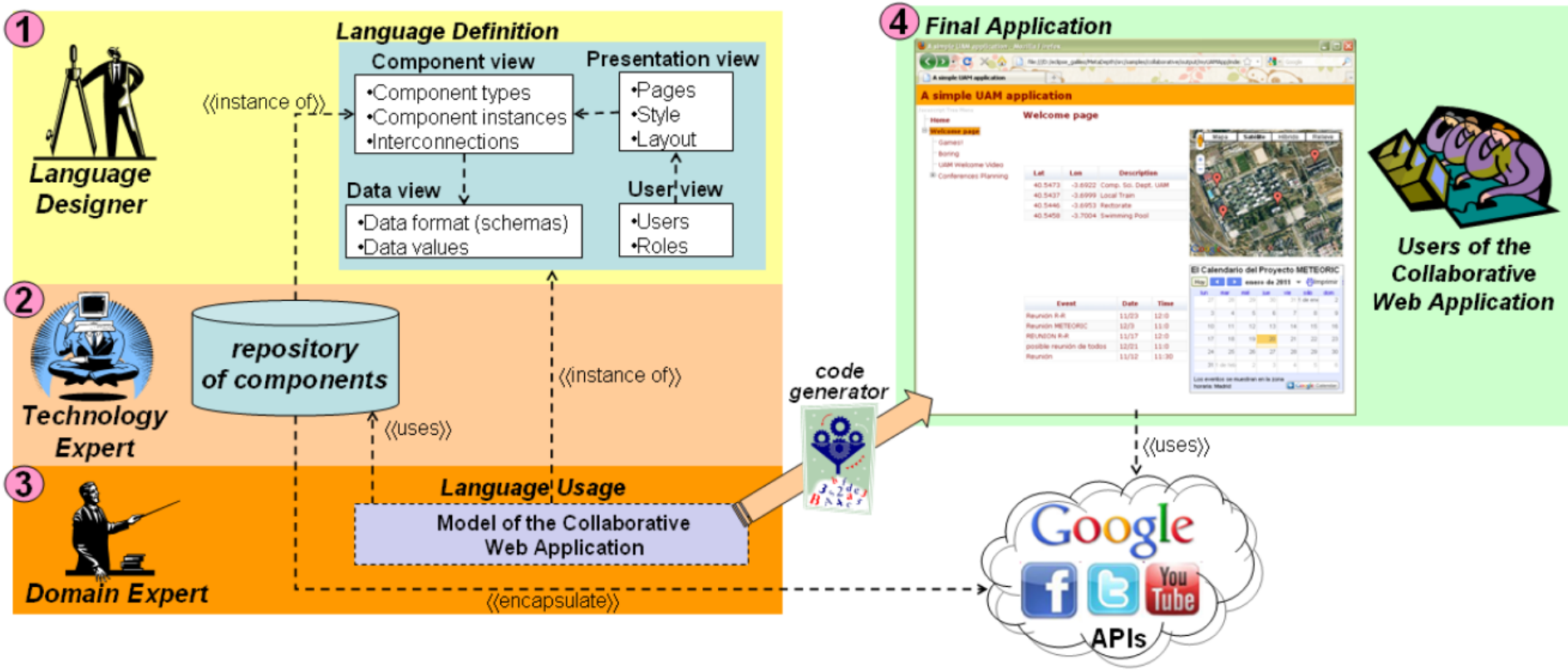
The top level can be customized for the process domain

- Family of DSLs for process modelling

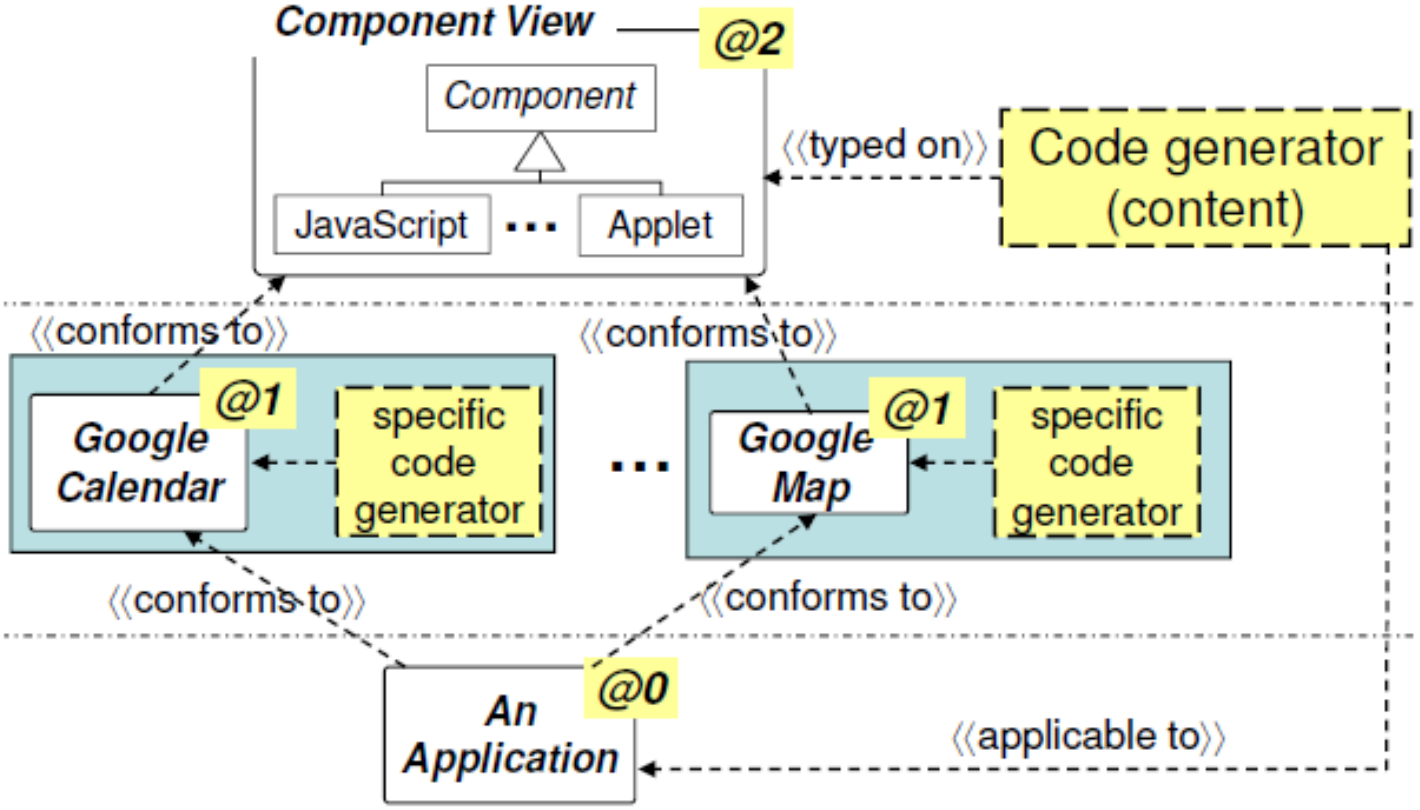
Transformations can be defined over the top level and reused across the whole family

- Code generators
- Model-to-model transformations
- In-place transformations
- Queries

COLLABORATIVE WEB APPLICATIONS



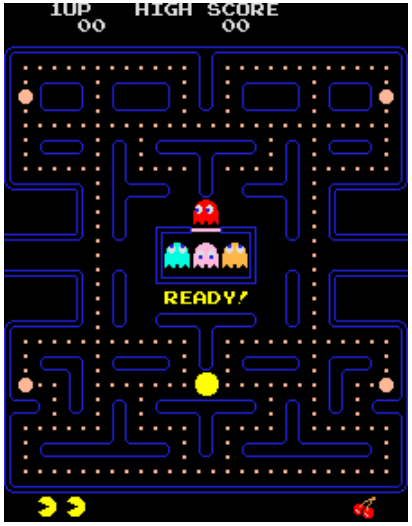
COLLABORATIVE WEB APPLICATIONS



de Lara, Guerra, Cobos, Moreno-Llorena: Extending Deep Meta-Modelling for Practical Model-Driven Engineering. Comput. J. 57(1): 36-58 (2014)

