Johannes-Kepler-Univ, Linz
6. Juni 2013

Fixing MOF + RDF (?)
A View from the ConceptBase Data Model

Manfred Jeusfeld
Tilburg University

Brief History of ConceptBase

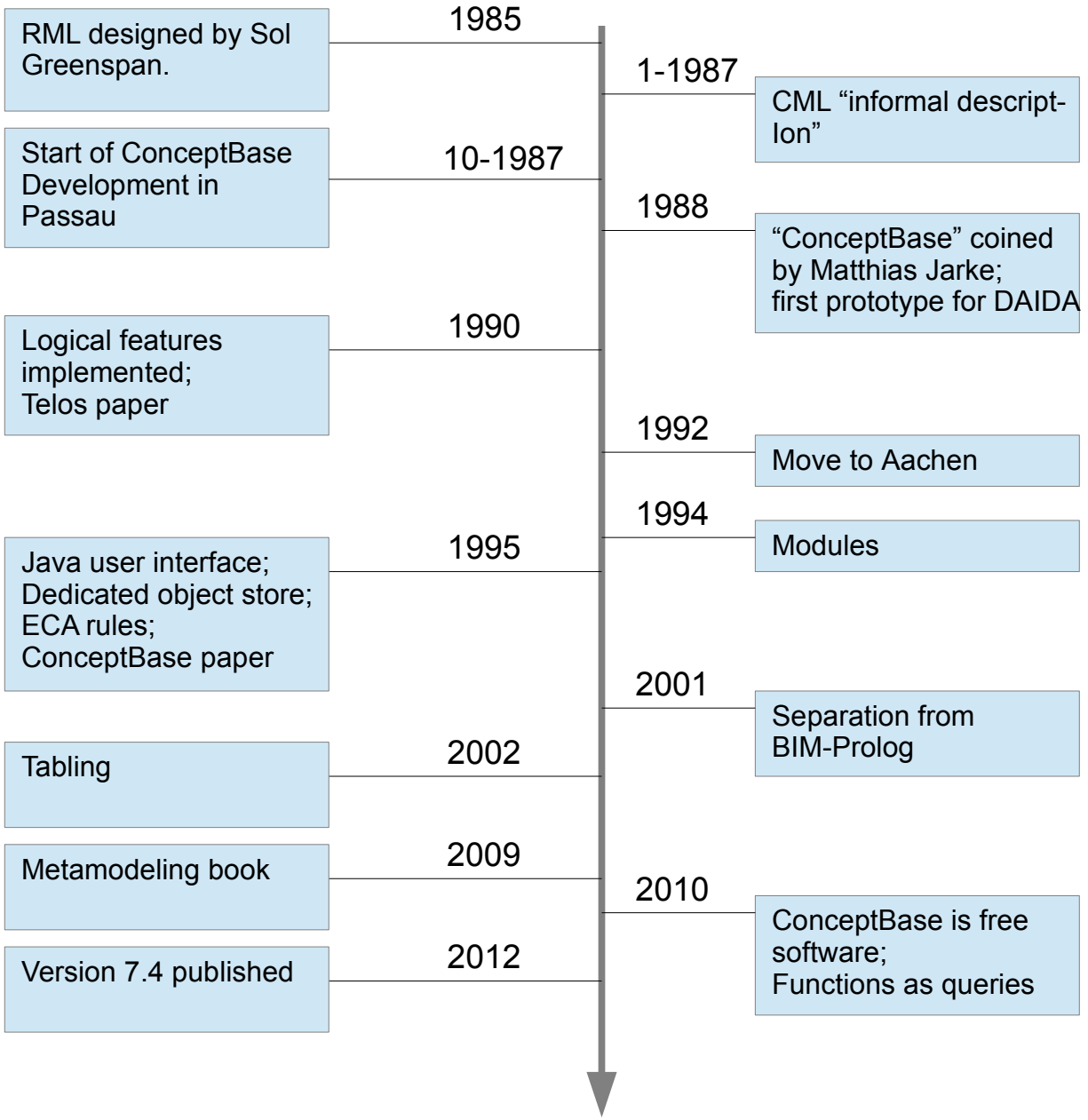


Figure on the instantiation in the 1987 CML paper

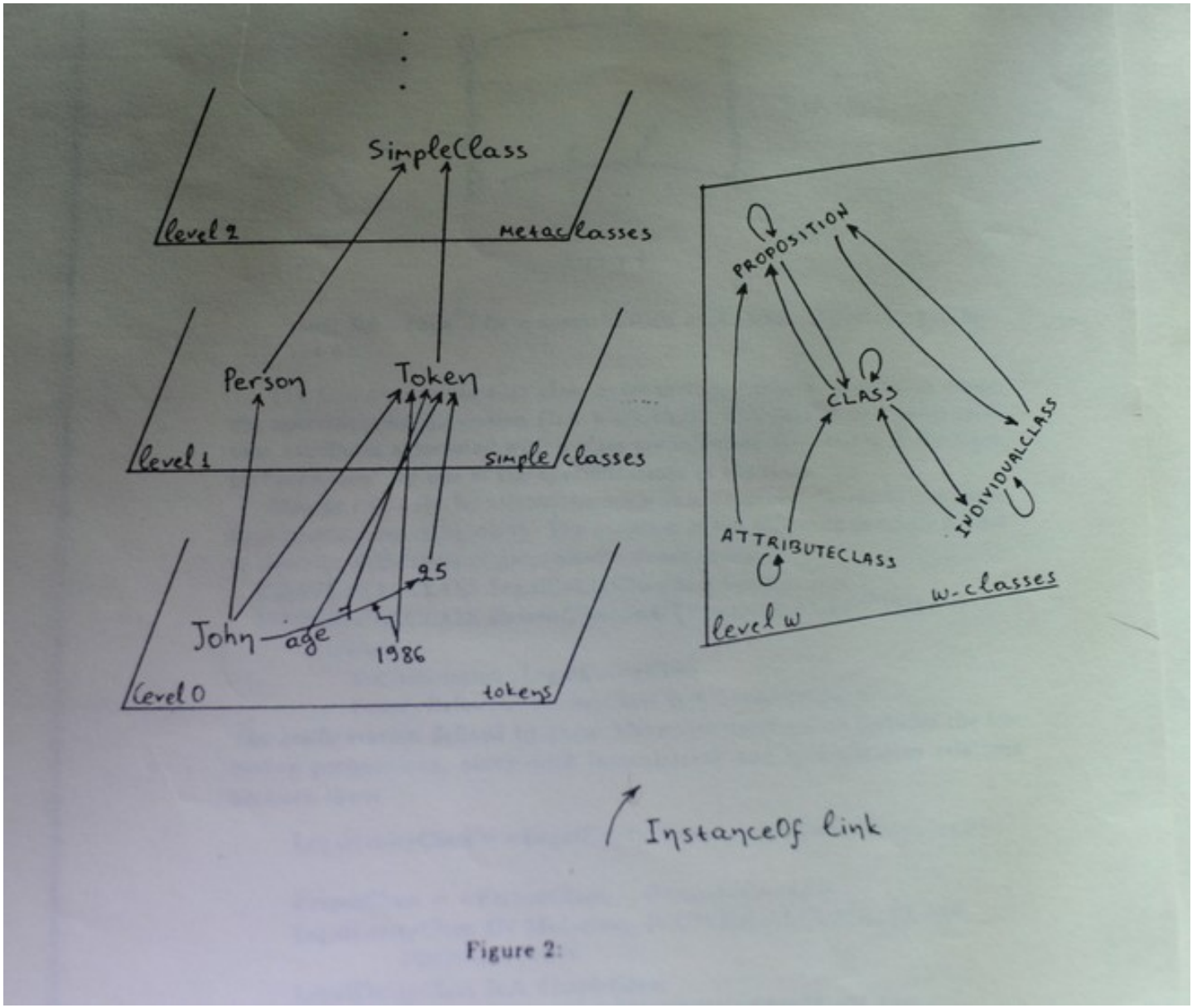
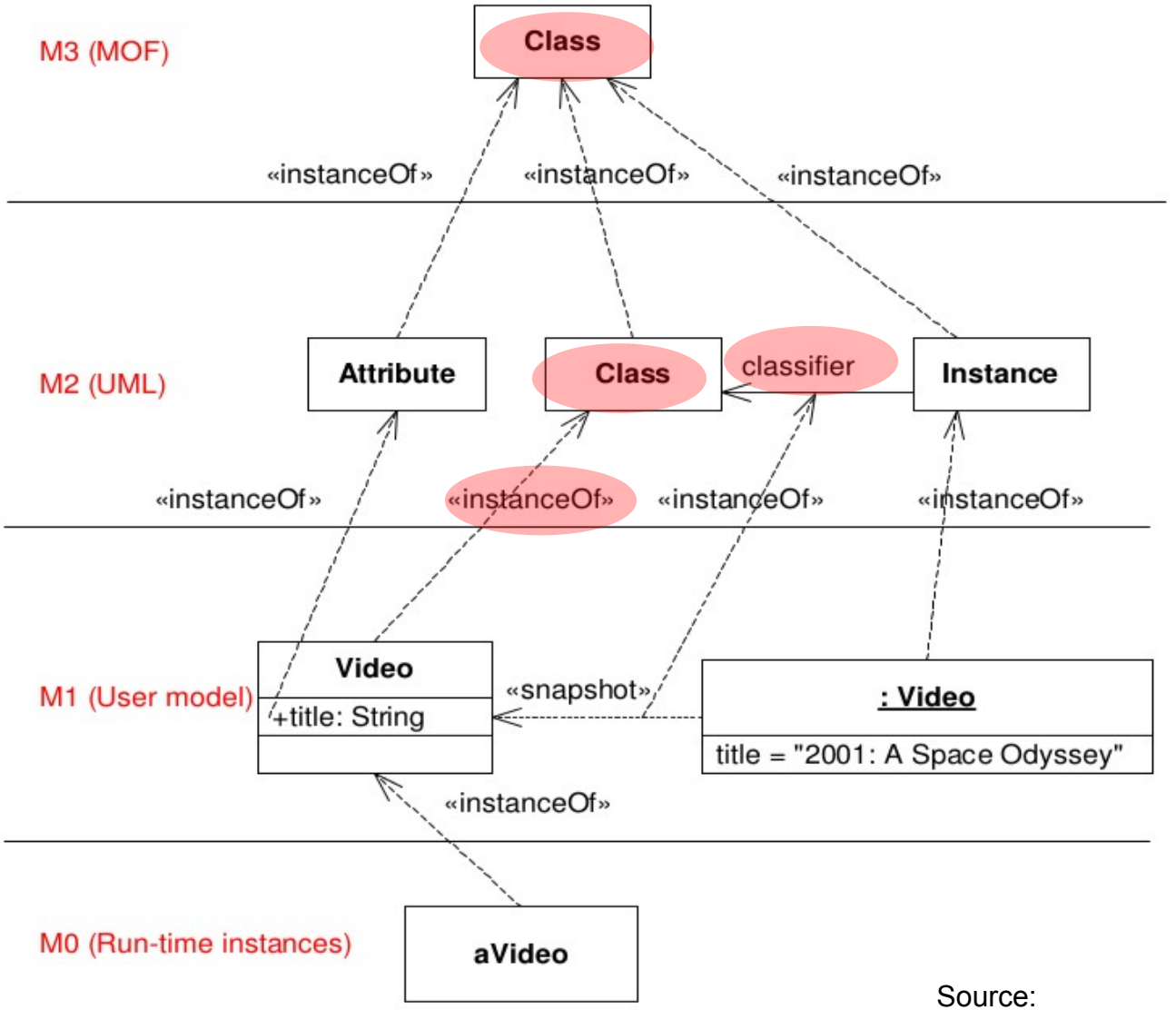


