Abstract vs Concrete Clabjects in Dual Deep Instantiation

Bernd Neumayr, Michael Schrefl
Johannes Kepler University Linz, Austria

MULTI 2014

Agenda

- Dual Deep Instantiation

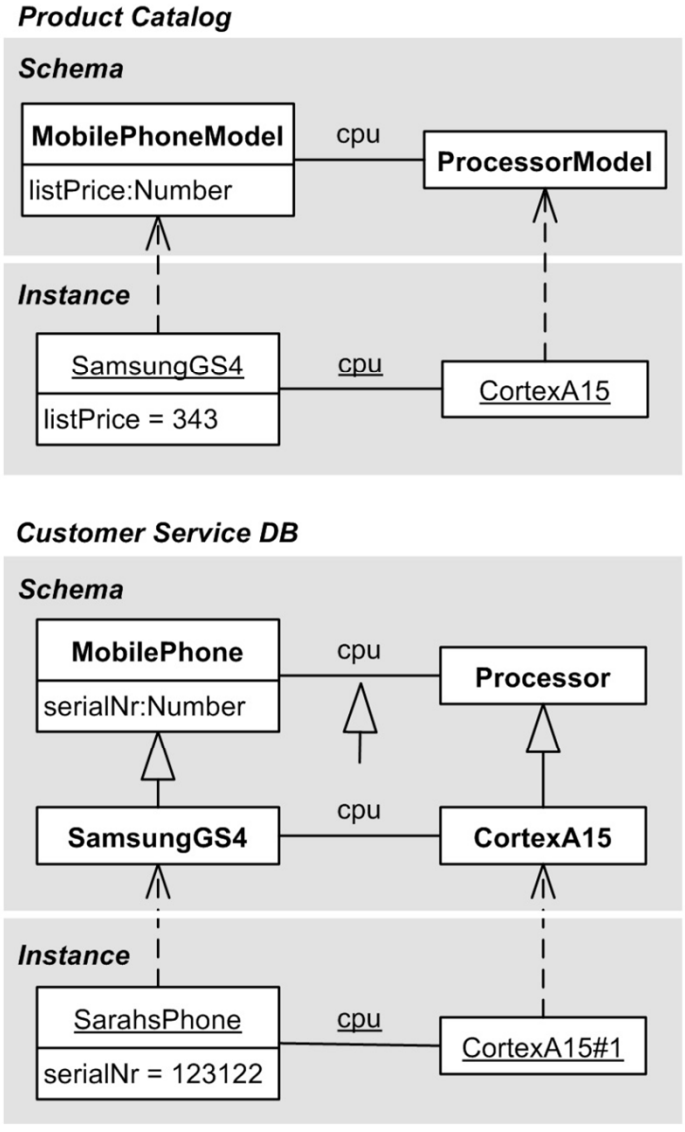
  Bernd Neumayr, Manfred A. Jeusfeld, Michael Schrefl, Christoph Schütz: Dual Deep Instantiation and Its ConceptBase Implementation. CAiSE 2014:503-517

  Research Goal: A modeling language, that allows for a ***compact and integrated representation of objects and relationships that play multiple roles at different levels of abstraction.*** Constraints should as far as possible be represented implicitly by the semantics of the language constructs without the need to take recourse to explicit constraints

- Abstract vs Concrete Clabjects in Dual Deep Instantiation

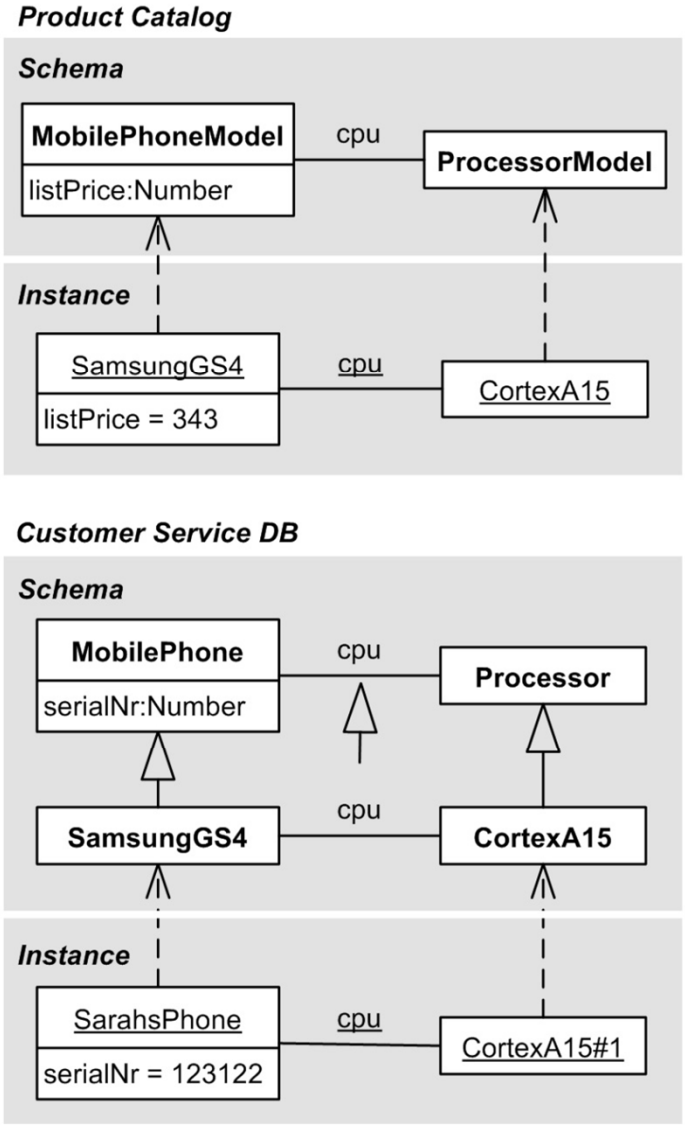
  Research Goal: Clarify the ***exact nature of superclabjects in generalization hierarchies***. Provide foundations for ***counting*** objects at different levels of abstractions for mandatory constraints and for multiplicity constraints

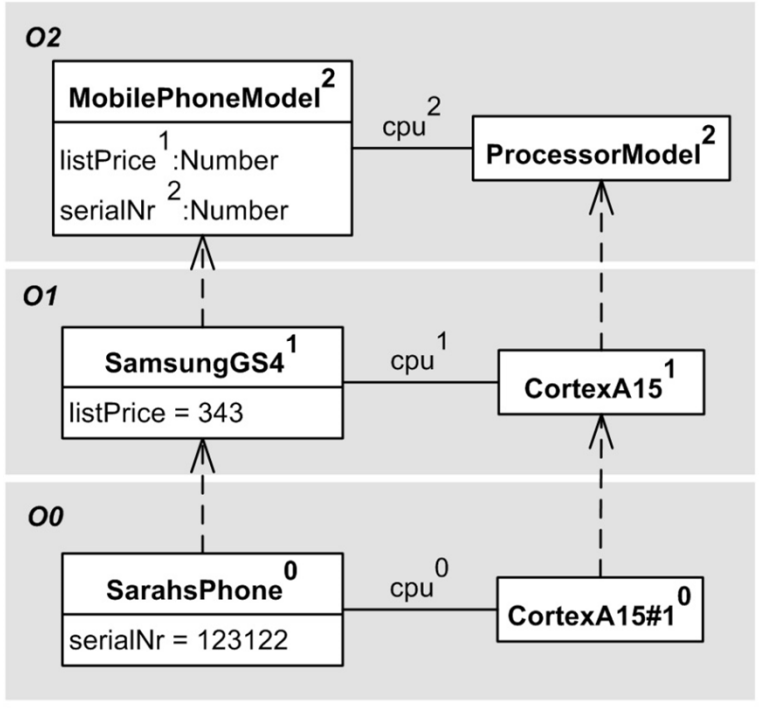
Starting Point:



What is represented at the instance level in one model, may be represented at the schema level in another model.

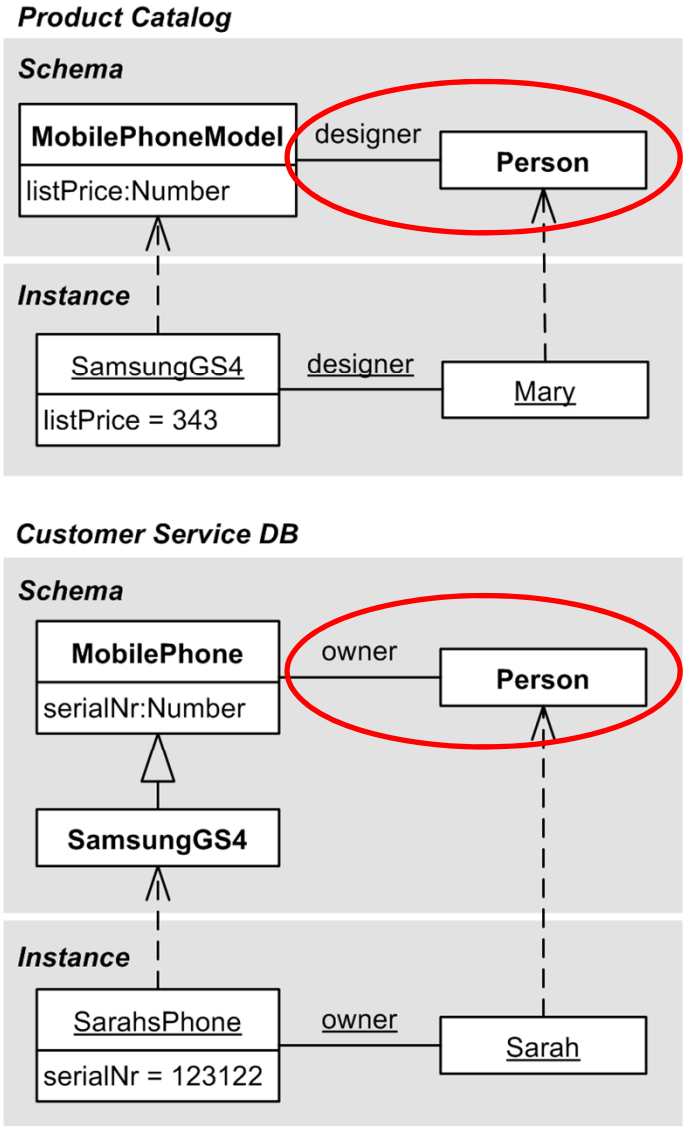
Starting Point: Deep Instantiation





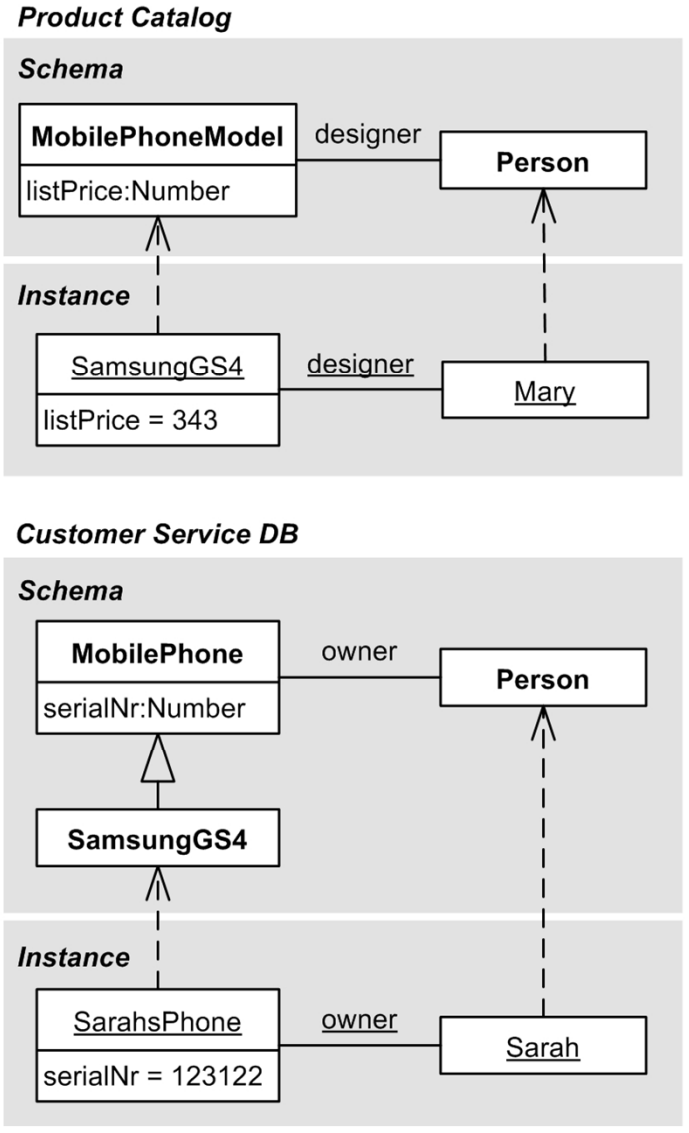
A multi-level model integrates these different levels of abstraction.

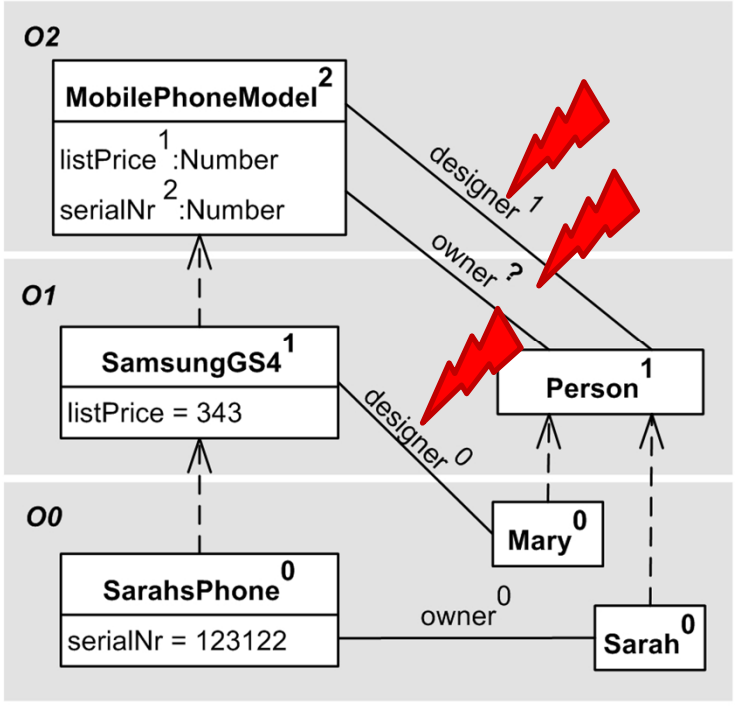
Starting Point: Deep Instantiation Limitations/Restrictions



Objects at different abstraction levels
may be related to
objects at the same abstraction level.