ARCADE/ADVENTURE GAMES

Sabre Wulf



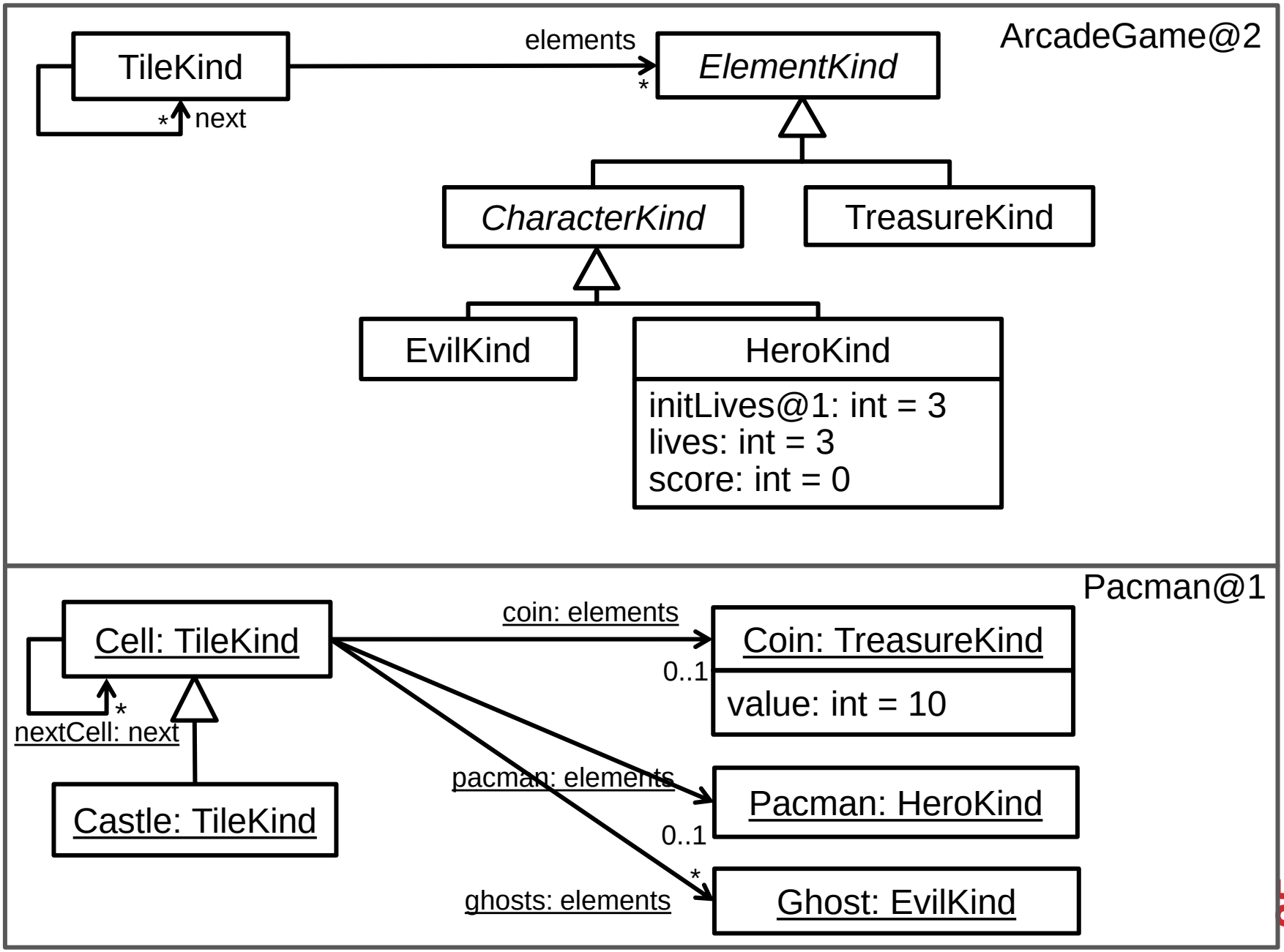
Pacman

Commando

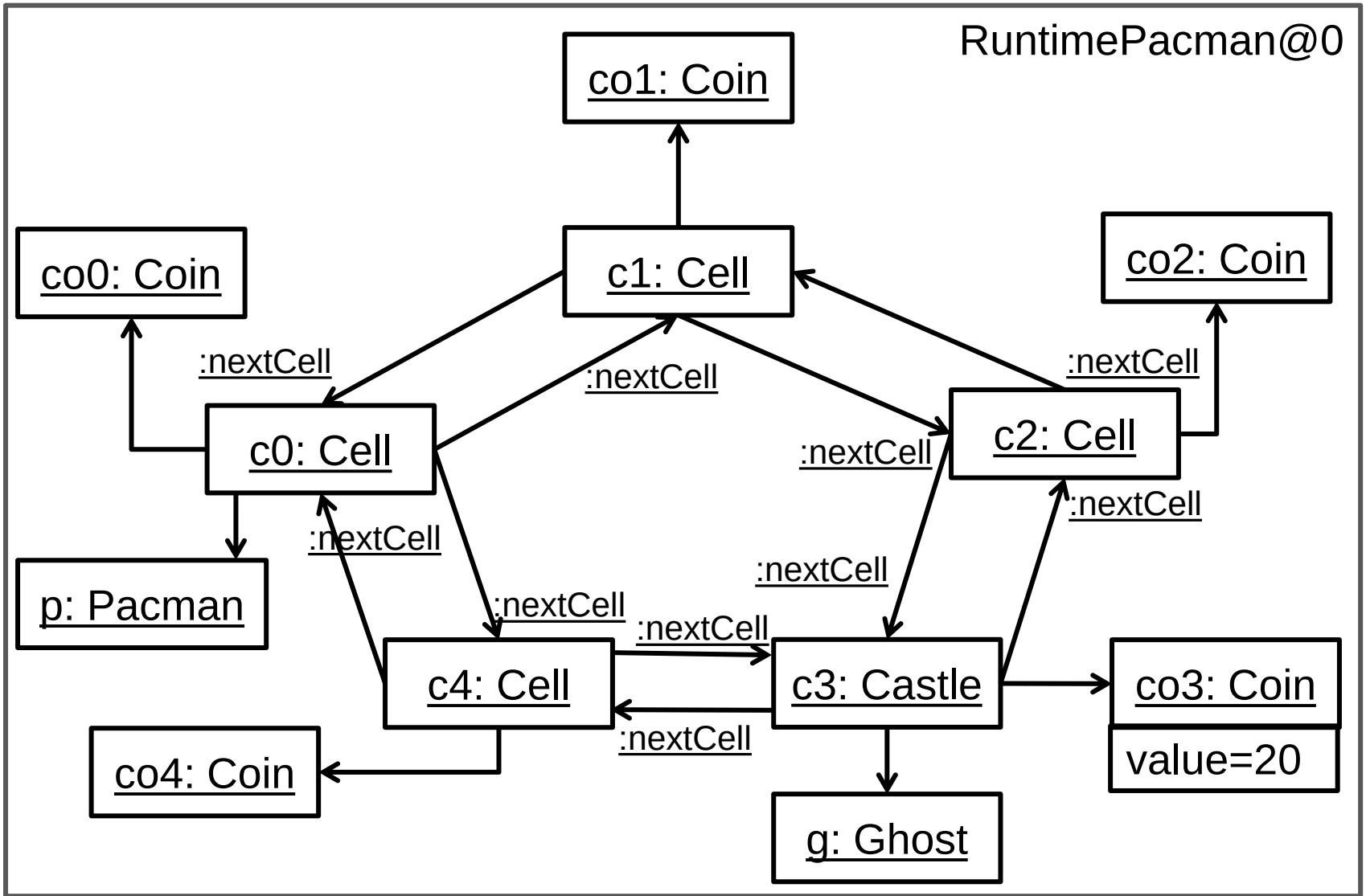
Atic atac

Gauntlet

MLM FOR ARCADE GAMES



MLM FOR ARCADE GAMES



AGENDA

Basic concepts of (potency-based) multi-level modelling

- Motivation
- Clabjects, potency, levels, OCA
- Examples

Tool Support: MetaDepth

- Case study: game development

Multi-level model management

- Constraints, transformations, code generation
- Multi-level DSLs

Advanced concepts

TOOLS!

Tool / Feature	*MetaDepth*	*Melanee*	*MultEcore*	*DPF Workbench*	*DeepRuby*	*DeepJava*	*DeepTelos*	*OMLM*	*ML2*	*DDI*	*FMMLx*
Mechanism	Native-deep (MLM meta-model)	Native-deep (MLM meta-model)	Automated promotion transformations	Native-deep (specifications)	Other (Meta-prog. in Ruby)	Other (Extension of Java)	Native-deep (Telos propositions)	Native-deep (MLM meta-model, Flora- 2)	Native-deep (powertypes)	Native-deep (Telos propositions)	Native-deep (Xcore extension with intrinsic feature)
Instance charact.	Deep (potency, leap potency, star potency)	Deep (potency, durability, mutability, star potency)	Deep (potency, range potency)	Shallow	Deep (dual potency)	Deep (potency)	Deep (most general instances)	Deep (potency, leap-potency)	Deep (regularity attributes)	Deep (dual potency)	Deep (level attribute)

...

(and more!)

Jácome-Guerrero, de Lara: TOTEM: Reconciling multi-level modelling with standard two-level modelling. Comput. Stand. Interfaces 69: 103390 (2020)

Textual multi-level modelling tool with a REPL

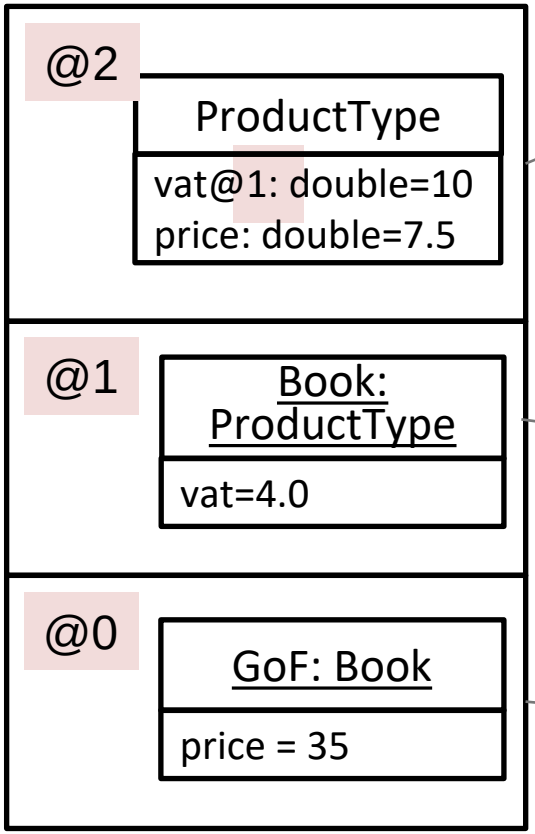
- Started in 2009
- Deep characterization based on clabjects/potency
- Orthogonal Classification Architecture
- http://metaDepth.org

Integrated with the Epsilon Languages for model management

- Constraints in EOL/EVL
- Derived attributes in EOL
- In-place transformations in EOL
- Model-to-model transformations in ETL
- Code generation in EGL

de Lara, Guerra: Deep Meta-modelling with MetaDepth. TOOLS (48) 2010: 1-20

METADEPTH



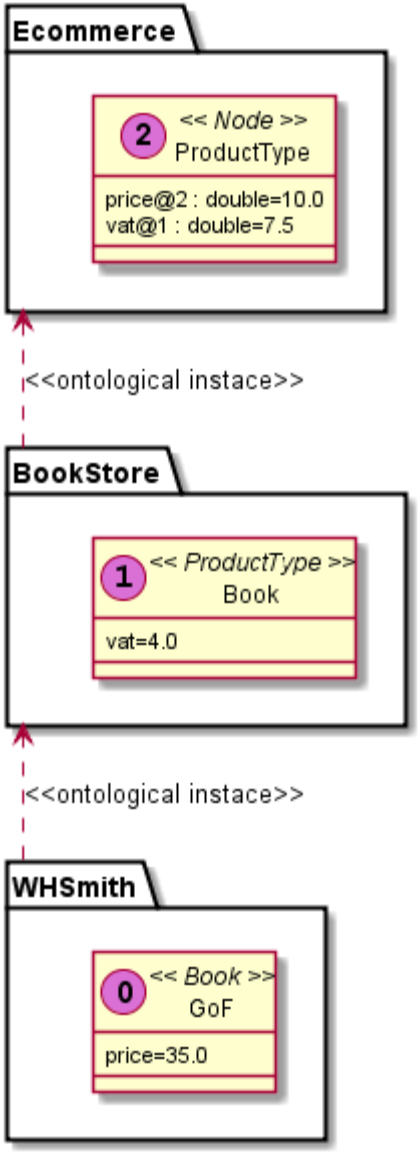

```
Model Ecommerce@2 {
  Node ProductType {
    vat@1 : double=10;
    price : double=7.5;
   }
}
```

```
Ecommerce BookStore{
  ProductType Book { vat = 4.0; }
}
```

```
BookStore WHSmith {
   Book GoF { price = 35; }
}
```

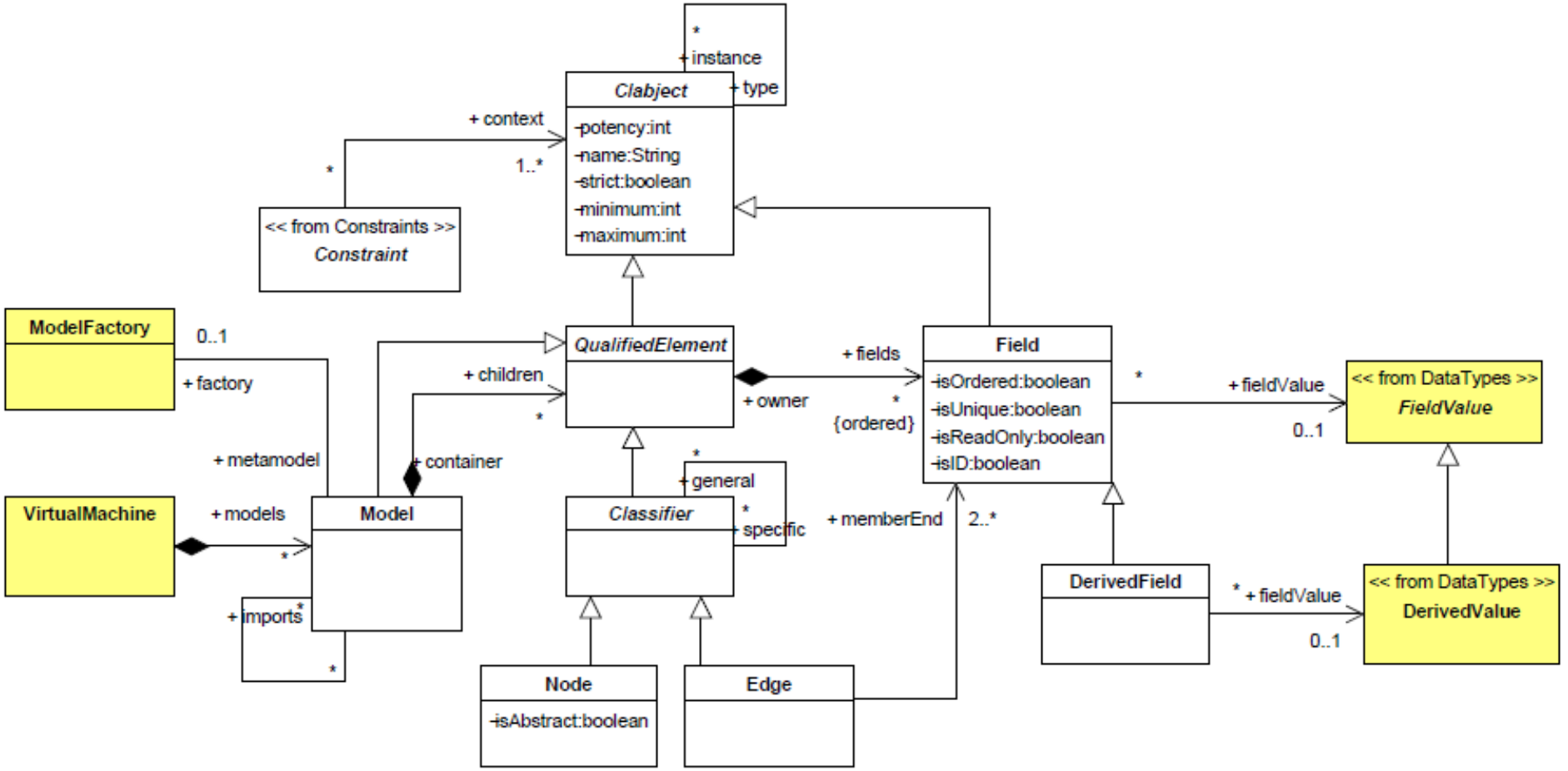
SAMPLE SESSION

```
> load “ProductSimple”
:: loading ../metadepth.samples/ProductSimple.mdepth
(6 clabjects created in 0.22 s).
> dump
:: dumping all
… (shows the models in memory)
> compile PlantUML
Executing...
:: compiling all models to PlantUML format,
generated file ./Ecommerce.txt
```