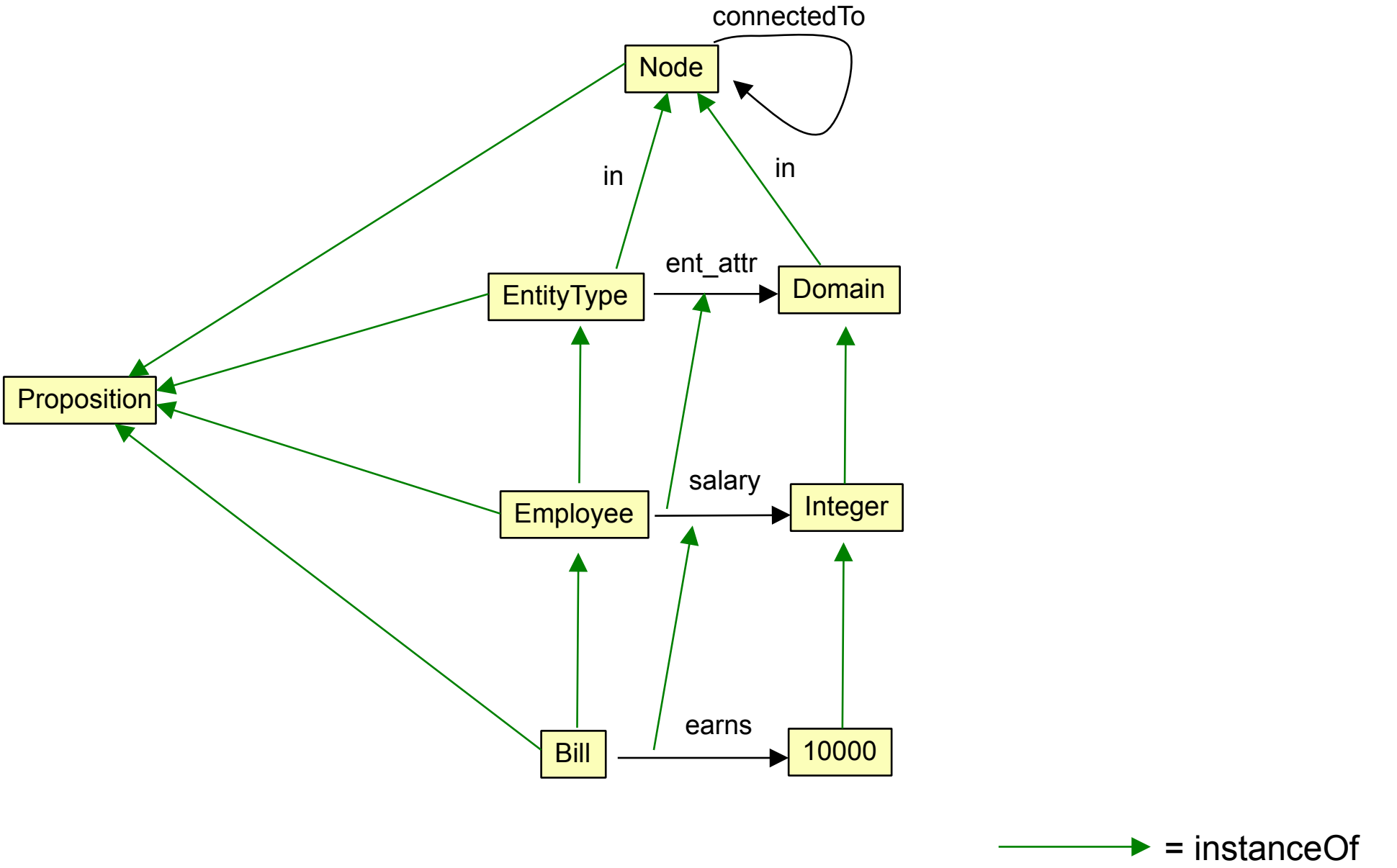
Figure 2:

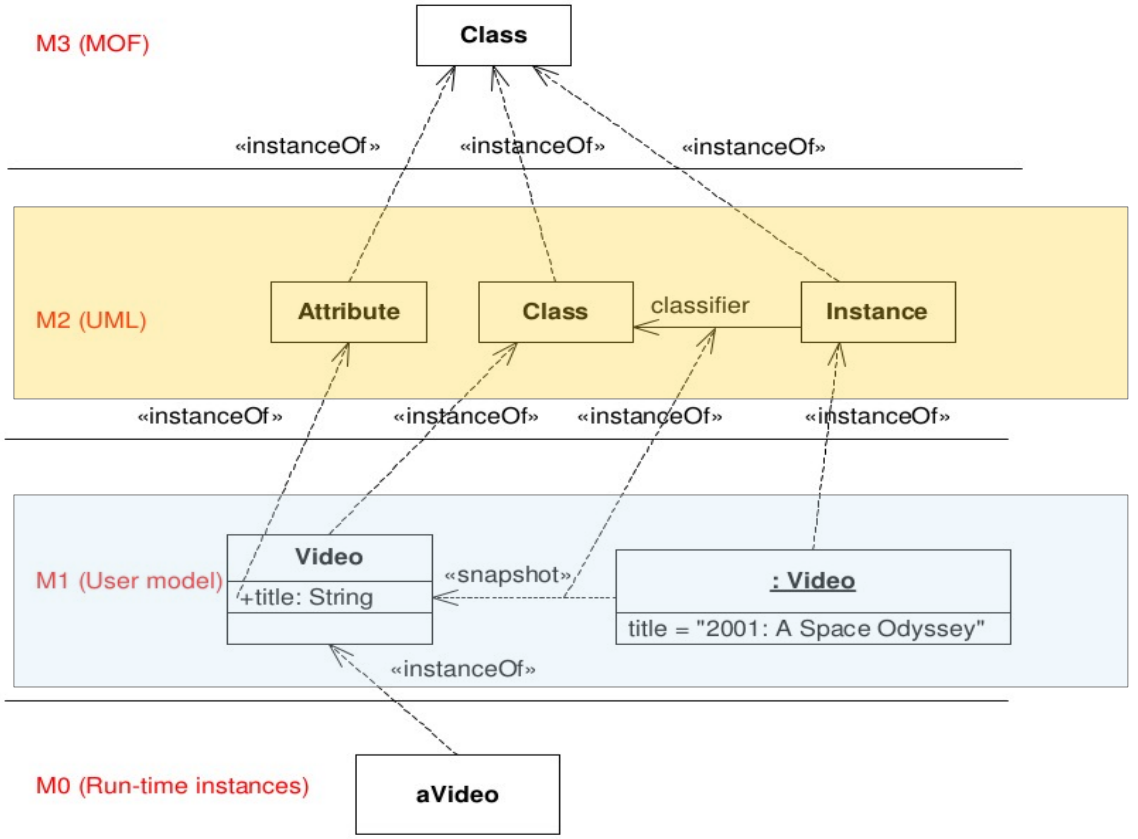
The Meta Object Facility (MOF)



Source:
OMG (2010): UML Infrastructure Version 2.3

Metamodeling with ConceptBase





Never changes

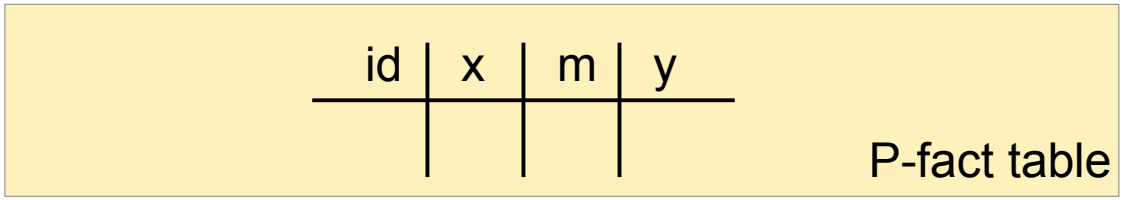
M2=Schema

M1=Instance

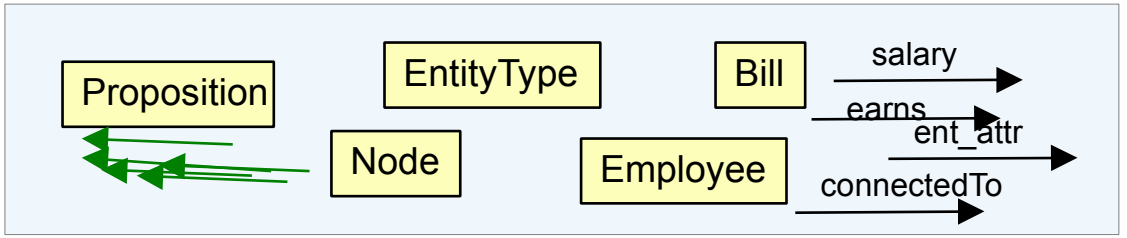
Not represented

OMG meta modeling is just about M2 and M1 (schema and its instance)

What is different then in ConceptBase?



Schema



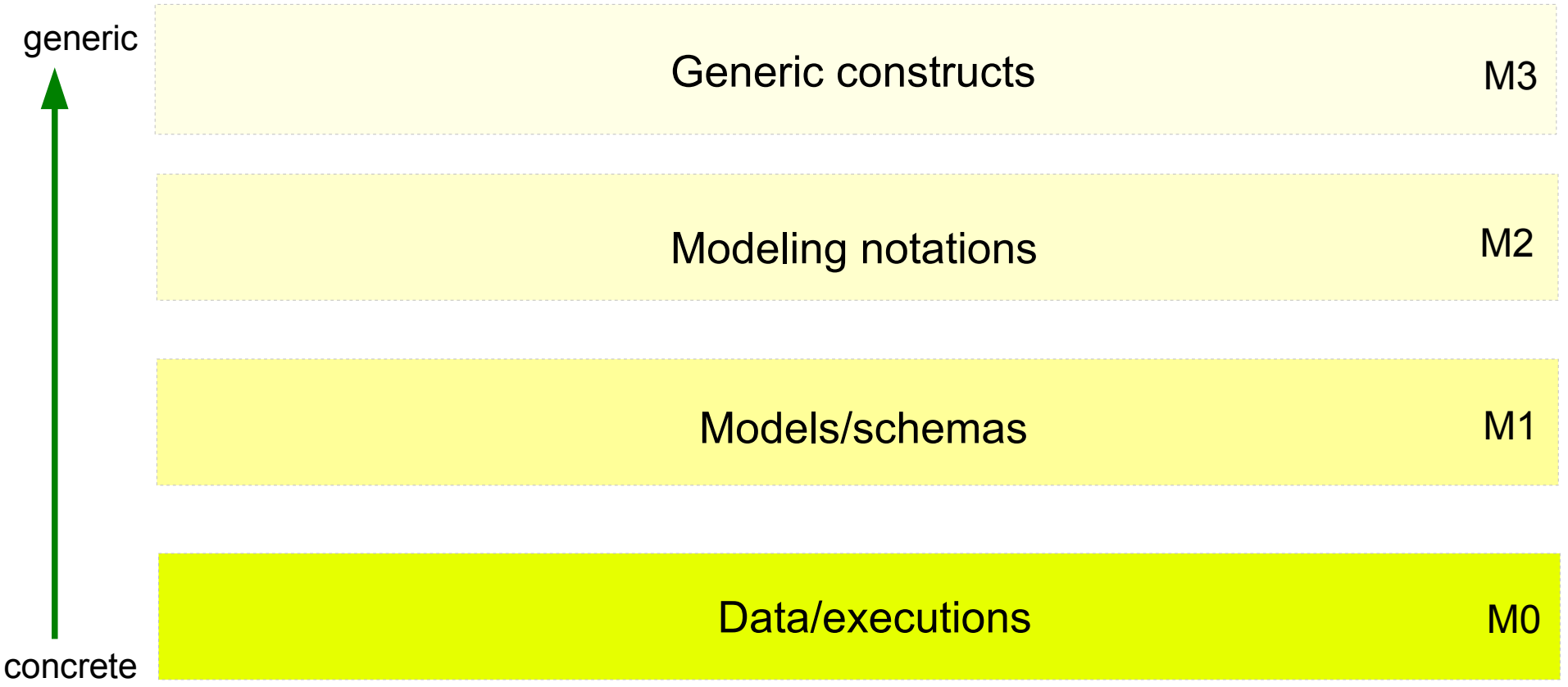
Instance

ConceptBase also has just one schema (the P fact table) and and its instance (the extension of the table) but schema and instance have nothing to do with the MOF levels M3,M2,M1,M0.

Instead, instantiation between two objects is just represented as a tuple in the P-fact table.

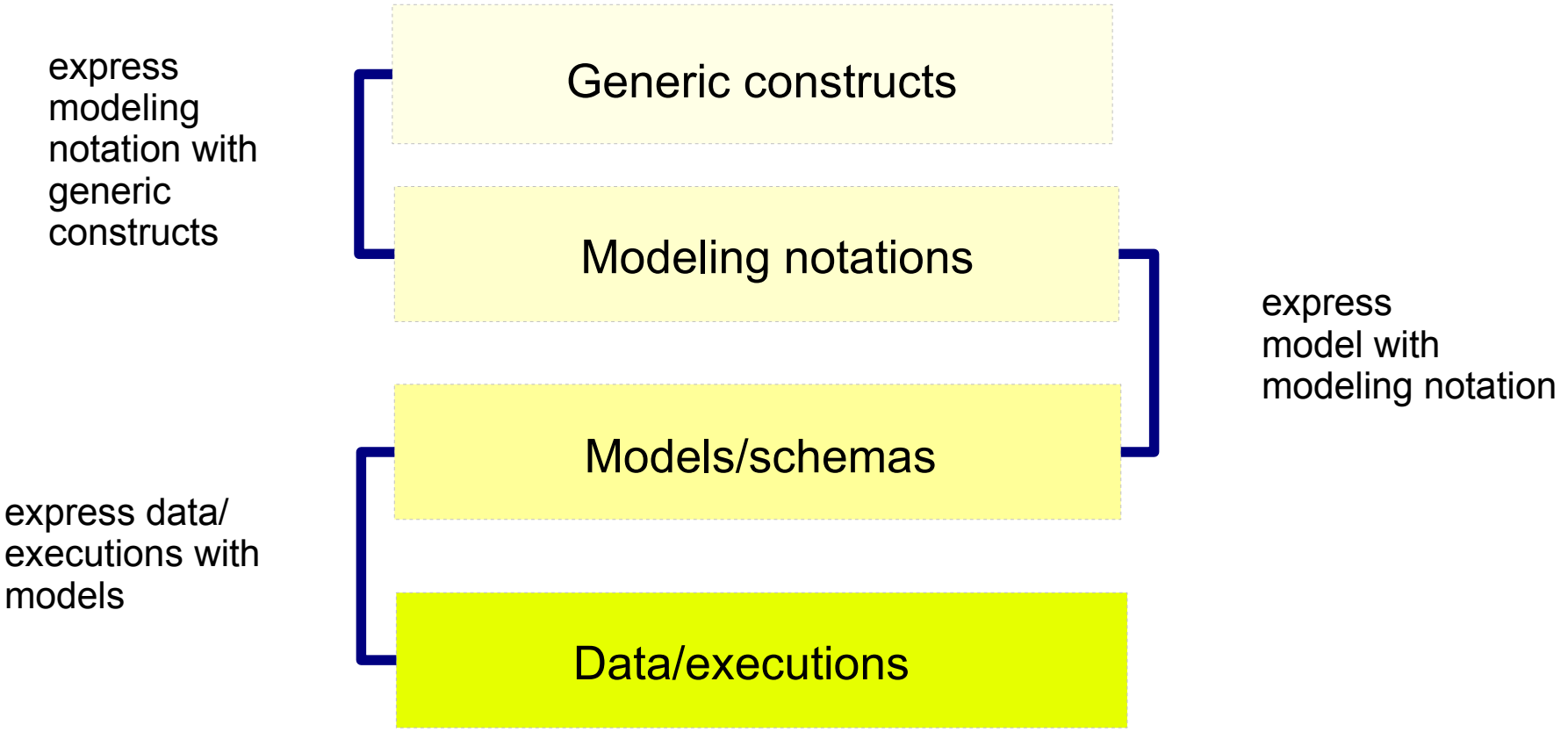
→ Any number of abstraction levels is supported and there is no constraint that forbids to associate objects from different abstraction levels.