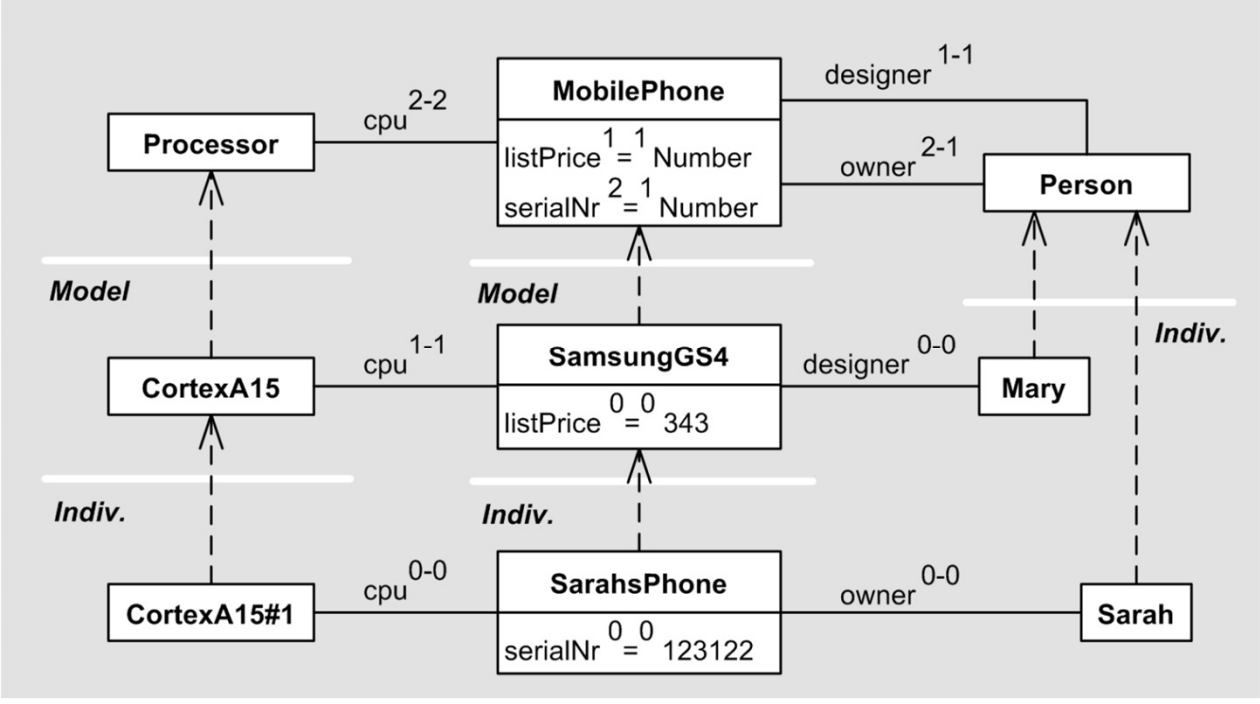
Starting Point: Deep Instantiation Limitations/Restrictions





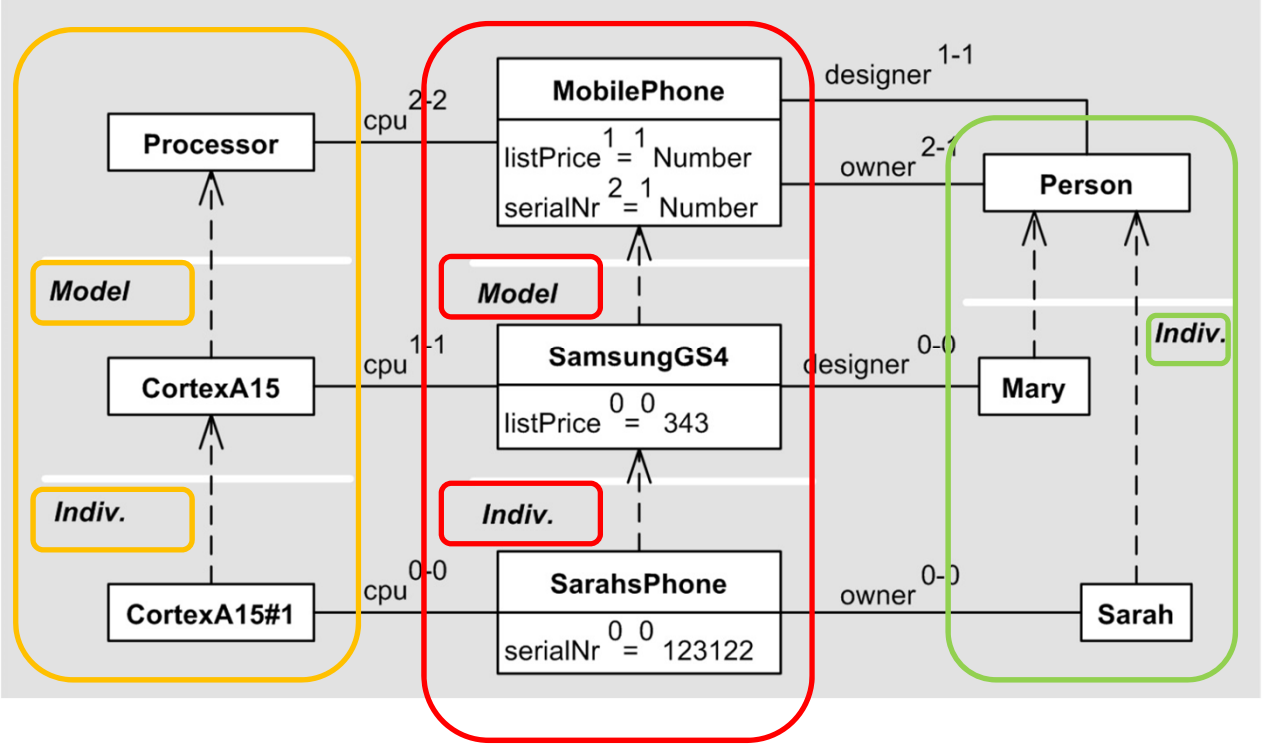
- With deep instantiation it is not possible to separately express the number of possible instantiation steps for the two ends of the relationship.
- relationships may not cross level boundaries (strict metamodeling)

Dual Deep Instantiation (DDI)



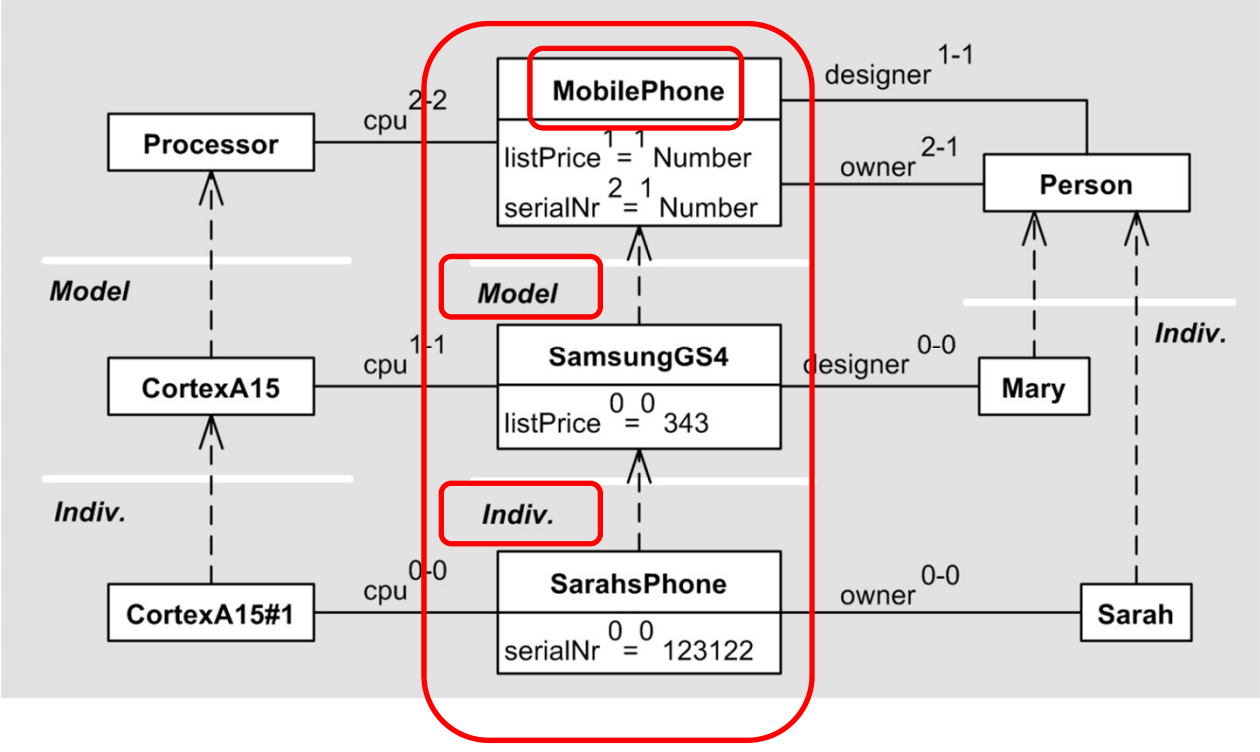
Uniform treatment of individual objects and classes as *objects* (= clabjects).

Dual Deep Instantiation (DDI)



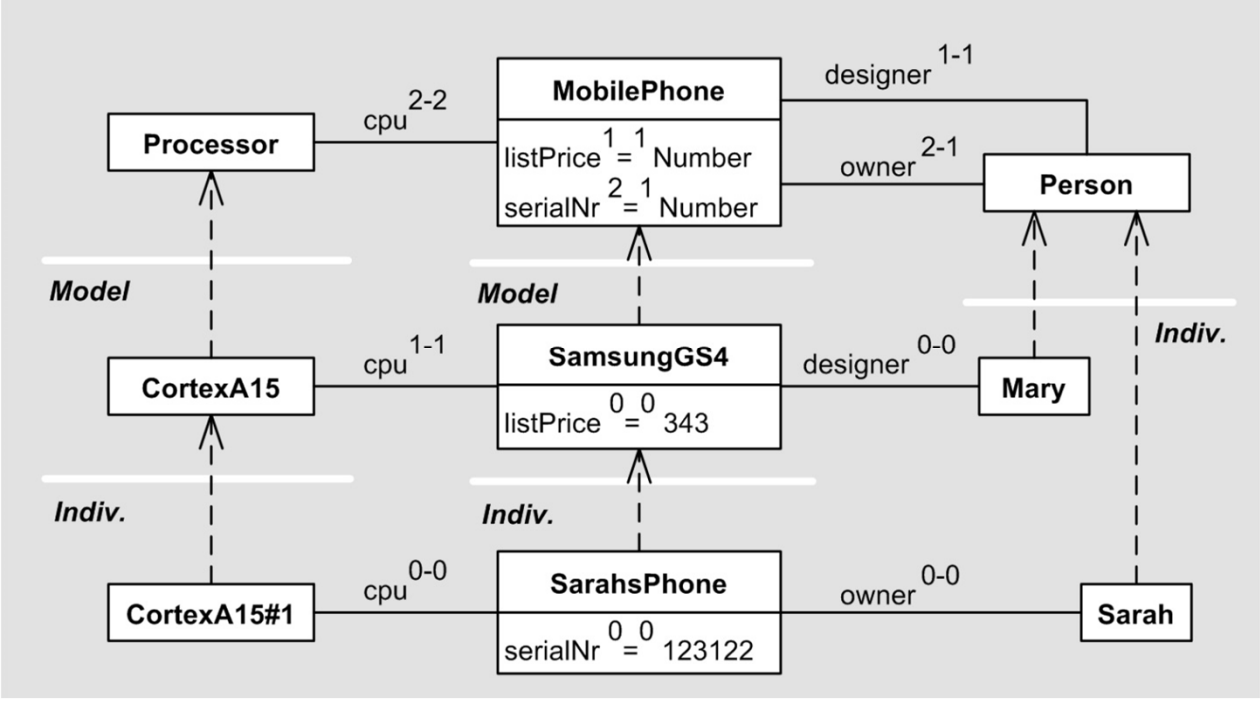
Instantiation levels are local to object hierarchies.

Dual Deep Instantiation (DDI)



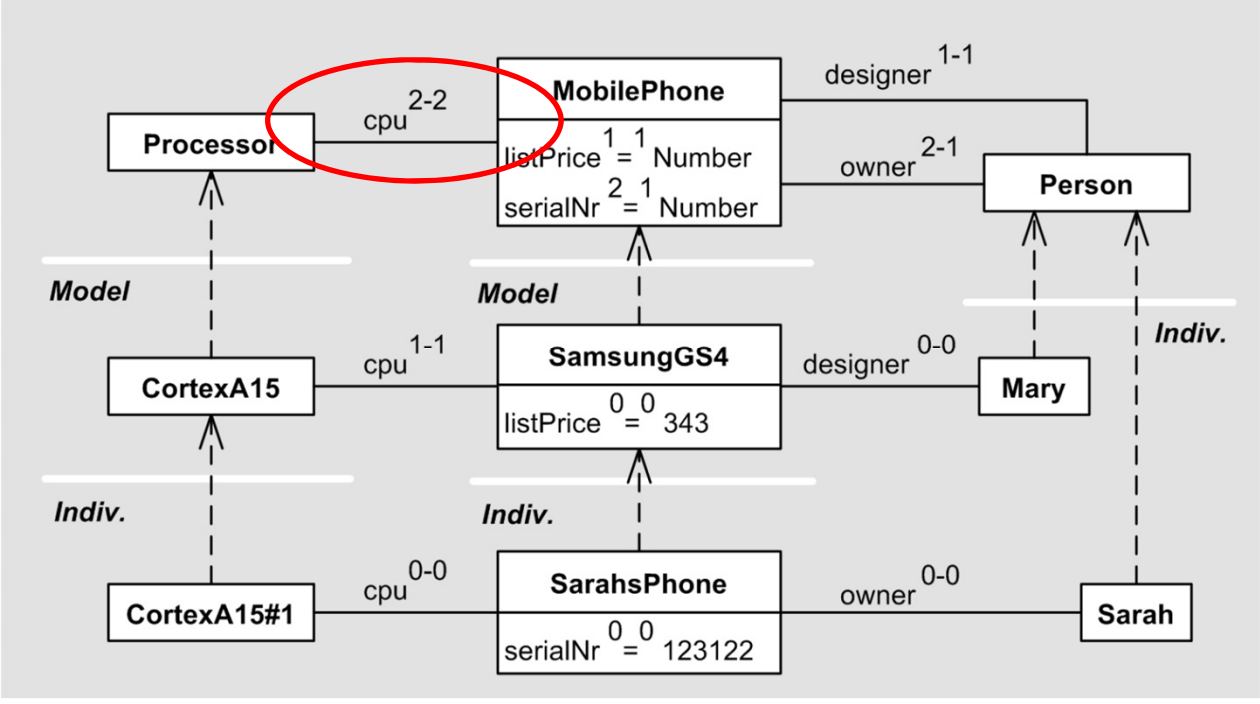
Object names are to be read together with names of instantiation levels.
“SamsungGS4 is a MobilePhone Model”
“SarahsPhone is a MobilePhone Individual”

Dual Deep Instantiation (DDI)



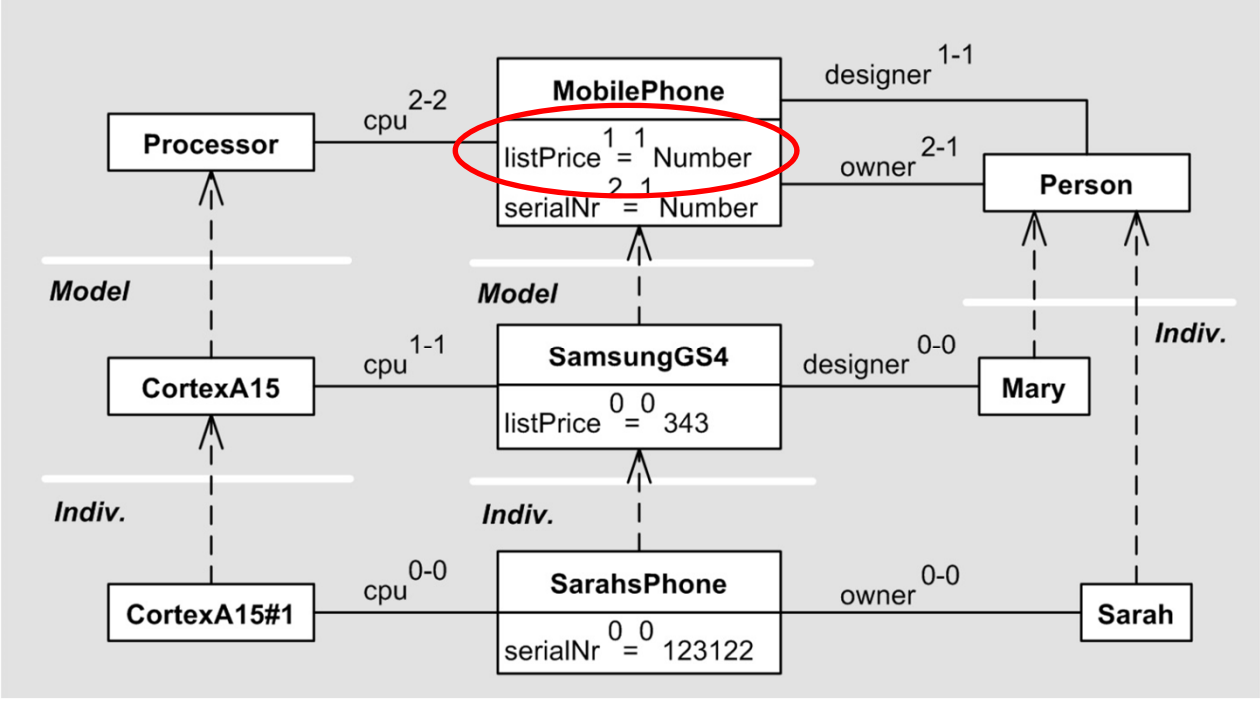
Uniform treatment of attributes and associations/links as bidirectional, binary relationships ...

Dual Deep Instantiation (DDI)



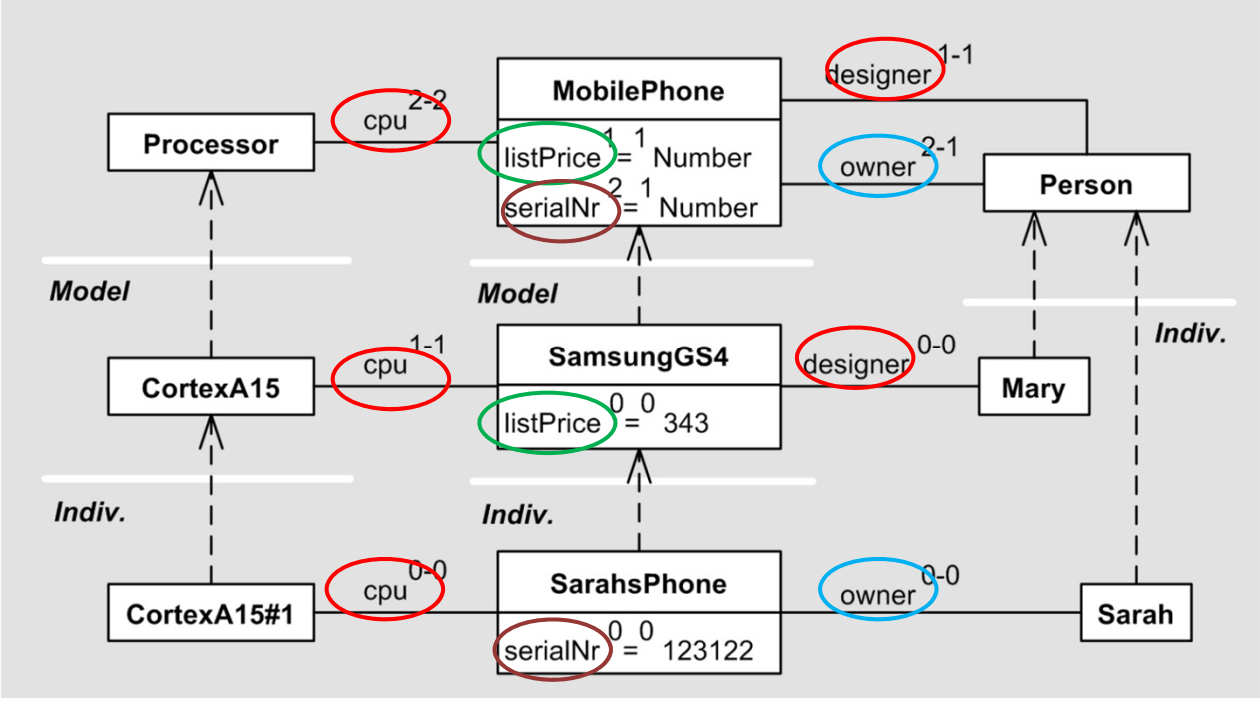
... with a *relationship-notation* in case source and target object of a relationship are part of the model ...