(generated png by PlantUML)

LINGUISTIC META-MODEL



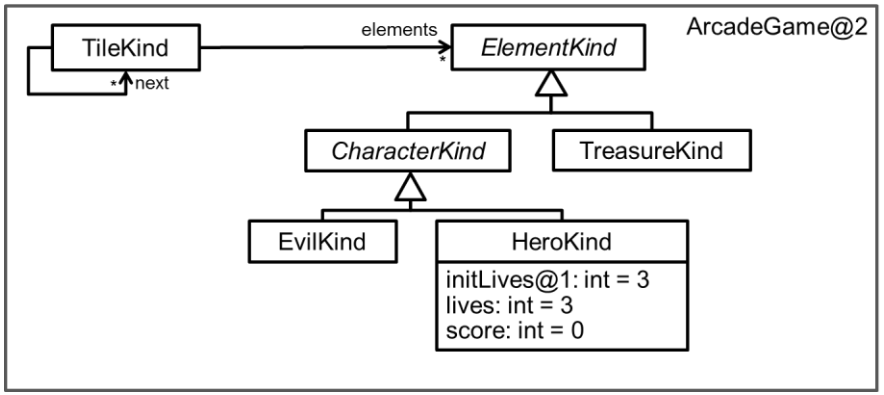
DERIVED FIELDS

```
Model Store@2 {
  Node ProductType{
    vat@1 : double = 7.5;
    price : double = 10;
    /finalPrice@2: double = $self.vat*self.price/100 +self.price$;
  }
}
```

Computation expressions for derived fields using the Epsilon Object Language (EOL)

Derived fields with primitive type, or references

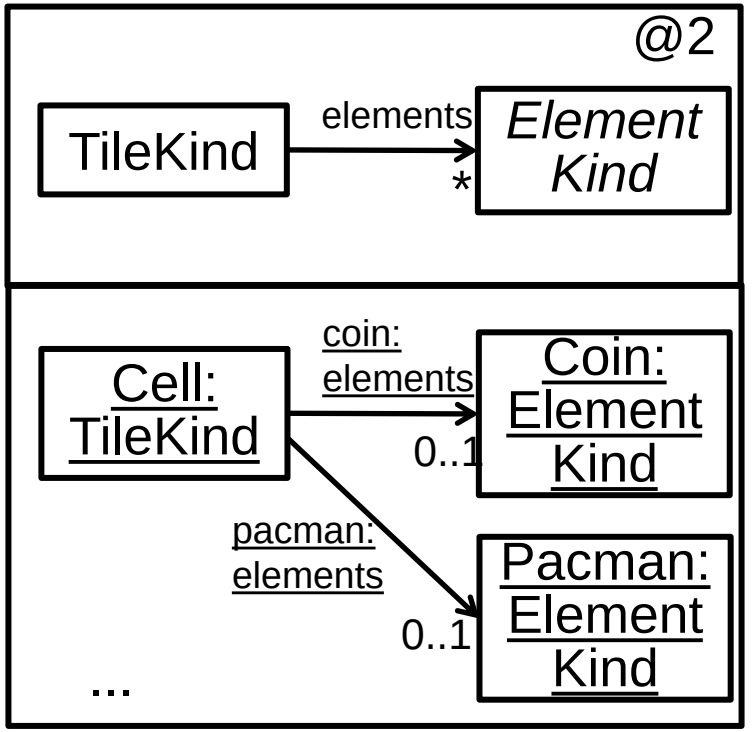
RUNNING EXAMPLE (1)




```
Model ArcadeGame@2 {
  Node TileKind {
    next       : TileKind[*];
    elements   : ElementKind[*];
  }

  abstract Node ElementKind{}

  abstract Node CharacterKind : ElementKind {  }

  Node HeroKind : CharacterKind {
    initLives@1 : int = 3;
    lives : int = 3;
    score : int = 0;
  }
  Node EvilKind : CharacterKind {}
  Node TreasureKind : ElementKind {}
}
```

RUNNING EXAMPLE (2)

```
ArcadeGame PacMan {
  TileKind Cell {
    nextCell : Cell[*] {next};
    coin     : Coin[0..1] {elements};
    pacman   : Pacman[0..1] {elements};
    ghosts   : Ghost[*] {elements};
  }
  TileKind Castle : Cell {}

  TreasureKind Coin {
    value : int=10;
  }

  HeroKind Pacman {
    initLives = 3;
  }

  EvilKind Ghost {}
}
```



RUNNING EXAMPLE (3)

```
PacMan RuntimePacman {
  Cell c0  { nextCell = [c1, c4]; coin = co0; pacman = p; }
  Cell c1  { nextCell = [c2, c0]; coin = co1; }
  Cell c2  { nextCell = [c3, c1]; coin = co2; }
  Castle c3{ nextCell = [c4, c2]; coin = co3; ghosts = g; }
  Cell c4  { nextCell = [c0, c3]; coin = co4;}

  Pacman p {}
  Ghost g {}
  Coin co0 {}
  Coin co1 {}
  Coin co2 {}
  Coin co3 { value = 20; }
  Coin co4 {}
}
```

AGENDA

Basic concepts of (potency-based) multi-level modelling

- Motivation
- Clabjects, potency, levels, OCA
- Examples

Tool Support: MetaDepth

- Case study: game development

Multi-level model management

- Constraints, transformations, code generation
- Multi-level DSLs

Advanced concepts

CONSTRAINTS

MetaDepth uses the Epsilon Object Language (EOL) to express constraints

- Also for operations and expressions for derived fields

EOL is a variant of OCL

- Permits side effects (assignments, object creation)
- Has imperative constructs (e.g., loops)

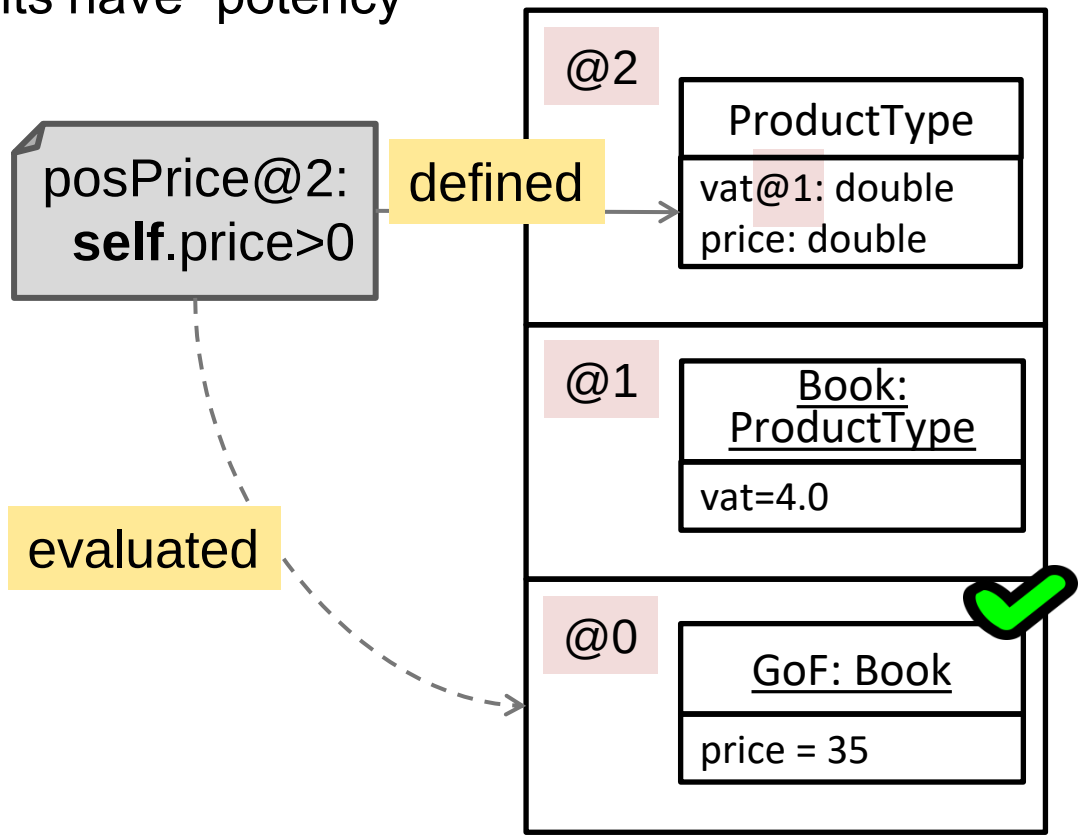
The basis of other Epsilon languages

- ETL for model-to-model transformation
- EGL for code generation

DEEP CONSTRAINT LANGUAGE

Level at which the constraint is defined vs level at which we want to evaluate it

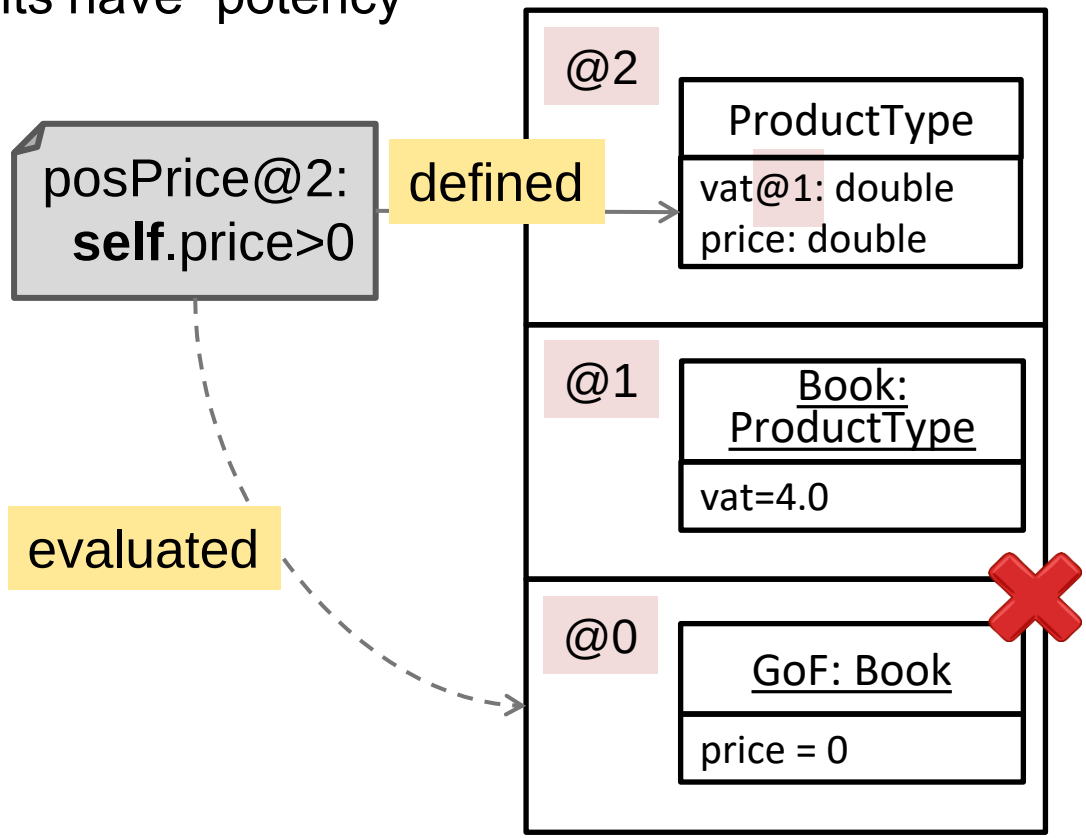
- Constraints have “potency”