Abstraction levels (IRDS,MOF)[1]



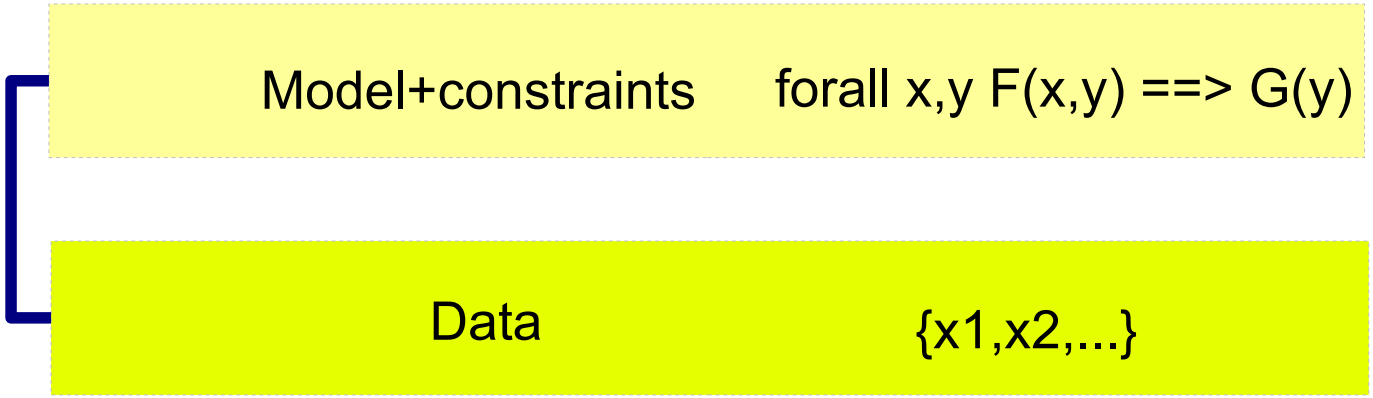
1) The OMG Meta Object Facility (MOF) is not limited to M0-M3 though in practice these 4 levels appear sufficient for metamodeling. Prior to MOF, the same abstraction levels were proposed in the ISO IRDS standard (1990). M0-M2 were pioneered by J. Abrial (1974)

Level pairs



- constraints on the higher level define allowed instantiations of concepts/constructs

Level pairs and “simple” constraints



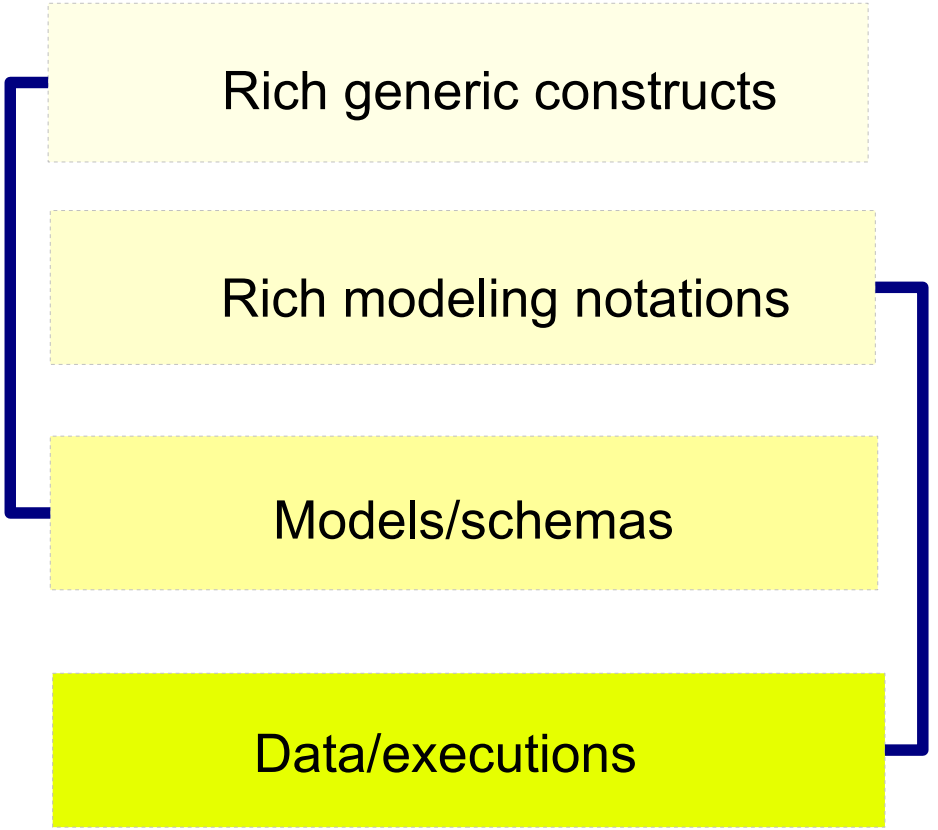
The lower level has to satisfy the constraints of the upper level. Constraints/rules are referring to the upper level by constants (or predicate/function names). Objects at the lower level are referred to by variables.

Typical example: Object Constraint Language of UML (OCL)
Classical logic (and Description Logic) follow the same scheme

Meta level pairs

constrain
the models
by expressing
the notation in
terms of
generic constructs
plus their
meta constraints

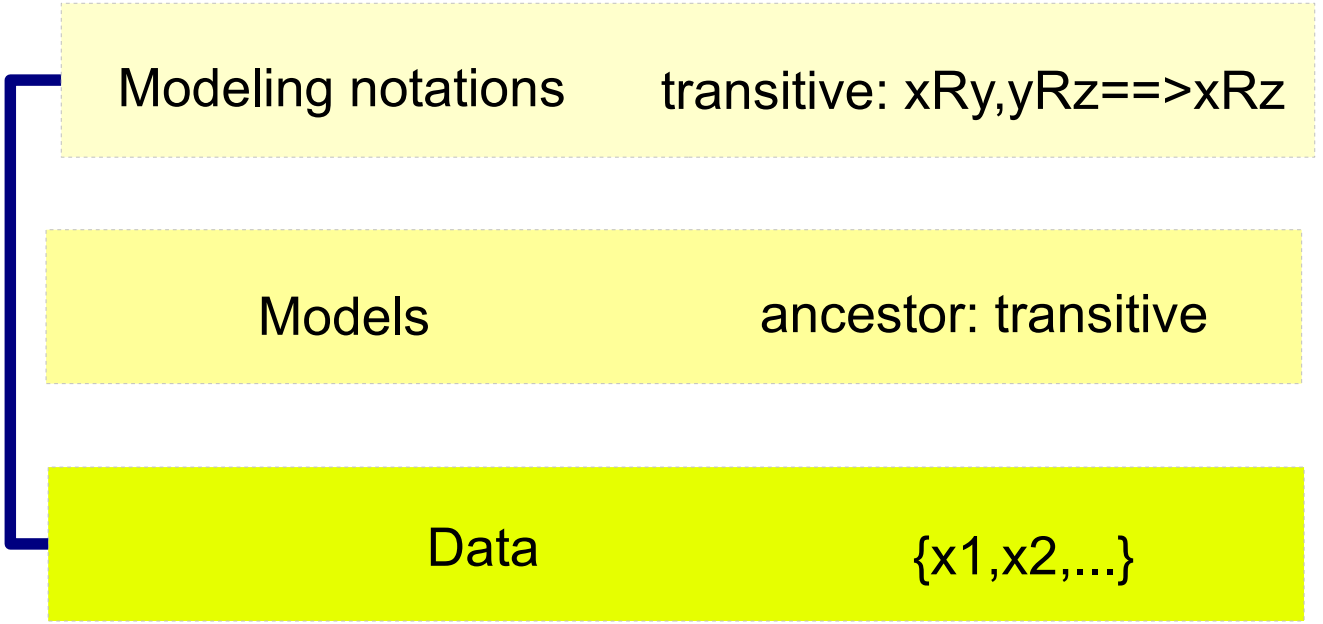
example:
generic construct
for model element



constrain data/
execution by
rich modeling
constructs

example: cardinality
construct

Meta level pairs and “generic” constraints



Define generic constraints at the meta level that operate on objects two levels below.

The specific problem statement

- Can we have a **library of re-usable generic constructs** including a specification of their semantics such that new modeling languages can be defined in terms on these constructs?

- We rely on 1st order logics (more precisely: Datalog with negation) for defining the constructs and their re-use: computable semantics, efficient implementations

The MOF abstraction levels

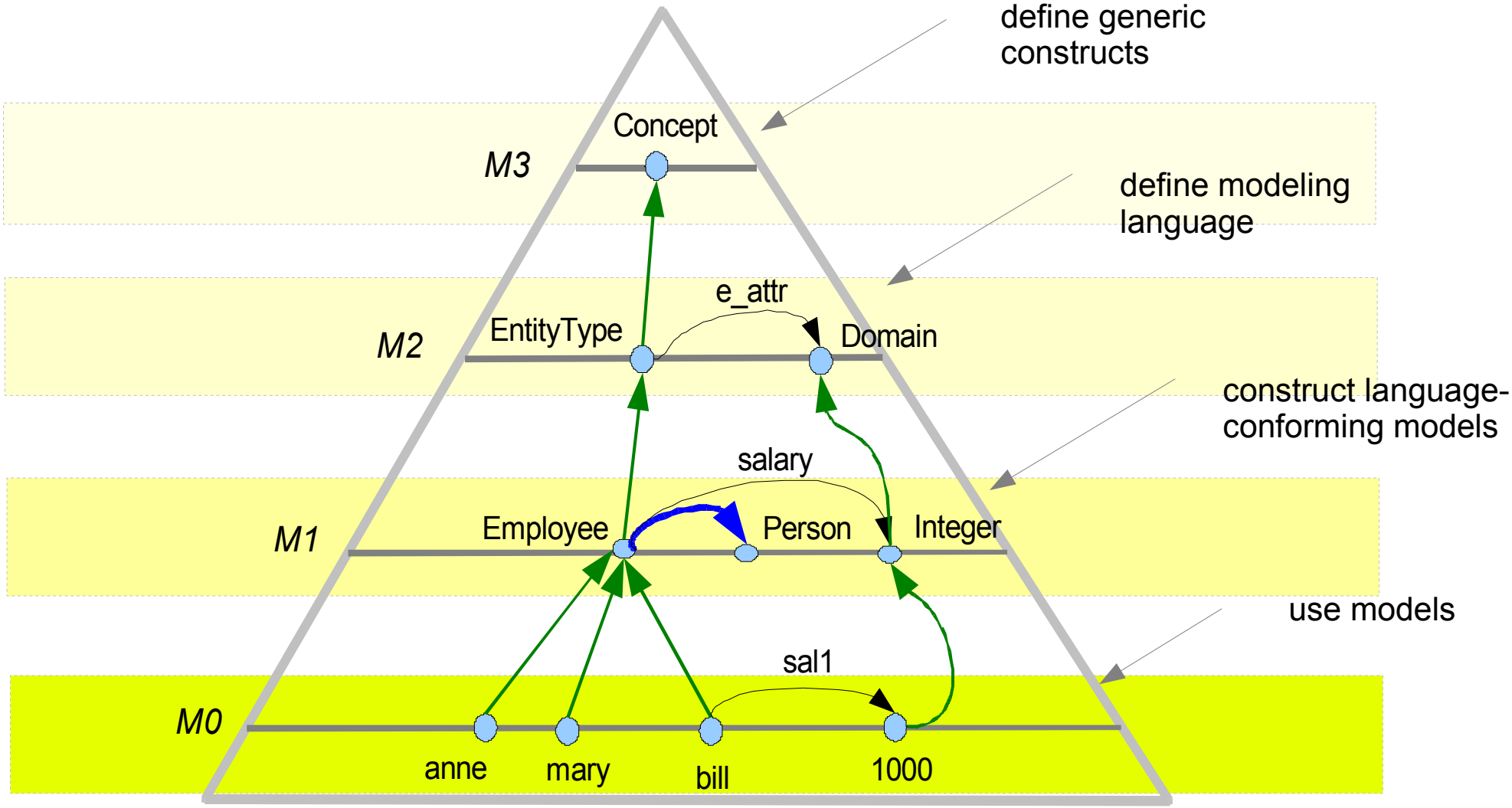


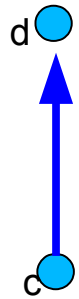
Figure 1: An interpretation of the MOF abstraction levels

PS: 'Concept' is a synonym for 'Node'!