Dual Deep Instantiation (DDI)



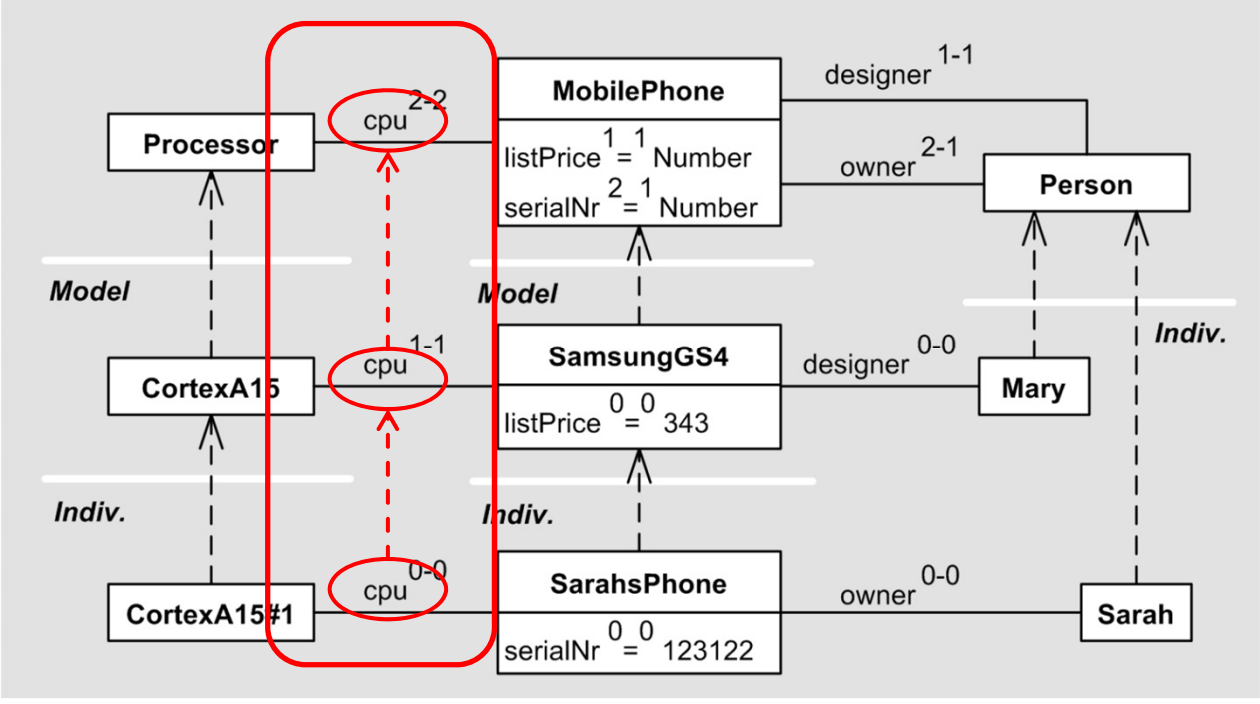
... and an *attribute-notation* for in-line representation of relationships.

Dual Deep Instantiation (DDI)



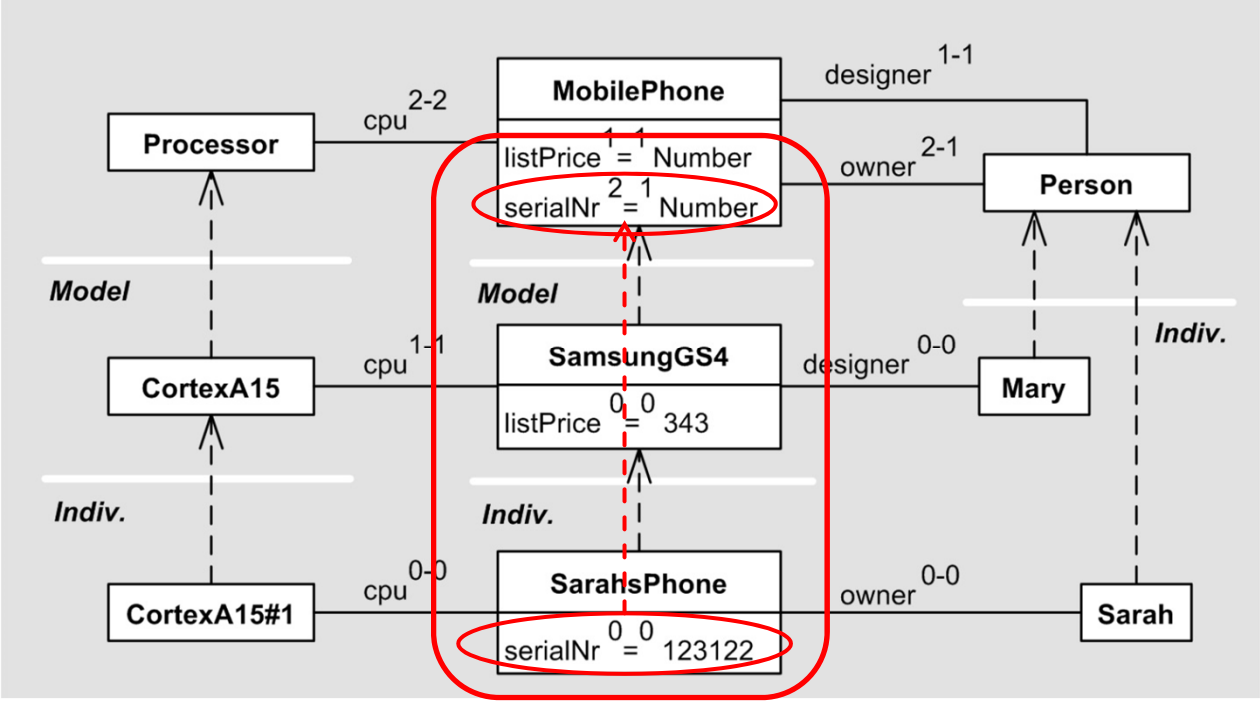
Every relationship has a relationship label.

Dual Deep Instantiation (DDI)



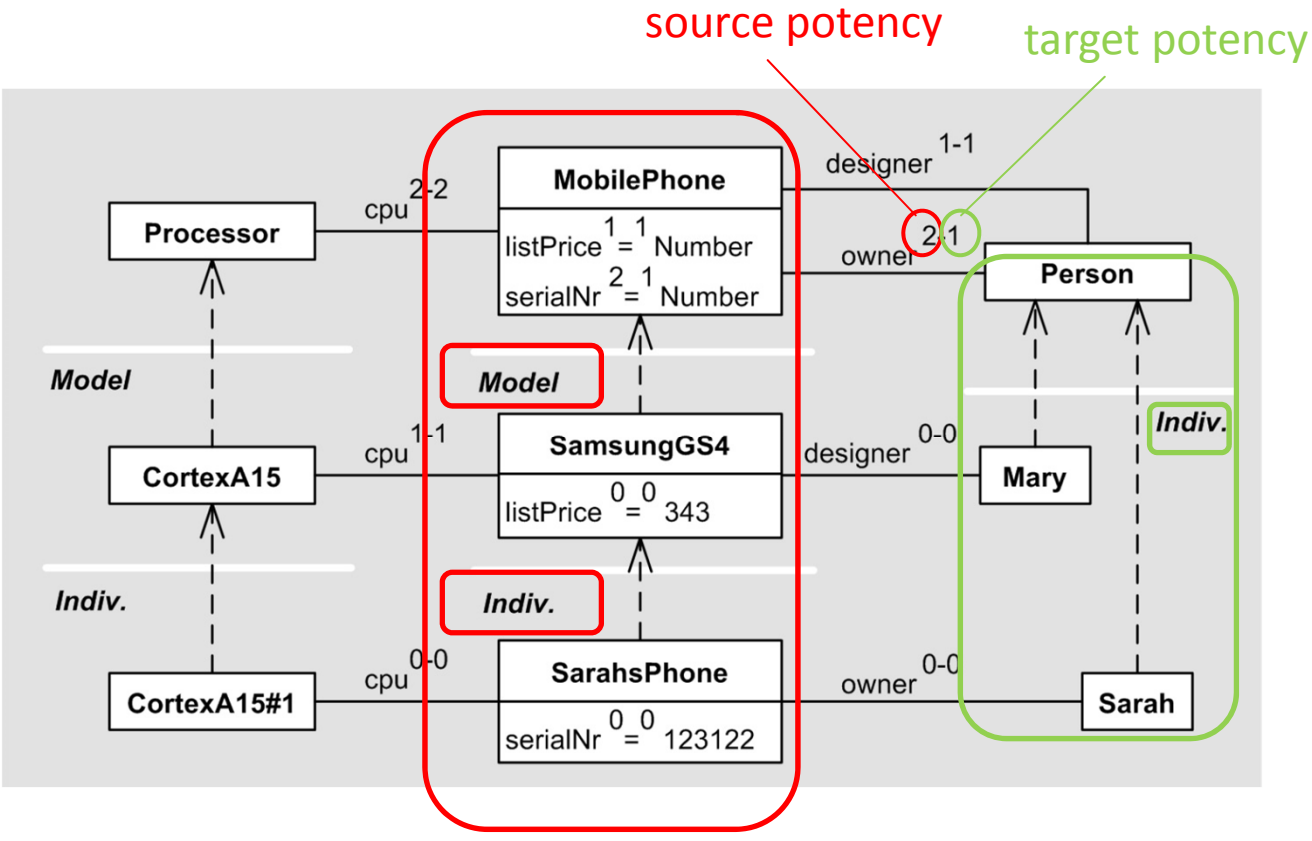
Relationship instantiation hierarchies are identified by the relationship label. Instantiation between relationships is not explicitly represented but derived from the object hierarchies.

Dual Deep Instantiation (DDI)



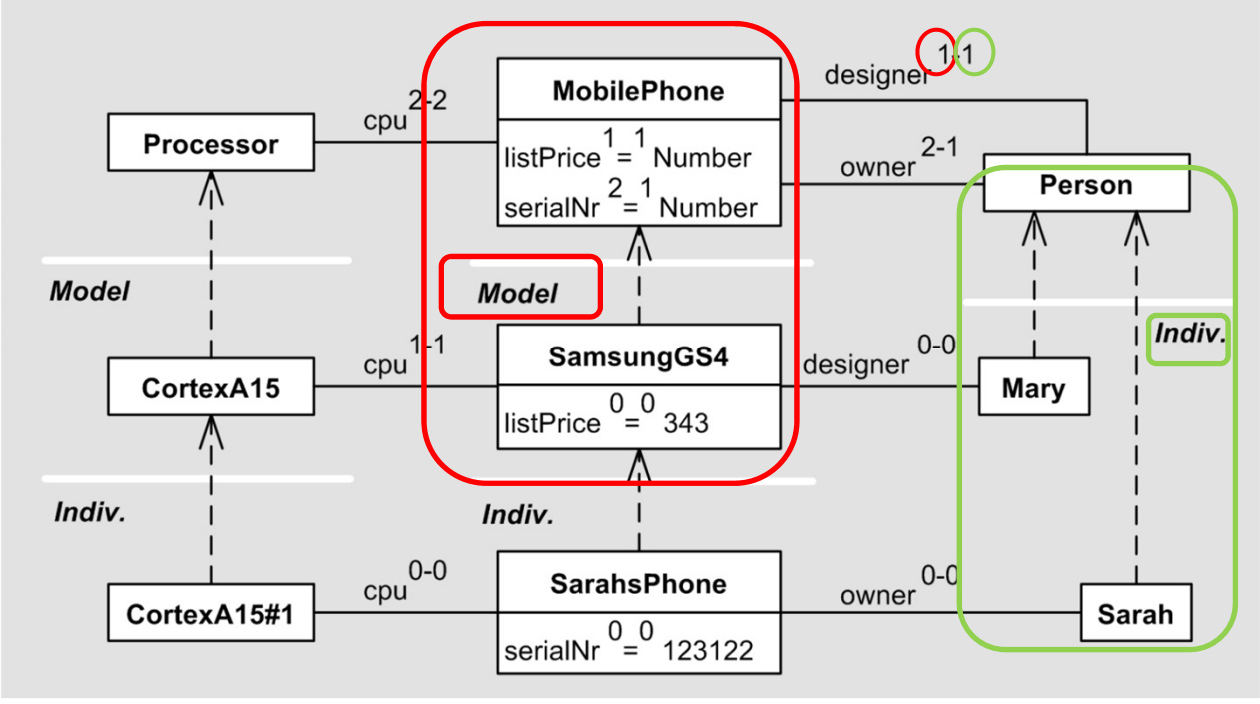
Relationship instantiation hierarchies are identified by the relationship label. Instantiation between relationships is not explicitly represented but derived from the object hierarchies.

Dual Deep Instantiation (DDI)



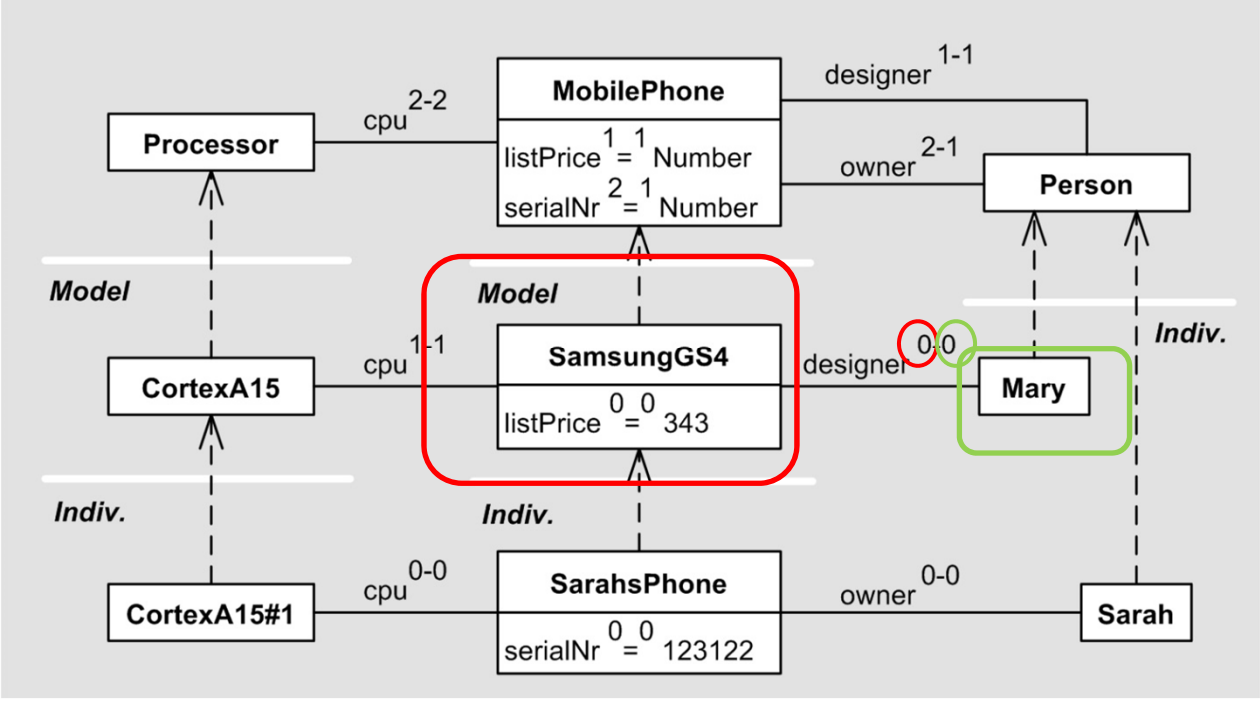
Every relationship has a source potency and a target potency to seperately indicate the depth of characterization for both ends of the relationship.