DEEP CONSTRAINT LANGUAGE

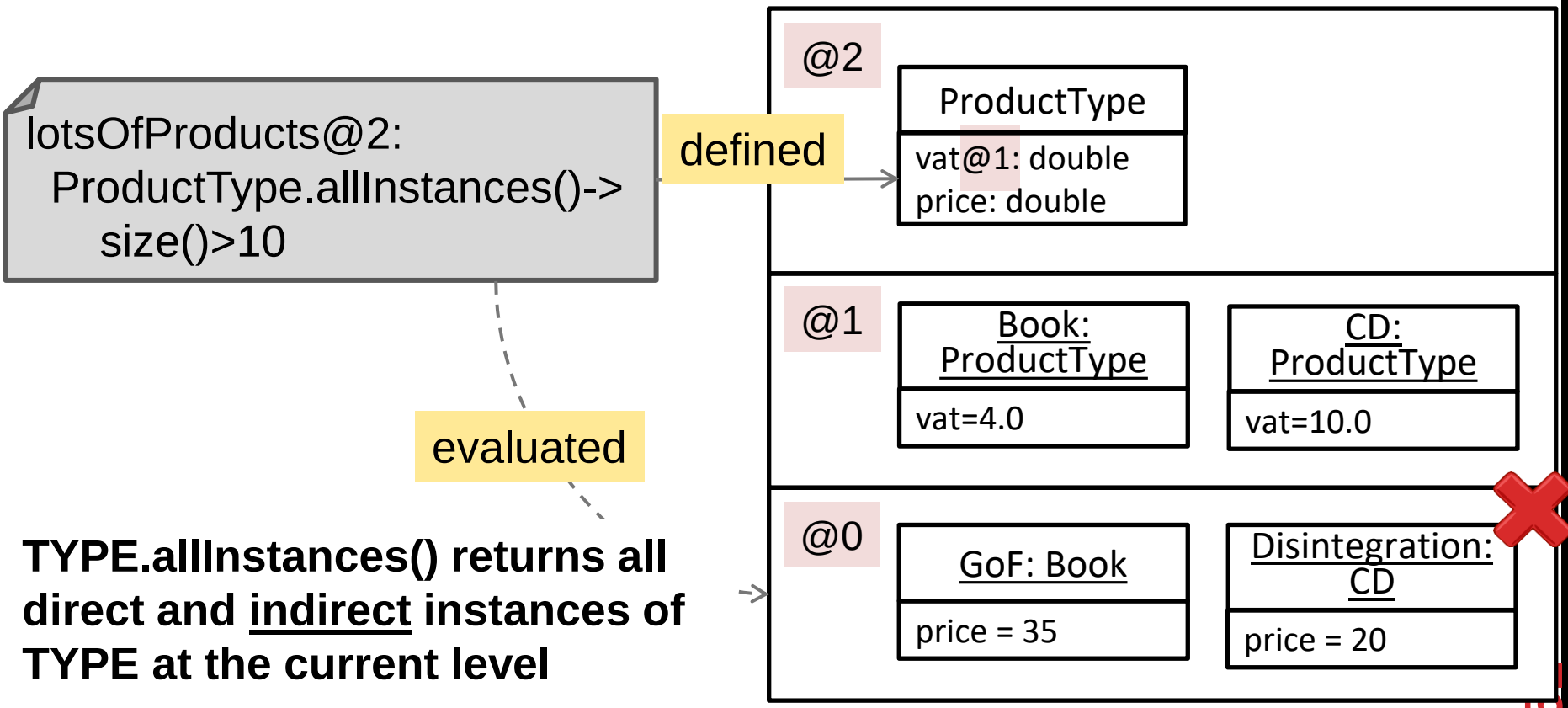
Level at which the constraint is defined vs level at which we want to evaluate it

- Constraints have “potency”



TRANSITIVE TYPING

We might not know type names at intermediate levels



TYPE.allInstances() returns all direct and indirect instances of TYPE at the current level

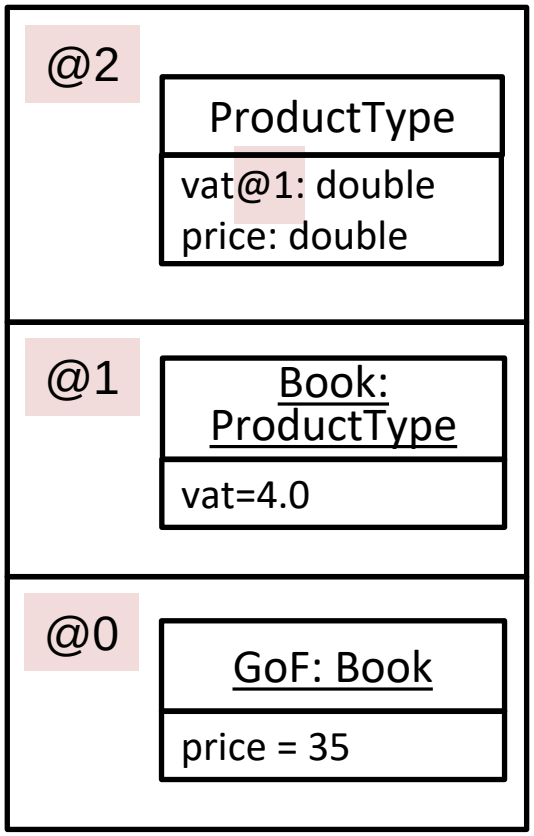
LEVEL NAVIGATION

Make (upwards) meta-level navigation implicit

- At level 0:

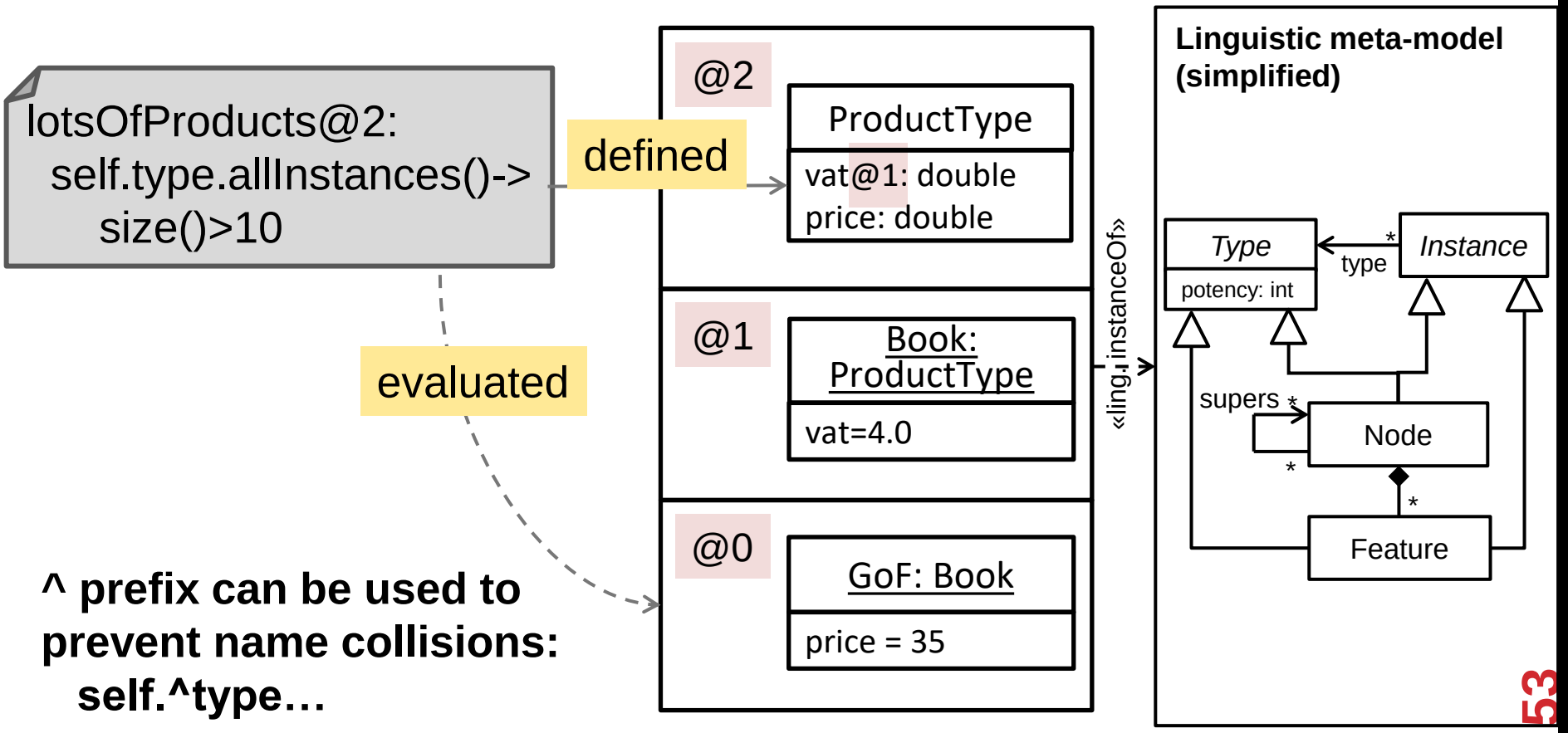
```
> GoF.vat.println();
> 4.0
```

Fields with instance facet in clabjects can be accessed in instance clabjects.



LINGUISTIC META-MODEL

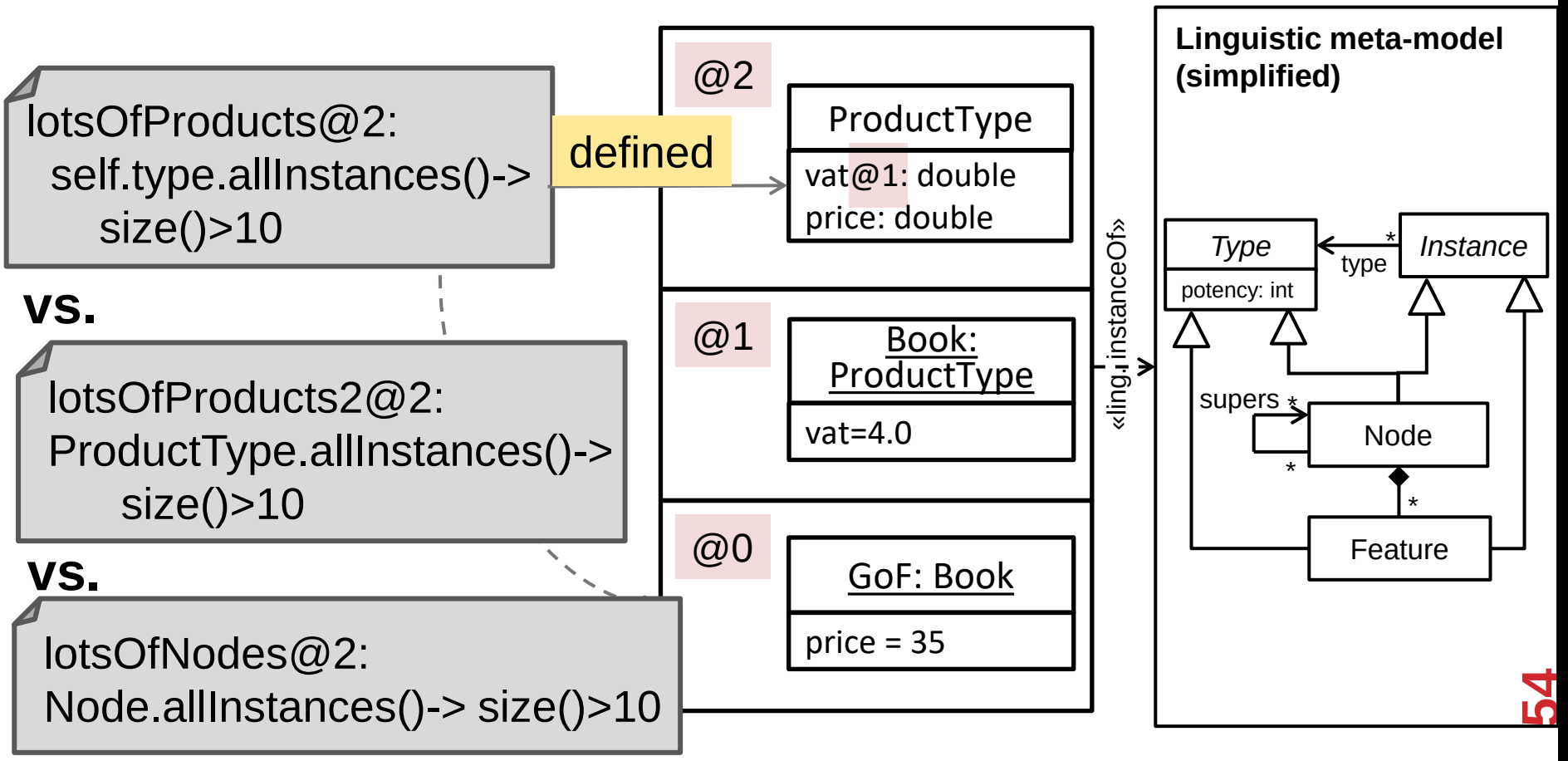
Transparent access to the linguistic meta-model in constraint expressions



^ prefix can be used to prevent name collisions:
self.^type…

LINGUISTIC META-MODEL

Transparent access to the linguistic meta-model in constraint expressions



RUNNING EXAMPLE

We'll add a few constraints to ensure that:

- There are no isolated tiles at level 0
- Treasures, Heroes and Evils are connected to one tile at level 0
- There is at least one hero at level 0
- There are no less evils than heroes at level 0

- There can be no Pacman in the castle

A derived attribute to calculate the number of (direct) reachable tiles at level 0.