Predicates

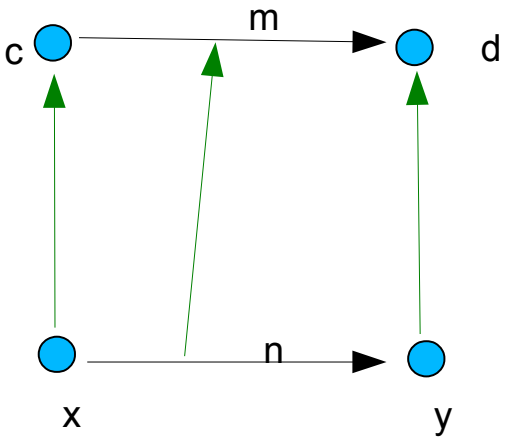
`In(x,c)`

The object x is an instance of object c (=its class)



`Isa(c,d)`

The object c is a specialization of d (=its superclass)

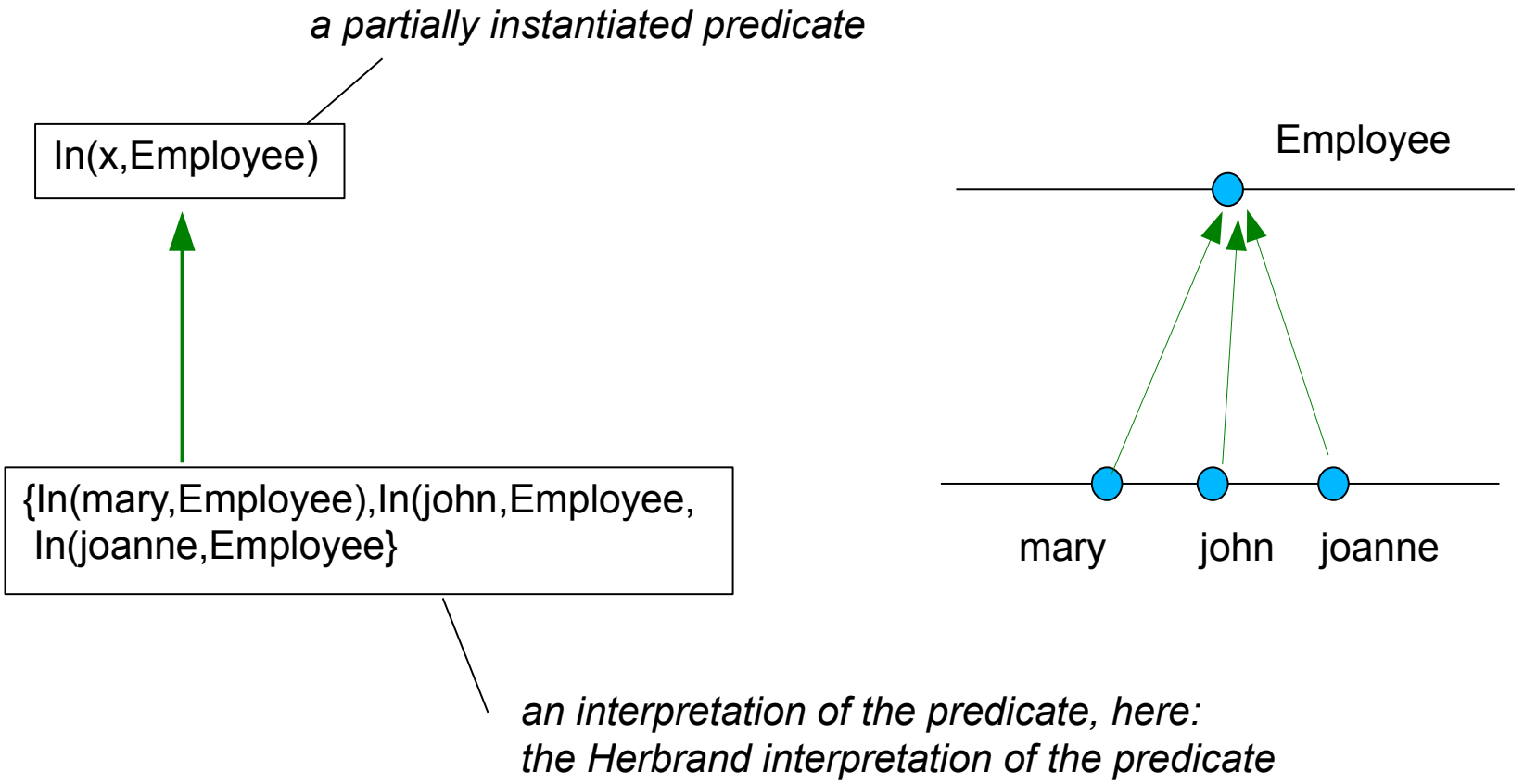


`AL(x,m,n,y)`

```
P(c!m,c,m,d)
P(x!n,x,n,y)
In(x!n,c!m)
```

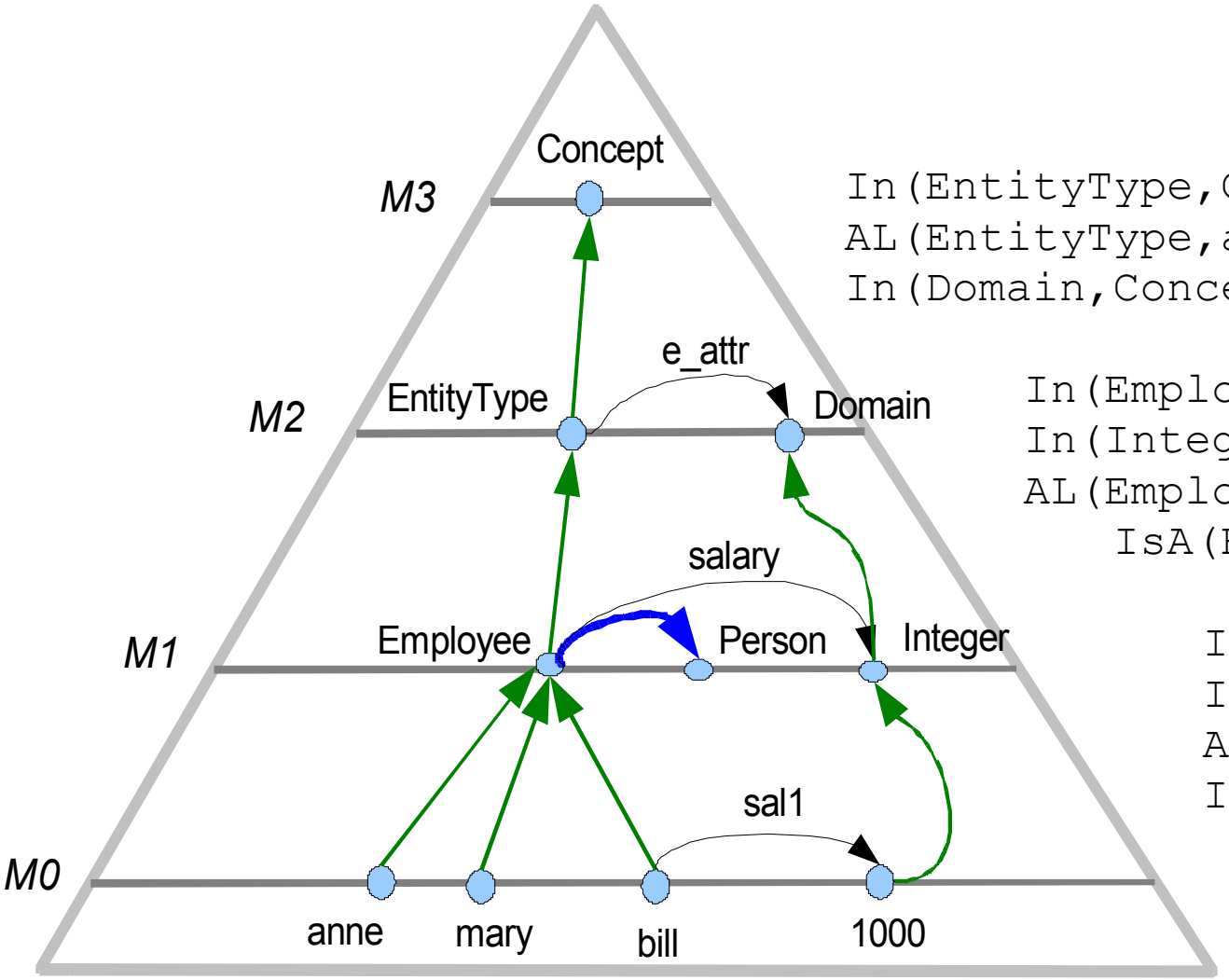
The object x has an attribute n with value y; this attribute has the class m (c!m)

Interpretation of predicates vs. classes



Datalog: compute the minimal Herbrand interpretation with a fixpoint operator

MOF in Logic




```
In(EntityType,Concept)
AL(EntityType,attribute,e_attr,Domain)
In(Domain,Concept)

In(Employee,EntityType)
In(Integer,Domain)
AL(Employee,e_attr,salary,Integer)
    IsA(Employee,Person)

In(anne,Employee)
In(bill,Employee)
AL(bill,salary,sal1,1000)
In(1000,Integer)
```

Figure 1: An interpretation of the MOF abstraction levels

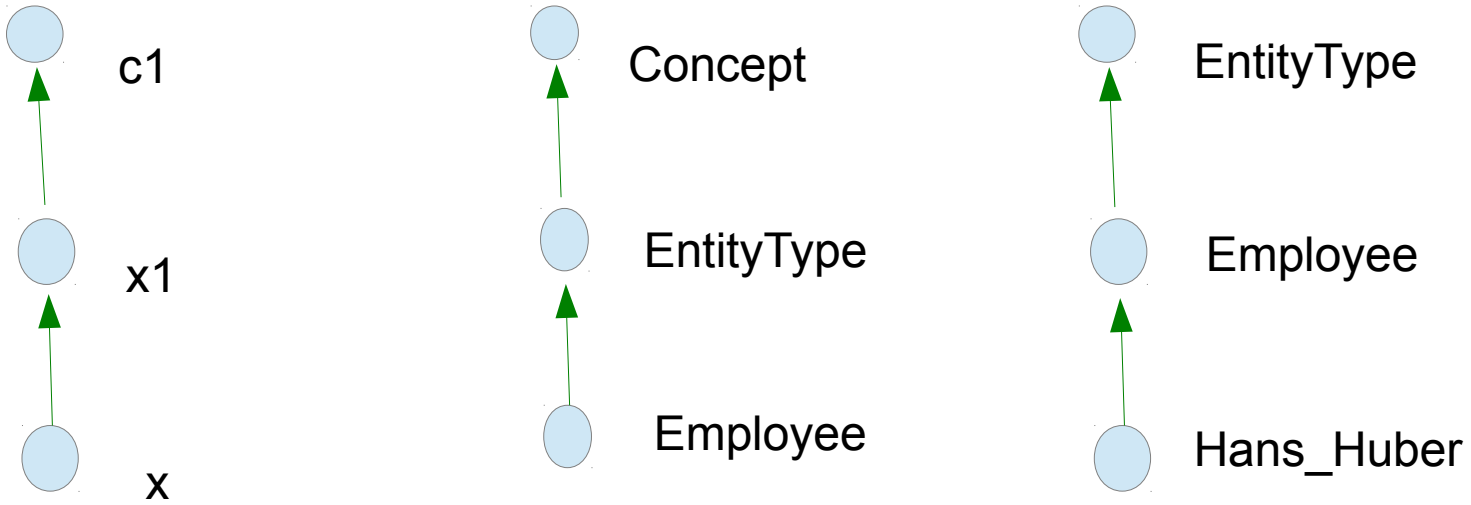
Some generic constructs

- 'key' property of certain attributes of a relation (or entity type)
- transitivity (symmetry,etc.) of a relation
- the fact that something is an entity and not a value
- the traceability of model elements

- Idea: formulate those constructs as a generic formula and use partial evaluation to map to concrete languages

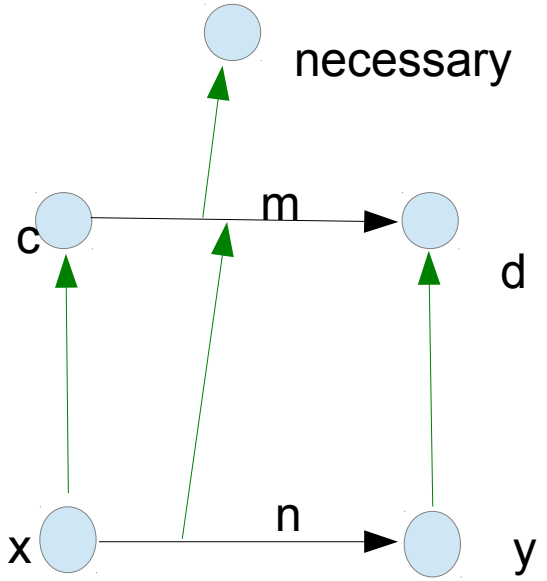
Definition

A variable occurrence x1 in a predicate p is called a *meta variable* iff p=In(x,x1), or p=AL(x,x1,n,y). A *generic formula* is a first order formula with at least one meta variable.



