ENGINEERING FAMILIES OF MODELLING LANGUAGES

A TALE OF THREE APPROACHES

Juan de Lara

Universidad Autónoma de Madrid (Spain)

Universidad Autónoma de Madrid

MLE’22 @ MODELS, Montreal

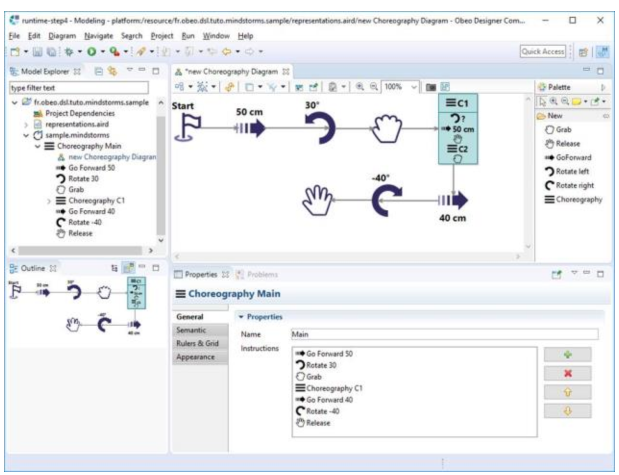
DOMAIN-SPECIFIC LANGUAGES

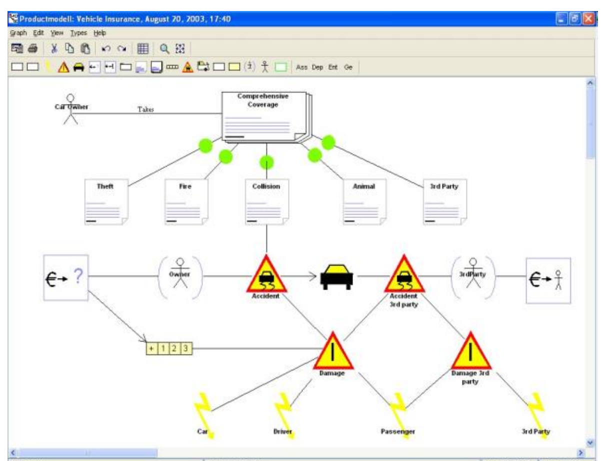
Models and modelling pervasive in software engineering

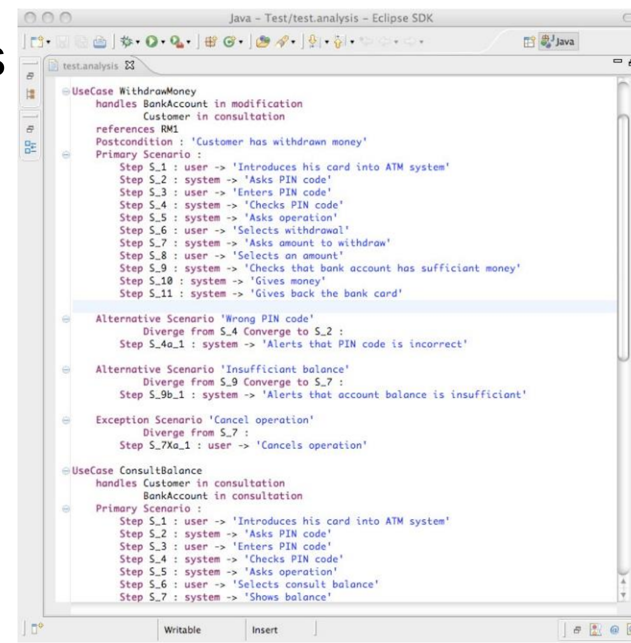
- Model-driven engineering
- Low-code development

Domain-specific modelling languages

- Targeting a specific domain and set of users
- Abstract, concrete syntax and semantics







DOES *ONE* LANGUAGE FIT *ALL*?

Users with different *backgrounds*

- Education, low-code development

Different *modelling needs* and *expressive power*

- Variants of Petri nets (w/o inhibitor, read arcs, weights, colours, etc)

Different *devices* and *interaction modalities*

- Mobile vs desktop vs digital whiteboard
- Graphical vs textual vs conversational

Different *domains*

- Generic language, specialized for different domains: educational process modelling, software process modelling

EXAMPLE
LEARNING A FOREIGN LANGUAGE

How do we learn a foreign spoken language?

- Slowly!
- Our sentences will be wrong: syntactic errors, bad pronunciation, lack of vocabulary

- The recipient (e.g., a teacher) adapts her "parser", pronunciation, speed and vocabulary to the level of knowledge of the student

The learning process occurs within a simpler language variant

As learning progresses, the variant used becomes more complex and complete

EXAMPLE
LEARNING A MODELLING LANGUAGE

How do we learn a modelling language (e.g. UML)?

- We present the different primitives of the language to the student
- Examples, exercises

- The student uses an editor (e.g., Eclipse based) that:

  Expects the user to be **fully proficient** with the syntax and semantics of the language

  Presents the user the **full language** “vocabulary”, even if the user has no knowledge of it

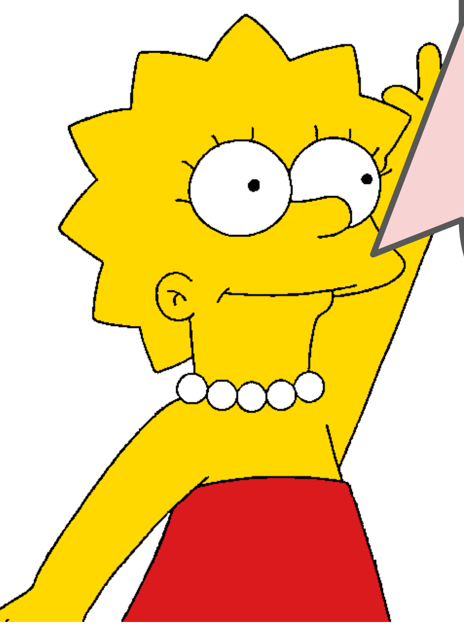
  Is **unable to understand** “almost correct” models, or may not even allow to construct or persist them

  Makes the “conversation” occur with a full-fledged language version, and **expects the “conversation” to be perfect**

EXAMPLE
LEARNING A MODELLING LANGUAGE

What can we do about this?

FROM LANGUAGES TO LANGUAGE FAMILIES



Let's define a language family to guide the learning process!

NAÏVE APPROACHES TO DEFINE LANGUAGE FAMILIES

Case-by-case approach (*clone-and-own*)

- Explicitly define each language of the family
- Exponential number of languages w.r.t. features
    - Class diagram with 4 features (inheritance, composition, aggregation, interfaces) leads to 2^4=16 languages
- No reuse, hard to maintain

Big language with all features

- Language too complex for the user
    - Perhaps that's precisely what we try to avoid with the family!
- What about alternative features?

THREE APPROACHES TO LANGUAGE FAMILIES

Annotative (superimposition, negative variability)

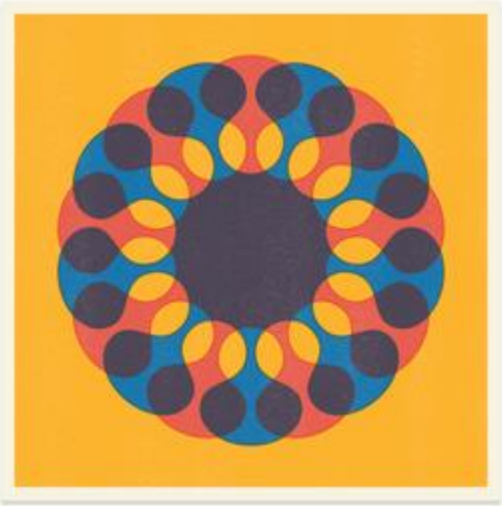
- Design: Overlap every language variant
- Configuration: Feature model to select the elements that are present in the language

Compositional (language modules)

- Design: Modules with language features
- Configuration: Select the modules that are present in the language

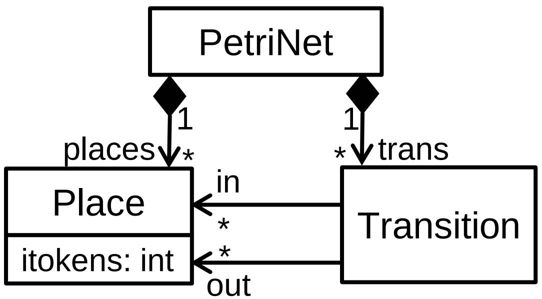
Multi-level modelling (specialise the language via instantiation)

- Design: Generic language with common primitives for all domains
- Configuration: Specialize the generic primitives for a domain

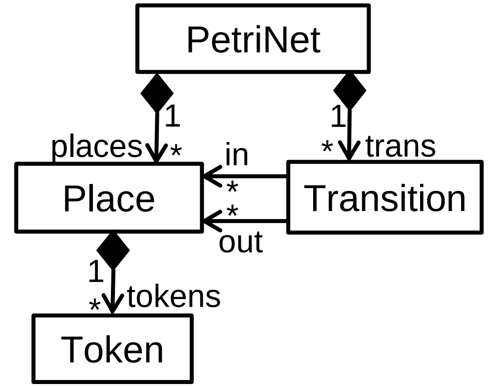


SUPERIMPOSITION

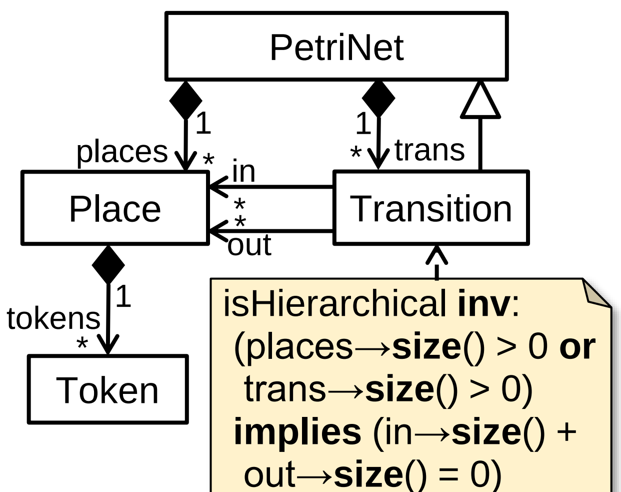
A FAMILY OF PETRI NET LANGUAGES



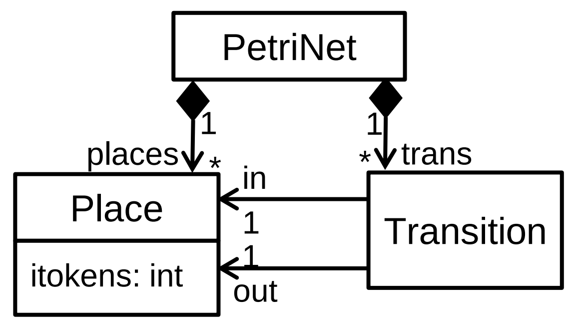
tokens as integers



tokens as objects

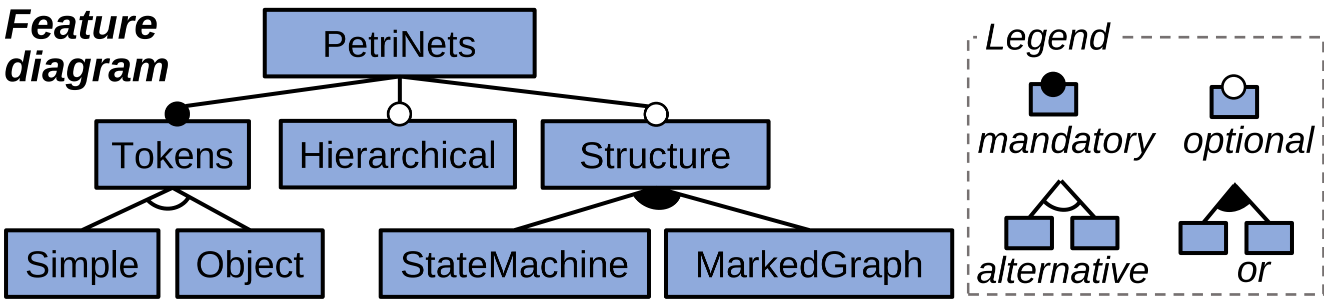


hierarchical nets



state machine nets

FEATURE MODEL



$$
\begin{aligned}
FM = \langle\ \{ & \text{PetriNets, Tokens, Simple, Object, Hierarchical,} \\
& \text{Structure, StateMachine, MarkedGraph} \}, \\
& \text{PetriNets} \wedge \text{Tokens} \wedge ((\text{Simple} \wedge \neg\text{Object}) \vee (\neg\text{Simple} \wedge \text{Object})) \wedge \\
& (\text{Structure} \Leftrightarrow (\text{StateMachine} \vee \text{MarkedGraph}))\ \rangle
\end{aligned}
$$

Model of the variability of a system

- Features + allowed feature combinations

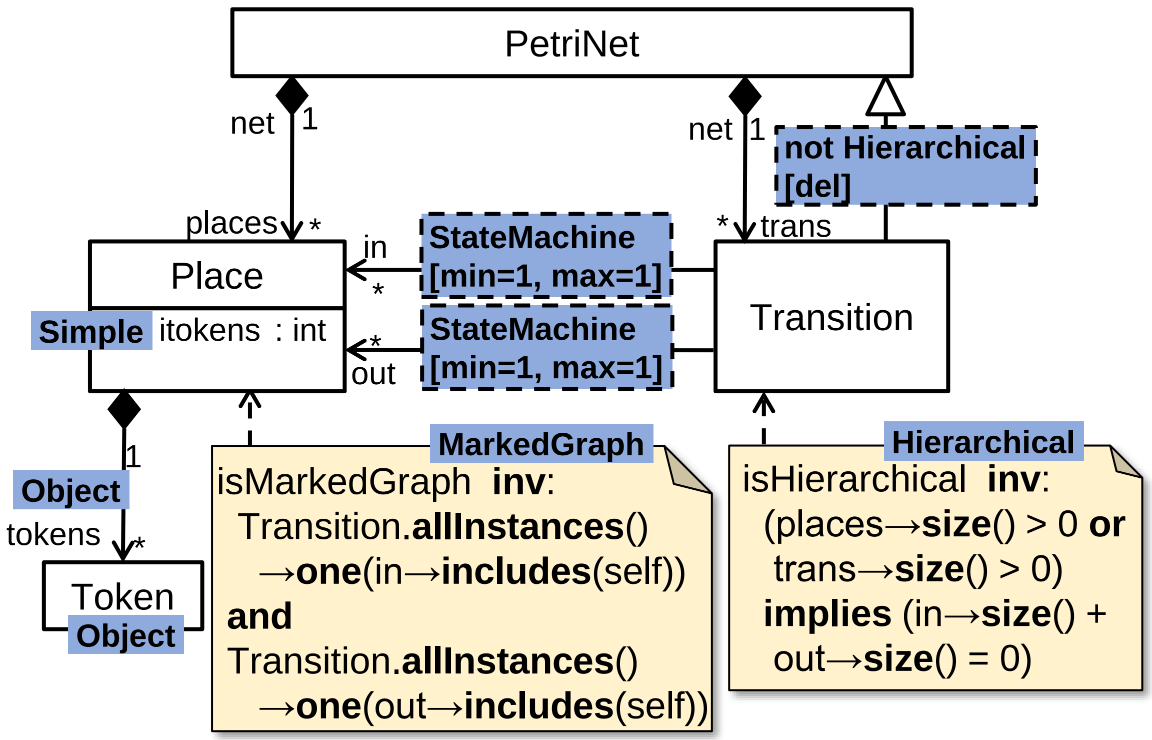
Configuration

- Set of features satisfying the constraints imposed by the feature model

Examples

- {PetriNets, Tokens, Simple}
- {PetriNets, Tokens, Object, Structure, MarkedGraph}