RUNNING EXAMPLE (LEVEL 2)

```
Model ArcadeGame@2 {
  Node TileKind {
    next      : TileKind[*];
    elements  : ElementKind[*];

    /numReachable : int = $self.next.size()$;
    noIsolated : $self.numReachable>0 or
                 TileKind.all.exists( t | t.next.includes(self))$
  }
```

no isolated tiles at level 0

```
  abstract Node ElementKind{
     connected : $TileKind.all.one( t | t.elements.includes(self))$
  }
```

treasures, heores and evils are connected to exactly one tile

```
  abstract Node CharacterKind : ElementKind {  }

  Node HeroKind : CharacterKind {
    initLives@1 : int = 3;
    lives : int = 3;
    score : int = 0;
  }
  Node EvilKind : CharacterKind {}
  Node TreasureKind : ElementKind {}
```

(global constraints) at least one hero, more evils than heroes

```
  someHero@2  : $HeroKind.all.size()>=1$
  moreEvils@2 : $EvilKind.all.size() >= HeroKind.all.size()$
}
```

RUNNING EXAMPLE (LEVEL 1)

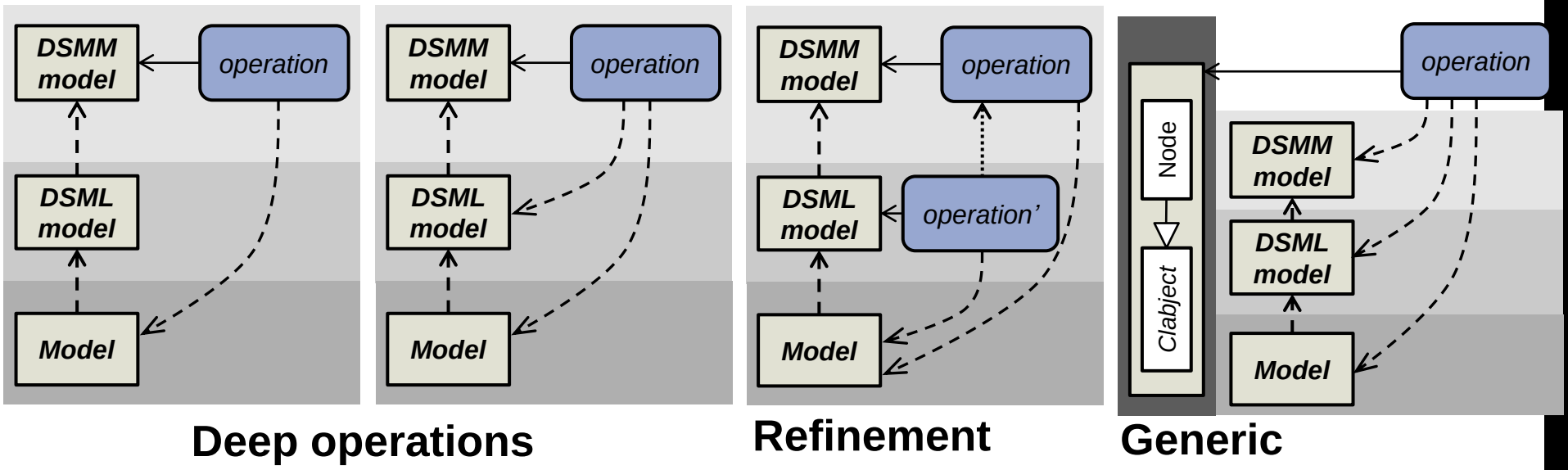
```
ArcadeGame PacMan {
  TileKind Cell {
    nextCell : Cell[*] {next};
    coin     : Coin[0..1] {elements};
    pacman   : Pacman[0..1] {elements};
    ghosts   : Ghost[*] {elements};
  }

  TileKind Castle : Cell {
    noPacmans : $not self.pacman.isDefined()$
  }

  TreasureKind Coin {
    value : int=10;
  }

  HeroKind Pacman {
    initLives = 3;
  }

  EvilKind Ghost {
  }
}
```

no Pacman in the Castle

Conflicts and consistency of (ML) constraints can be analyzed:

Guerra, de Lara: Automated analysis of integrity constraints in multi-level models. Data Knowl. Eng. 107: 1-23 (2017)

MULTI-LEVEL MODEL MANAGEMENT



Legend

defined on — applicable to - - - - - - redefines ·····

IN-PLACE MODEL TRANSFORMATION

Infrastructure for a family of DSLs

- Operations can be attached to meta-classes
- Operations with potency

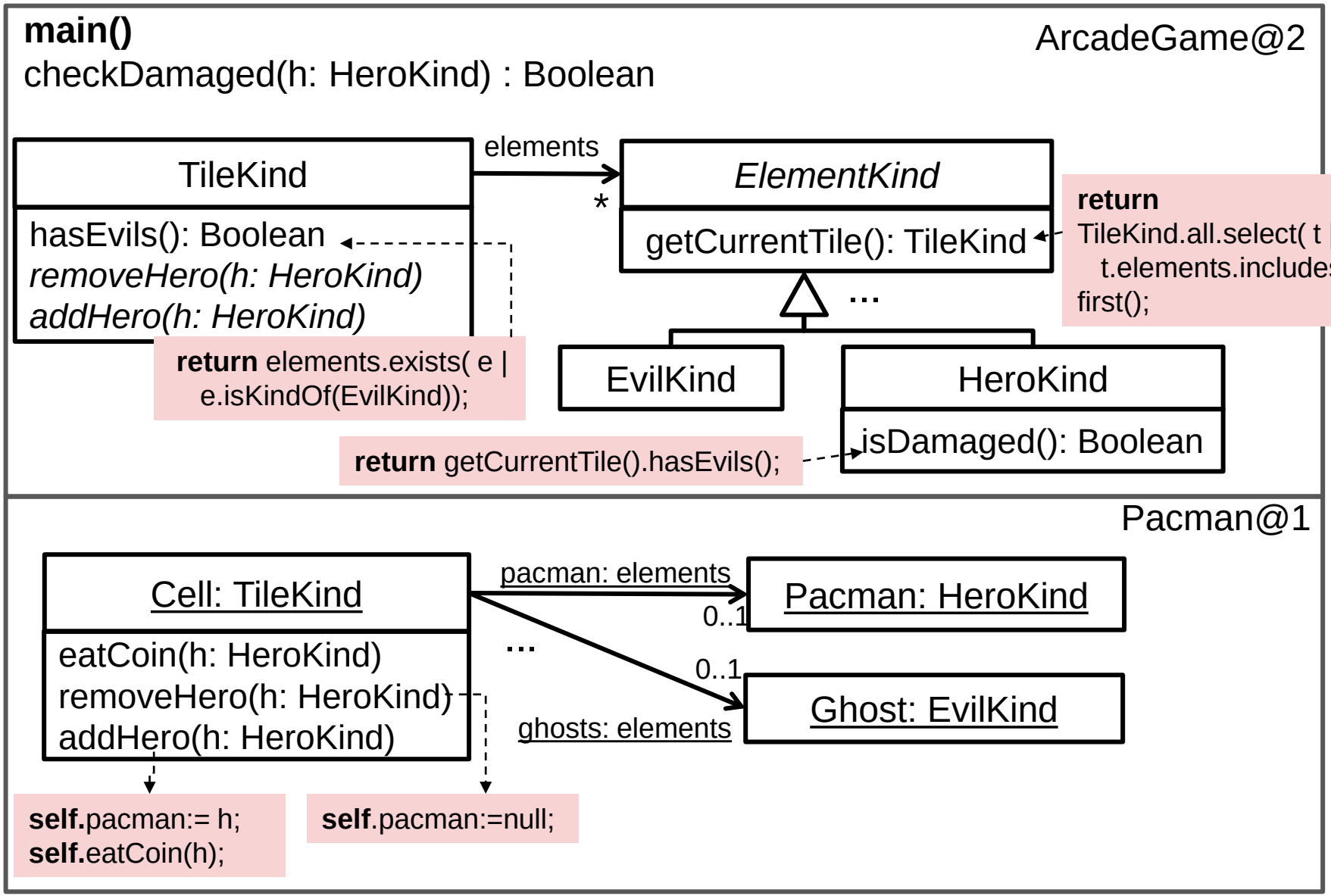
Design similar to an object oriented framework

- Common operations
- Operations expected to be overriden at next meta-level

For the running example:

- A game engine for the DSL (autonomous play)
- Core of the engine will be common (basic movement rules, game over and winning condition)
- Some operations can be overriden at level 1 (ie., particular to Pacman)

ARCADE GAME SIMULATOR



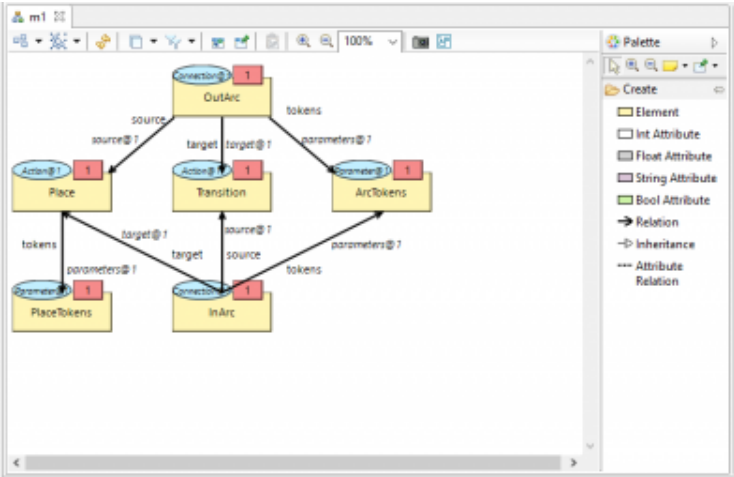
OTHER APPROACHES

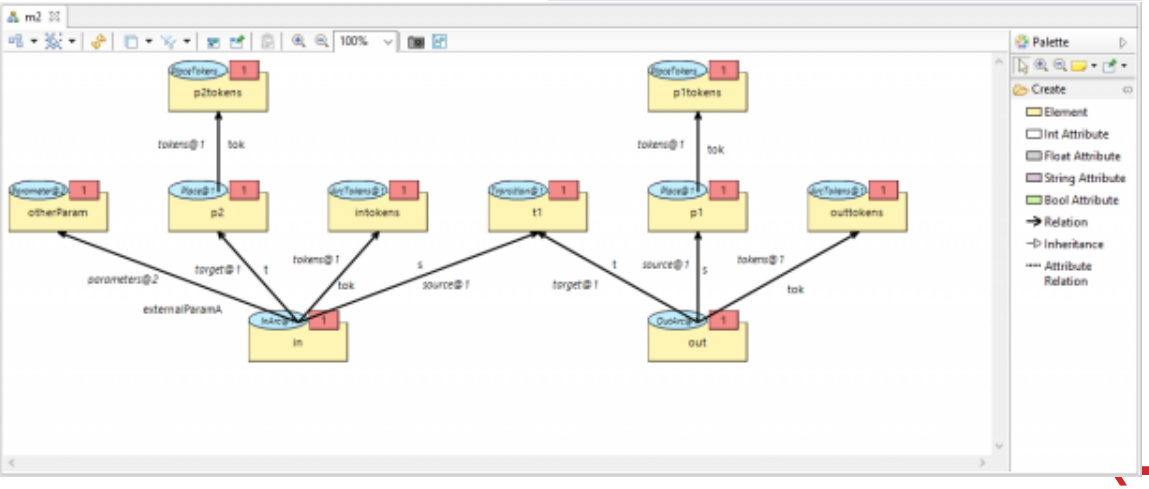
Multecore (https://ict.hvl.no/multecore/)