Example: the 'necessary' construct

```
forall x,a,c,m,d In(a,necessary)
and  P(a,c,m,d) and In(x,c) ⇒
exists y,n In(y,d) and
AL(x,m,n,y)
```

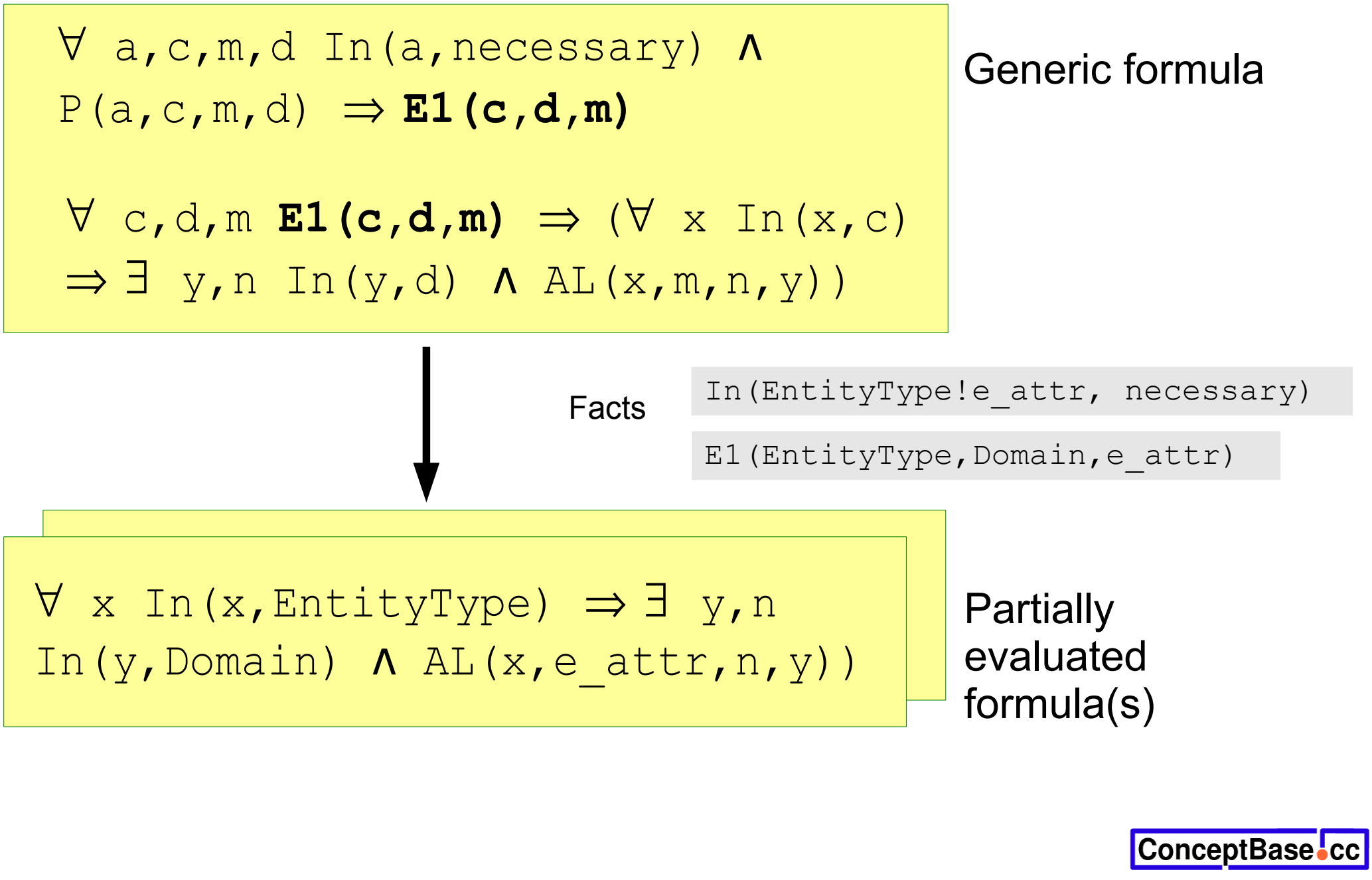


Certain attributes of concepts are declared as 'necessary', i.e. each instance x of class c must have a filler y for the necessary attribute m.

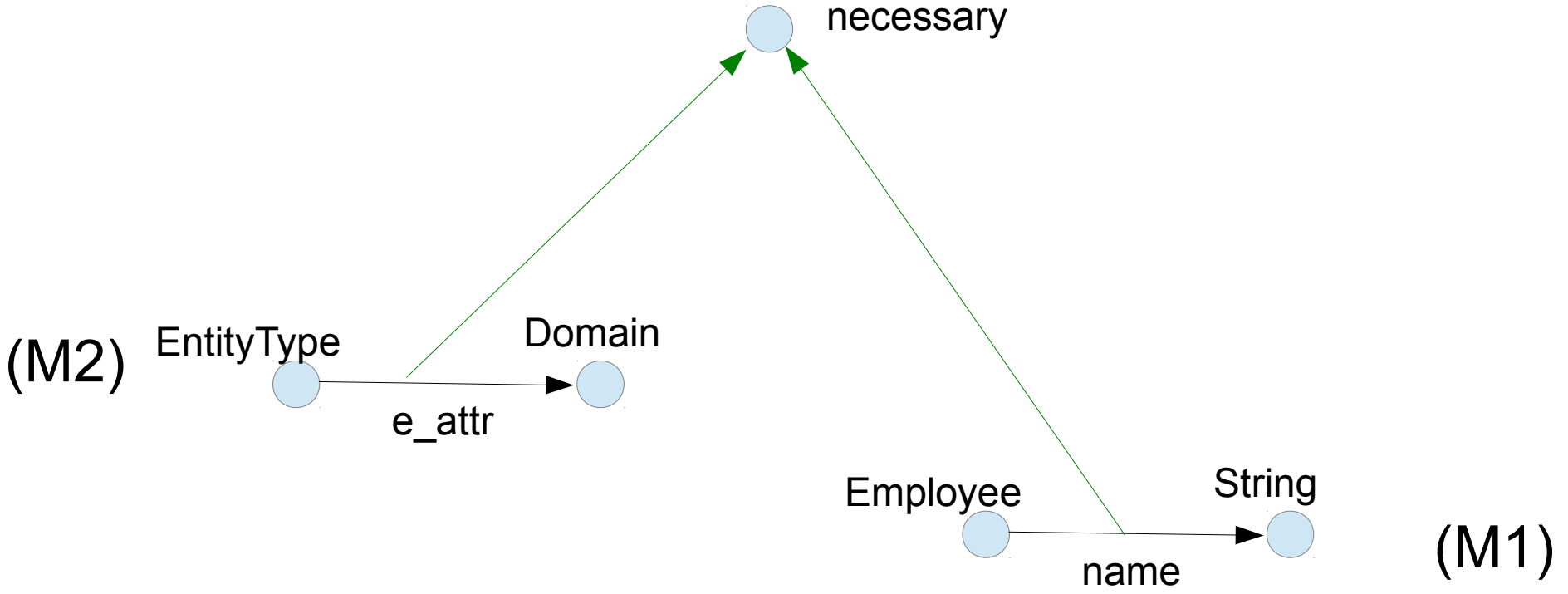
Partial evaluation

- transform meta formula into a **normal form** (exposing the so-called E predicate)
- replace the E-predicate by its interpretation (=current set of facts matching the E-predicate)
- repeat until no more meta variables

→ Result: conjunction (disjunction) of formulas without meta variables; components of the formula are defining the semantics of the **use** of a construct

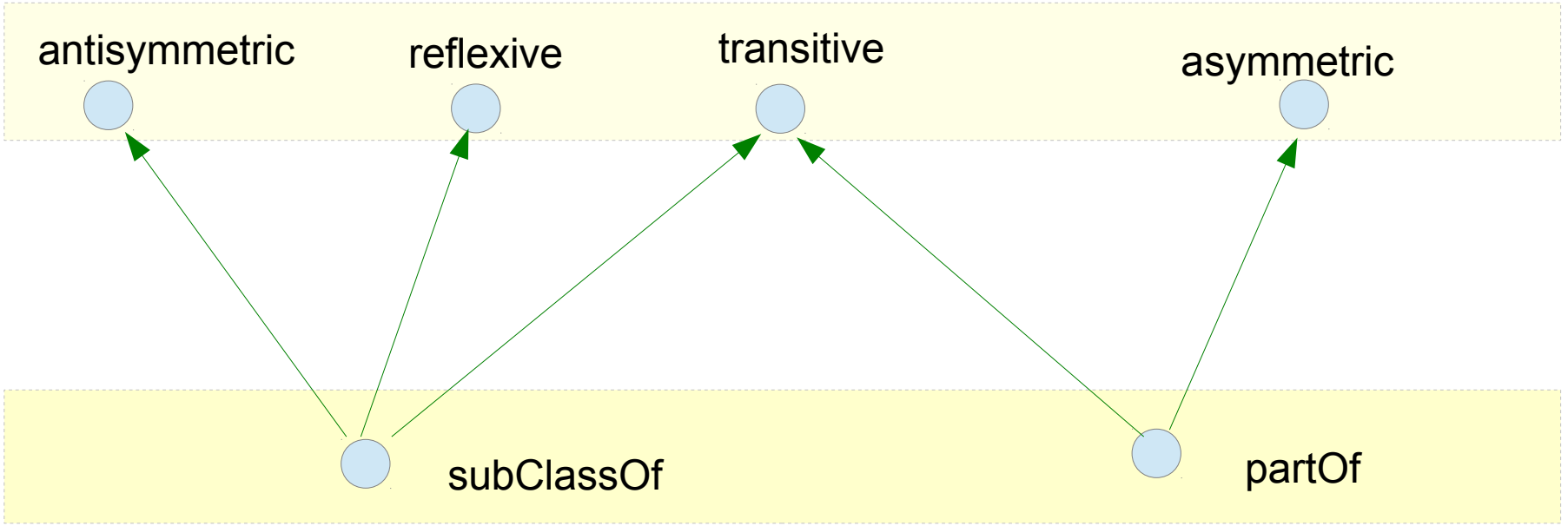


The 'necessary' construct is now (very) reusable!



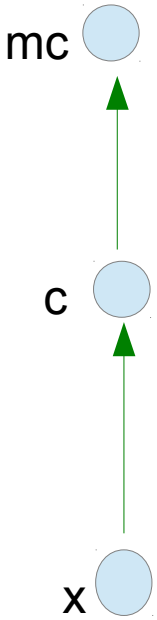
- The 'necessary' construct has been defined independently from the abstraction level (M0-M3). Hence, it can be used for defining modeling language constructs (e_attr) as well for defining model element (name attribute of Employee)

Another example



We just need to define those basic relational properties once and then re-use their definitions for defining modeling constructs

A useful shortcut: In2(x,mc)



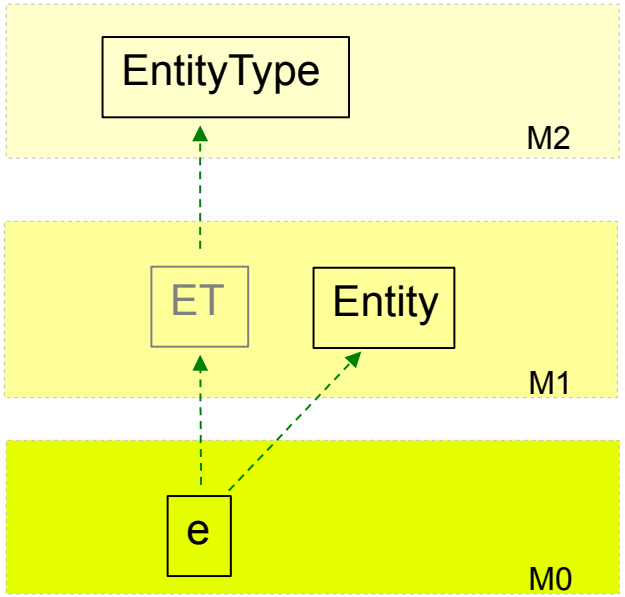
$$\forall\ x,c,mc\ \ In(x,c) \wedge In(c,mc) \Rightarrow In2(x,mc)$$

The formula is evaluated as a deductive rule (deriving In2). It relates objects to their meta classes. Instead of partially evaluating it, we just replace each `In2(x,mc)` by

$$\exists\ c\ \ In(x,c) \wedge In(c,mc)$$

Infix syntax for In2(x,mc) : (x [in] mc)

Distinguish entities from values (1)



"An 'entity' is an instance of some entity type ET."