150% META-MODEL

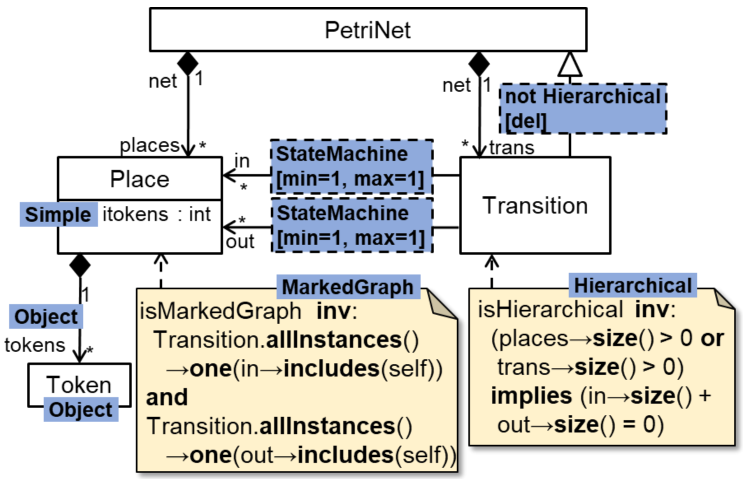


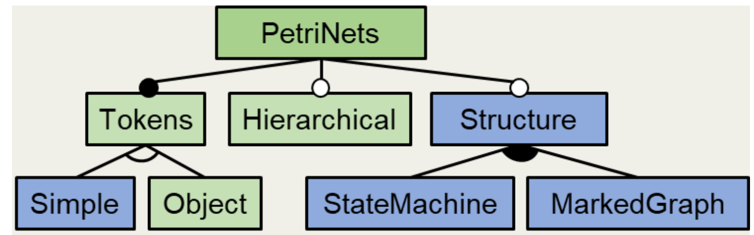
Overlap the meta-models of all language variants

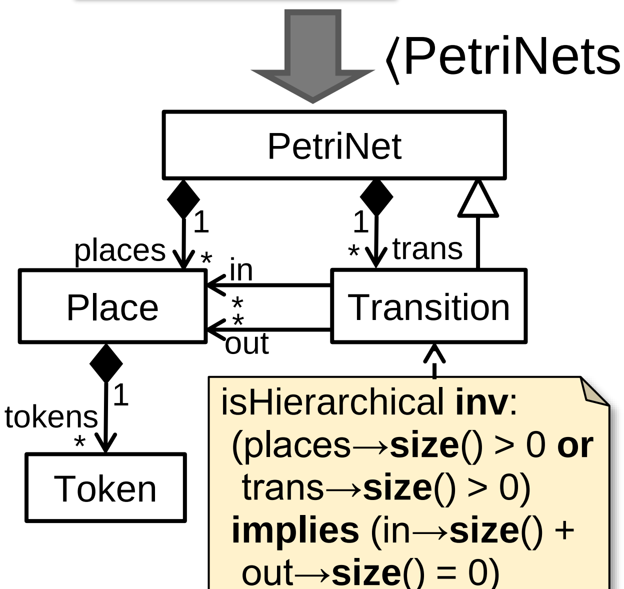
Presence conditions: formulae over the features

Negative variability: remove what is not selected by a configuration

CONFIGURATION AND DERIVATION







⟨PetriNets, Tokens, Object, Hierarchical⟩

(SOME) CHALLENGES

Identify non-consistent combinations of features

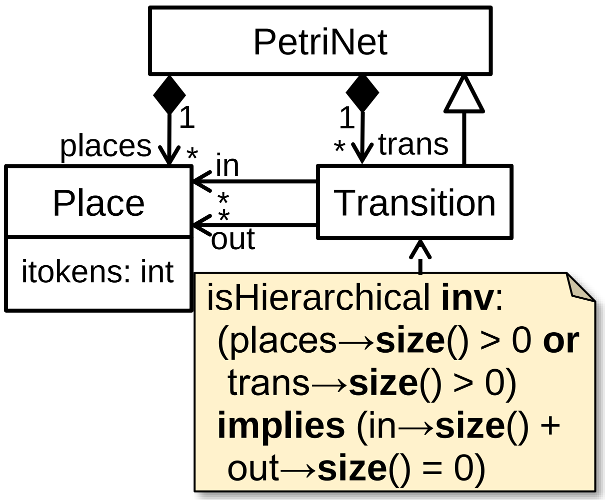
- Two variants conflict if their integrity constraints clash
- “Hierarchy and StateMachine cannot be meaningfully combined”

Instantiability properties for each language

- “no Petri net in any variant can have a negative number of tokens”
- Checking this property on a per-language basis would be ineffective!

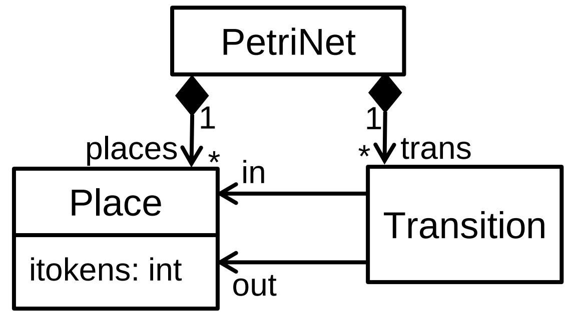
Guerra, de Lara, Chechik, Salay: Property Satisfiability Analysis for Product Lines of Modelling Languages. IEEE Trans. Software Eng. 48(2): 397-416 (2022)

VACUOUS FEATURE COMBINATIONS

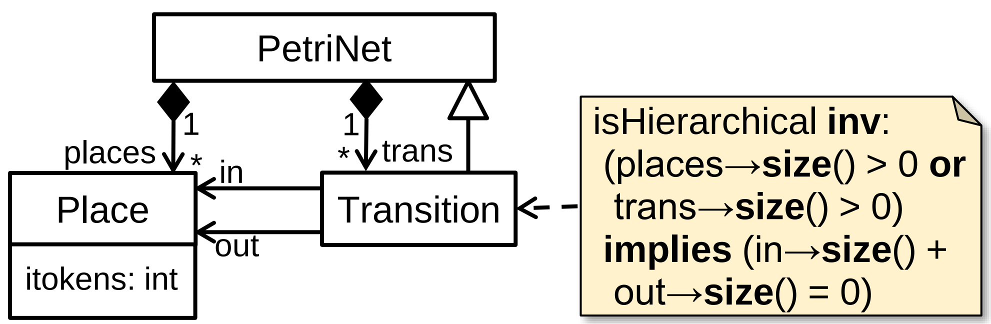


hierarchical nets

+



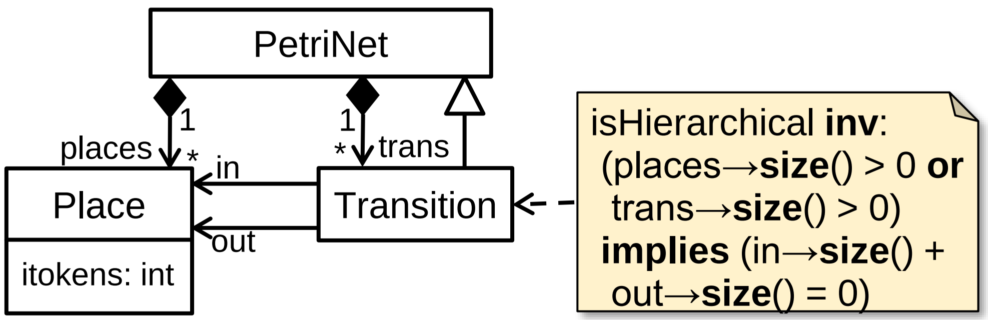
state machine nets

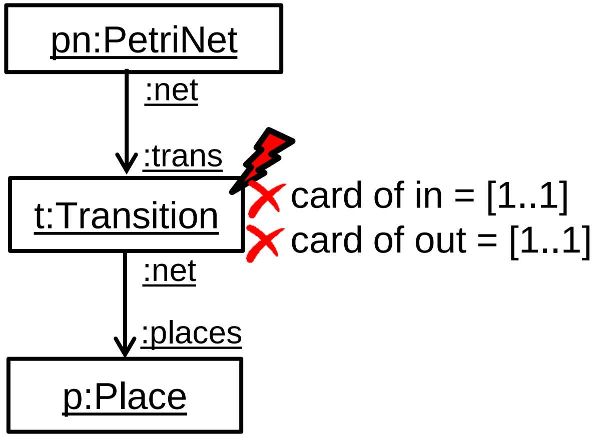


hierarchical state machine nets

(the meta-model is fine, and it is instantiable, BUT…)

VACUOUS FEATURE COMBINATIONS



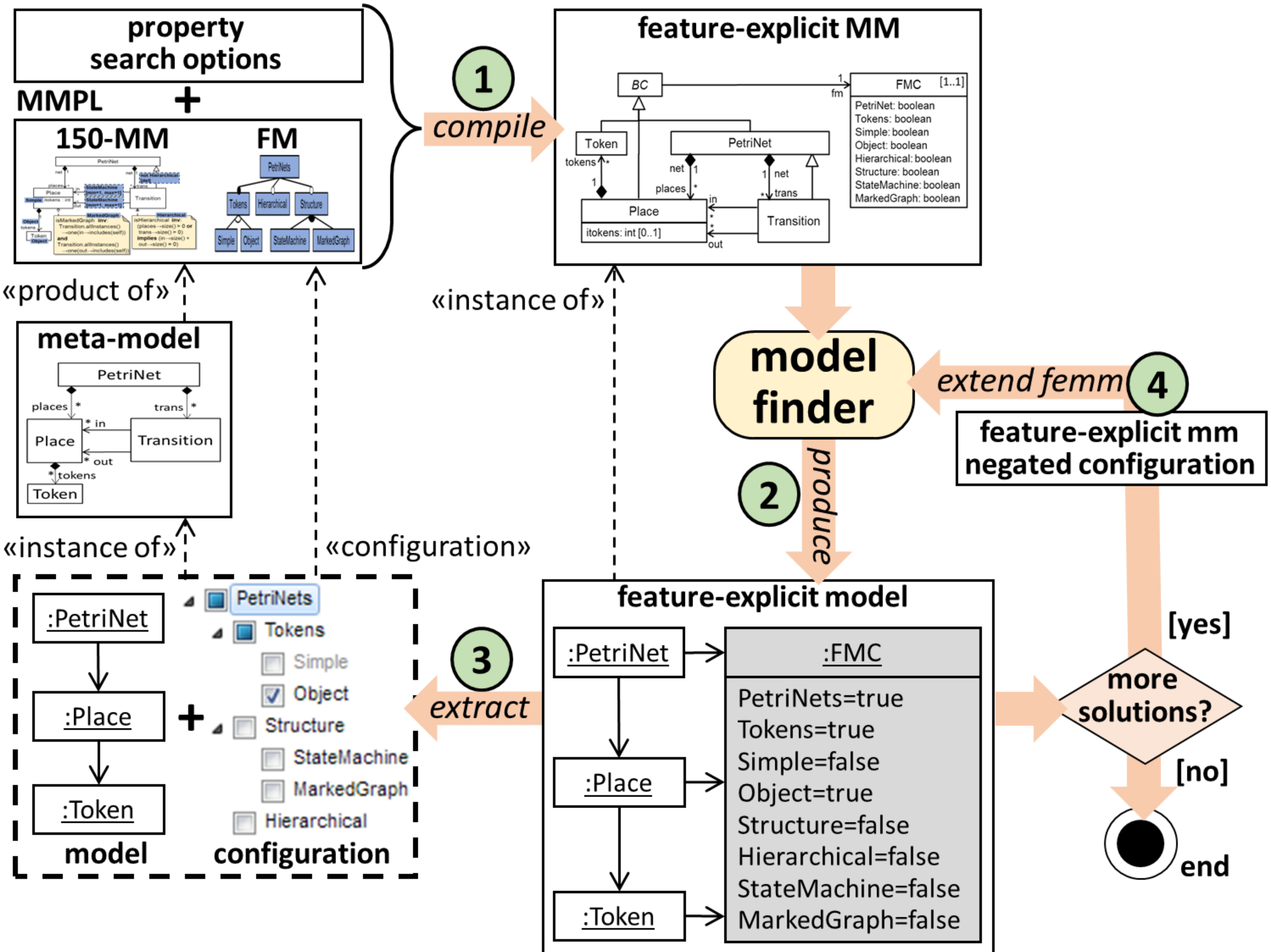


Hierarchical state machine nets

We cannot exercise the features introduced by "Hierarchical"

- `Transition.places` and `Transition.trans` need to be empty
- This means, we cannot really have hierarchical transitions
- Hierachical and StateMachine cannot be meaningfuly combined