Dual Deep Instantiation (DDI)



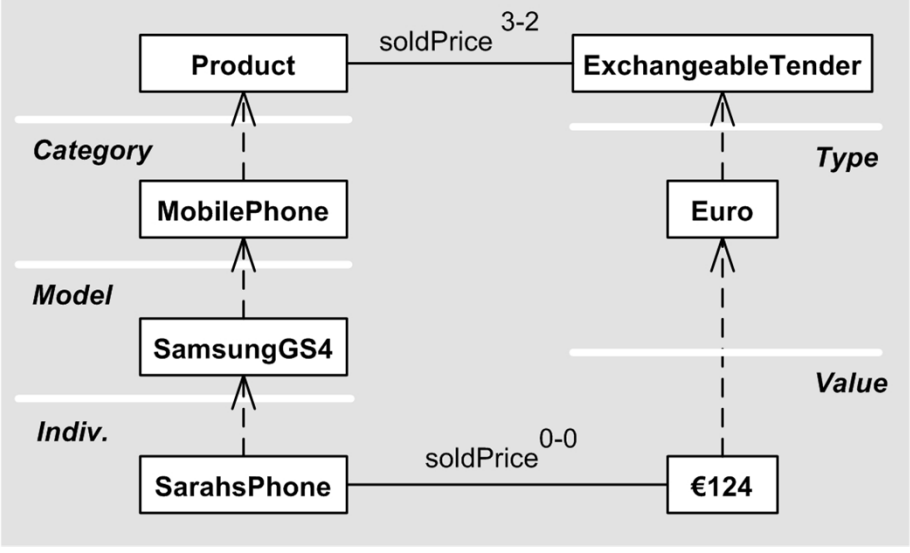
Every relationship has a source potency and a target potency to seperately indicate the depth of characterization for both ends of the relationship.

Dual Deep Instantiation (DDI)



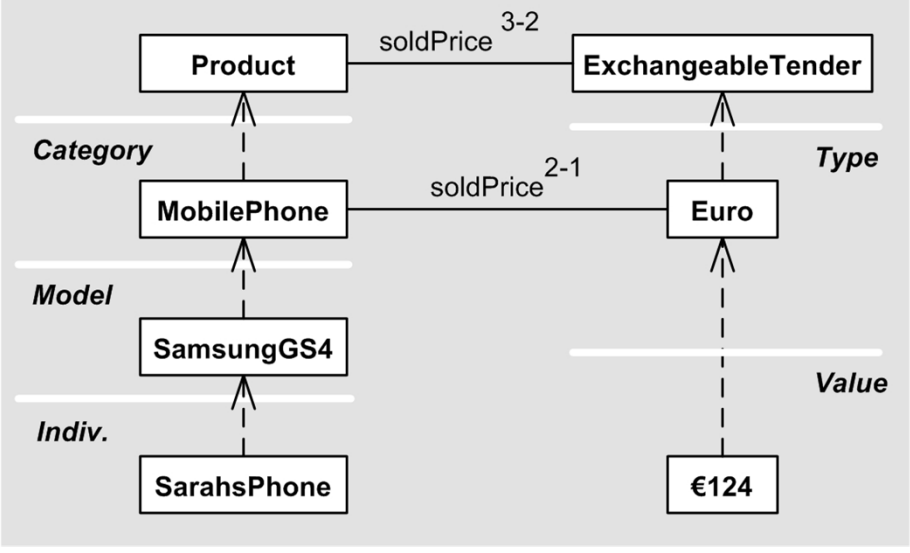
Every relationship has a source potency and a target potency to seperately indicate the depth of characterization for both ends of the relationship.

Dual Deep Instantiation (DDI)
Skip Levels



Relationships may skip instantiation levels.

Dual Deep Instantiation (DDI)
Referential Integrity Constraints

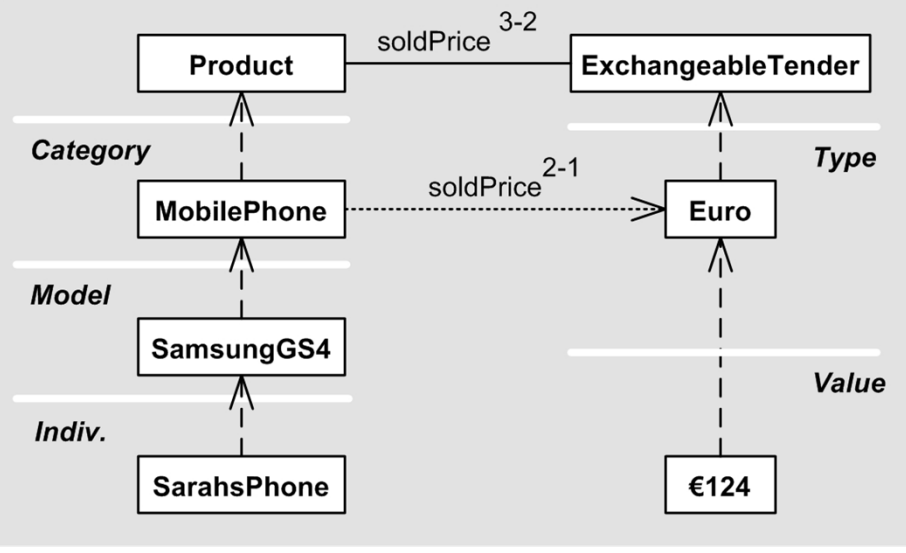


Relationships impose referential integrity constraints on both roles

„mobile phones are paid only in Euro“

„only mobile phones are paid in Euro“

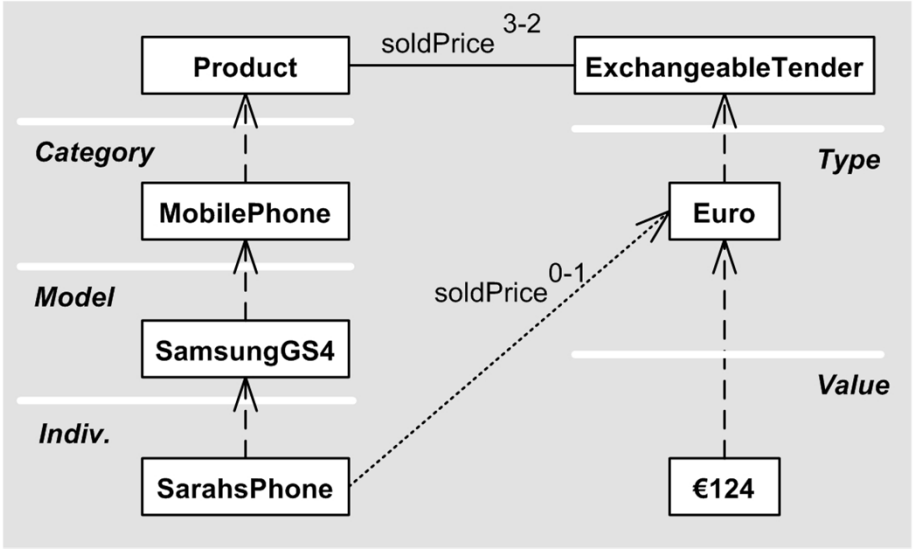
Dual Deep Instantiation (DDI)
Referential Integrity Constraints



Directed relationships impose referential integrity constraints only on one role:

„mobile phones are paid only in Euro“

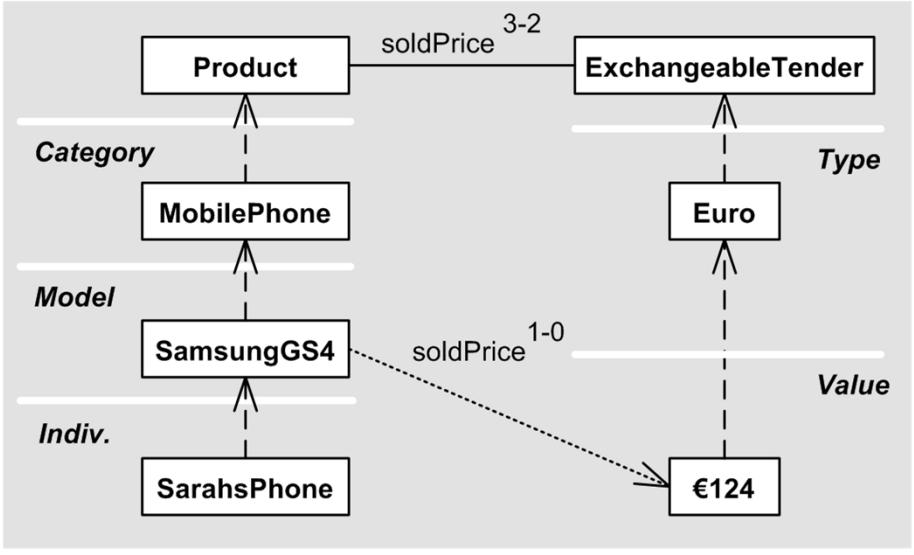
Dual Deep Instantiation (DDI)
Self-constraining Objects



Objects may constrain themselves:

„Sarah‘s phone is paid in Euro“

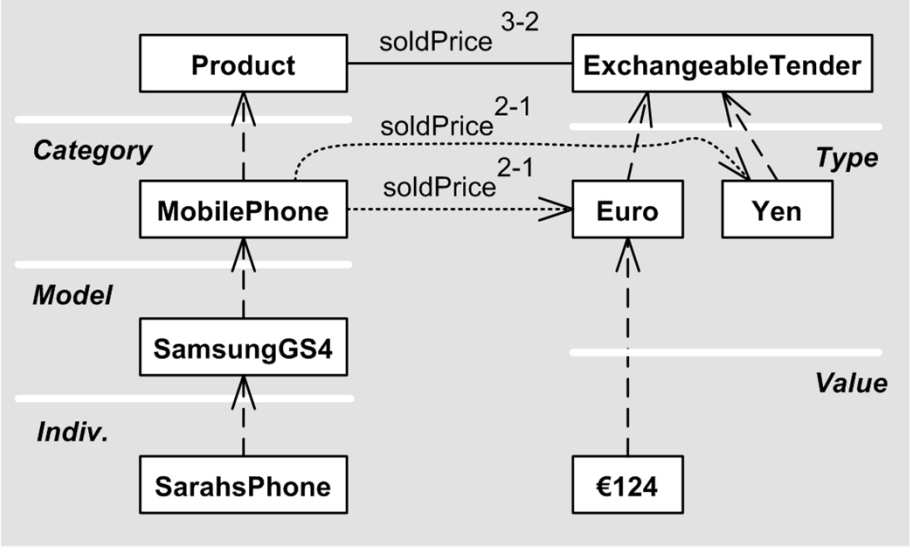
Dual Deep Instantiation (DDI)
„Shared Values“



Shared values (may be refined in a co-variant manner):

„If a SamsungGS4 individual is sold, then the price is €124“

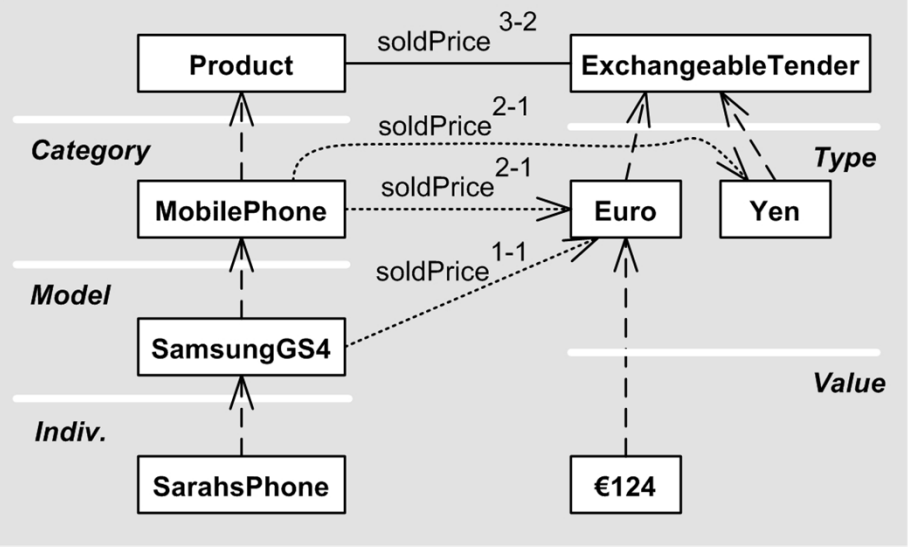
Dual Deep Instantiation (DDI)
Multiple Values



Multiple Values:

„Mobile phones are paid in Euro or Yen ...“

Dual Deep Instantiation (DDI)
Multiple Values



Multiple Values:

„ ... but Samsung GS4 individuals are paid only in Euro“

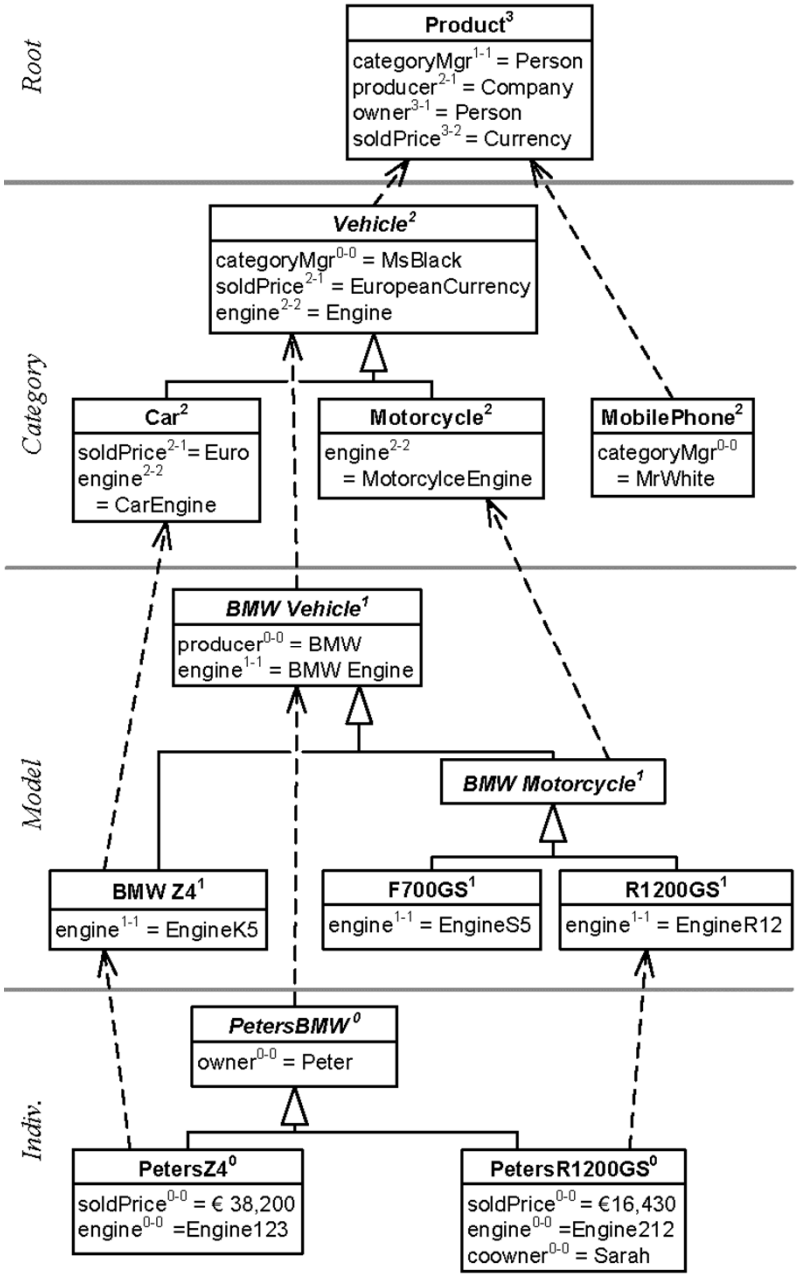
In the CAiSE paper you further find

- **Formalization** of the DDI approach
- A rather short analysis of how to represent DDI models in UML using the powertype pattern and (many) explicit OCL constraints
- Restrictions of the approach
- Extending DDI with Generalization
- Description of the ConceptBase implementation