- A tool for MLM within Eclipse
- Graphical modelling
- Formal theory based on graph transformation [WMR20]
- By translation into rewriting logic (MAUDE)

test.multecore

```
module my101
model: "http://example/model"

rule CreatePart {
  meta {
    P1: mm[0]!Part
    M1: mm[0]!Machine
    cr: mm[0]!Machine.creates
    [M1.cr = P1]
  }

  from {
    m1: M1
  }

  to {
    p1: P1
    m1: M1
    c: cr
    [m1.c = p1]
  }
}
```





Wolter, Macías, Rutle: Multilevel Typed Graph Transformations. ICGT 2020, *163-182*

MODEL-TO-MODEL TRANSFORMATION

Many scenarios:

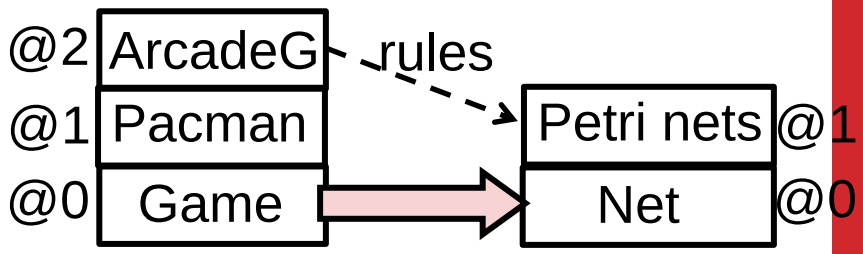
- Deep transformations (eg. defined at @2, applied at @0)
- Co-transformations (eg. defined at @2, applied at @1 and @0)
- Refining transformations (overriding rules at @1)
- Reflective & linguistic transformations (generic, using linguistic types)
- Multi-level target (eg. creating models at @1 and @0)

For the running example:

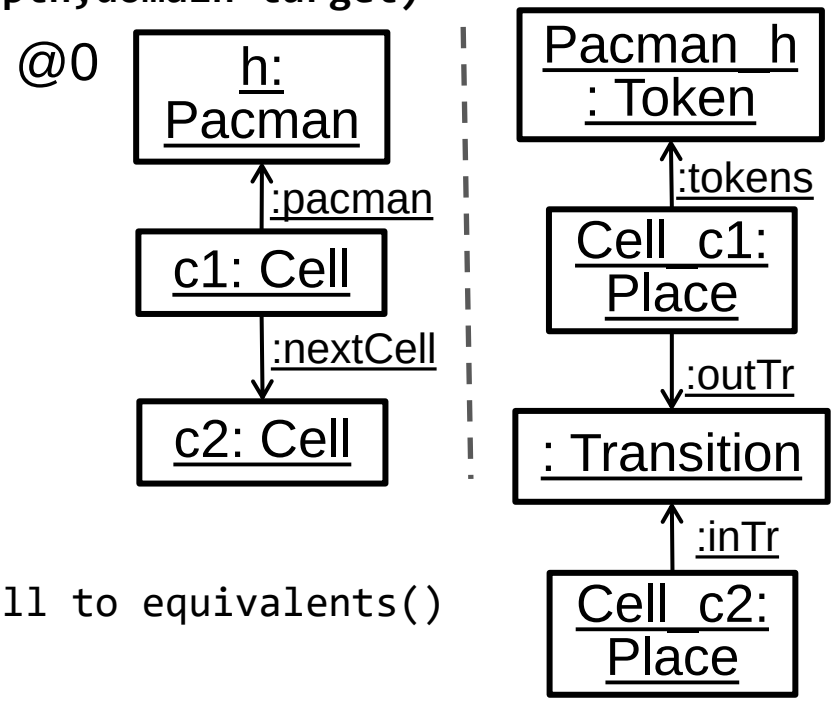
- A (deep) transformation into Petri nets
- Reusable for any concrete adventure game

de Lara, Guerra, Sánchez Cuadrado. Model-driven engineering with domain-specific meta-modelling languages. SoSyM 14(1): 429-459 (2015)

MODEL-TO-MODEL TRANSFORMATION

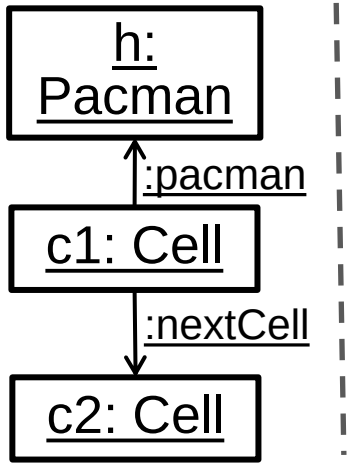


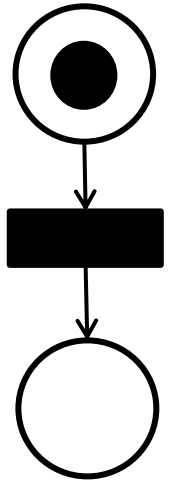

```
@metamodel(name=ArcadeGame,file=Arcade.mdepth,domain=source)
@metamodel(name=PetriNet,file=PetriNet.mdepth,domain=target)
rule CharacterToToken
  transform h : SLevel0!CharacterKind
  to        p : Target!Token {
     p.^name := h.^type+'_'+h.^name;
  }

rule TileToPlace
  transform t : SLevel0!TileKind
  to        p : Target!Place  {
    p.name := t.type.name+'_'+t.^name;
    p.^name := t.type.name+'_'+t.^name;
    p.tokens ::= t.elements;// implicit call to equivalents()
    for ( n in t.next) {
      var tr := new Target!Transition;
      tr.name := t.^name+' to '+n.^name;
      tr.^name := t.^name+'_to_'+n.^name;
      p.outTr := tr;
      n.equivalents().first().inTr := tr;
    }
}
```



MODEL-TO-MODEL TRANSFORMATION

```
@metamodel(name=ArcadeGame,file=Arcade.mdepth,domain=source)
@metamodel(name=PetriNet,file=PetriNet.mdepth,domain=target)
rule CharacterToToken
  transform h : SLevel0!CharacterKind
  to        p : Target!Token {
     p.^name := h.^type+'_'+h.^name;
  }

rule TileToPlace
  transform t : SLevel0!TileKind
  to        p : Target!Place  {
    p.name := t.type.name+'_'+t.^name;
    p.^name := t.type.name+'_'+t.^name;
    p.tokens ::= t.elements;// implicit call to equivalents()
    for ( n in t.next) {
      var tr := new Target!Transition;
      tr.name := t.^name+' to '+n.^name;
      tr.^name := t.^name+'_to_'+n.^name;
      p.outTr := tr;
      n.equivalents().first().inTr := tr;
    }
}
```

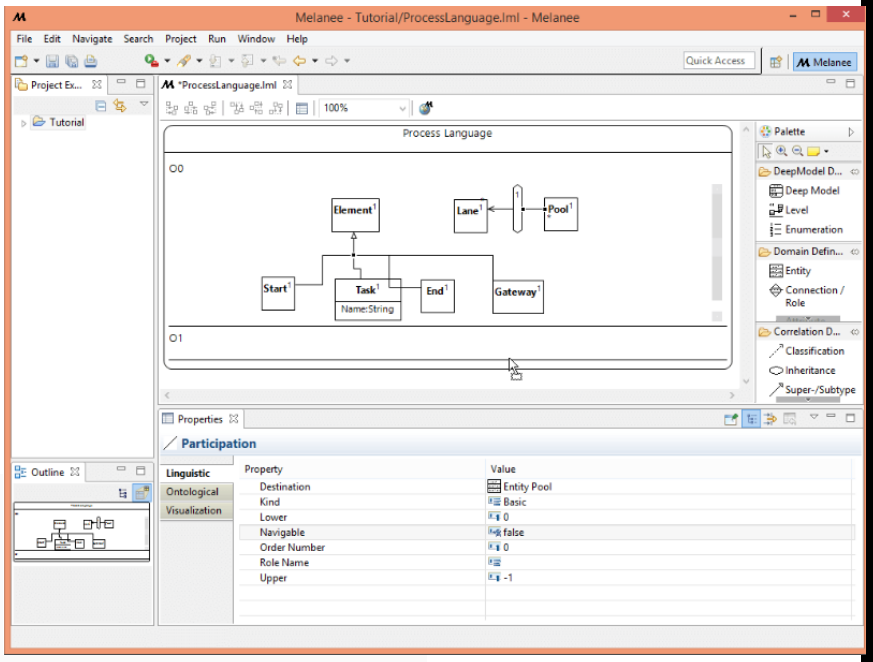




OTHER APPROACHES

Melanee (http://www.melanee.org**)**

- A tool for MLM within Eclipse
- Graphical modelling (and more)
- Enhancement of the ATL transformation language for multi-level modelling [AGT15]




```
rule Component2Class {
 from s : SUM!O0.Component 1
 to t : STRUCTURE!Class (
  name <- s._l_.name
 )
 do {
  for (i in s._l_.getModelDirectInstances())
        {
   thisModule.createInstances(i, t); }
...
```

Atkinson, Gerbig, Tunjic. Enhancing classic transformation languages to support multi-level modeling. Softw. Syst. Model. 14(2): 645-666 (2015)

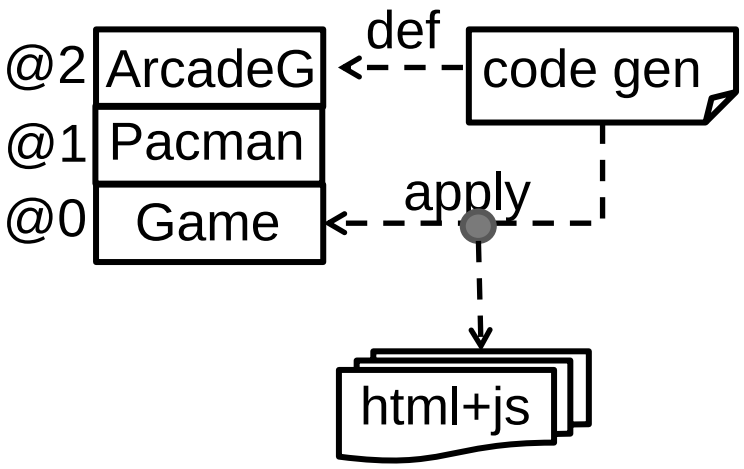
CODE GENERATION

Visualization of the game

- vis.js (http://visjs.org)