ANALYSIS VIA MODEL FINDING

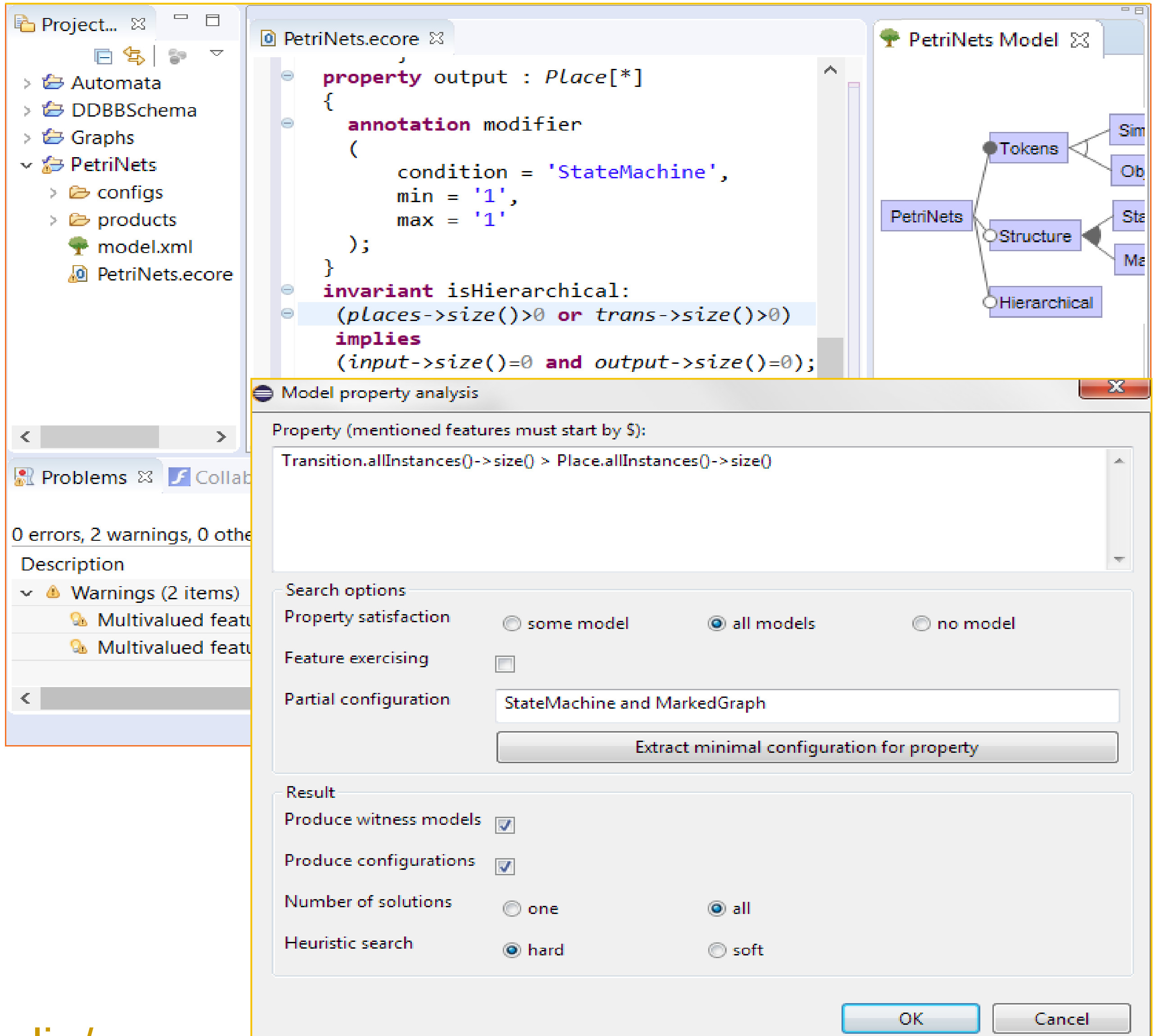


TOOL SUPPORT: MERLIN

Eclipse plugin,
FeatureIDE

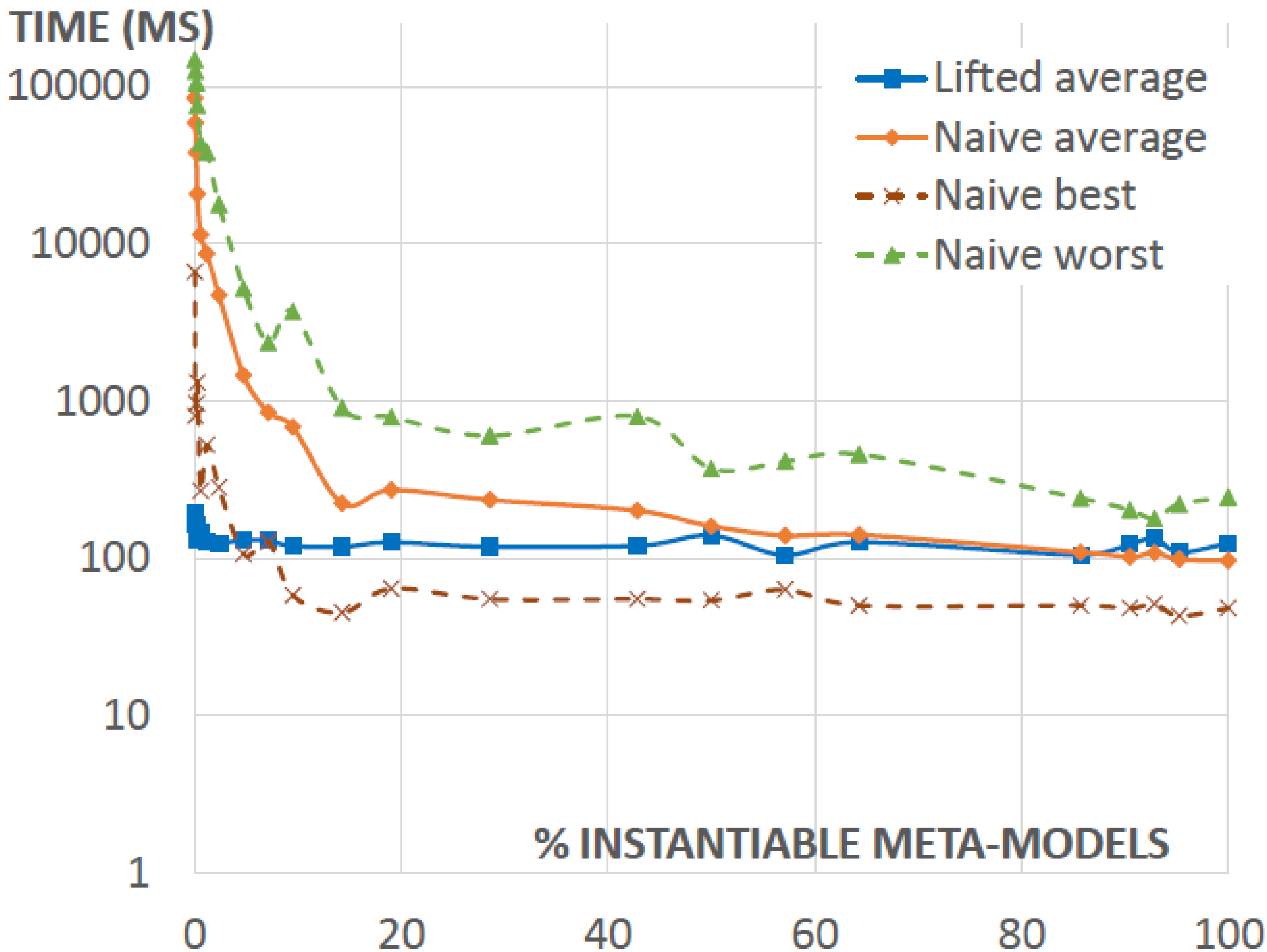
Product lines
of transformations

More advanced
analysis via
partial
configurations



http://miso.es/tools/merlin/

SOME EXPERIMENTS



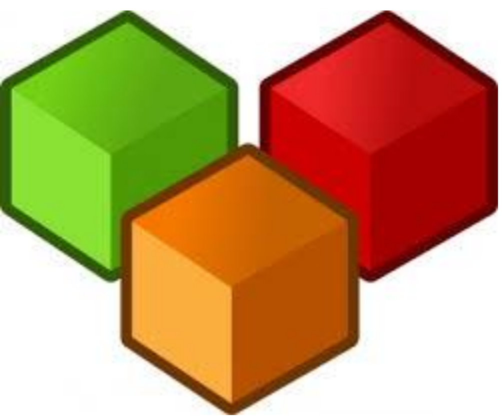
NICE BUT…

Doesn't the 150% meta-model become a "big ugly monster"?

- We can use slicing to visualize parts of it

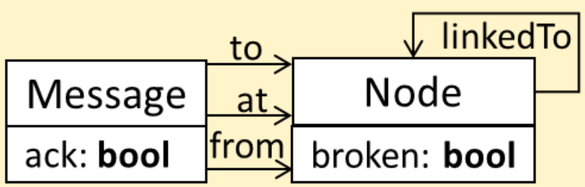
Is the approach really extensible?

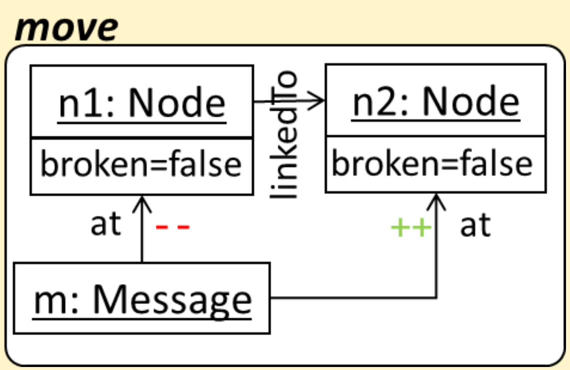
- We still need to dive into the 150% meta-model to add a new feature, and also change the feature model



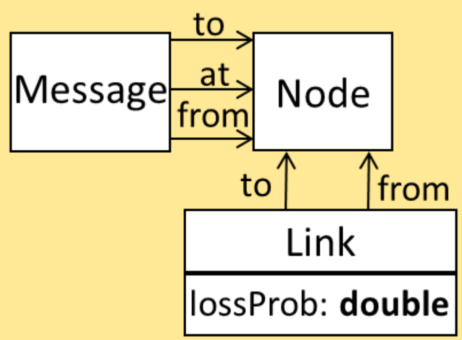
LANGUAGE MODULES

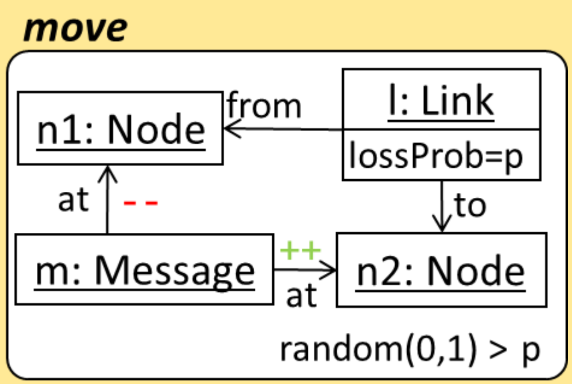
EXAMPLE
A DSL FAMILY FOR NETWORKING



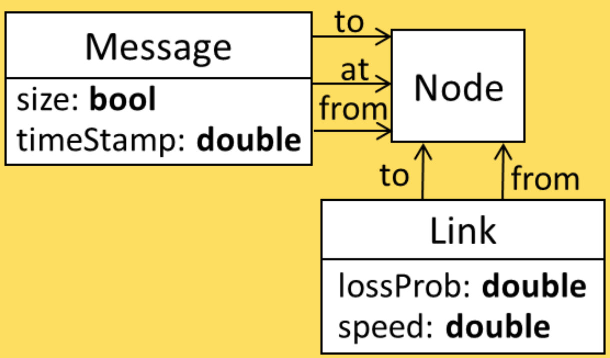


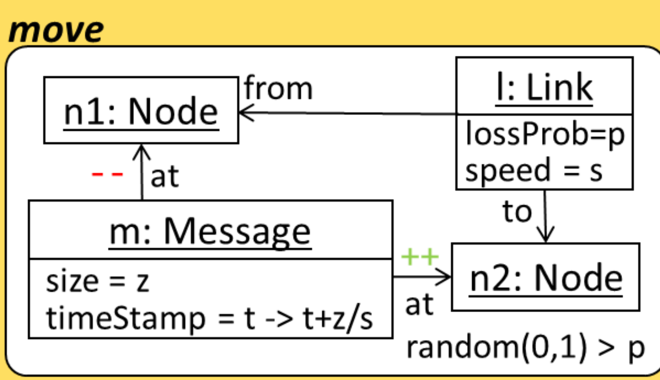
Simple link with node failures and acks





Rich links with communication failures





Rich links with communication failures and time

THE APPROACH

Language product line

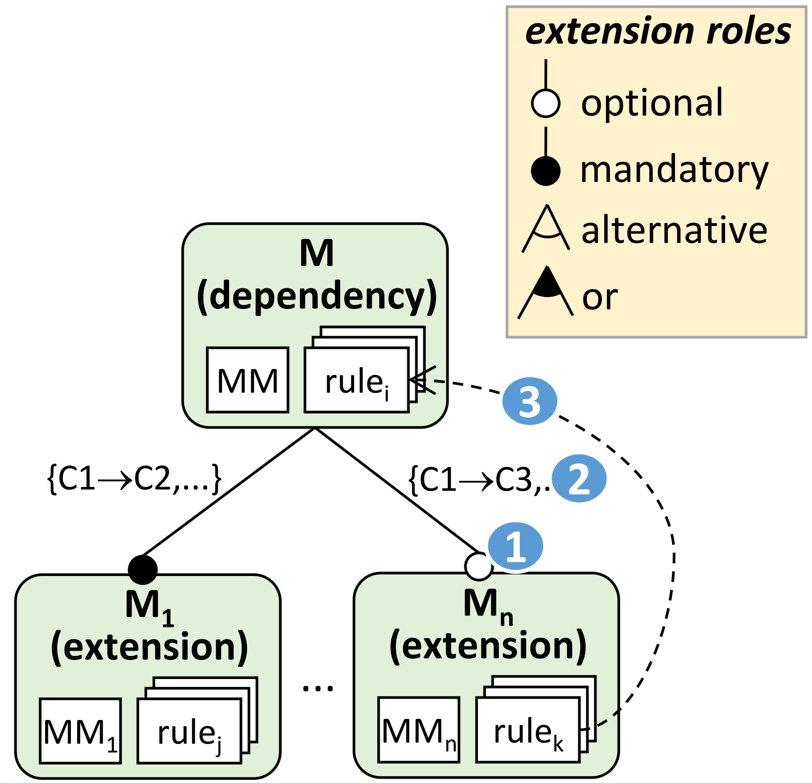
Language modules

- Meta-model
- Transformation rules

Module dependencies

Module extensions

- Extension roles
- As in feature models



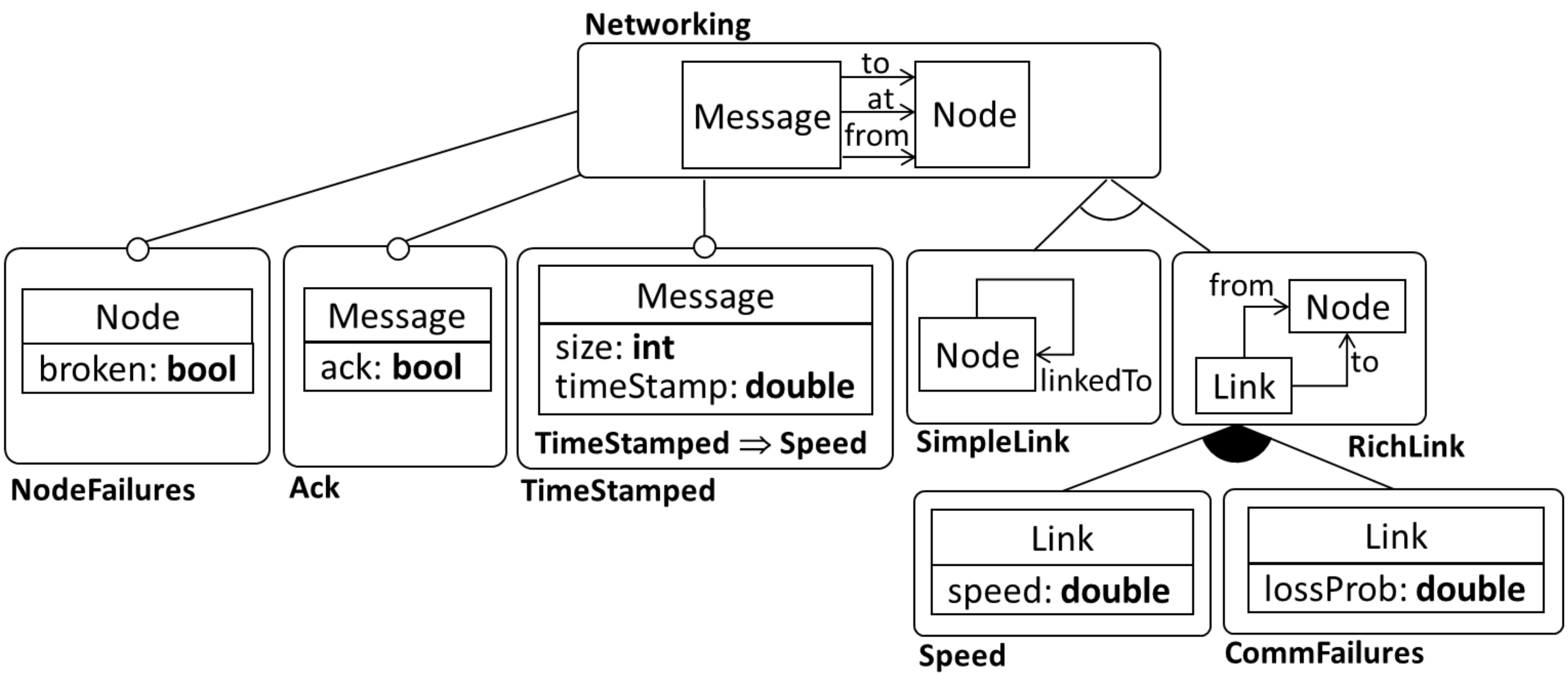
1 role of M_n in dependency

2 inclusion of MM_n in MM

3 rule extension

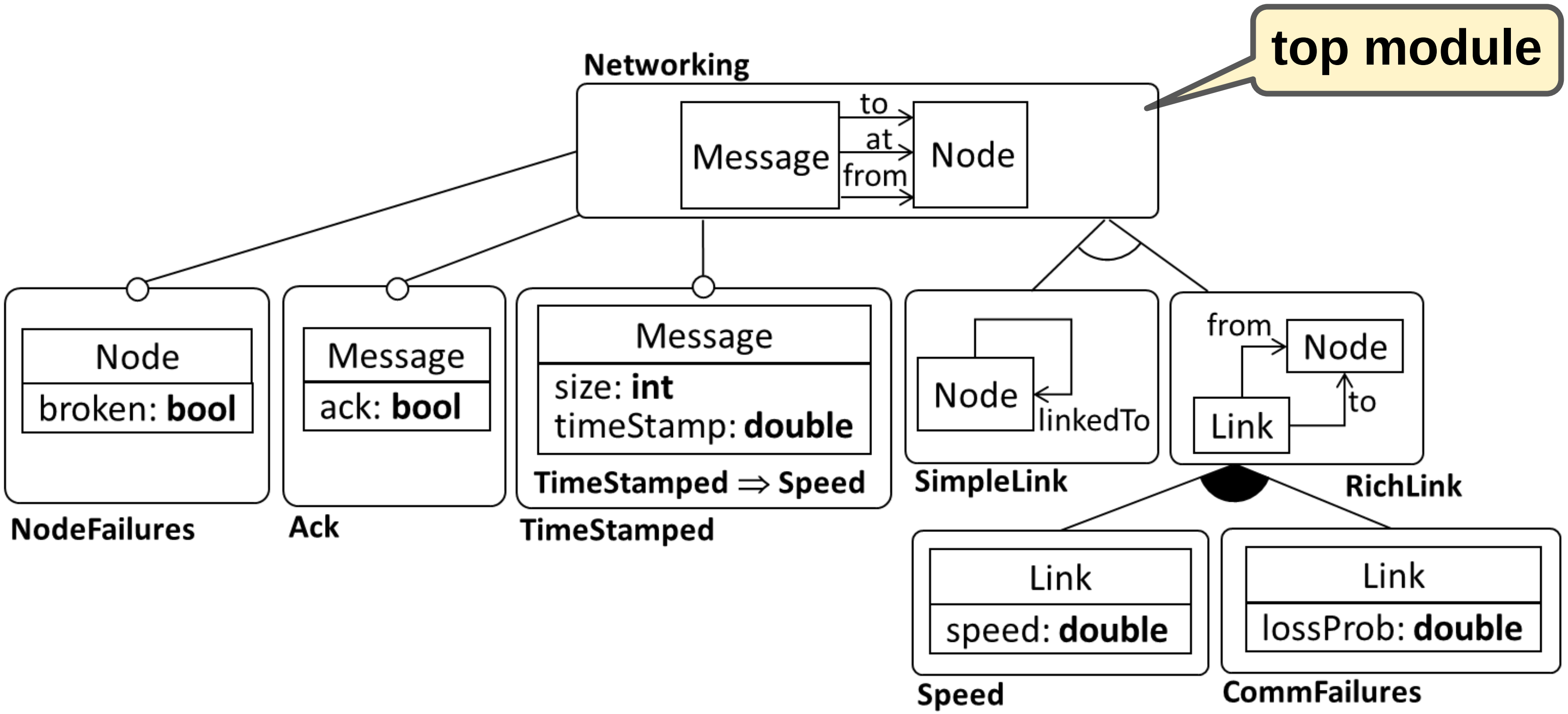
de Lara, Guerra, Bottoni. “Modular Language Product Lines. A Graph Transformation Approach”. Proc. MoDELS’2022 (**on Wednesday!**)