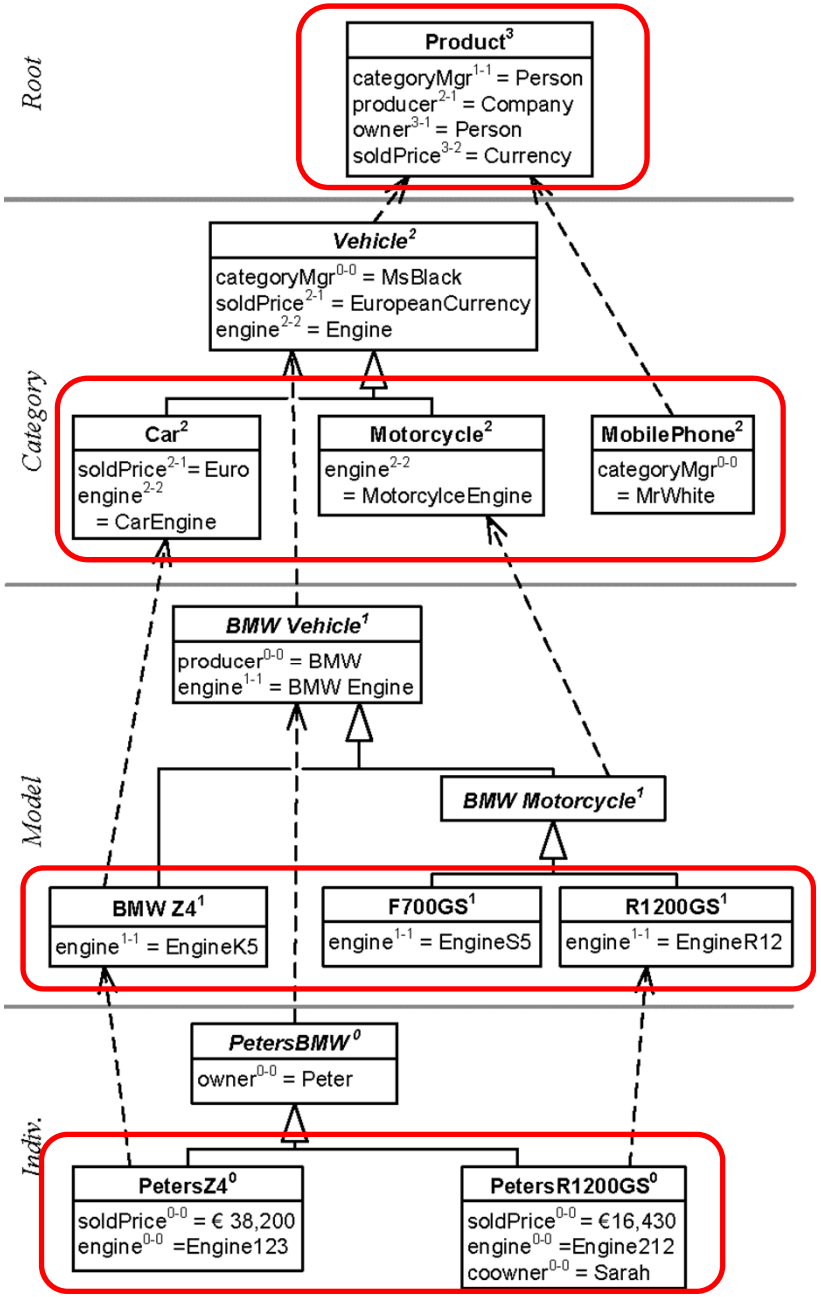
Abstract vs Concrete Clabjects in DDI (MULTI Paper)

- Simplified Setting:
  - only uni-directional, single-valued relationships
- Contributions:
  - Clarification of Abstract vs Concrete Clabjects
  - Abstract Superclabject Rule (relaxation of the ‘abstract superclass rule’)
  - Multilevel Mandatory Constraints (see paper)
  - Inheritance Mechanisme for Dual Inheritance (see paper)
  - Formalization of the simplified setting and extensions (see paper)

Abstract vs Concrete Clabjects



Abstract vs Concrete Clabjects



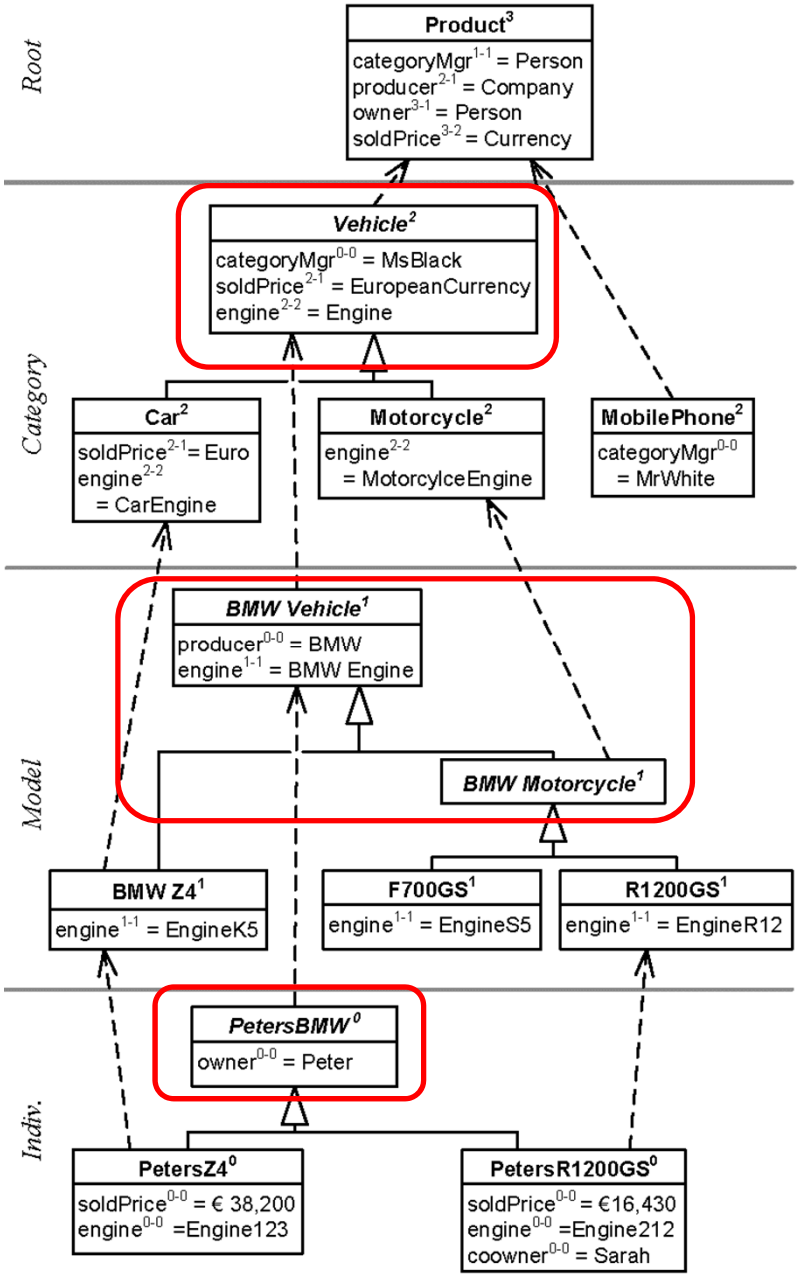
Concrete clabjects

combine aspects of concrete classes and concrete objects.

Only concrete clabjects are treated as proper objects.

„A class that has the ability to create instances is referred to as instantiable or concrete" (Hürsch, 1993)

Abstract vs Concrete Clabjects



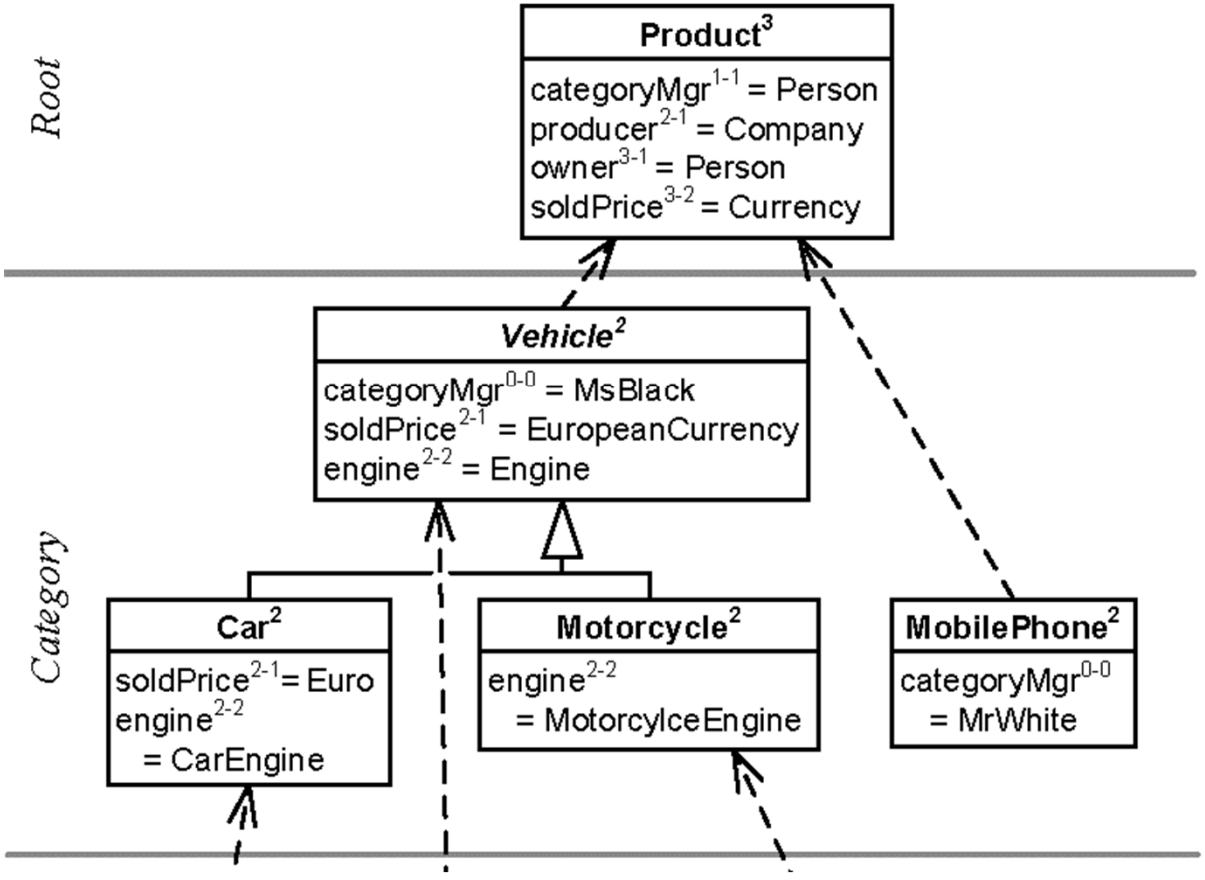
Abstract clabjects

combine aspects of abstract classes and abstract objects.

Abstract classes are classes that may not be instantiated directly.

„An abstract object captures
what is universal about a set of instances
but resides at the same logical level
as the instances“ (Kühne, 2009)

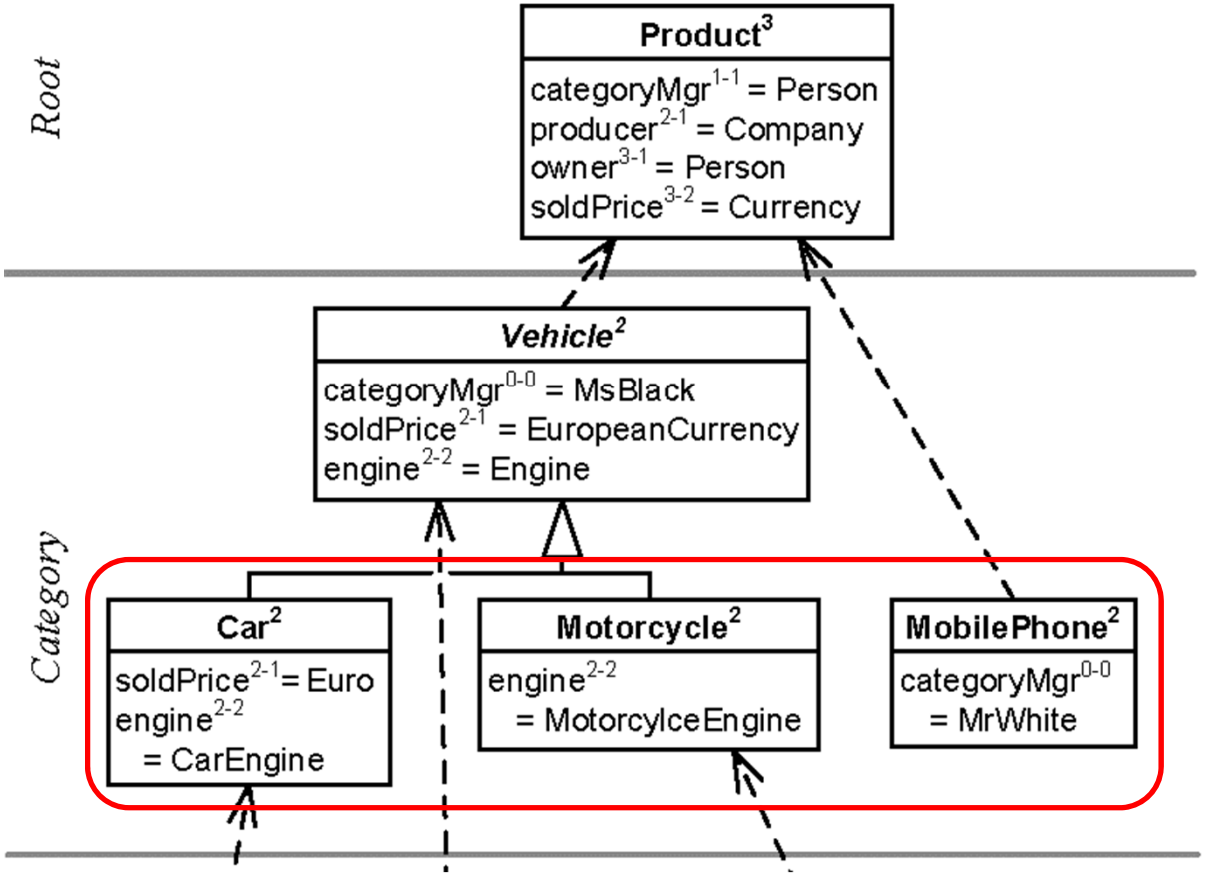
Abstract vs Concrete Clabjects



Counting clabjects (foundation for multiplicity constraints):

Number of product categories?

Abstract vs Concrete Clabjects

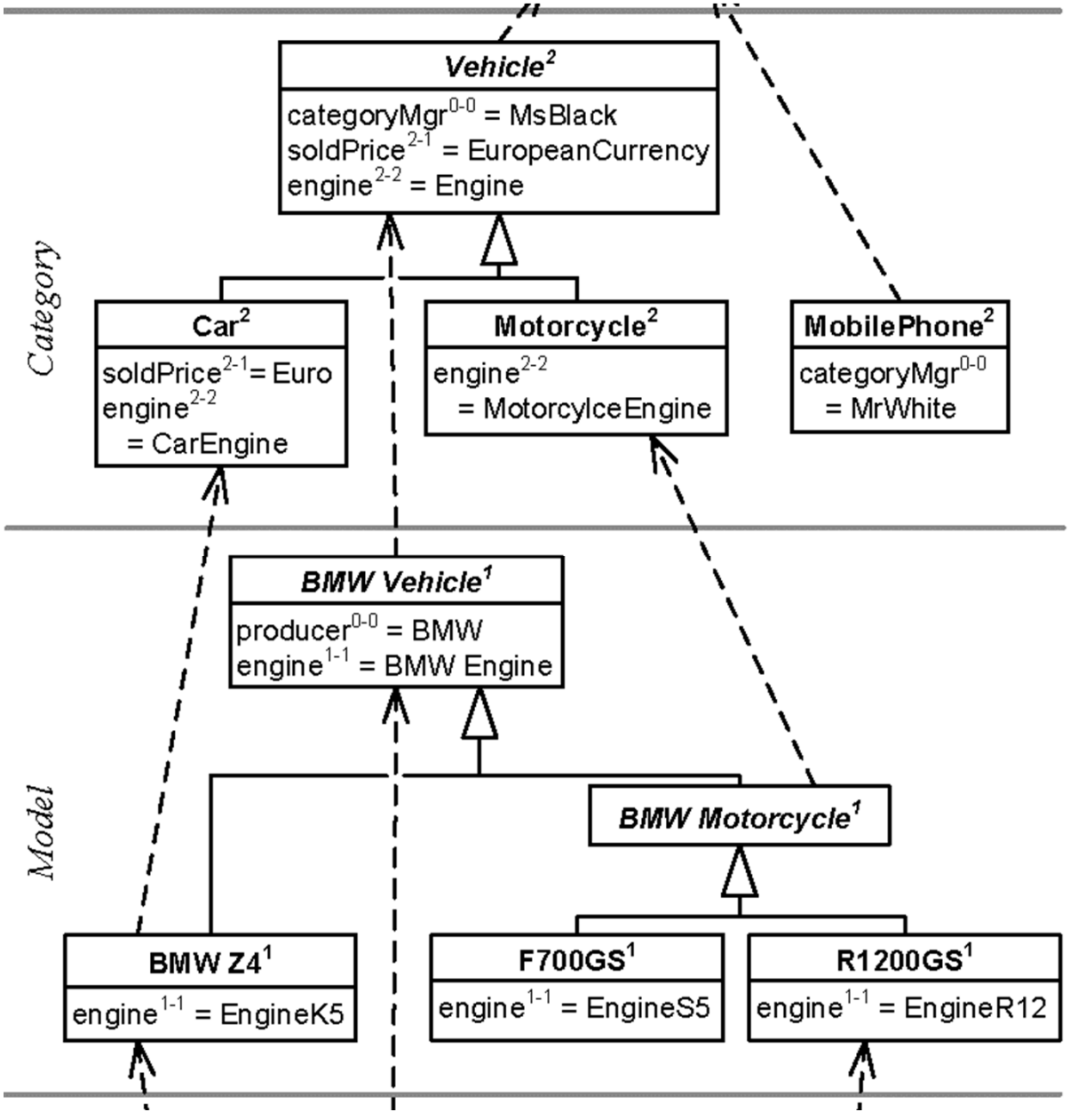


Counting clabjects (foundation for multiplicity constraints):