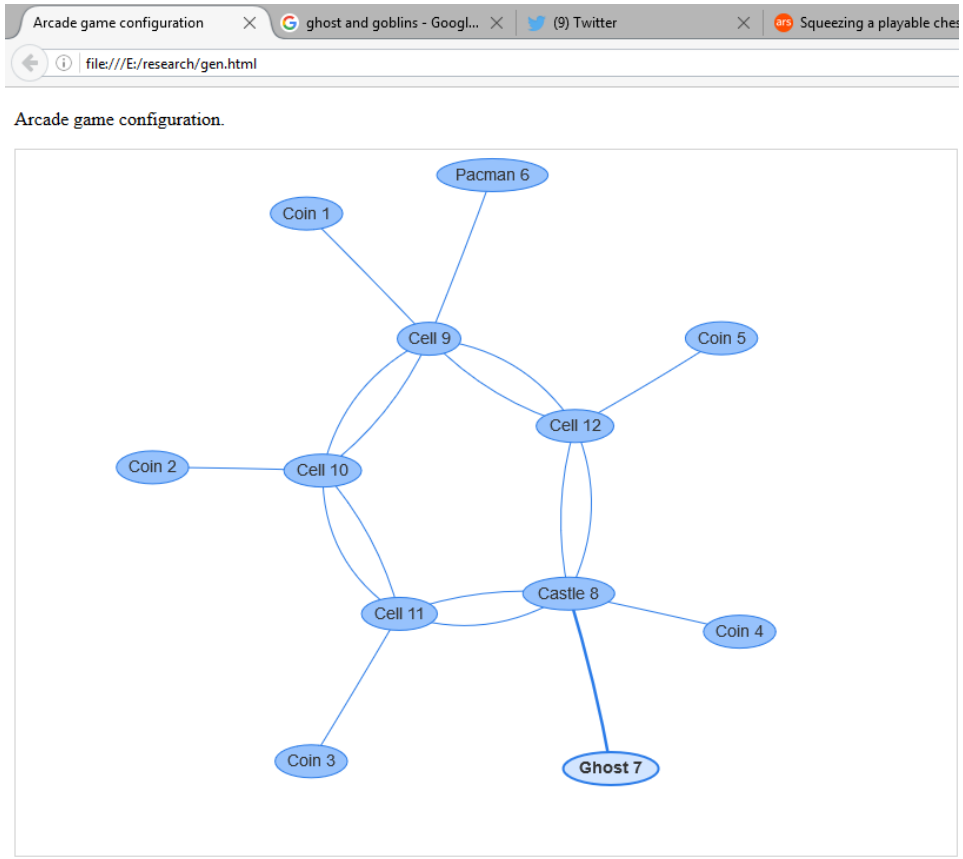
Code generator

- Deep (reusable for the DSL)
- Generic (linguistic types)




```
<!doctype html>
<html>
<head>
…
<script type="text/javascript">
  // create an array with nodes
  var nodes = new vis.DataSet([
  [%  var counter : Integer := 0;
      // Generic
      for(t in Level0!Node.all) {
        counter := counter+1;
  %]
  [%if (counter > 1) {%], [%}%]
    {id: [%=counter%],
     label: '[%=t.type%] [%=counter%]'}
    [% t.~identif := counter; %]
  [%}%]
  ]);
…
</html>
```

GENERATED CODE




```
<script type="text/javascript">
  // create an array with nodes
  var nodes = new vis.DataSet([
    {id: 1, label: 'Coin 1'}
,     {id: 2, label: 'Coin 2'}
,     {id: 3, label: 'Coin 3'}
,     {id: 4, label: 'Coin 4'}
,     {id: 5, label: 'Coin 5'}
,     {id: 6, label: 'Pacman 6'}
,     {id: 7, label: 'Ghost 7'}
,     {id: 8, label: 'Castle 8'}
,     {id: 9, label: 'Cell 9'}
,     {id: 10, label: 'Cell 10'}
,     {id: 11, label: 'Cell 11'}
,     {id: 12, label: 'Cell 12'}
  ]);
```

AGENDA

Basic concepts of (potency-based) multi-level modelling

- Motivation
- Clabjects, potency, levels, OCA
- Examples

Tool Support: MetaDepth

- Case study: game development

Multi-level model management

- Constraints, transformations, code generation
- Multi-level DSLs

Advanced concepts

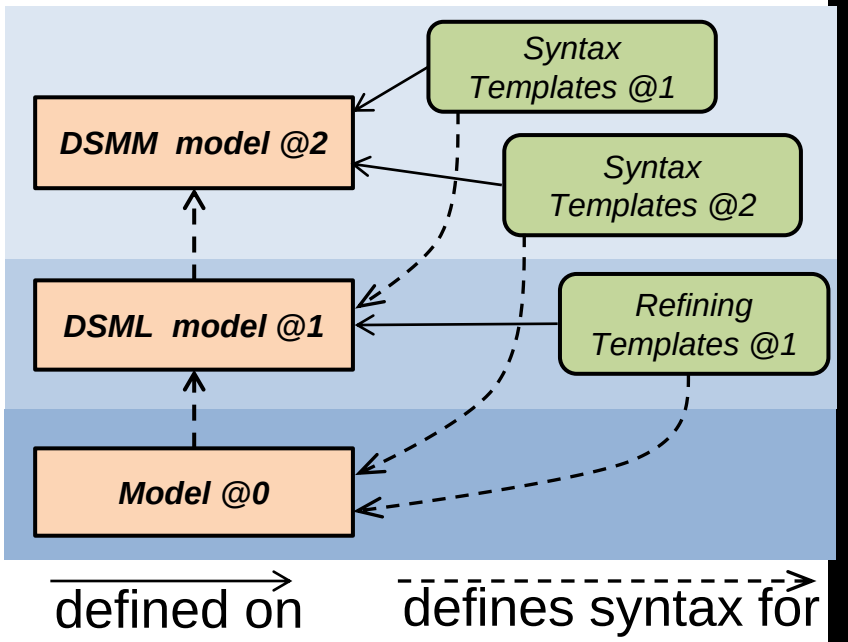
CUSTOMIZING THE TEXTUAL SYNTAX

MetaDepth supports a uniform textual concrete syntax accross meta-levels.

Define customized syntax

- Pen(8%) instead of
- Product Pen {VAT=8;}

Challenges imposed by a multi-level setting.



TEXTUAL CONCRETE SYNTAX

```
strict Model Ecommerce@2 {
  strict Node Product {
    VAT@1 : double;
    price : double;
  }
}
```

Meta-Level 2

```
Syntax for Ecommerce [".ecommerce_mm"] {
  template@1 TEcommerce for Model Ecommerce:
    "id '{' &TProduct * '}' "
  template@1 TProduct for Node Product:
    "id '(' #VAT '%' ')' "
}
```

```
Stationer {
  Pen(8%)
}
```

Meta-Level 1

```
Syntax for Ecommerce [".ecommerce"] {
  template@2 DeepProds for Model Ecommerce:
    "typename id '{' &DeepProd * '}' "
  template@2 DeepProd for Node Product:
    "typename id '(' #price '€' ')' "
}
```

```
Stationer WHSmith {
  Pen parker(5.0€)
}
```

Meta-Level 0

REFINEMENT TEMPLATES

We can design a special syntax for specific clabjects of potency 1

```
strict Model Ecommerce@2 {
  Node Product {
    VAT@1 : double;
    price : double;
  }
}
```

template@1 TNail **for Node** Nail:
"id '(' #price '€' ',' #caliber ',' #length ')' **"**

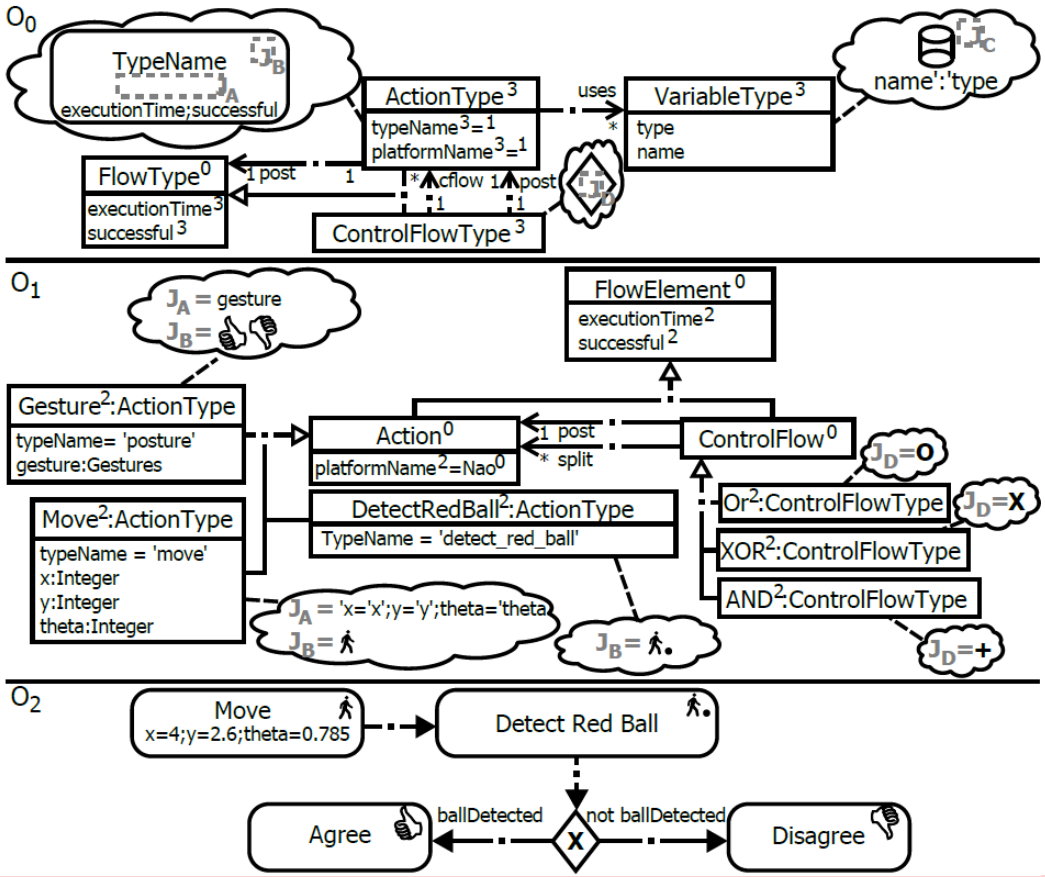
```
Hardware {
  Nail (8.0%){
    caliber: double;
    length: double;
    bigger: $self.caliber>=0.1$
  }
}
```

```
Hardware DoItBest {
  n1(0.1€, 0.1, 10)
}
```

GRAPHICAL CONCRETE SYNTAXES

Melanee

- Graphical
- Form-based
- Virtual Reality



Atkinson, Gerbig: Aspect-oriented Concrete Syntax Definition for Deep Modeling Languages. MULTI@MoDELS 2015: 13-22

Gerbig: Deep, Seamless, Multi-format, Multi-notation Definition and Use of Domain-specific Languages. University of Mannheim, Germany, Dr. Hut 2017