LANGUAGE PRODUCT LINE *EXAMPLE*



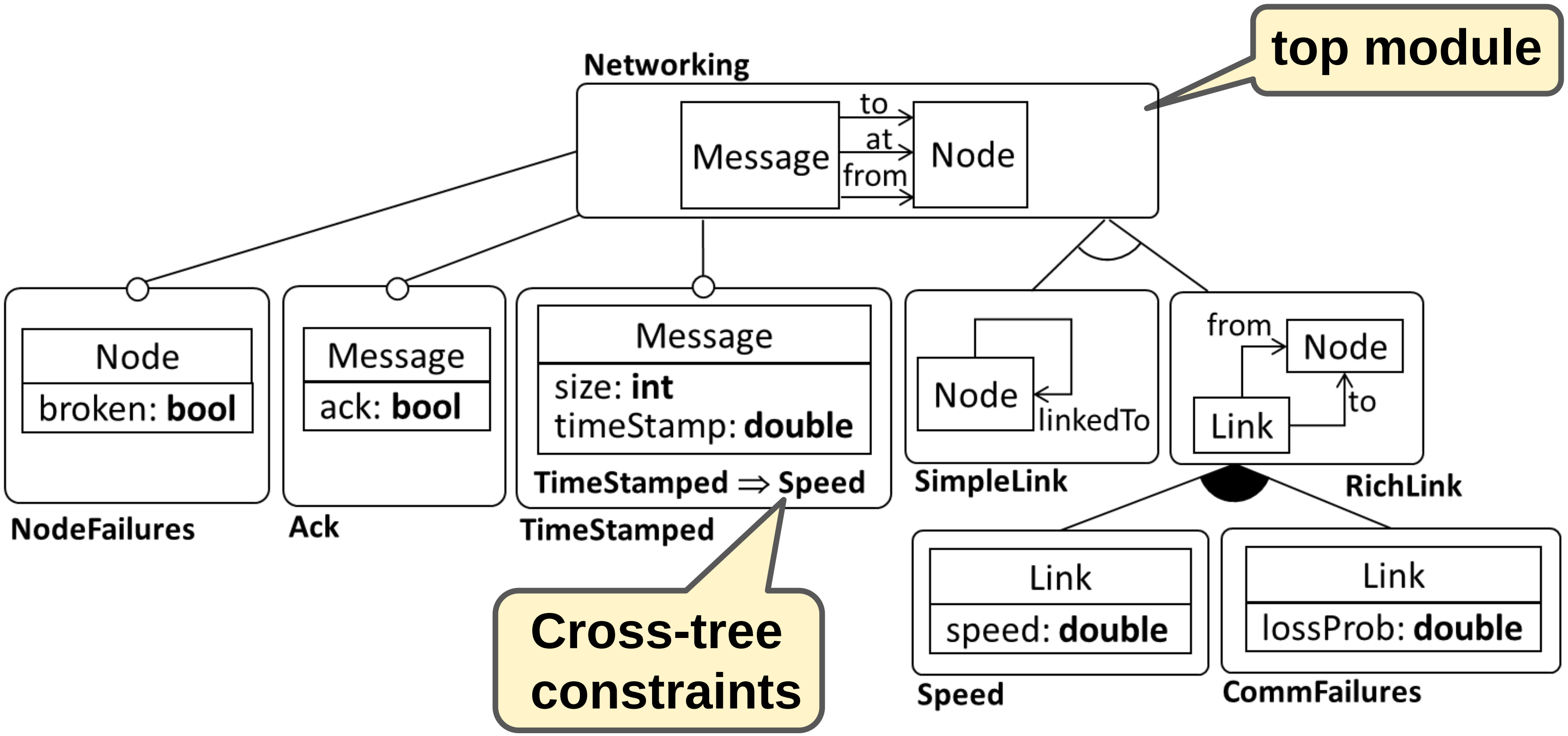
Meta-model elements are identified by name

LANGUAGE PRODUCT LINE *EXAMPLE*



Meta-model elements are identified by name

LANGUAGE PRODUCT LINE
EXAMPLE

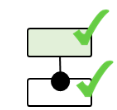


Meta-model elements are identified by name

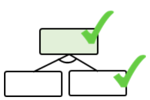
USING THE PRODUCT LINE: CONFIGURATIONS

A set of modules such that

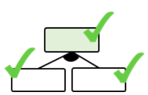
- All top modules are selected
- If a module is **selected**, then the configuration needs selecting:



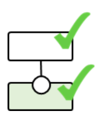
1. **all** mandatory extension modules



2. **exactly one** alternative extension modules



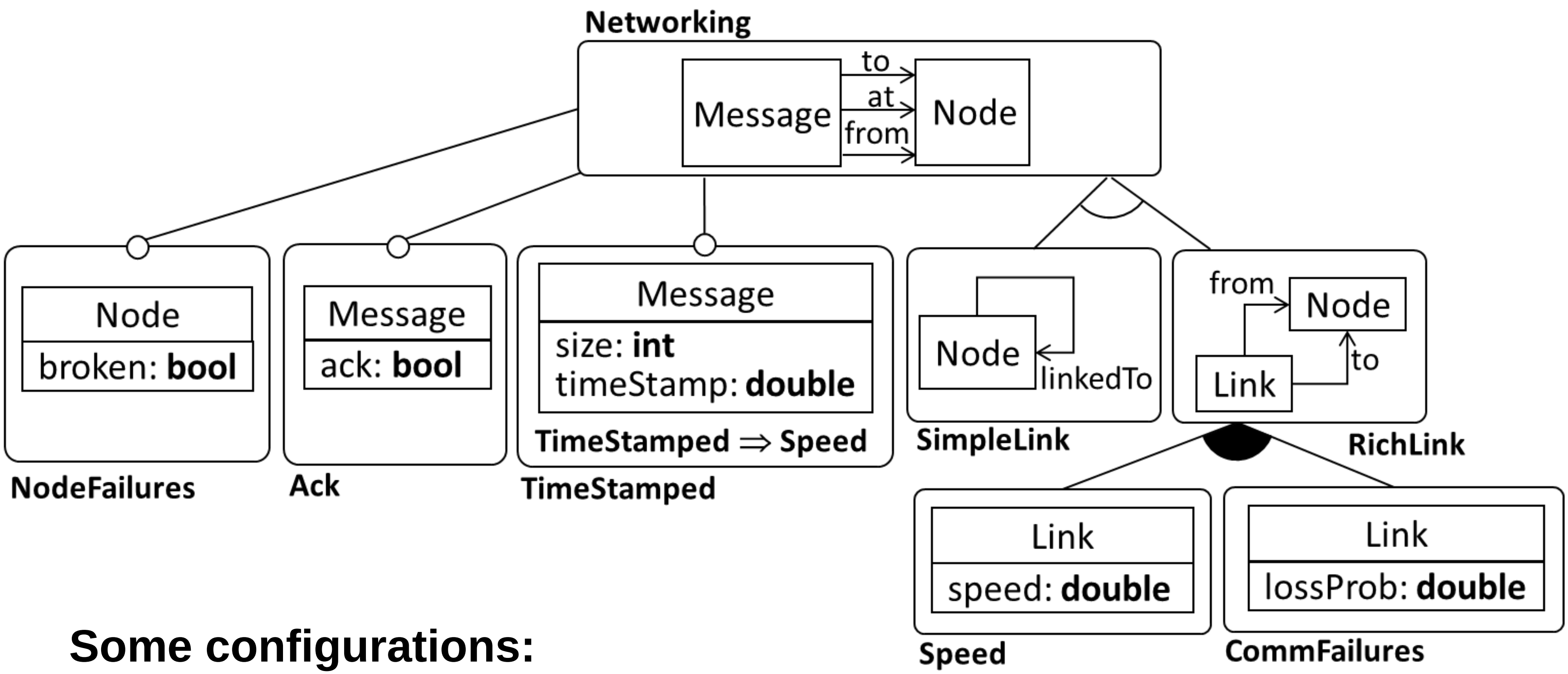
3. **at least one** *OR* extension module



4. **its dependency**

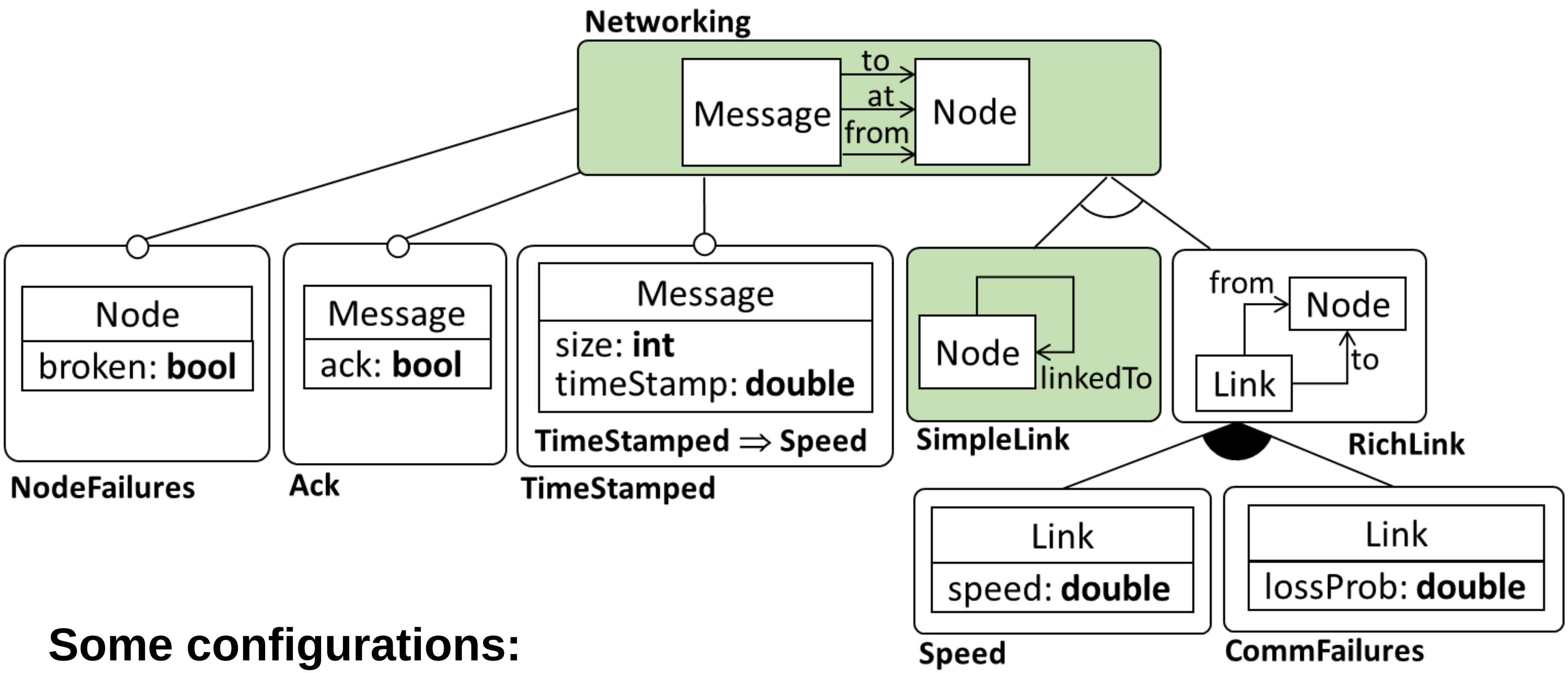
- The cross-tree constraints evaluate to true

CONFIGURATIONS *EXAMPLE*



Some configurations:

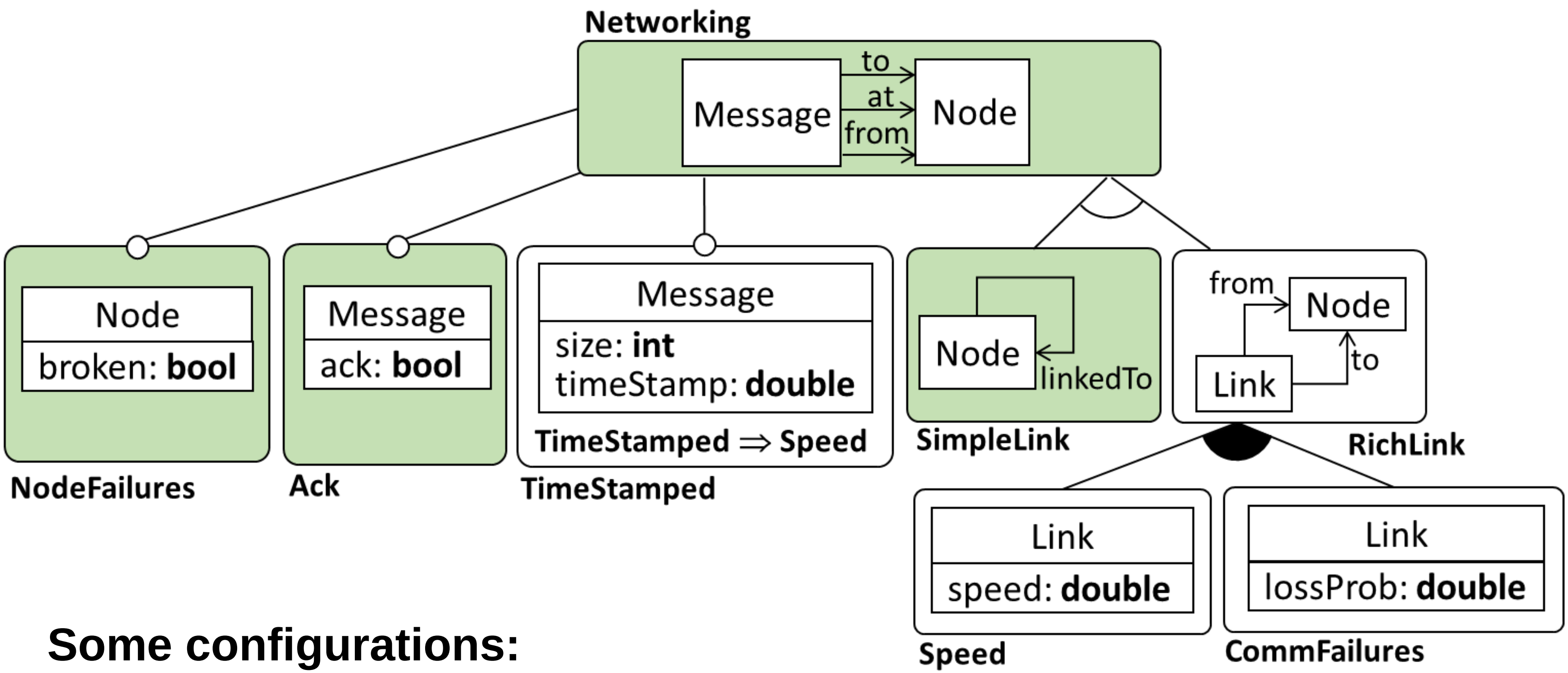
CONFIGURATIONS *EXAMPLE*



Some configurations:

- {Networking, SimpleLink}

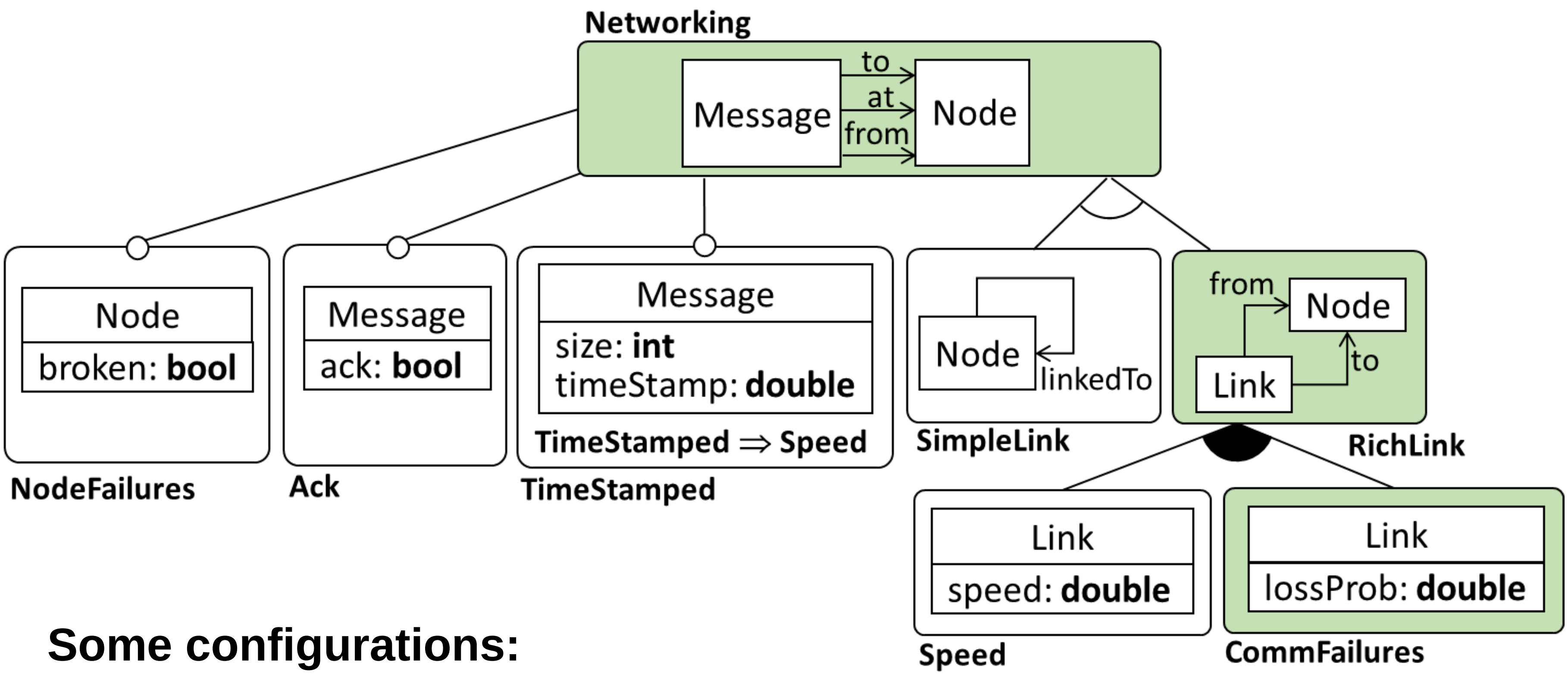
CONFIGURATIONS *EXAMPLE*



Some configurations:

- {Networking, SimpleLink}
- {Networking, SimpleLink, NodeFailures, Ack}

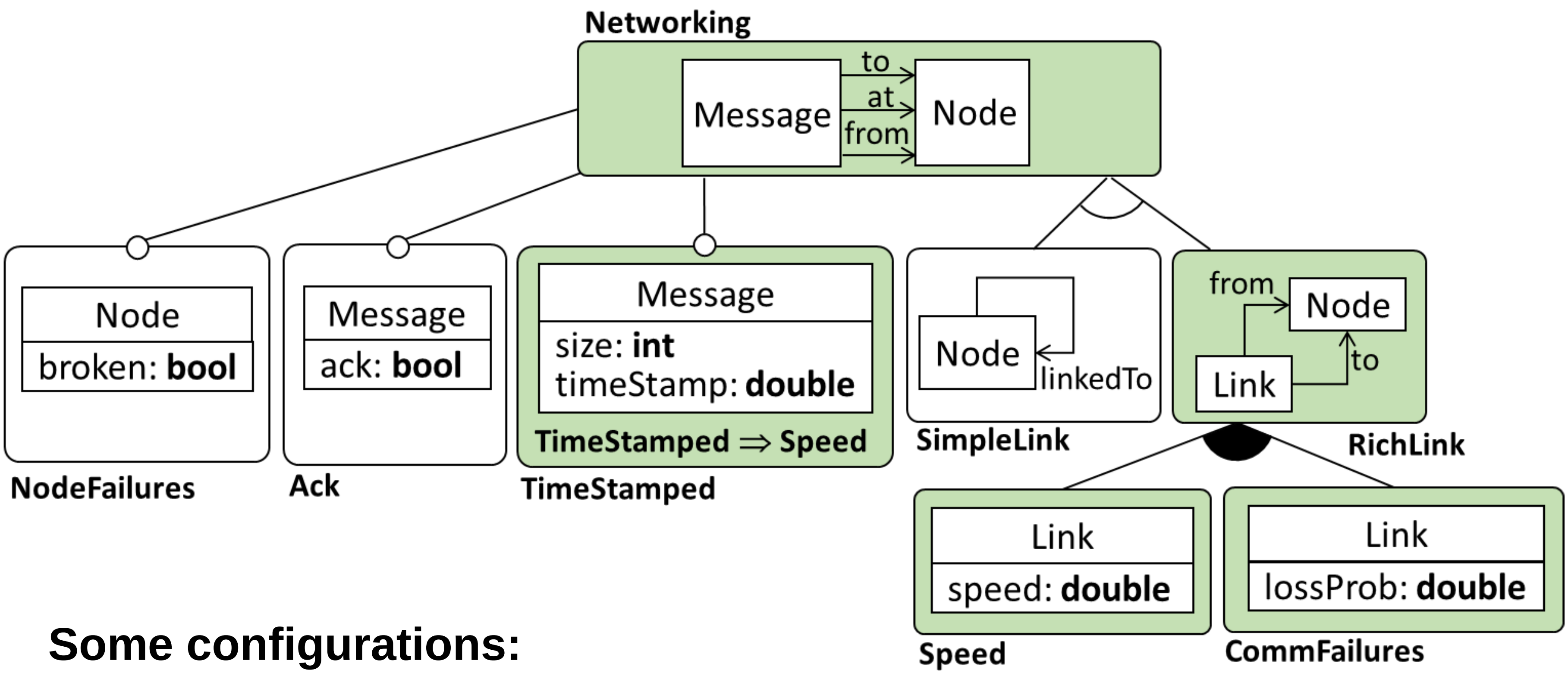
CONFIGURATIONS *EXAMPLE*



Some configurations:

- {Networking, SimpleLink}
- {Networking, SimpleLink, NodeFailures, Ack}
- {Networking, RichLink, CommFailures}

CONFIGURATIONS
EXAMPLE



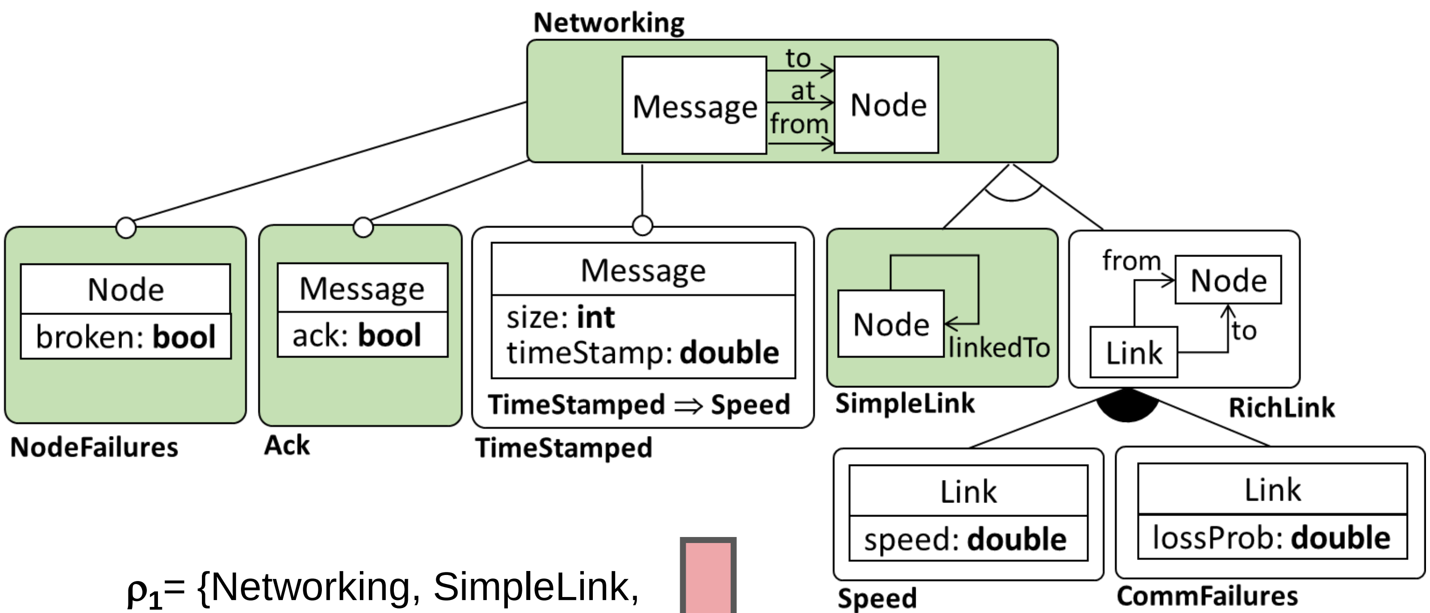
Some configurations:

- {Networking, SimpleLink}
- {Networking, SimpleLink, NodeFailures, Ack}
- {Networking, RichLink, CommFailures}
- {Networking, RichLink, CommFailures, TimeStamped, Speed}

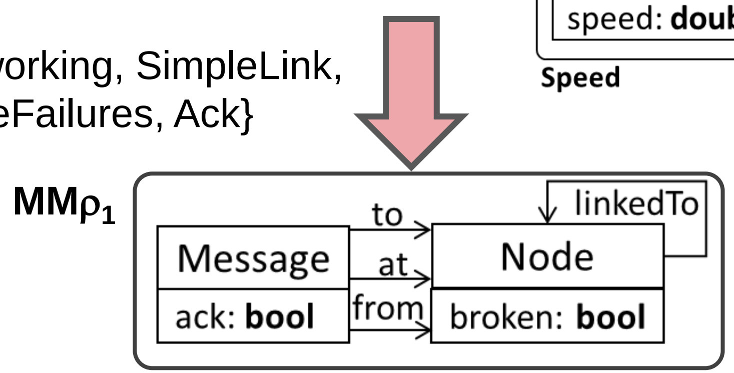
DERIVATION: GETTING THE META-MODEL

Given a configuration

- Merge the meta-model fragments of all modules (co-limit)



ρ_1= {Networking, SimpleLink, NodeFailures, Ack}



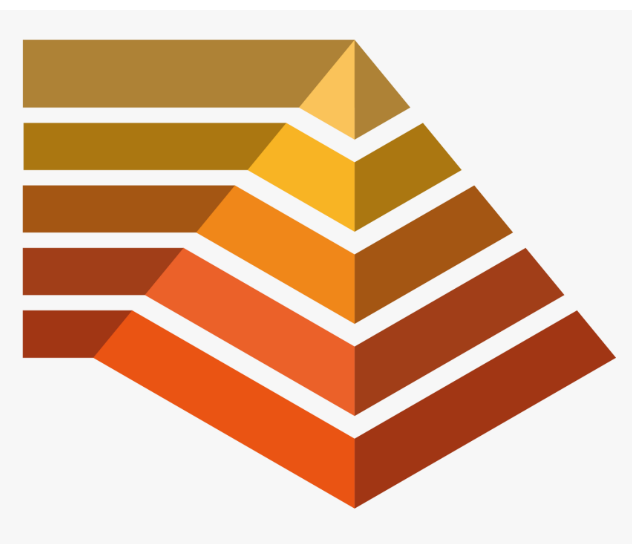
NICE BUT...

Still closed variability

- How do I refine the DSL to my domain?

We need a notion of “open variability”

- Guided refinement
- Still allows defining a coherent language family



MULTI-LEVEL MODELLING

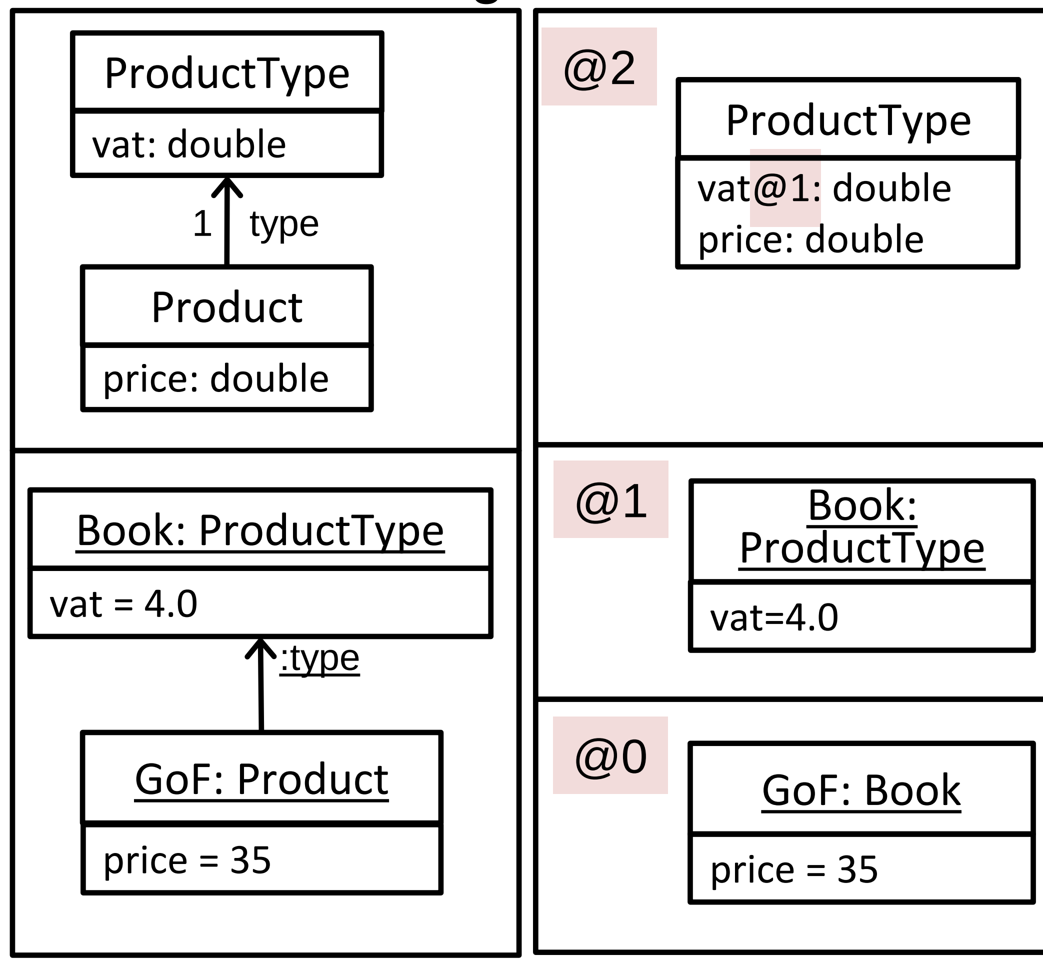
MULTI-LEVEL MODELLING

Use an arbitrary number of meta-levels

Model elements have both a type and an instance facet

Potency to control characteristics of instances beyond the next meta-level below

Standard modelling **Multi-level**



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

CLABJECT = CLASS + OBJECT

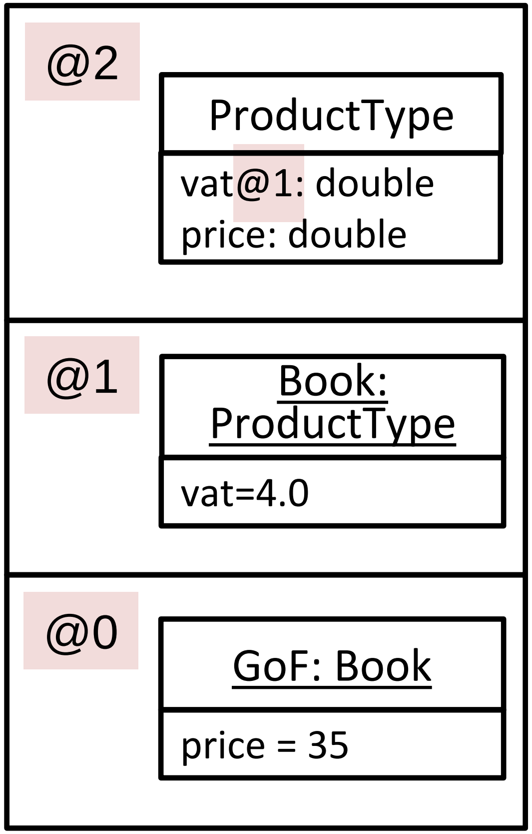
Elements have a combined type and instance facet