Number of product categories?
3

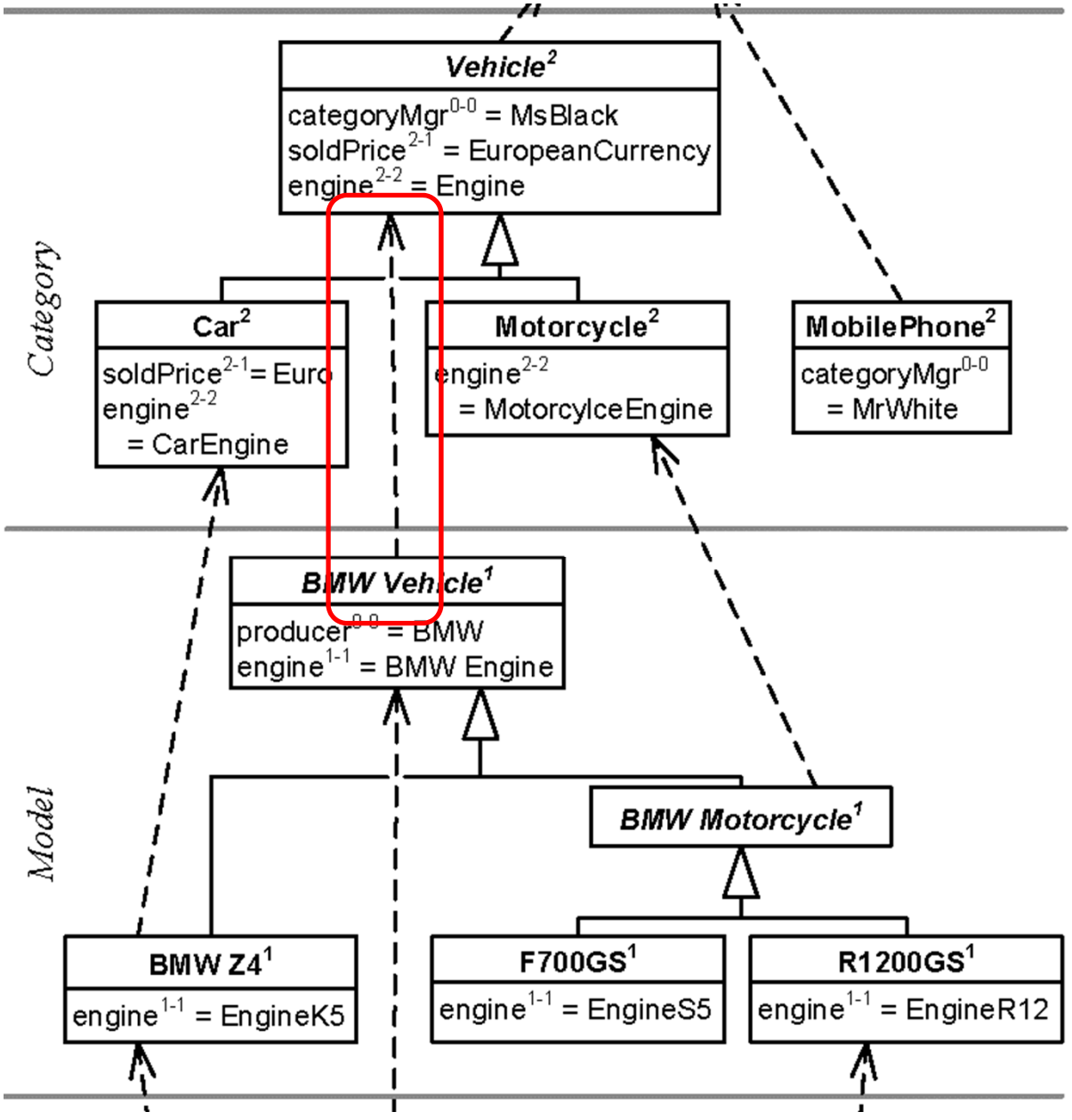
Abstract clabject Vehicle is not a proper product category. It represents common properties of product categories Car and Motorcycle.

Abstract vs Concrete Clabjects



Different kinds of instantiation relationships:

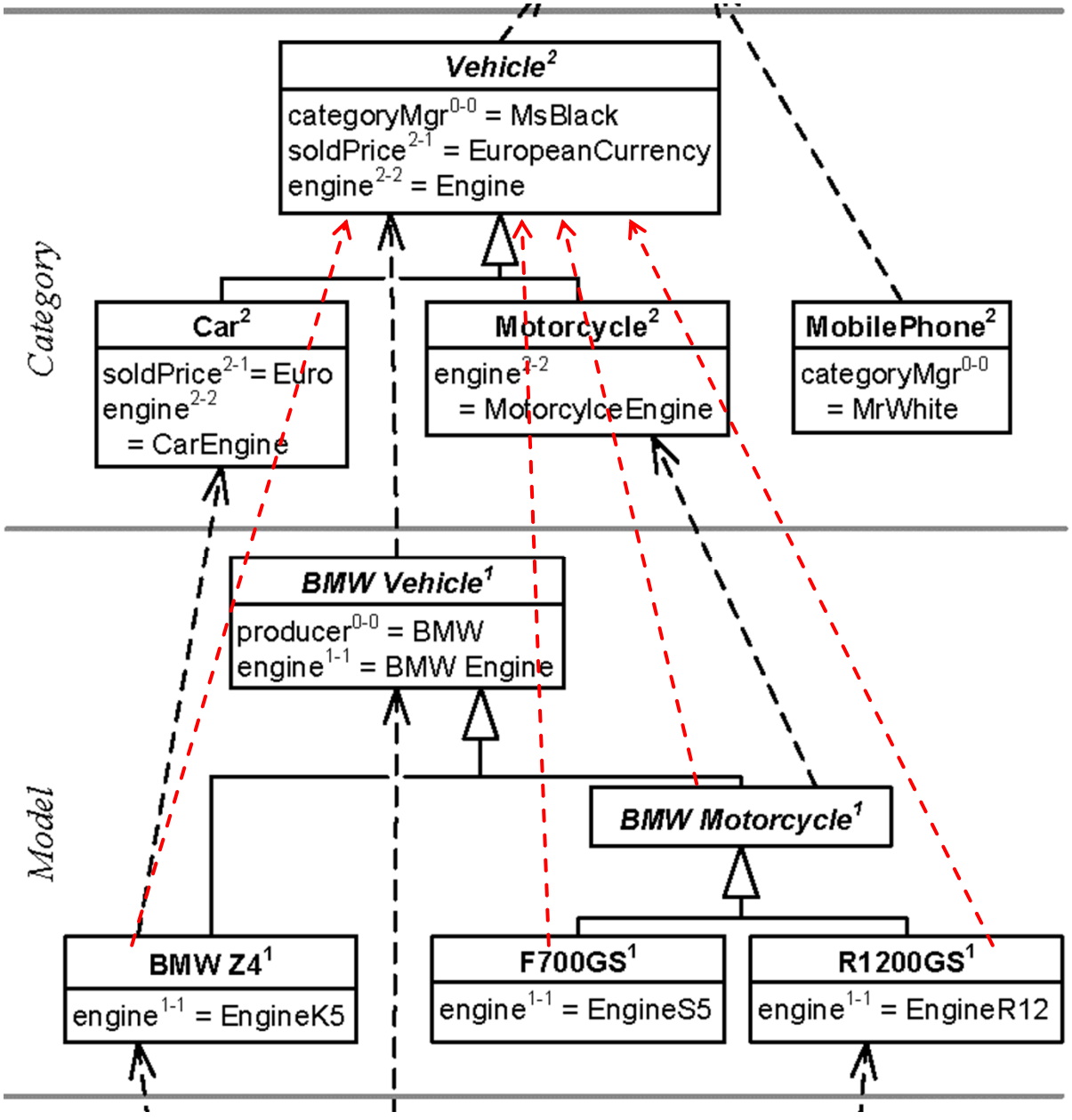
Abstract vs Concrete Clabjects



Different kinds of instantiation relationships:

Shared abstract instantiation

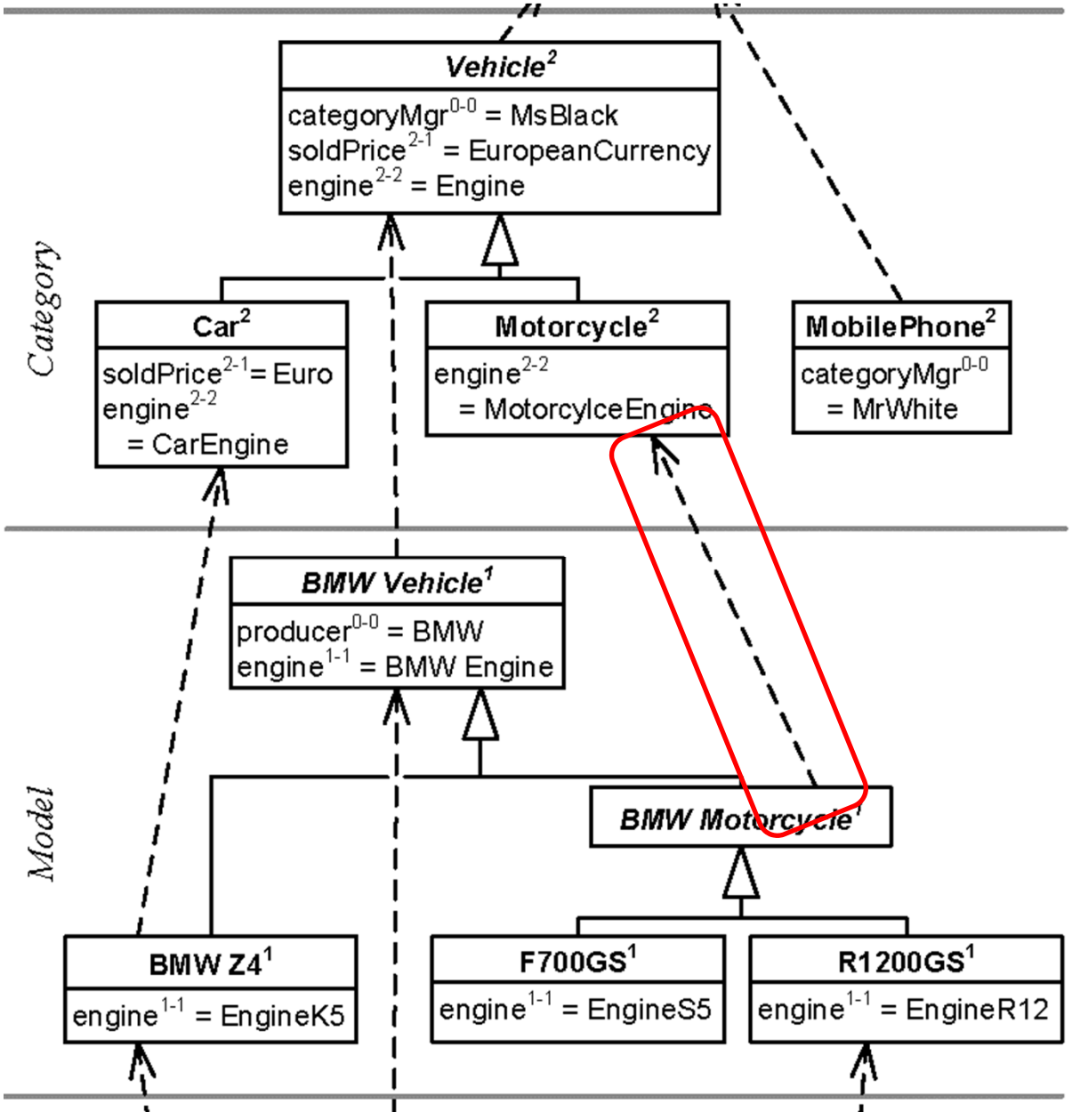
Abstract vs Concrete Clabjects



Different kinds of instantiation relationships:

Shared abstract instantiation

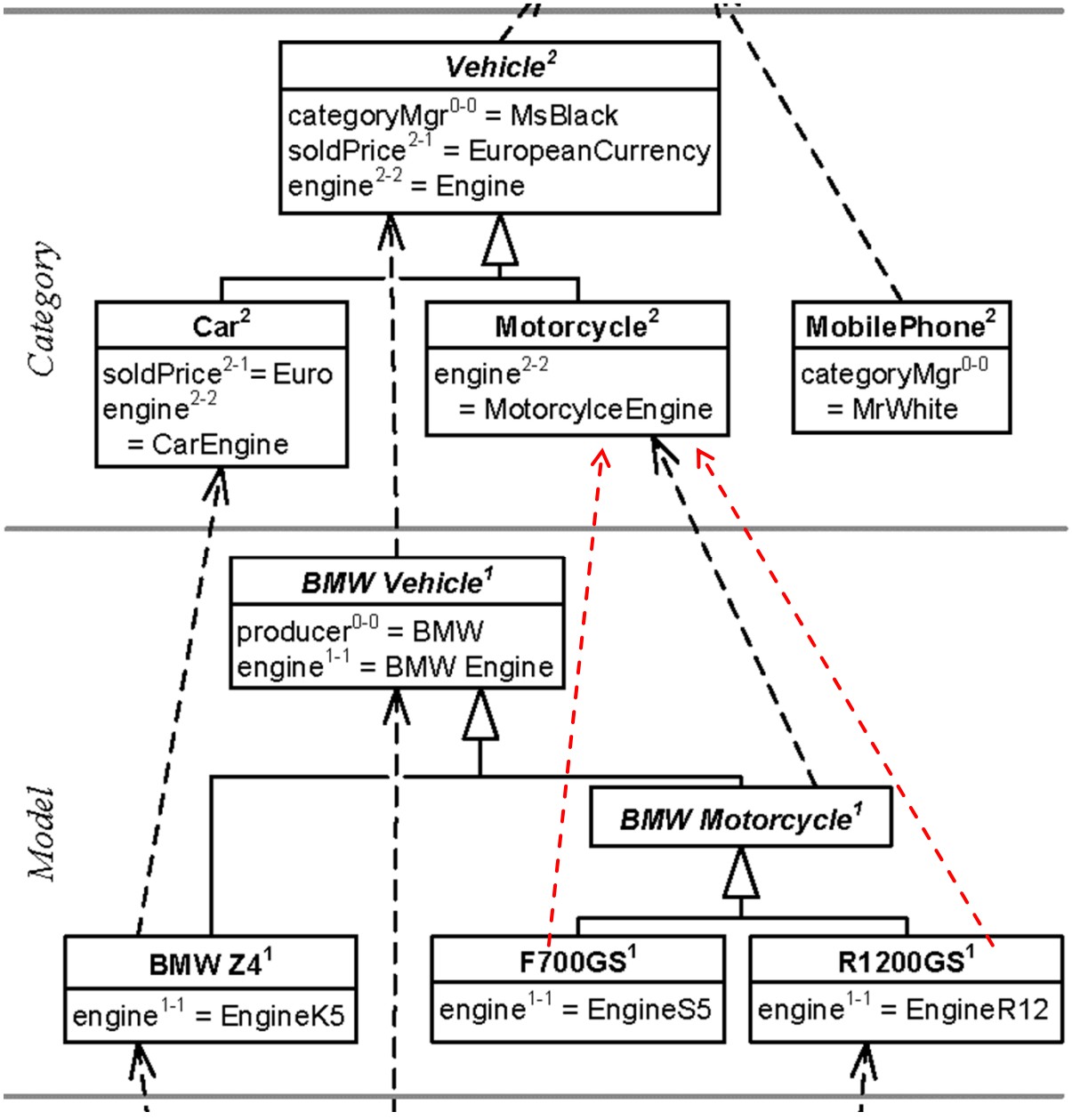
Abstract vs Concrete Clabjects



Different kinds of instantiation relationships:

Shared concrete instantiation

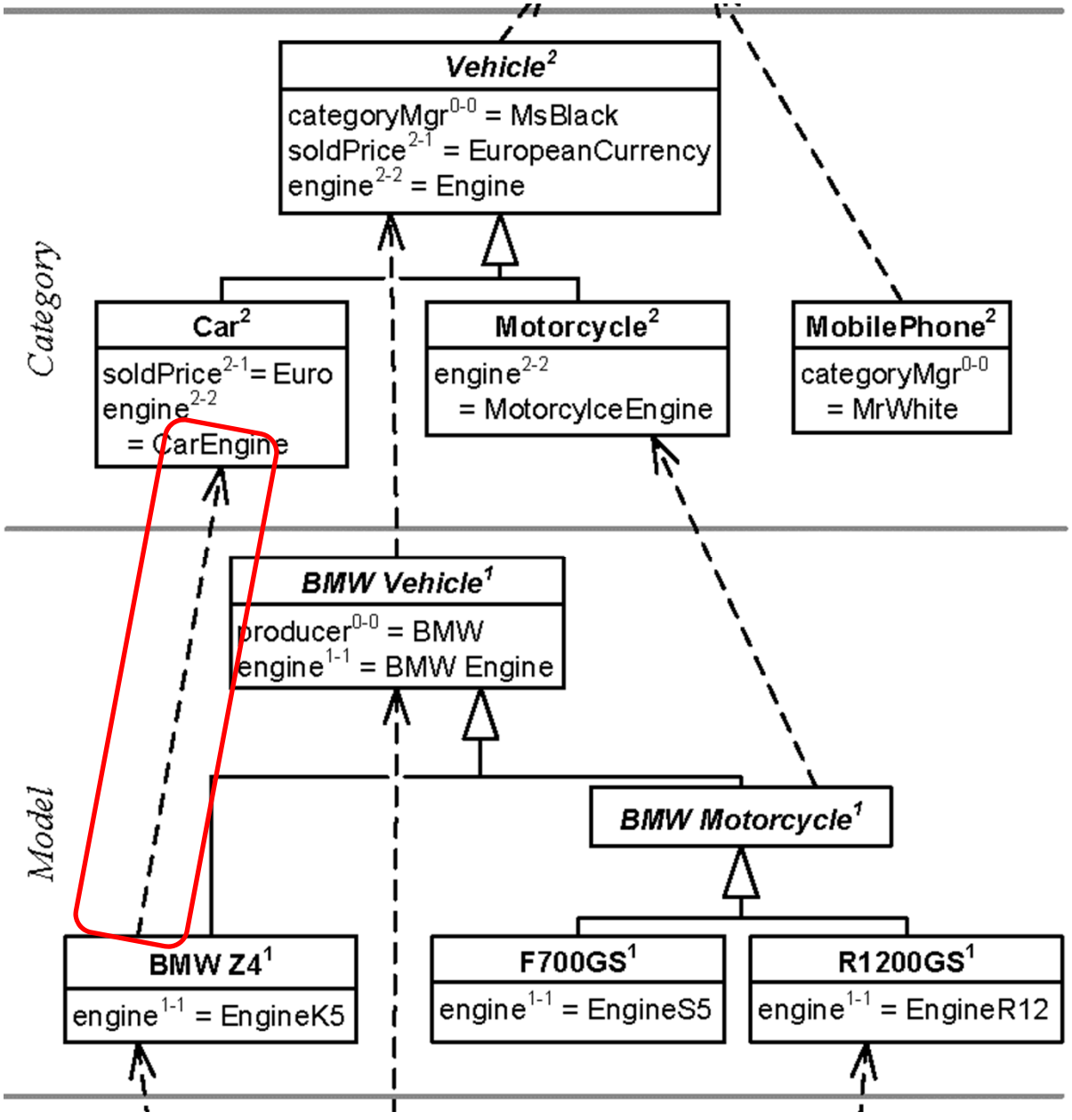
Abstract vs Concrete Clabjects



Different kinds of instantiation relationships:

Shared concrete instantiation

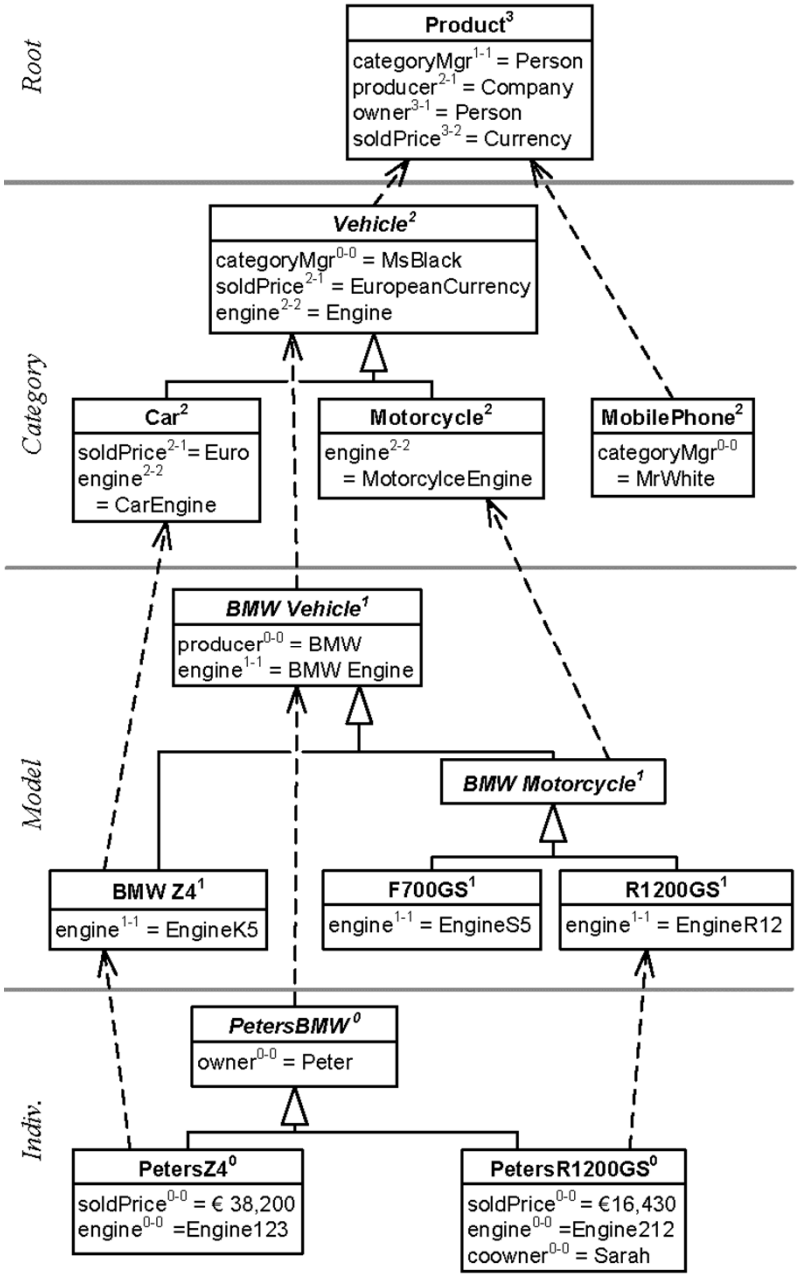
Abstract vs Concrete Clabjects



Different kinds of instantiation relationships:

Immediate concrete instantiation

Abstract vs Concrete Clabjects



Abstract superclabject rule

All superclabjects are abstract in that they have no direct concrete instances (but they may have abstract instances).

guarantees (with other formalization)

- that every concrete clabject is member of exactly one concrete clabject of the instantiation level above
- that every property is instantiated according to mandatory constraints

In the MULTI paper you further find

- Inheritance Mechanism for ‚Dual Inheritance‘
- Mandatory constraints at multiple levels
- Formalization of the approach

Ongoing Work

- Object-oriented programming with DDI
  (implemented in Ruby)
- Extension of DDI and its ConceptBase Implementation with
  - Multiplicity constraints at multiple instantiation levels
  - Specialization of relationships (sub-hierarchies)
  - Define different sub-languages (profiles) for different modeling needs (maybe hiding the complexities of working with potencies)

Future Work

- Combine DDI with multiple inheritance, roles, mixins, ...
- Further explore the use of level names instead of numeric potencies
- Analyse the ontological foundations of DDI
- Evaluate DDI in real-world settings

Thanks for your attention!

Additional Slides

VODAK
the first system with deep instantiation

Wolfgang Klas: Metaclasses in VODAK and their application in database integration. Technical Report. GMD-IPSI, Integrated Publication and Information Systems Institute. **1990**

Wolfgang Klas, Michael Schrefl: Metaclasses and Their Applications, Data Model Tailoring and Database Integration. Lecture Notes in Computer Science 943, Springer **1995**, ISBN 3-540-60063-9

From the abstract: *„... some object-oriented systems organize classes into **metaclasses** which define common properties of their class instances. But in none of them metaclasses can be used to **define common properties of instances of their instances**. Metaclasses in VODAK, a distributed object-oriented database system developed at IPSI, have been extended in that direction.“*

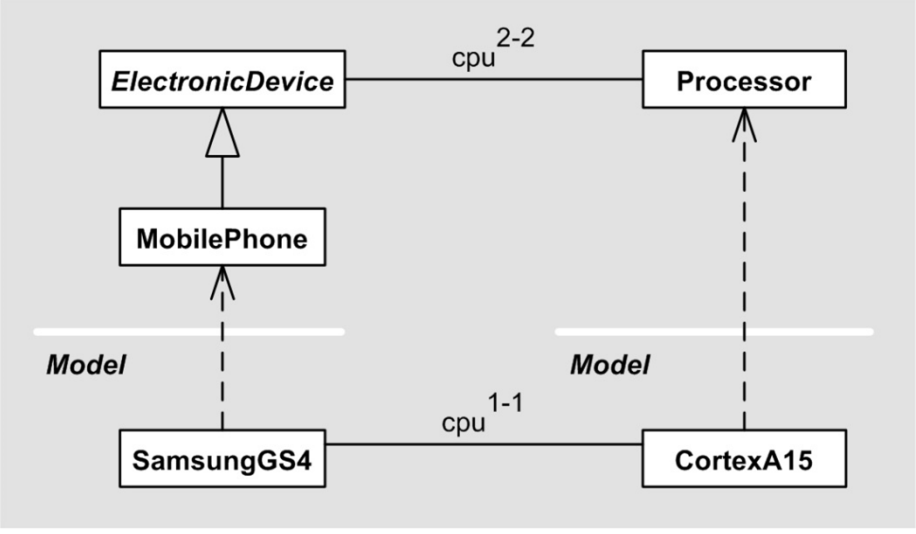
Implementation in ConceptBase

- Prototypical environment for storing DDI models and analyzing their consistency
- Implementation strategy:
  - create Telos metamodel to store DDI models
  - define predicates by deductive rules
  - encode DDI axioms as integrity constraints
- available for download and own experimentation and extension at: http://conceptbase.cc/ddi

Restrictions of DDI

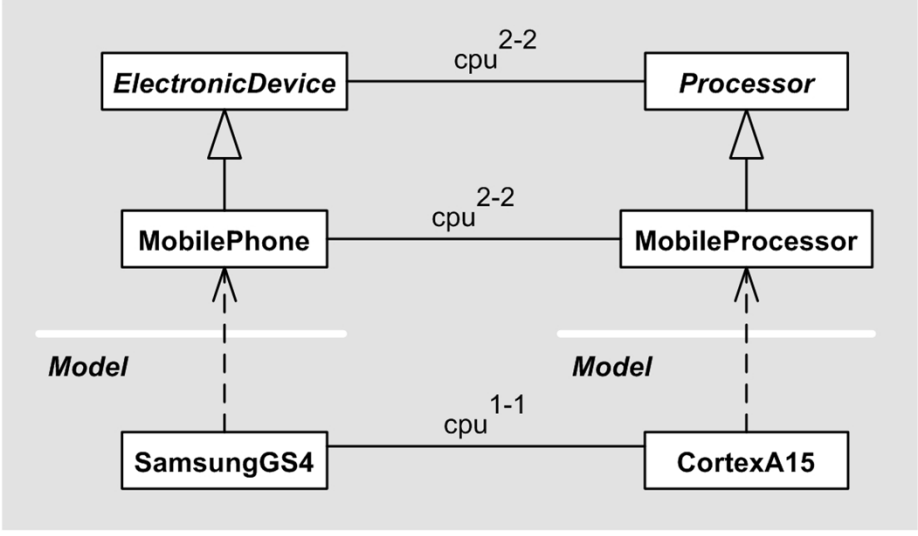
- Single instantiation
- Acyclicity of instantiation hierarchy
- Unique introduction of relationship labels
  - every relationship instantiation hierarchy has a single root relationship
- Local stratification of relationships
  - checked for relationships with the same label
  - ensures that every relationship is direct instantiation of at most one other relationship
  - reduce overall complexity of approach
  - avoids potential conflicts of domain and range constraints

Extending DDI with Generalization



Sub-objects inherit from super-objects

Extending DDI with Generalization
Co-Variant Refinement

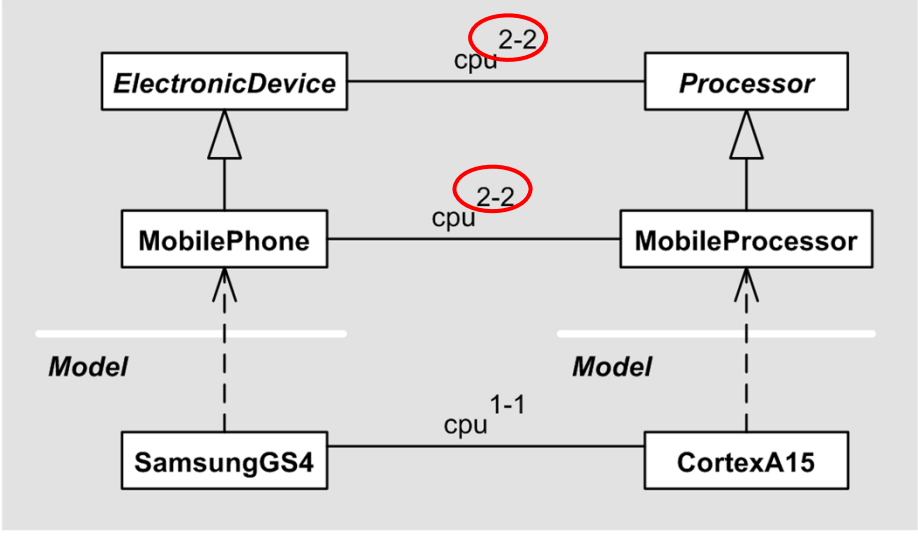


Bi-directional Refinement:

„mobile phones have only mobile processors as cpu“

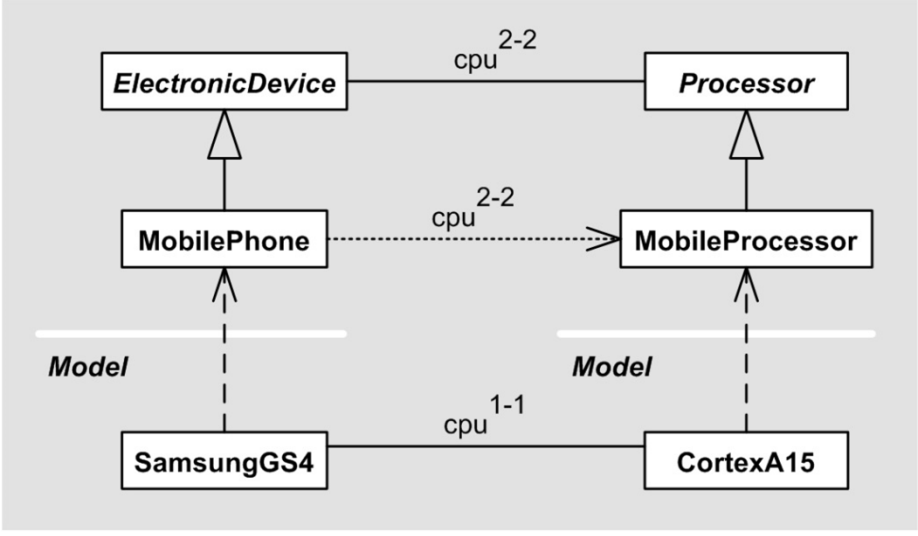
„mobile processors are only cpu of mobile phones“

Extending DDI with Generalization
Co-Variant Refinement



Potencies of sub- and super-relationships are the same

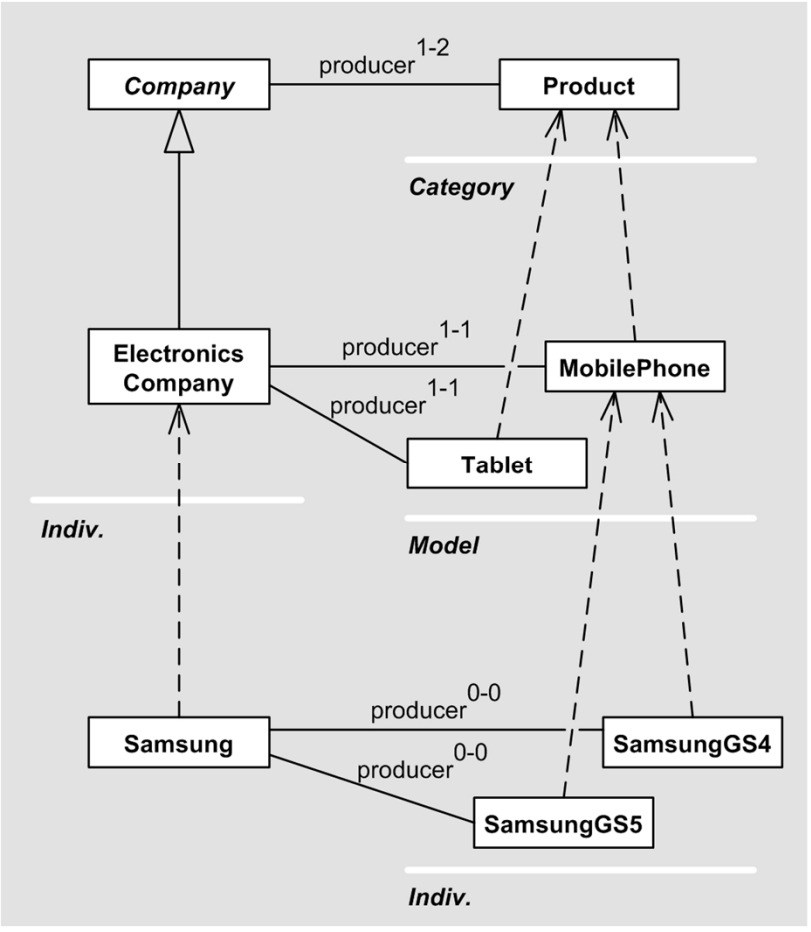
Extending DDI with Generalization
Co-Variant Refinement