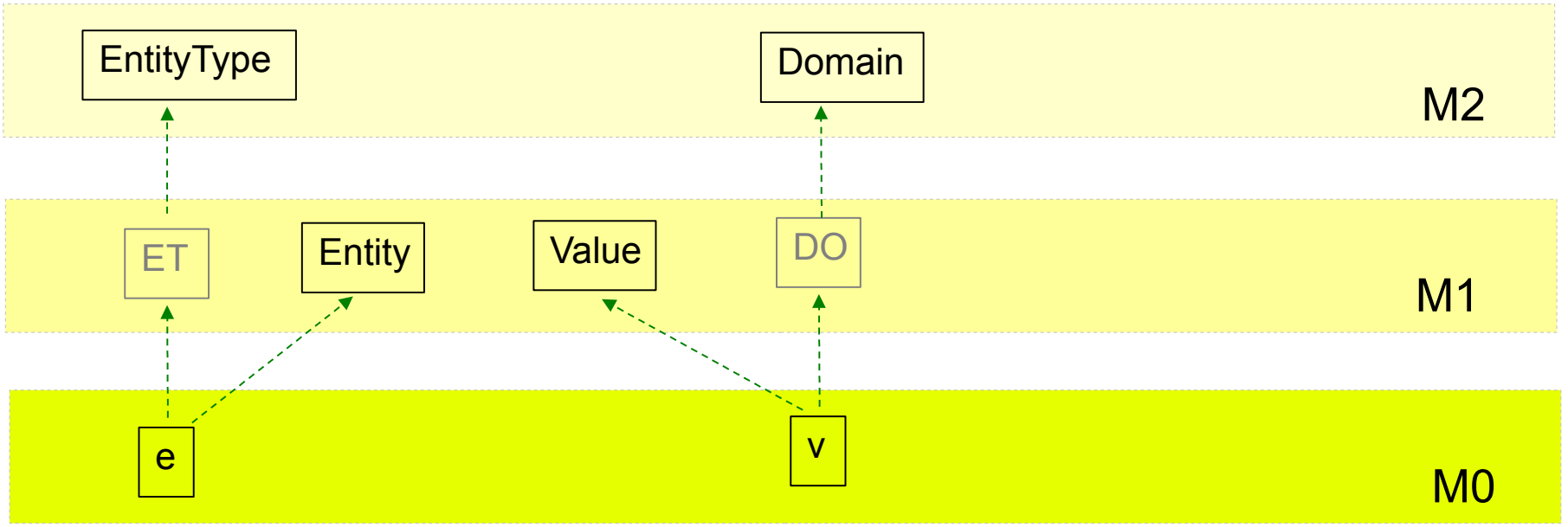
```
forall e,ET
   In(e,ET) and In(ET,EntityType)
 ==> In(e,Entity)
```

```
In2(e,EntityType)
```

"No entity type may have an empty interpretation."

```
forall ET In(ET,EntityType) ==>
   (exists e In(e,Entity) and (e in ET))
```

Distinguish entities from values (2)

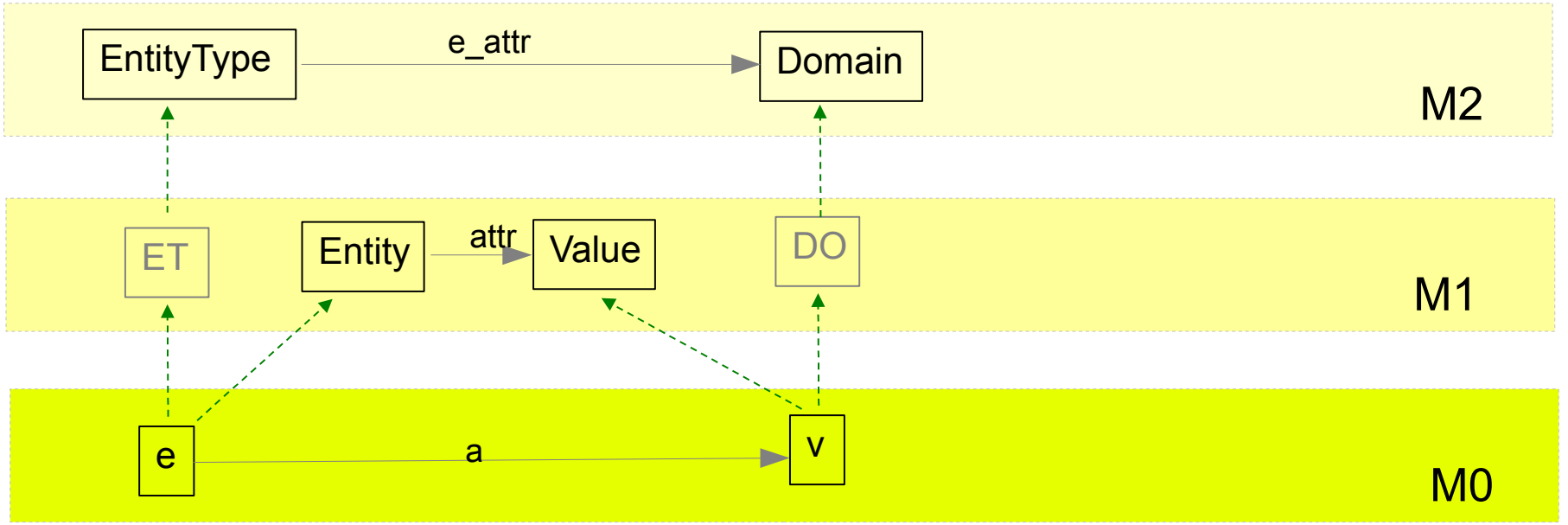


Make sure that entities and values are disjoint sets:

```
forall e In2(e,EntityType) ==> In(e,Entity)
forall v In2(v,Domain) ==> In(v,Value)
forall e In(e,Entity) ==> not In(e,Value)
forall v In(v,Value) ==> not In(v,Entity)
```

Mario Bunge: Treatise on Basic Philosophy. Vol 3, Ontology I: The Furniture of the World, Reidel, Boston, 1977.

Distinguish entities from values (3)



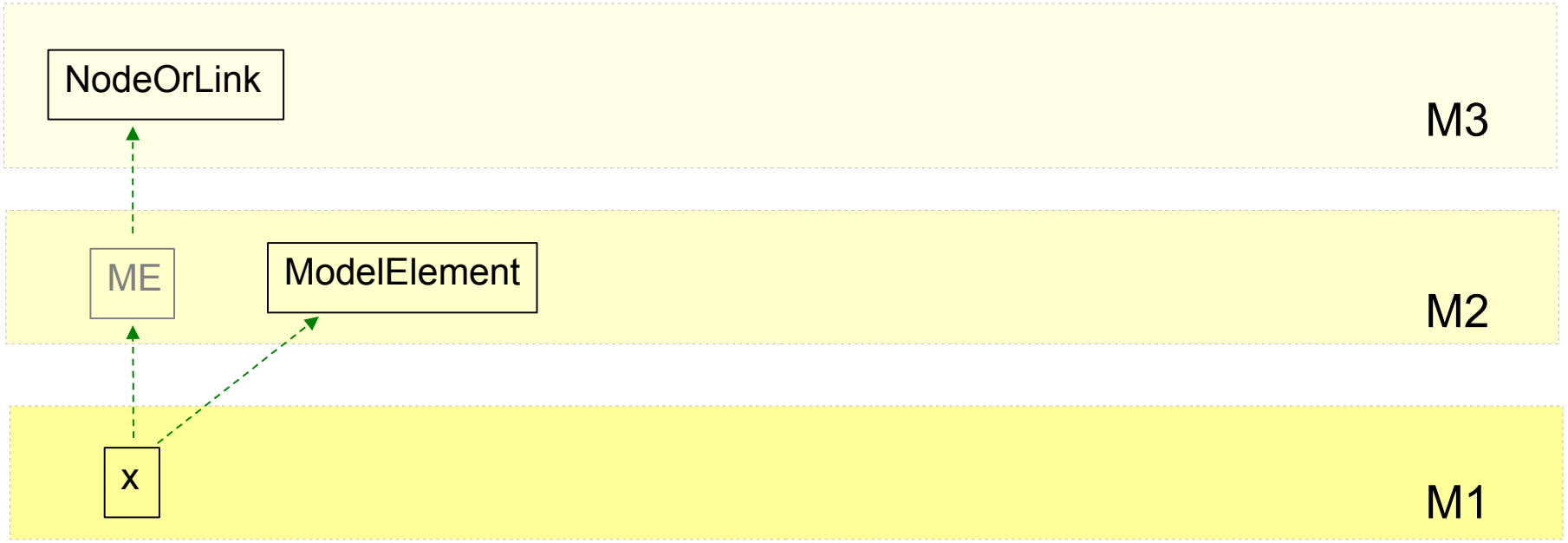
If we add the link 'attr' (attribute between Entity and Value), then we can use this schema for querying the data level M0 independent from the domain model (ERD).
Example query: give me all entities that share a value ...

A complete specification is available from
http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d2230805/MacroFormulas.sml.txt

Traceability (1)

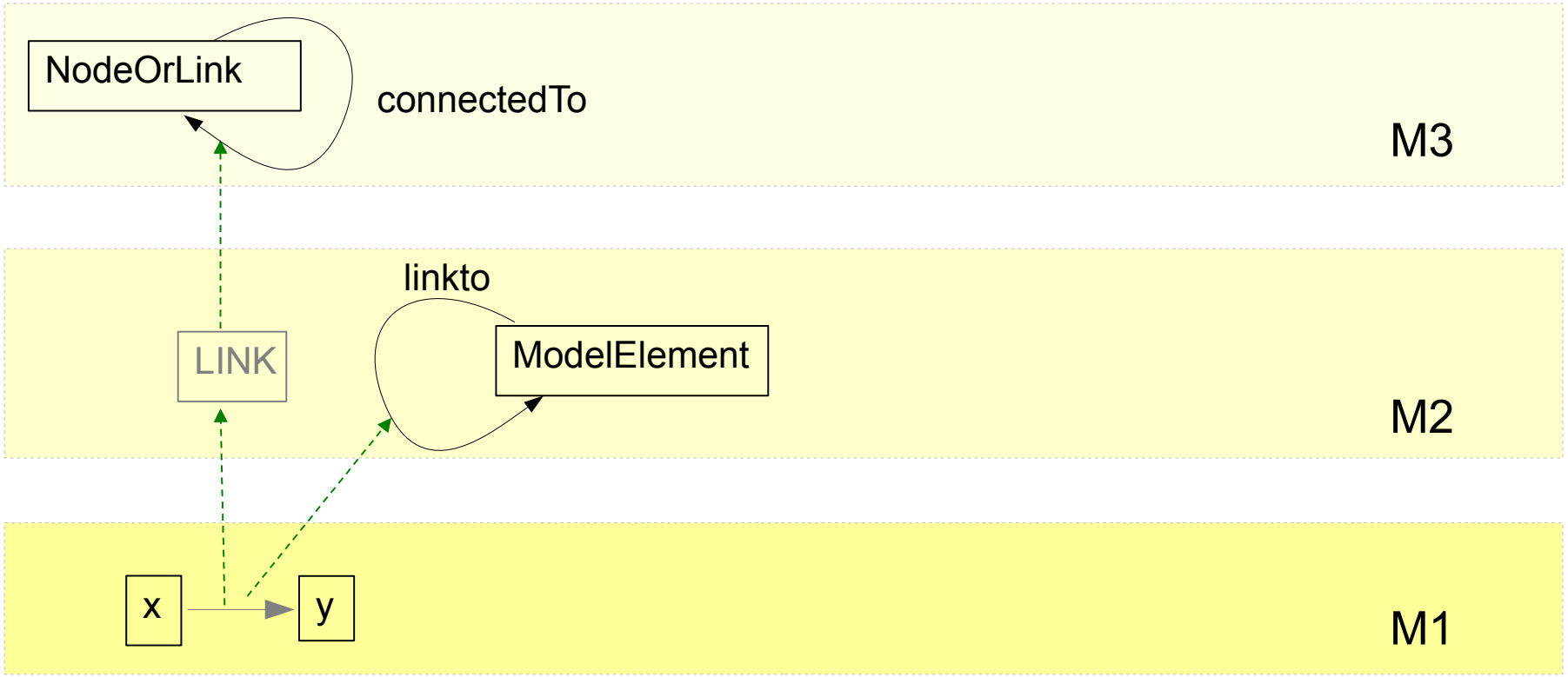
"A model element is simply anything at the ML."

```
forall x In2(x,NodeOrLink)  ==> In(x,ModelElement)
```



The class ModelElement is useful to define some general constructs such as linkage between model elements and then use that for traceability queries (see next slides).