AGENDA

Basic concepts of (potency-based) multi-level modelling

- Motivation
- Clabjects, potency, levels, OCA
- Examples

Tool Support: MetaDepth

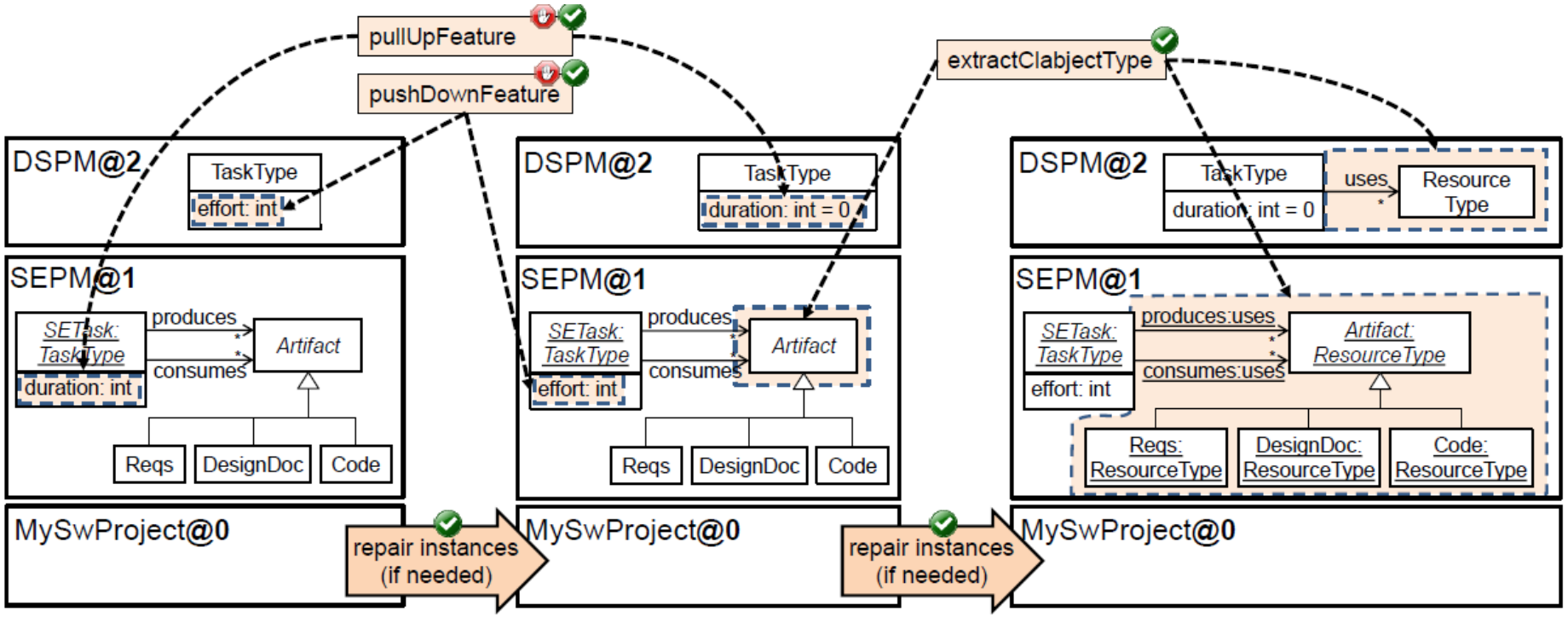
- Case study: game development

Multi-level model management

- Constraints, transformations, code generation
- Multi-level DSLs

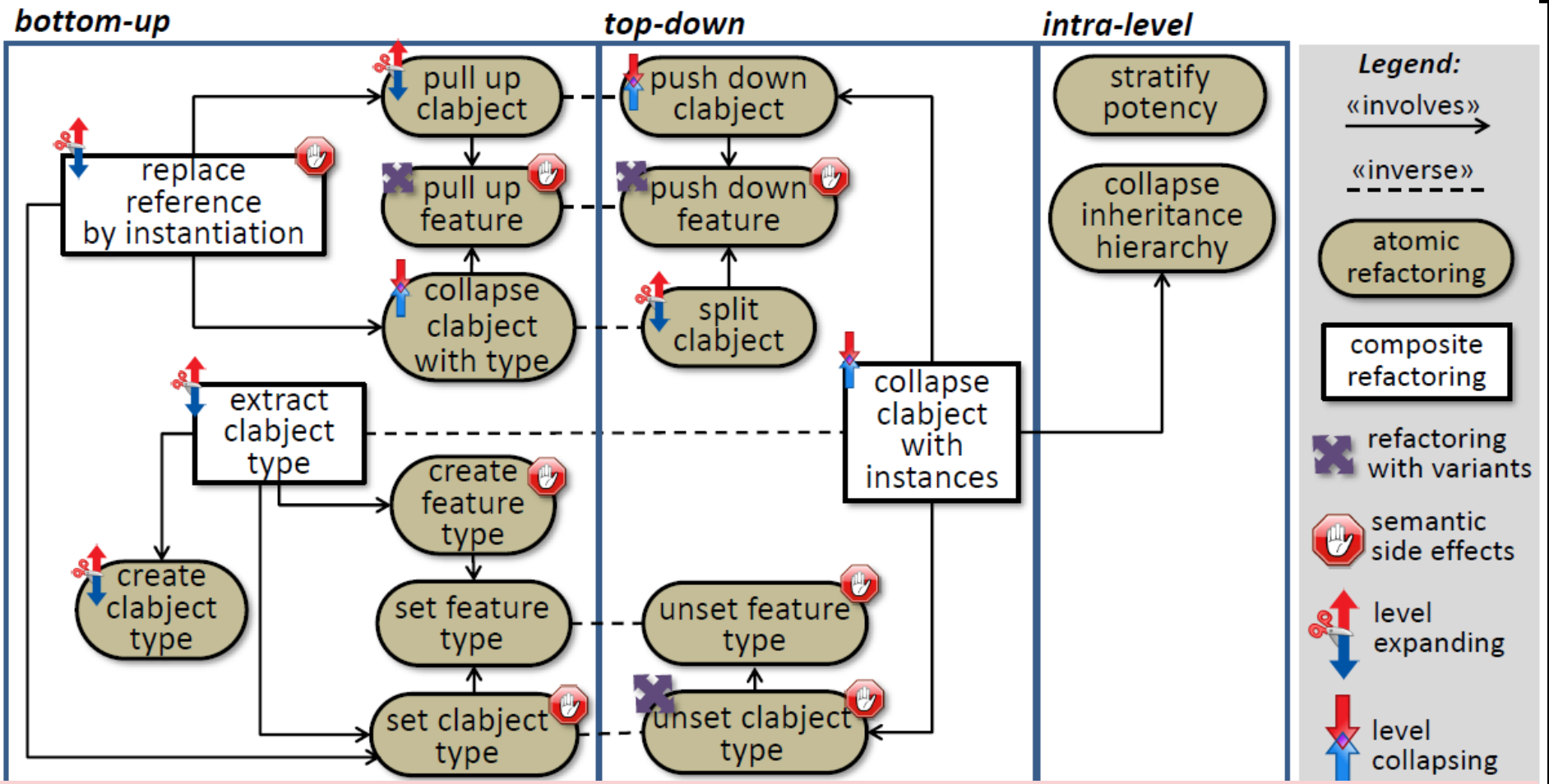
Advanced concepts

MULTI-LEVEL REFACTORINGS



Refactoring multi-level models. 2018. Juan de Lara, Esther Guerra. ACM Transactions on Software Engineering and Methodology, Vol 27(4)

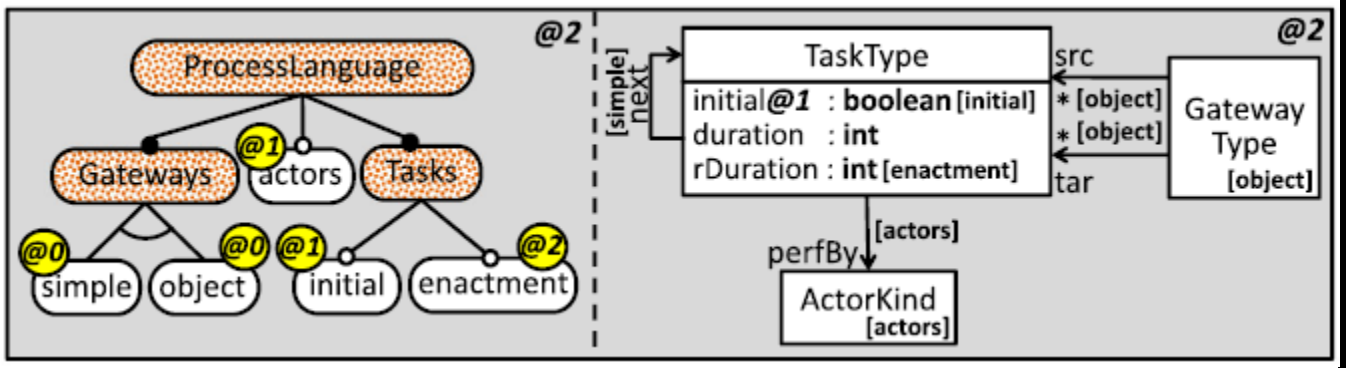
MULTI-LEVEL REFACTORINGS



Refactoring multi-level models. 2018. Juan de Lara, Esther Guerra. ACM Transactions on Software Engineering and Methodology, Vol 27(4)

MULTI-LEVEL MODEL PRODUCT LINES

ML languages with variability



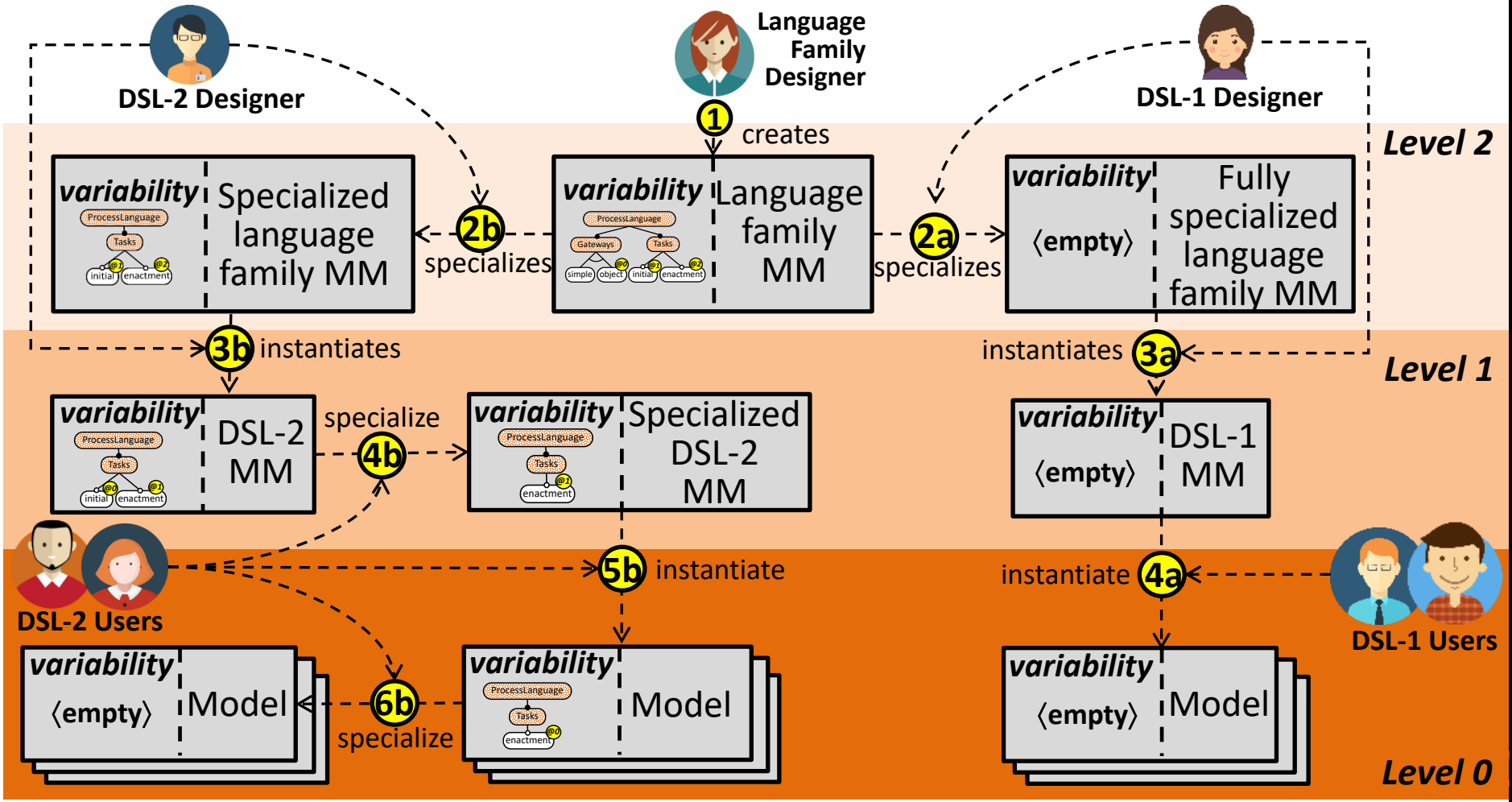
Users can configure the languages at different levels using a multi-level feature model

Open variability: instantiation

Closed variability: configuration

de Lara, J., Guerra, E. Language Family Engineering with Product Lines of Multi-level Models. *Form Asp Comp* (2021).

USING MULTI-LEVEL MODEL PRODUCT LINES



CONCLUSIONS AND SUMMARY

Multi-level modelling

- Simplifies modelling in some scenarios
- Relevant in practice (OMG specs, process modelling, architectural languages)

Families of DSLs

- Common semantics and services
- Defined through MDE techniques

Tooling

- MetaDepth (but many others are available)

WHAT'S NEXT?

Use in real projects

- (Large) case study reports
- Is potency flexible enough in practice?

Bridges with two-level modelling

- EMF
- Automated rearchitecture from 2-level into multi-level

Bridges between multi-level modelling tools

- Interoperability with other tools

THANKS!

Juan.deLara@uam.es

@miso_uam

http://www.miso.es

https://metadepth.org/