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

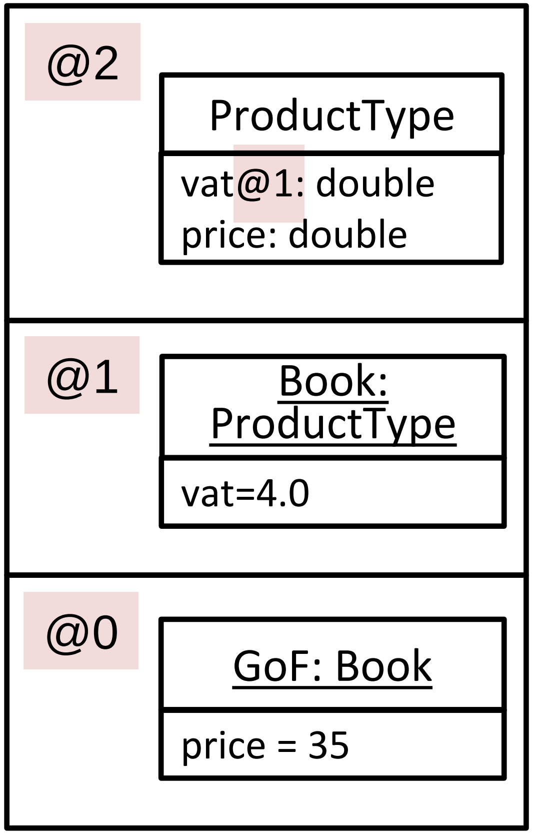
Used to characterize instances beyond the next meta-level

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

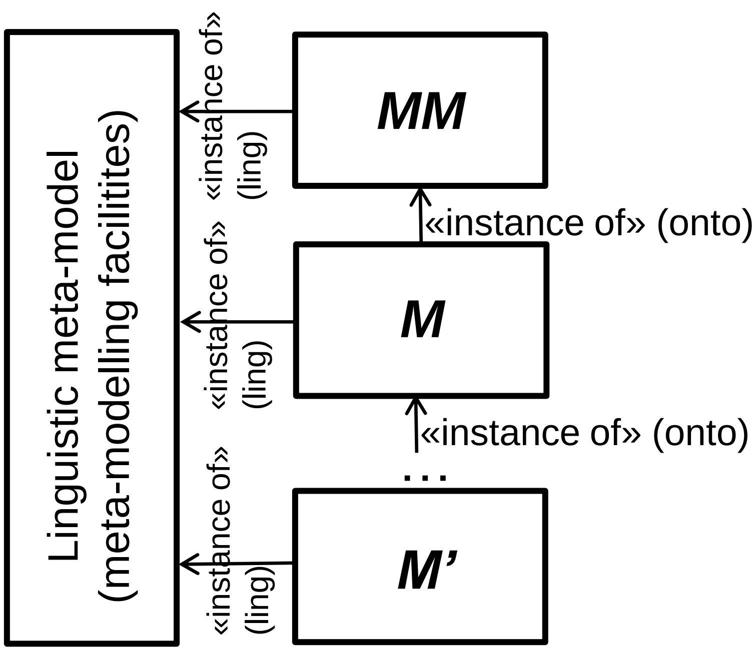
By default elements take the potency of their containers



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

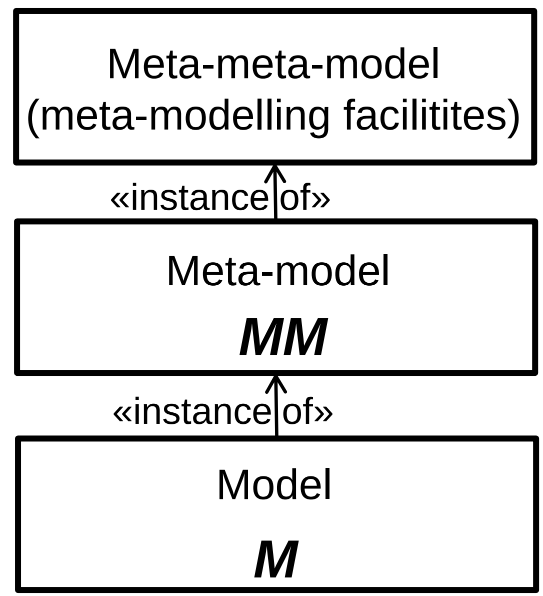
ORTHOGONAL CLASSIFICATION (OCA)

Multi-level



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

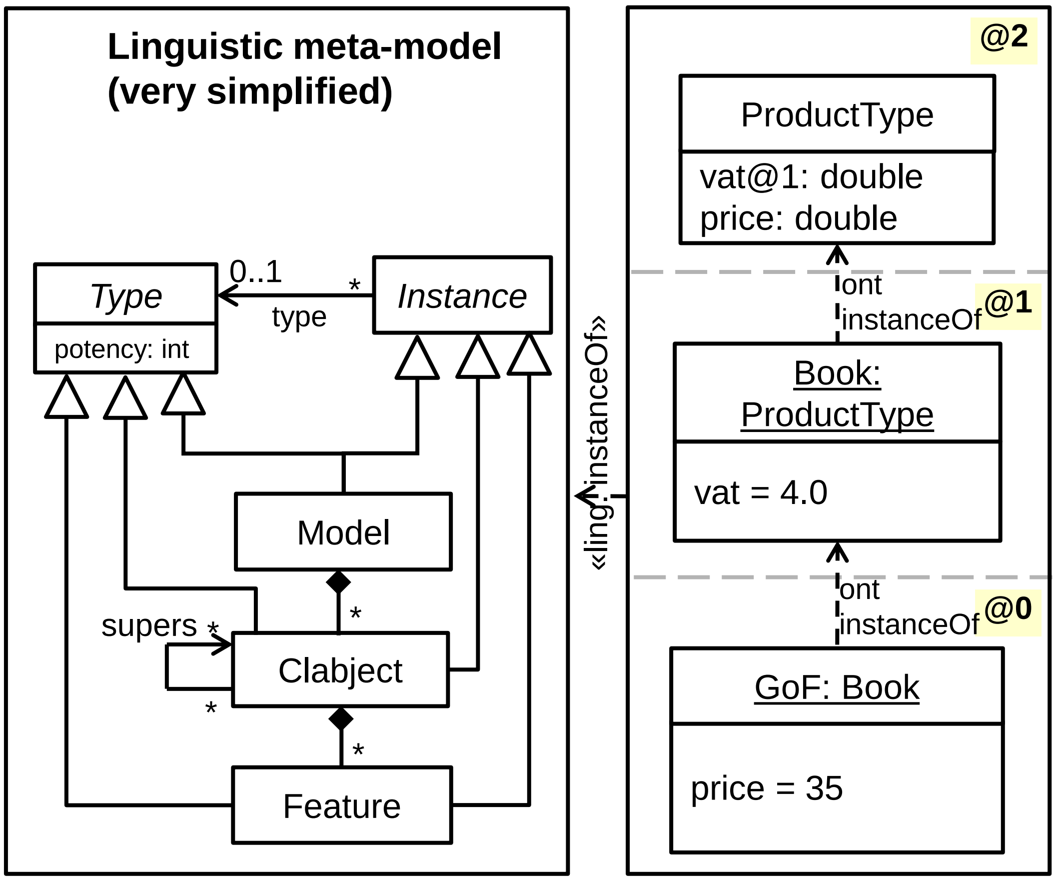
Two-Level (eg., EMF)



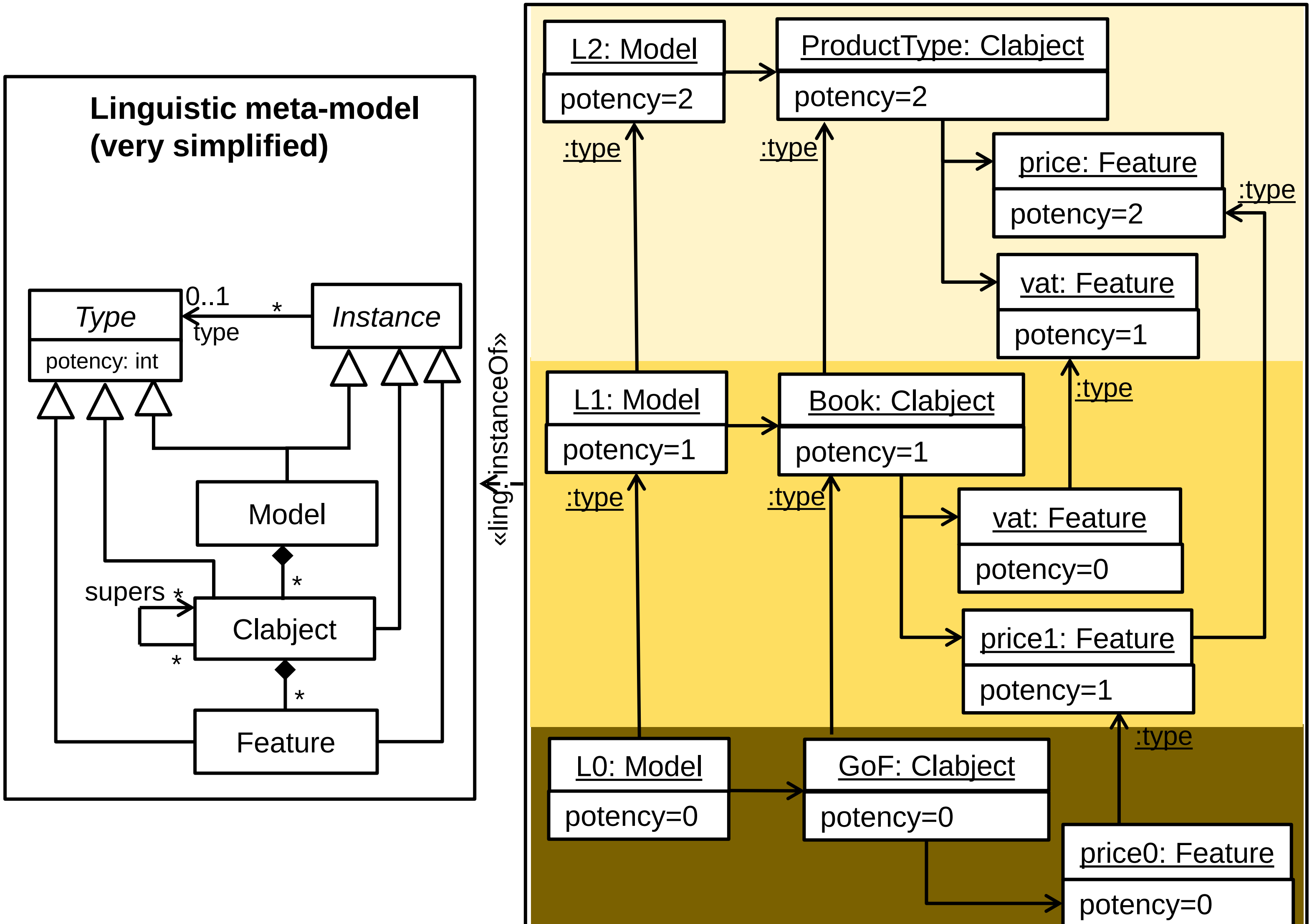
- Types and meta-modelling facilities only at the meta-model level

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

ORTHOGONAL CLASSIFICATION (OCA)



LINGUISTIC VIEW



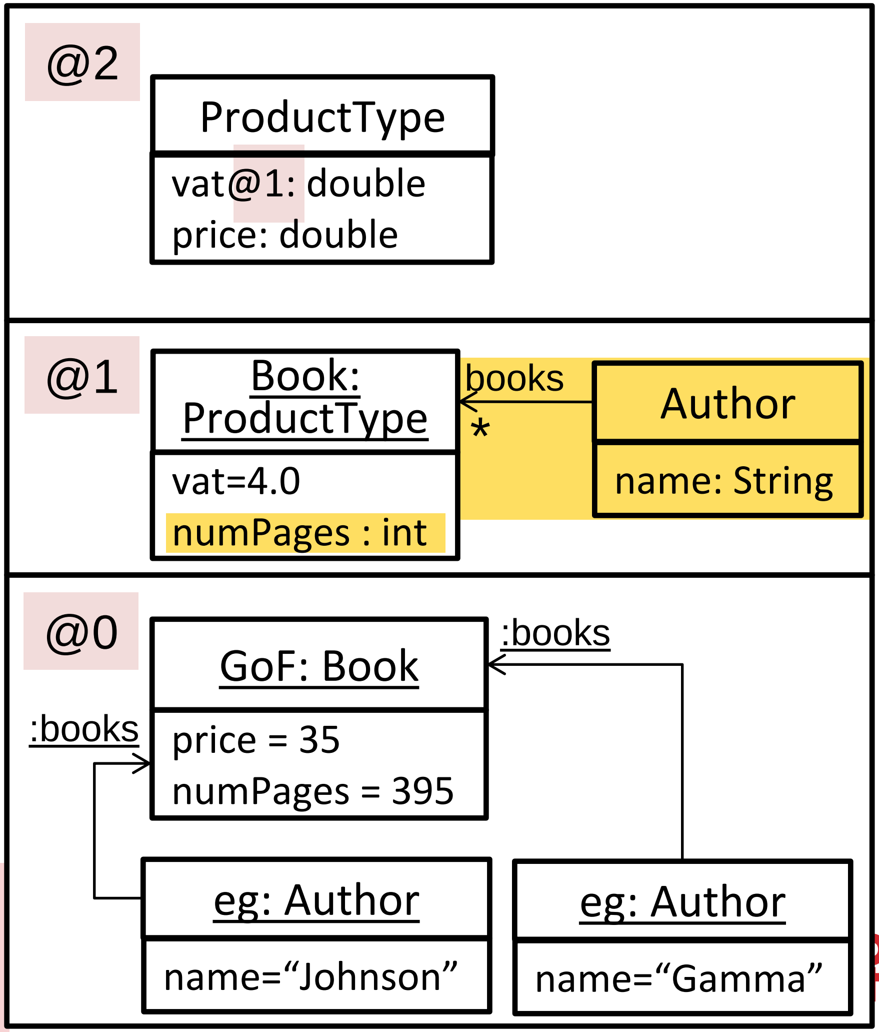
LINGUISTIC EXTENSIONS

Elements with no ontological type

- Ontological typing is optional
- Linguistic typing is mandatory

New clabjects or features

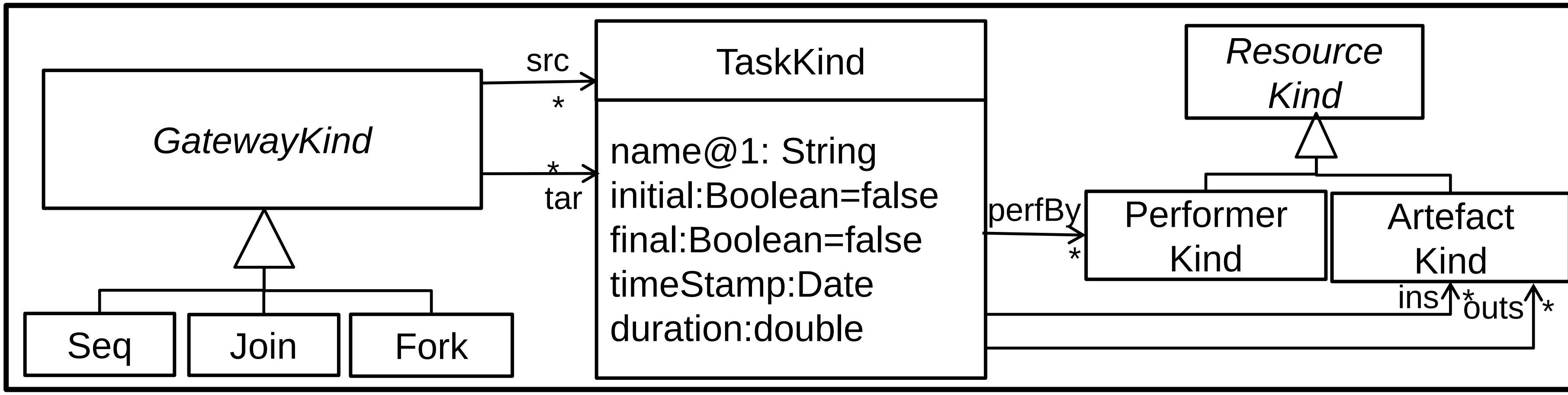
Not everything can be anticipated at the top-most level