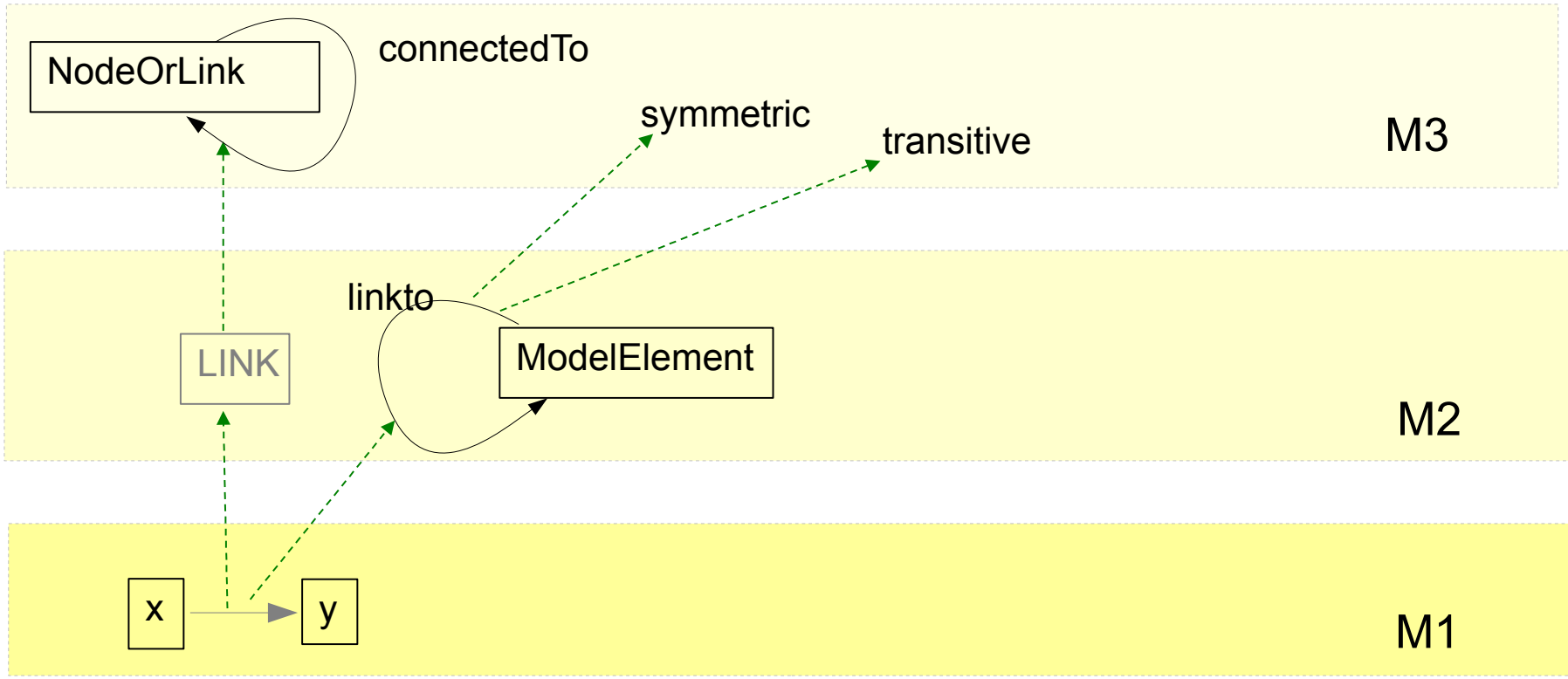
Traceability (2)




```
forall x
  In2(x,NodeLink!connectedTo) ==> In(x,ModelElement!linkto)
```

Traceability (3)

Connectedness can be checked in terms of linkto



This definition allows to formulate queries in terms of ModelElement (M2) and to check whether two model elements (M1) are linked to each other.

ConceptBase definitions for traceability

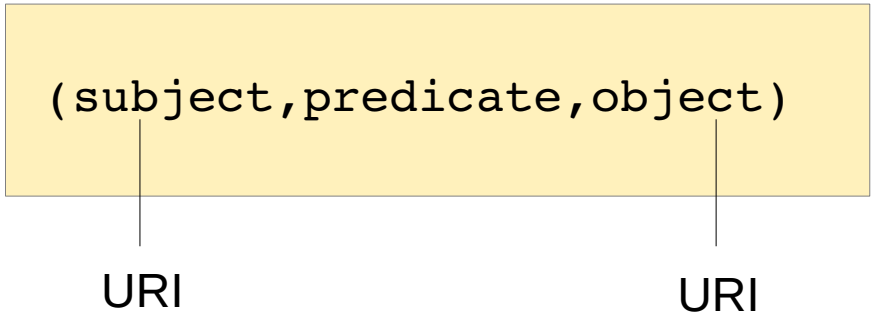
```
ModelElement in Class with
  symmetric,transitive
    linkto: ModelElement
  rule
    me1: $ forall x/VAR (x [in] NodeOrLink)  ==> (x in ModelElement) $;
    me2: $ forall link/VAR (link [in] NodeOrLink!connectedTo)
              ==> (link in ModelElement!linkto) $
end

UnconnectedModelElement in GenericQueryClass isA ModelElement with
  parameter
    start : ModelElement
  constraint
    uc : $ not (start linkto this) $
end
```

This query checks whether two model elements (M1) are linked to each other, regardless of the modeling notation at M2!

ConceptBase and the Resource Description Framework (RDF, RDFS)

RDF statements are represented as triples



Examples:

```
(Johannes_Kepler, wrote, Astronomia_Nova)

(mammal, type, resource)
(human, subClassOf, mammal)
(horse, subClassOf, mammal)

(book, type, resource)

(Johannes_Kepler, type, human)
```

Reification with RDF: regard the statement itself as a resource (object)

```
(Johannes_Kepler, wrote, Astronomia_Nova)
```

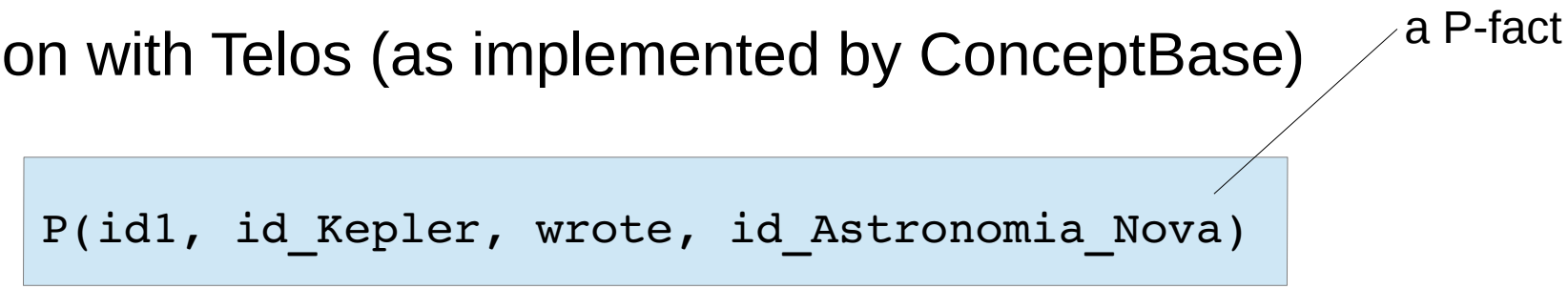
```
(statement1, type, statement)
(statement1, subject, Johannes_Kepler)
(statement1, predicate, wrote)
(statement1, object, Astronomia_Nova)

(Wikipedia, claims, statement1)
```

So, four triples instead just one!

What about reifying the reified statements?

Reification with Telos (as implemented by ConceptBase)




```
P(id2, id_Wikipedia, claims, id1)
```

- each P-fact quadruple has its own identity
- so, reification is a built-in feature
- even instantiation statements have their own identity

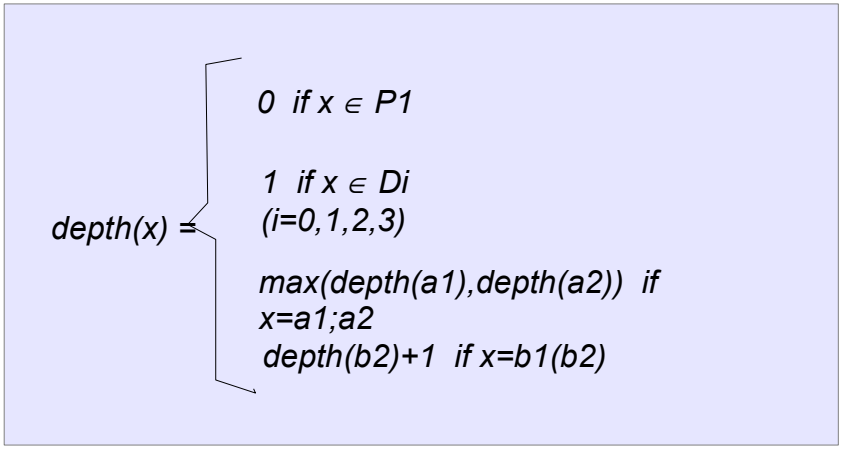
```
P(id3, id2, instanceOf, id_Utterance)
```

More on this: see paper by M. Wolpers on O-Telos-RDF

More features

Evaluating depth of nesting in ConceptBase

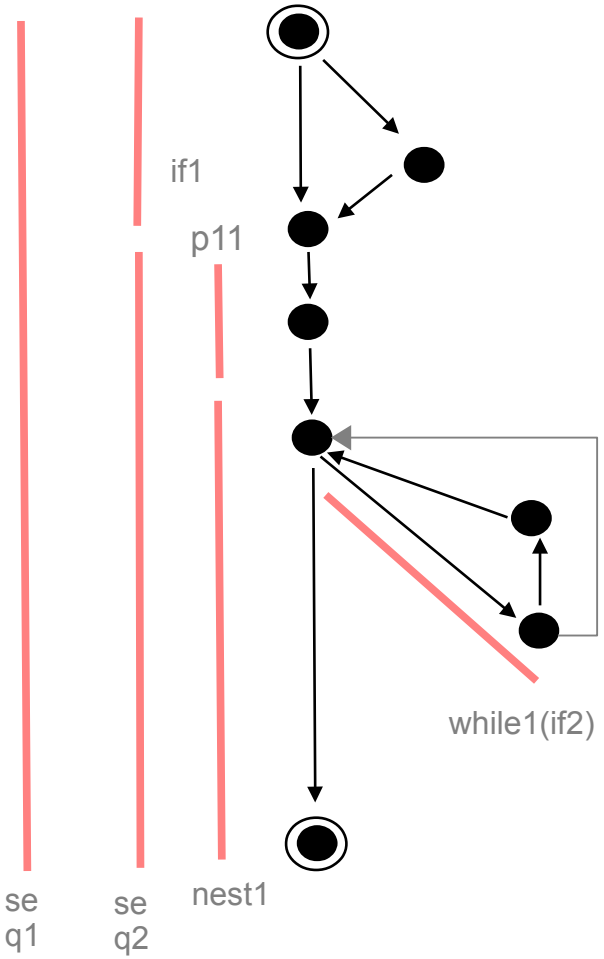
- depth-of-nesting is defined on the parse tree of a flowgraph, e.g. Seq(P1,Nest(D2,D1)). This parse tree is a bit difficult to create with just deductive query classes. But as soon as we have it, we can rather easily implement depth-of-nesting by recursive functions on ConceptBase

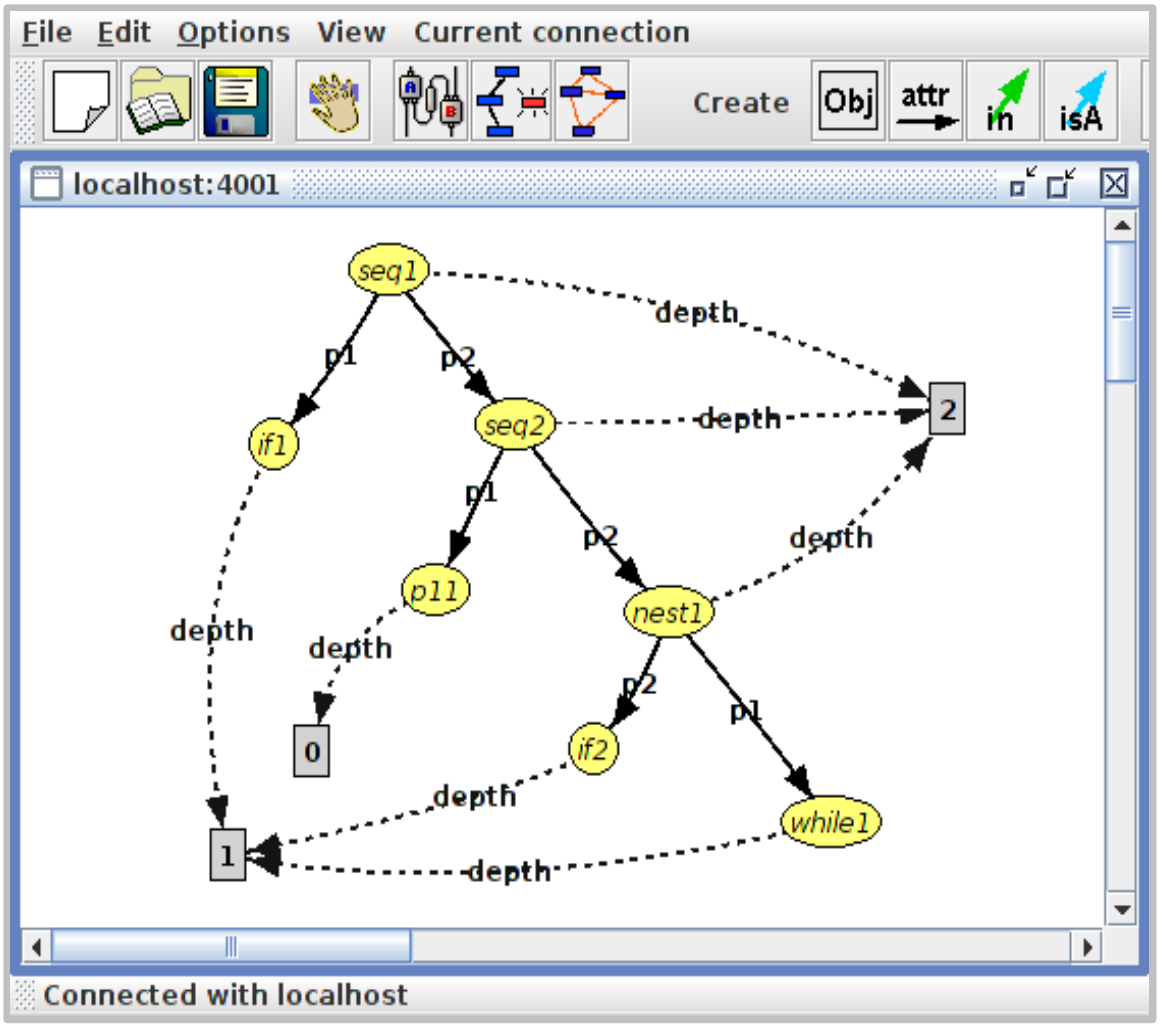



```
Function depth isA Integer with
  parameter
    x: Construct
  constraint
     cval: $ (x in P1) and (this = 0) or
             (x in D0) and (this = 1) or
             (x in D1) and (this = 1) or
             (x in D2) and (this = 1) or
             (x in D3) and (this = 1) or
             (exists a1,a2/Construct (x in Seq)
                       and (x part/p1 a1) and (x part/p2 a2)
                       and (this = maxf(depth(a1),depth(a2))) ) or
             (exists b2/Construct (x in Nest) and (x part/p2 b2)
                       and (this = depth(b2)+1) )
           $
end
```

Displaying depth of nesting in ConceptBase

example flowgraph





Complete example in:

http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d3060630/SeqNestMetric.sml.txt

... to summarize

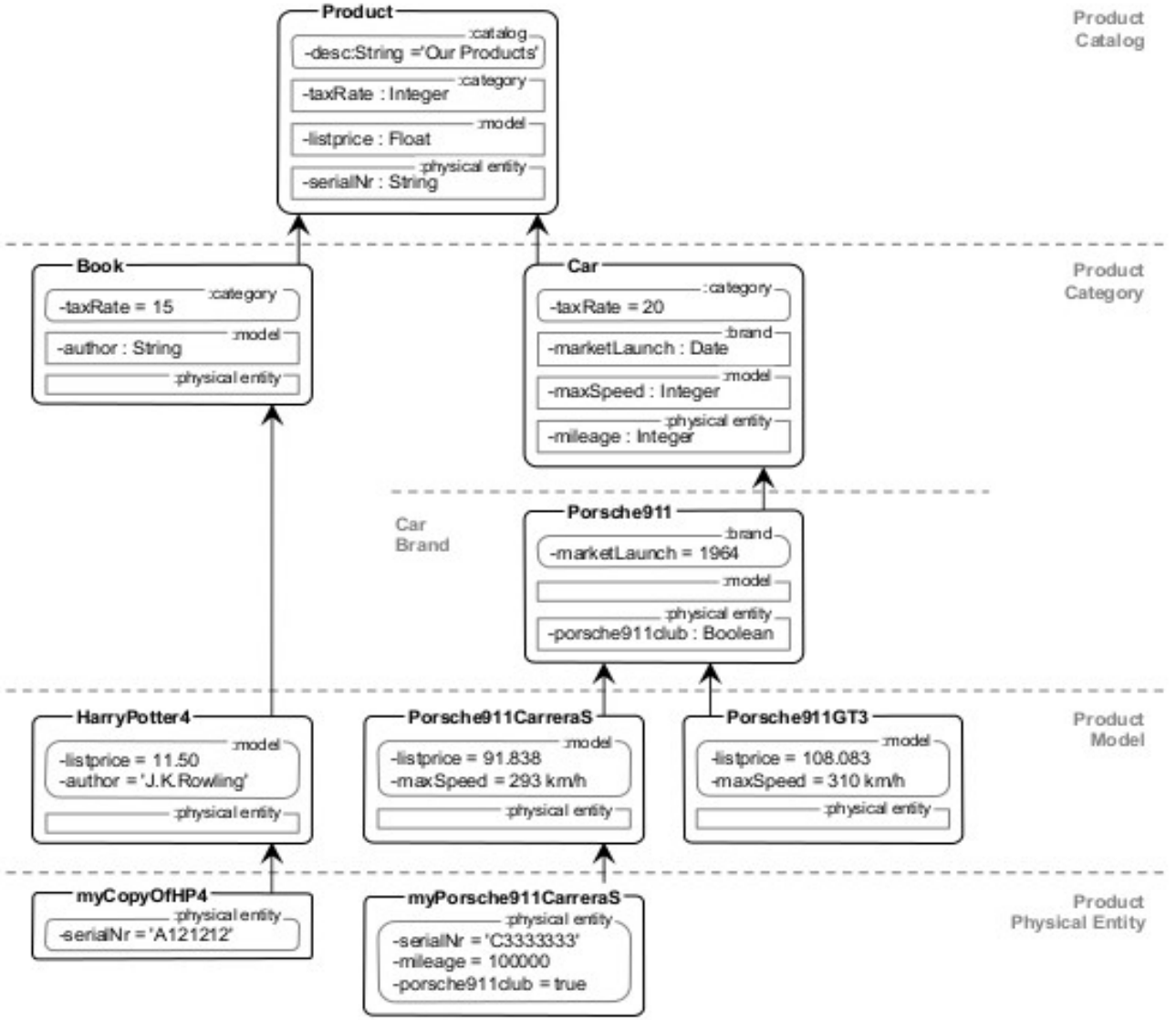
MOF vs. ConceptBase:

"Leave MOF levels out of the language!"

RDF vs. ConceptBase:

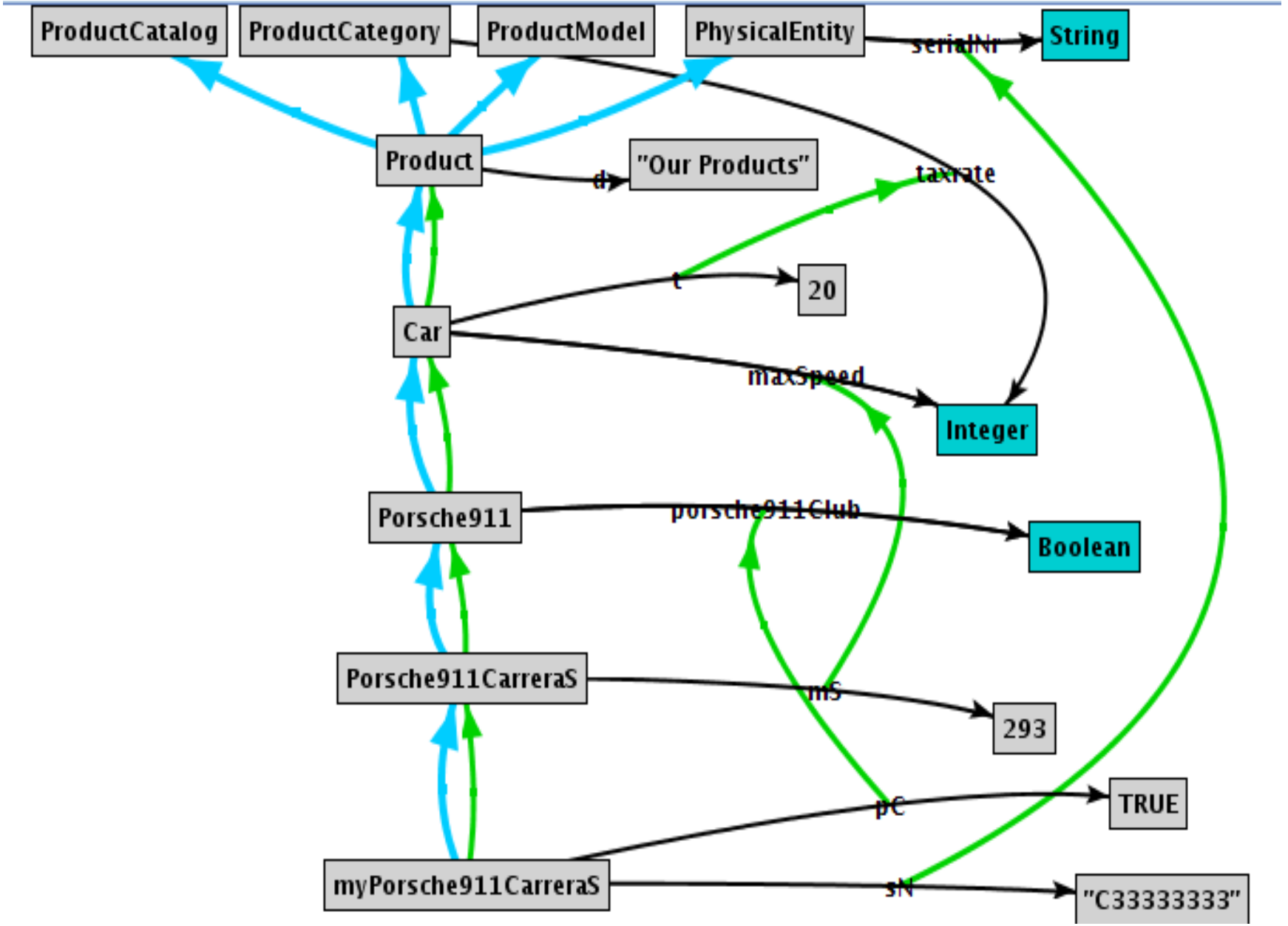
"Reify each statement!"

Multi-level modeling: example from Neumayr,Grün,Schrefl 2009



Hypothesis: Concretization is combination of instantiaion and specialization

Multi-level modeling: a first attempt with ConceptBase



To do list

- faster translation of object identifiers to memory addresses
- modules (=set of P-facts) with multiple super-modules
- from client-server to peer-to-peer
- Parallel Datalog engine

...

More on ...

http://conceptbase.cc

Some PhD thesis utilizing ConceptBase

Manfred Jeusfeld
Thomas Rose
Martin Staudt
Hans Nissen
Gerhard Steinke
Dominik Schmitz
Armin Eberlein
Christoph Quix
Stefan Eherer
Patrick Chen
Willem-Jan vd Heuvel
Kees Leune
Martin Wolpers
Quan C. Dang
Stephanie Kethers
Bettina v. Buol
Peter Szczurko
Klaus Pohl
Panos Vassiliadis
Mohamed Dahchour
Ralf Klamma
Birgit Bayer
Daniel Gross
Vinay K. Chaudhri
Günter Gans
Hadhami Dhraief
(J. Jayasinghe Arachchige)

Major contributions to the source code of ConceptBase were made by: Lutz Bauer (module system), Rainer Gallersdörfer (object store), Manfred Jeusfeld (CB server, logic foundation, function component), Eva Krüger (integrity component), Thomas List (object store), **Hans Nissen** (module system), **Christoph Quix** (CB server, view component), Christoph Radig (object store), René Soiron (optimizer), **Martin Staudt** (CB server, query component), Kai von Thadden (query component), and Hua Wang (answer formatting).

Additional contributions came from Masoud Asady, Markus Baumeister, Ulrich Bonn, Stefan Eherer, **Michael Gebhardt**, Dagmar Genenger, Michael Gocek, Rainer Hermanns, David Kensche, André Klemann, Rainer Langohr, Tobias Latzke, Xiang Li, Yong Li, Farshad Lashgari, Andreas Miethsam, Martin Pöschmann, Achim Schlosser, Tobias Schöneberg, Claudia Welter, Thomas Wenig, and others.

Thank you, folks!