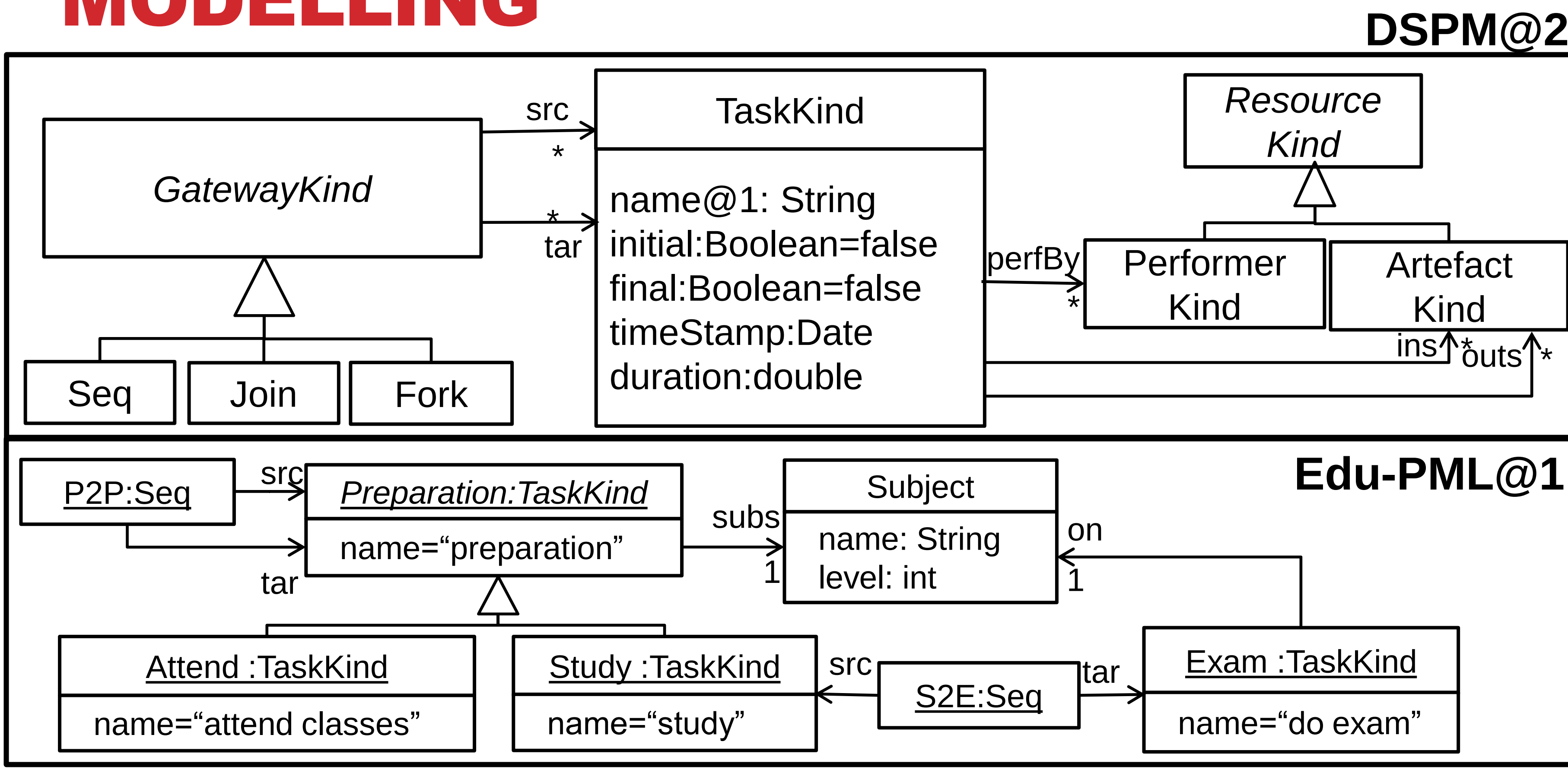
de Lara, Guerra. 2010. Deep meta-modelling with metaDepth. In TOOLS'10 (LNCS), Vol. 6141. Springer, 1–20.

DOMAIN SPECIFIC PROCESS MODELLING

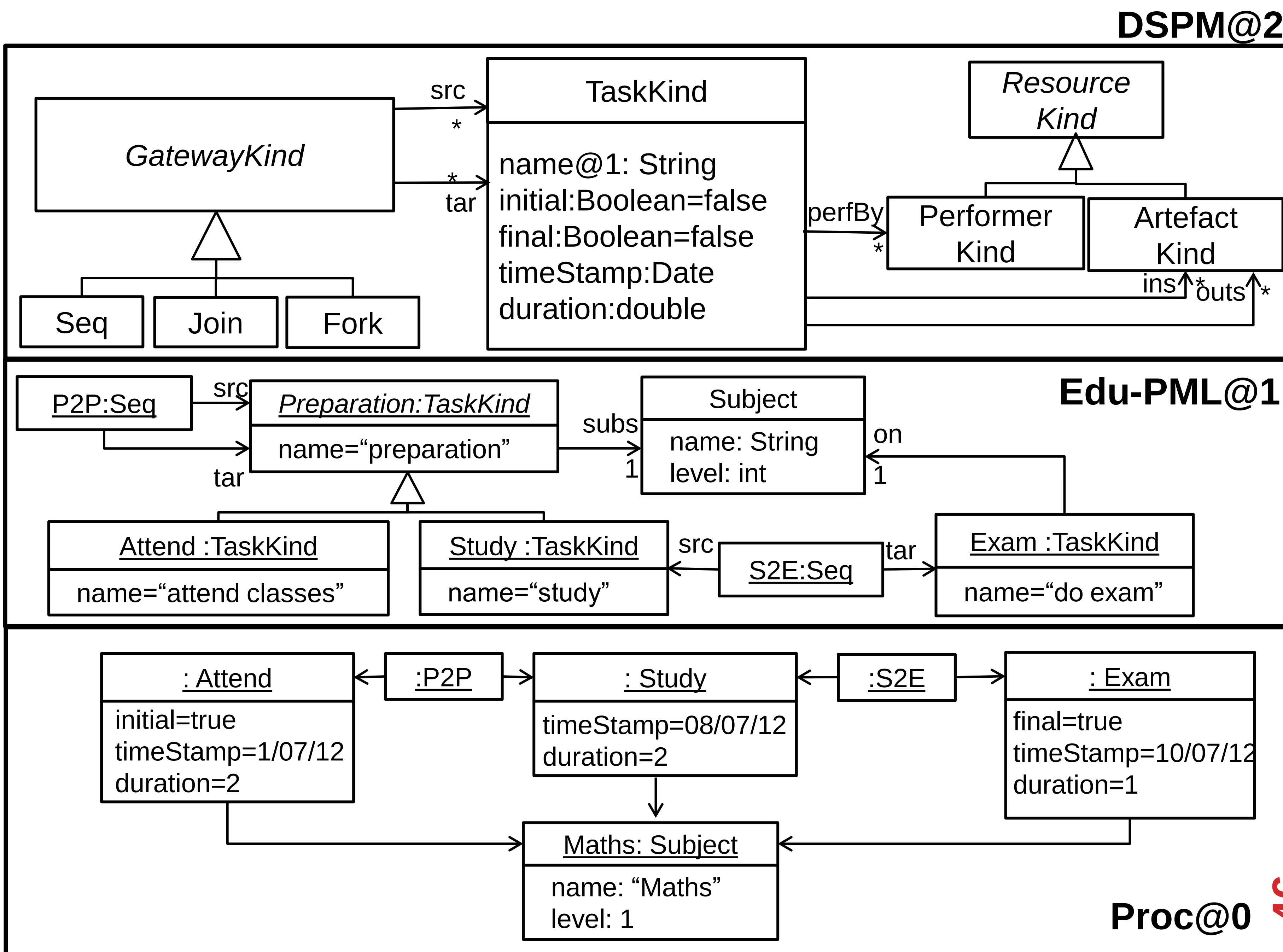
DSPM@2



DOMAIN SPECIFIC PROCESS MODELLING

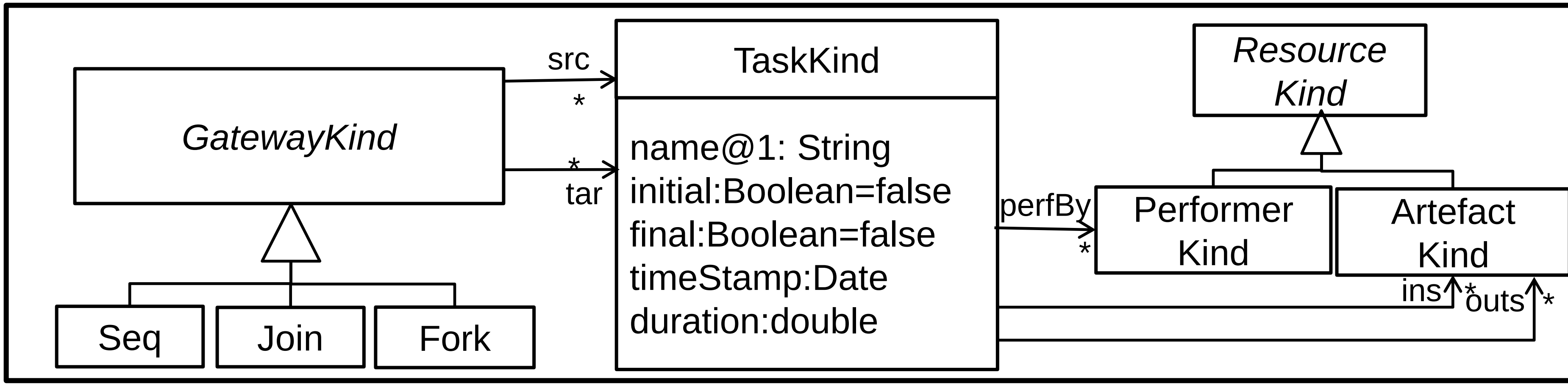


de Lara, Guerra, Sánchez Cuadrado. *Model-driven engineering with domain-specific meta-modelling languages*. Software and System Modeling 14(1): 429-459 (2015)

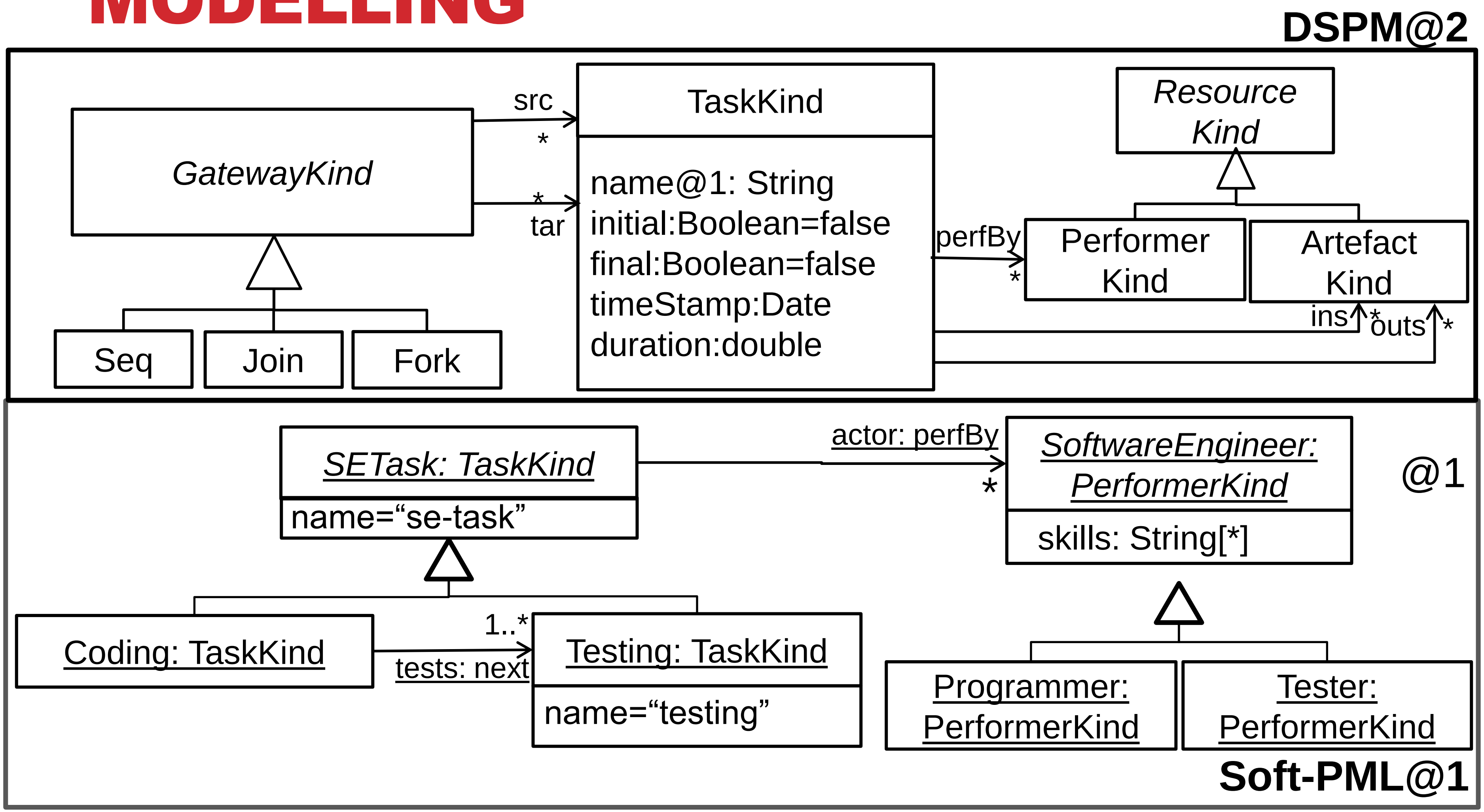

DSPM@2
GatewayKind
src
*
*
tar
TaskKind
name@1: String
initial:Boolean=false
final:Boolean=false
timeStamp:Date
duration:double
Resource Kind
perfBy
*
Performer Kind
Artefact Kind
ins
*
outs
*
Seq
Join
Fork
Edu-PML@1
P2P:Seq
src
tar
Preparation:TaskKind
name="preparation"
subs
1
Subject
name: String
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



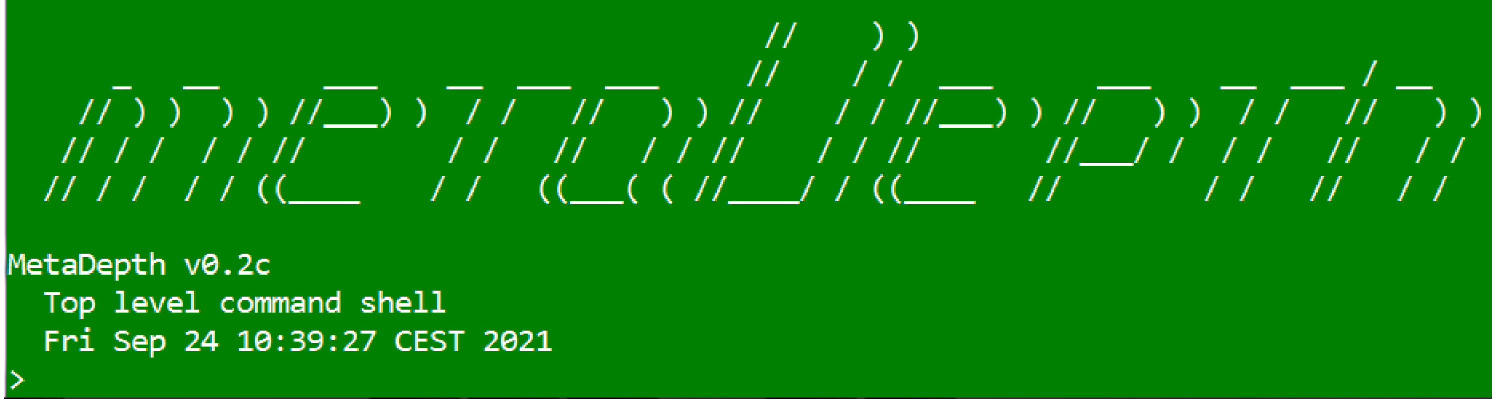
ADVANTAGES

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



Textual multi-level modelling tool with a REPL

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

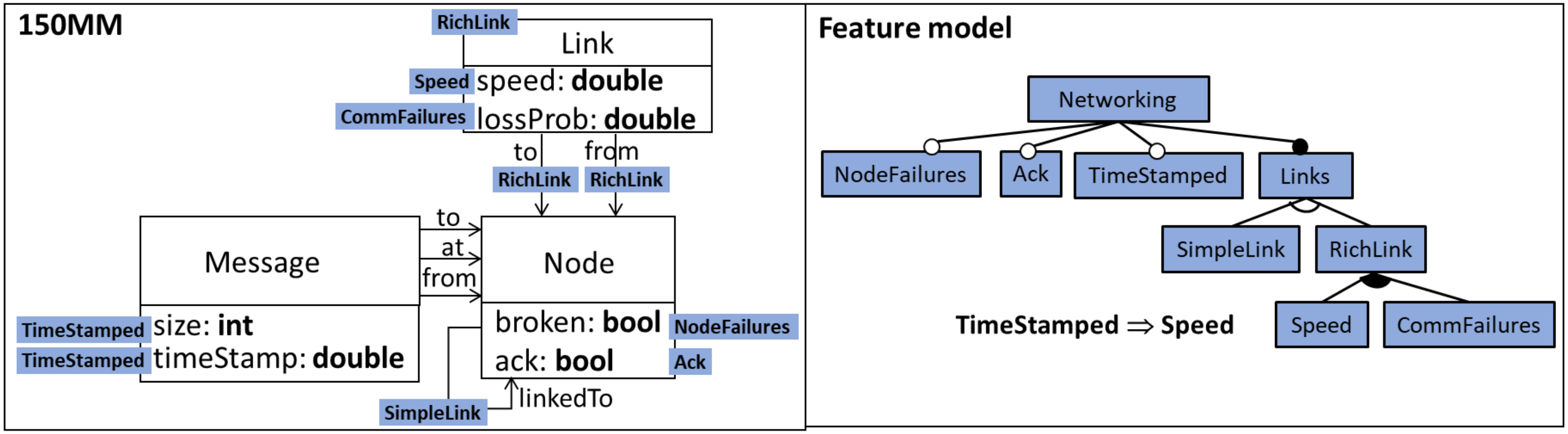
Integrated with the Epsilon Languages for model management

- 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

DISCUSSION AND OPEN LINES

ANNOTATIVE APPROACH



✗ **Not modular**

✗ **Large 150MMs may become difficult to understand (visualization mechanisms) (*)**

~ **Requires two artefacts (150MM + Feature model)**

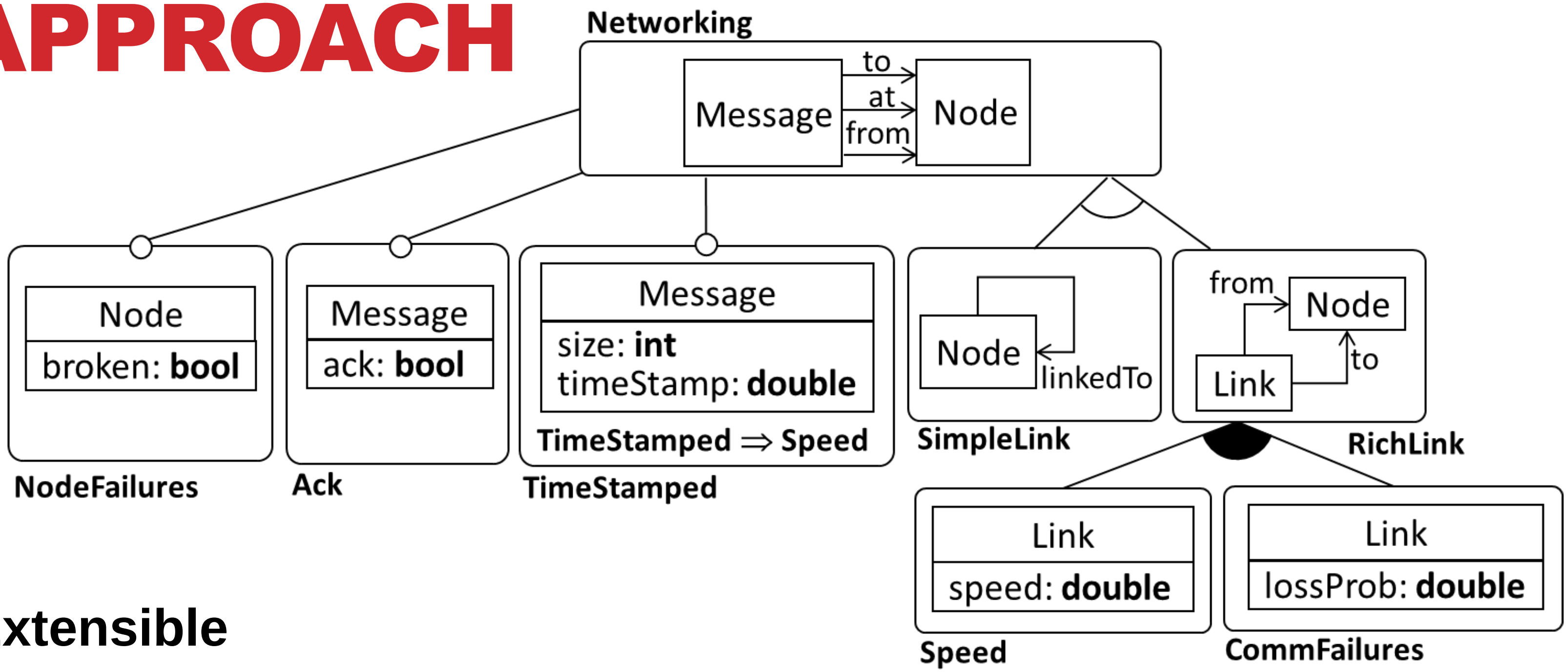
✓ **PCs permit flexible reuse of elements across variants**

- f1 ∨ f2 on a single element would require two modules

✓ **Good for analysis via model finding**

(*) Mahmood, W., Strüber, D., Anjorin, A. et al. “Effects of variability in models: a family of experiments”. Empir Software Eng 27, 72 (2022)

COMPOSITIONAL APPROACH



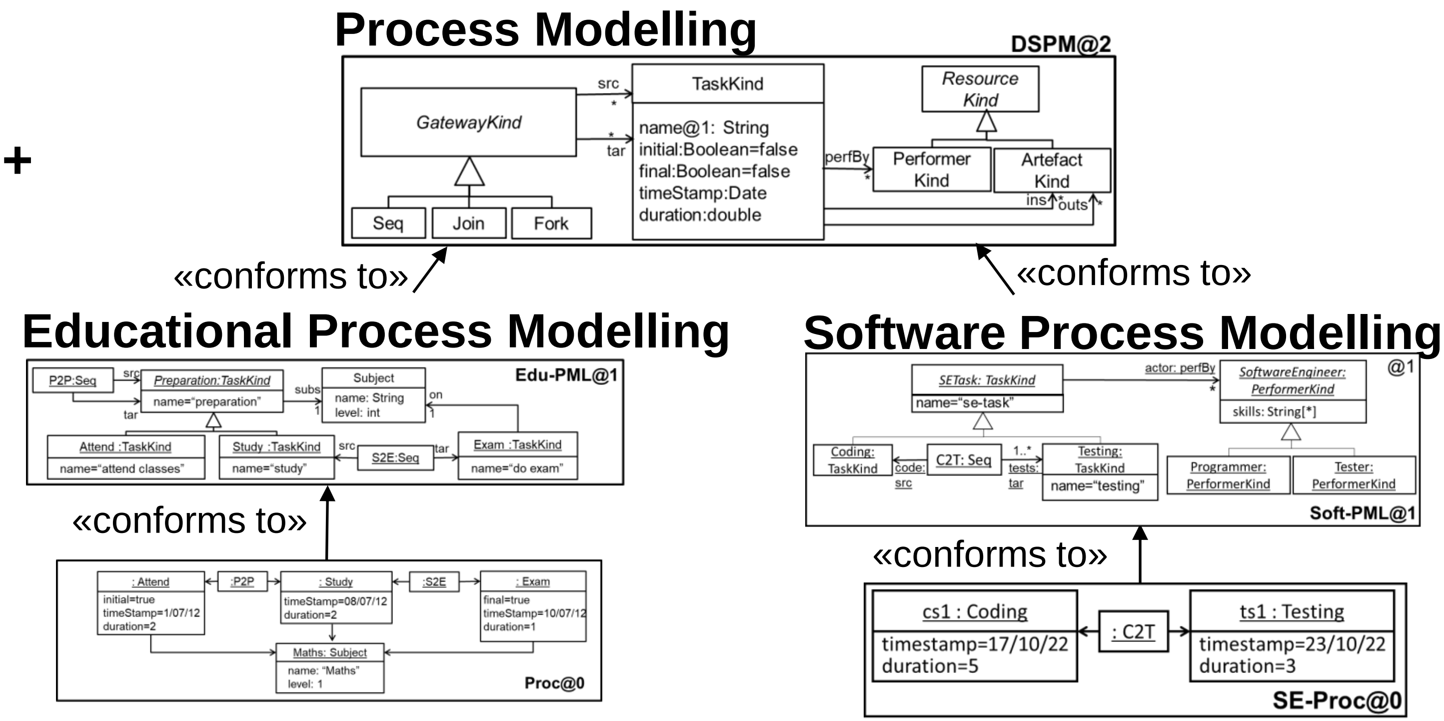
- ✓ **Extensible**
- ✓ **Language features described modularly**
- ✓ **Modules provide both structure and behaviour**
- ✓ **Modules could be reused in different LPLs**
- ✗ **Large families may lead to fragmentation difficult to understand**
- ✗ **Behaviour definition needs to merge the meta-models**
  - And some analysis too

MULTI-LEVEL MODELLING

Generic Process Language + Services

Domains

Models



✓ **Open variability**

- Refinement

✓ **Domain-specific meta-modelling**

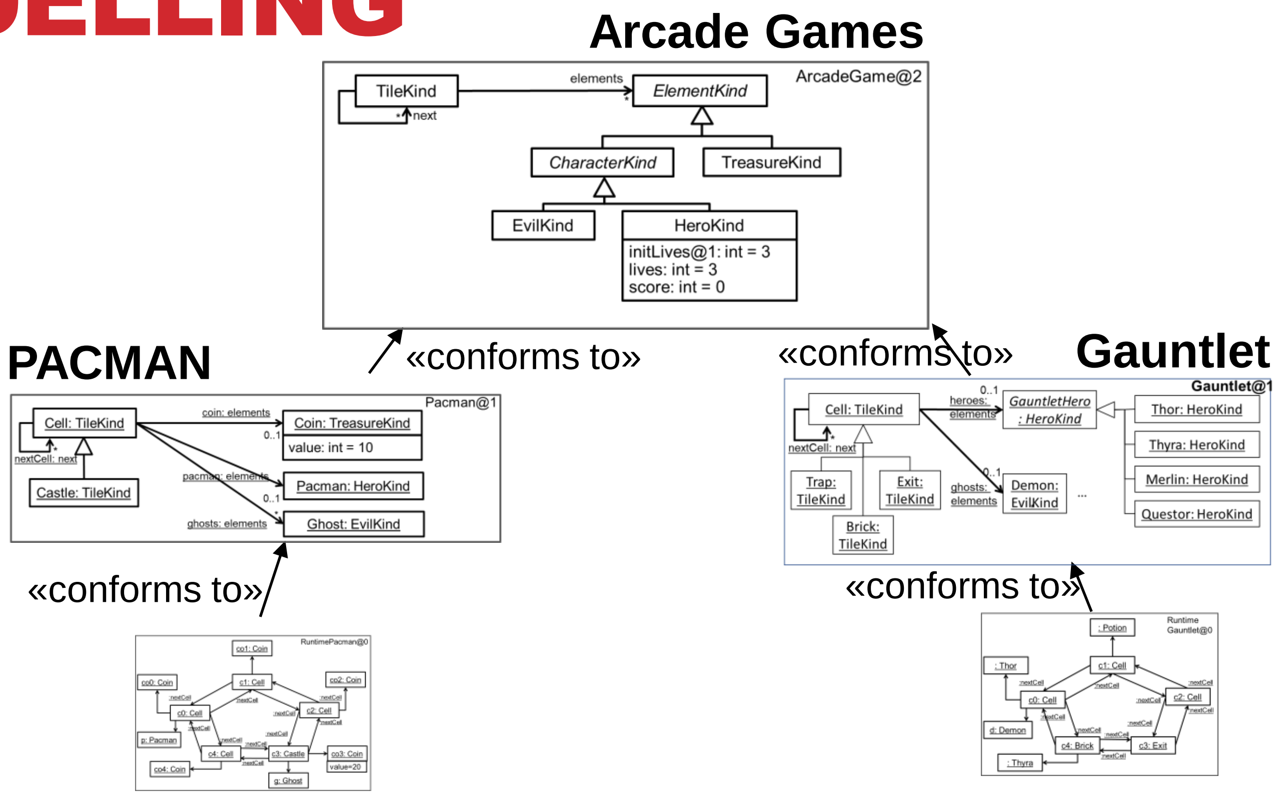
✗ **Interoperability: requires specific technology**

MULTI-LEVEL MODELLING

Generic Arcade Language + Services

Game Types

Games at run time



✓ **Open variability**

- Refinement

✓ **Domain-specific meta-modelling**

✗ **Interoperability: requires specific technology**

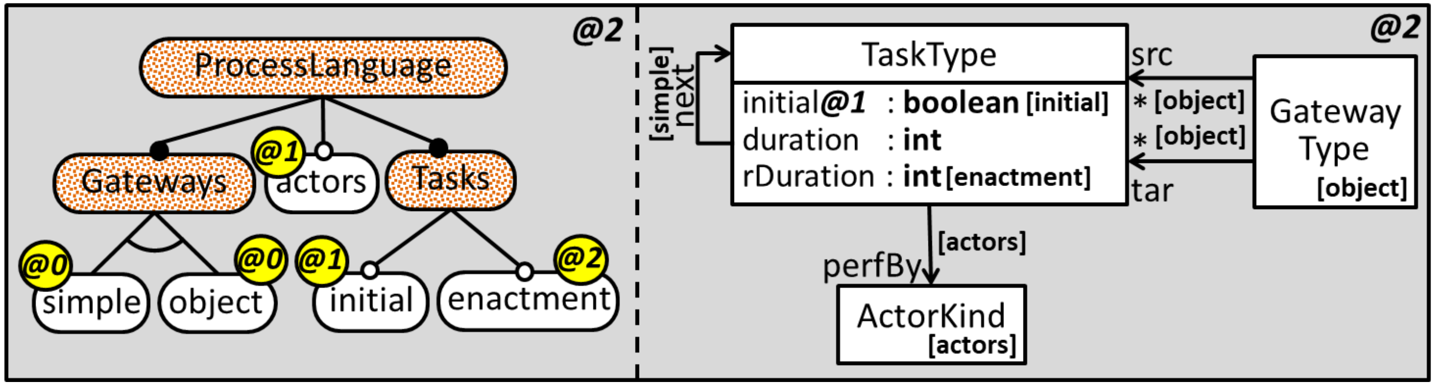
CAN WE COMBINE…?

✓ **Compositional + annotative**

- Compositional for family language design
- Annotative for analysis

✓ **Multi-level + annotative**

- Configurable language, which can be refined (*)



? **Multi-level + Compositional**

(*) de Lara, Guerra. "Language Family Engineering with Product Lines of Multi-level Models". Formal Aspects Comput. 33(6): 1173-1208 (2021)

OPEN LINES

Techniques

- Analysis
- Concrete syntax
- Combination open + closed

Applications for

- Education (akin to gradual programming languages*)
- Low-code development to support citizen developers with wide range of skills

(*) https://www.hedycode.com/

THANKS!

Thanks to my co-authors: Marsha Chechik, Esther Guerra, Rick Salay, Jesús Sánchez Cuadrado

Juan.deLara@uam.es
@miso_uam