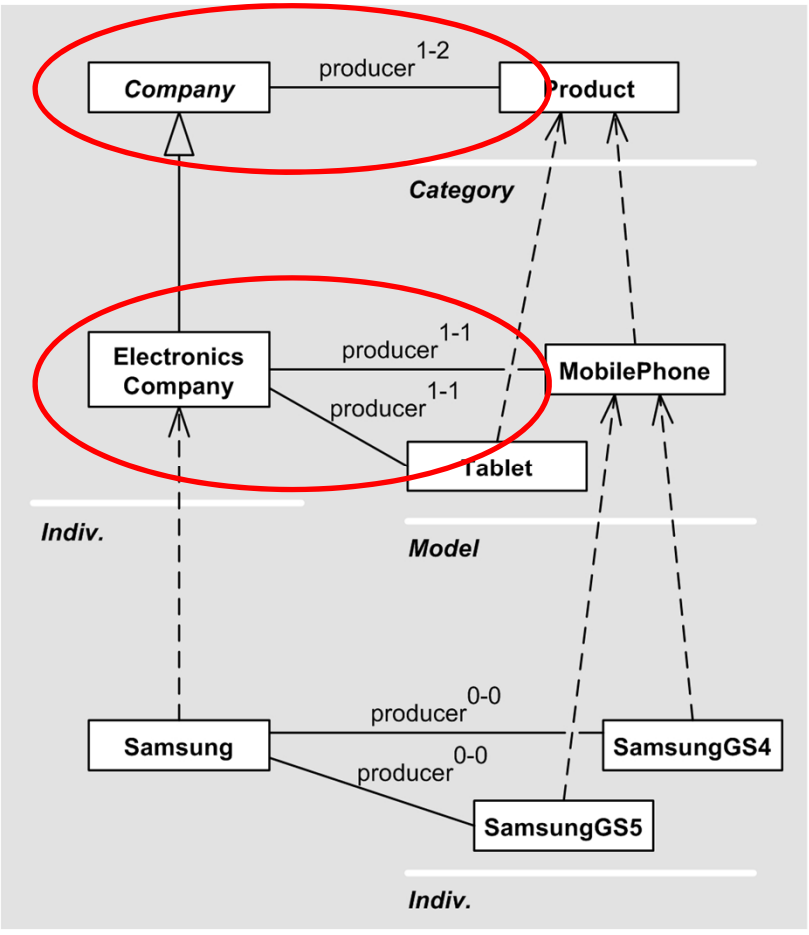
Uni-directional Refinement:

„mobile phones have only mobile processors as cpu (but mobile processors may be cpu of other electronic devices)“

Extending DDI with Generalization

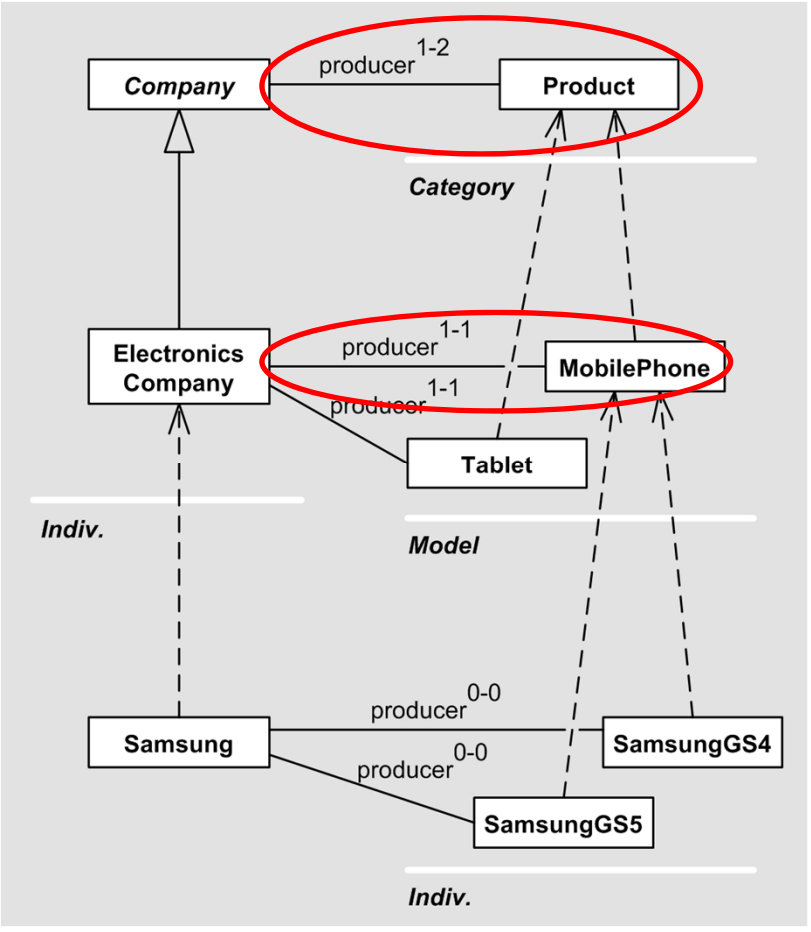


Extending DDI with Generalization



An object may instantiate the relationships of its super-object ...

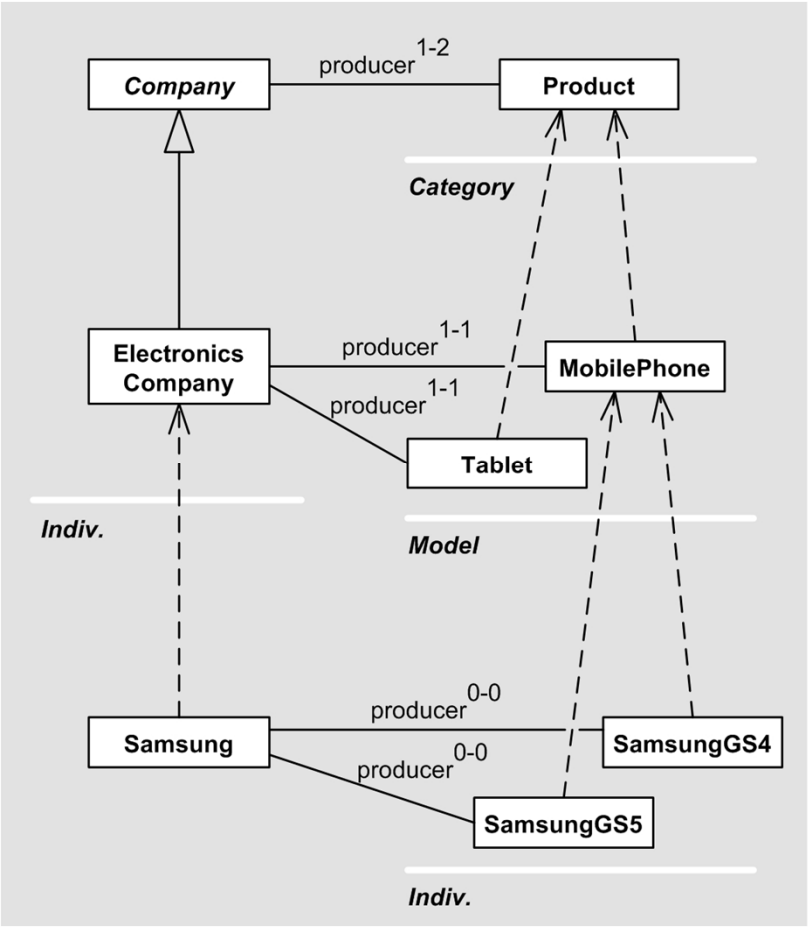
Extending DDI with Generalization

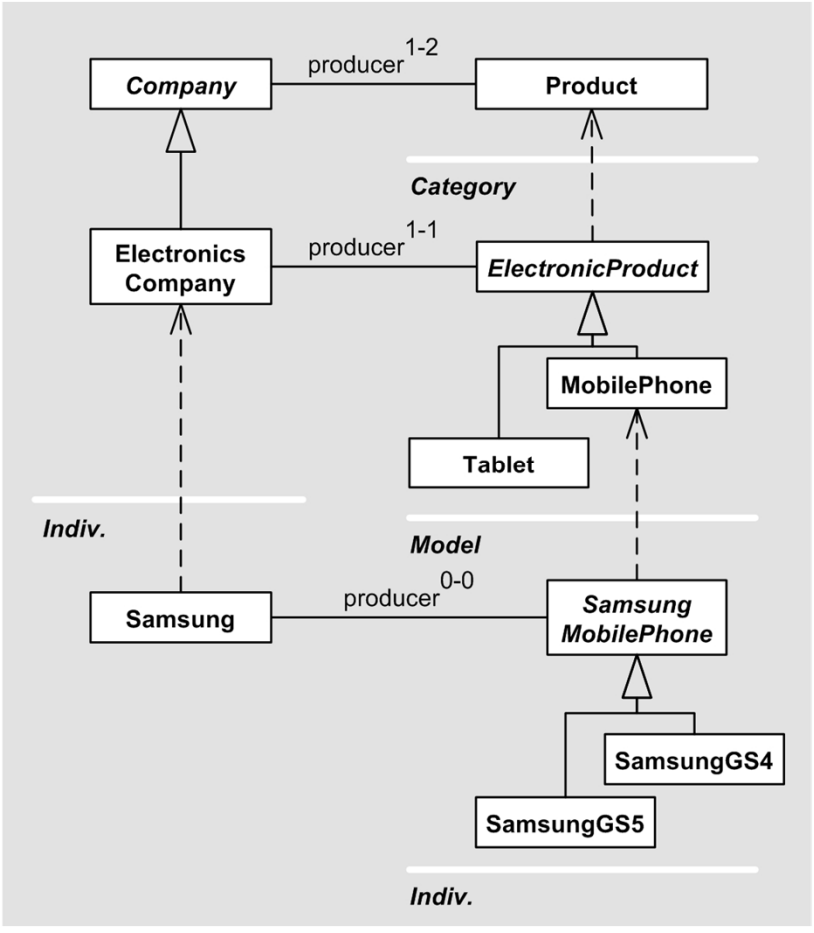


And viewed from the oppossite direction:

An object may refine the relationships of its class object

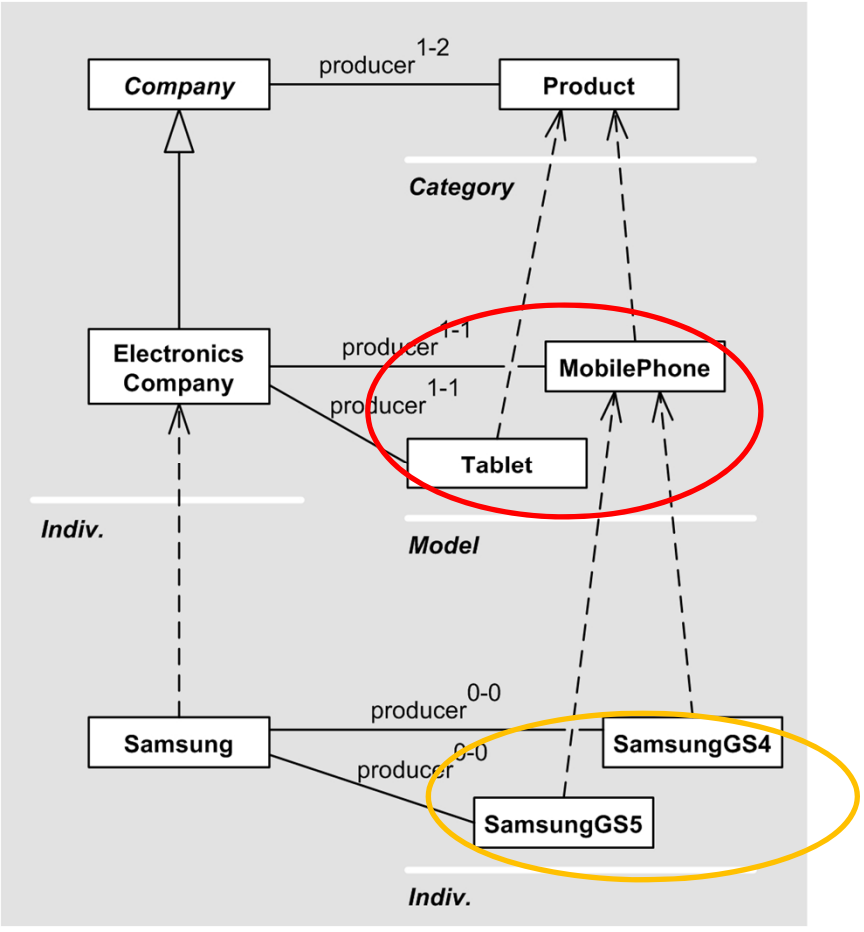
Extending DDI with Generalization

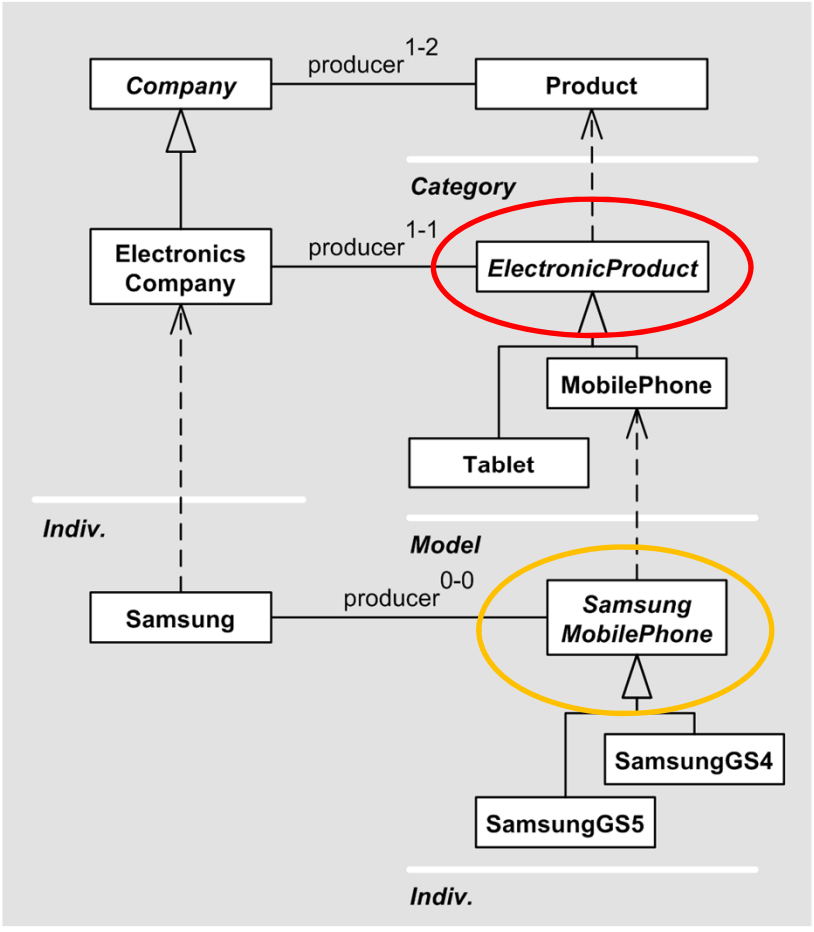




Generalization may be applied to objects at every instantiation level ...

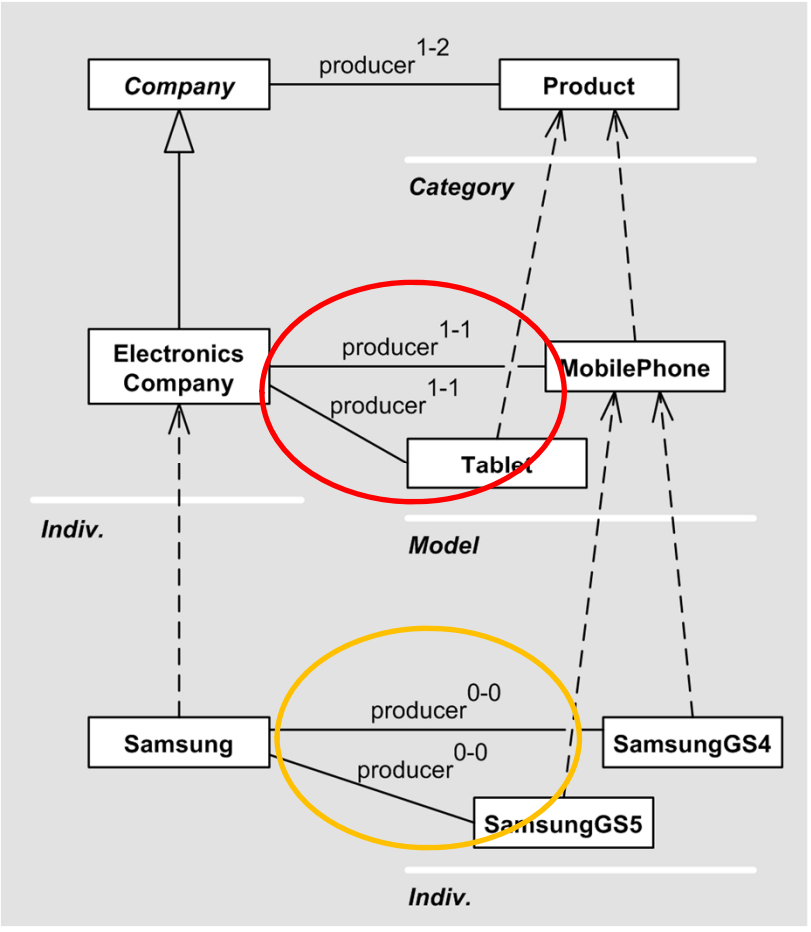
Extending DDI with Generalization

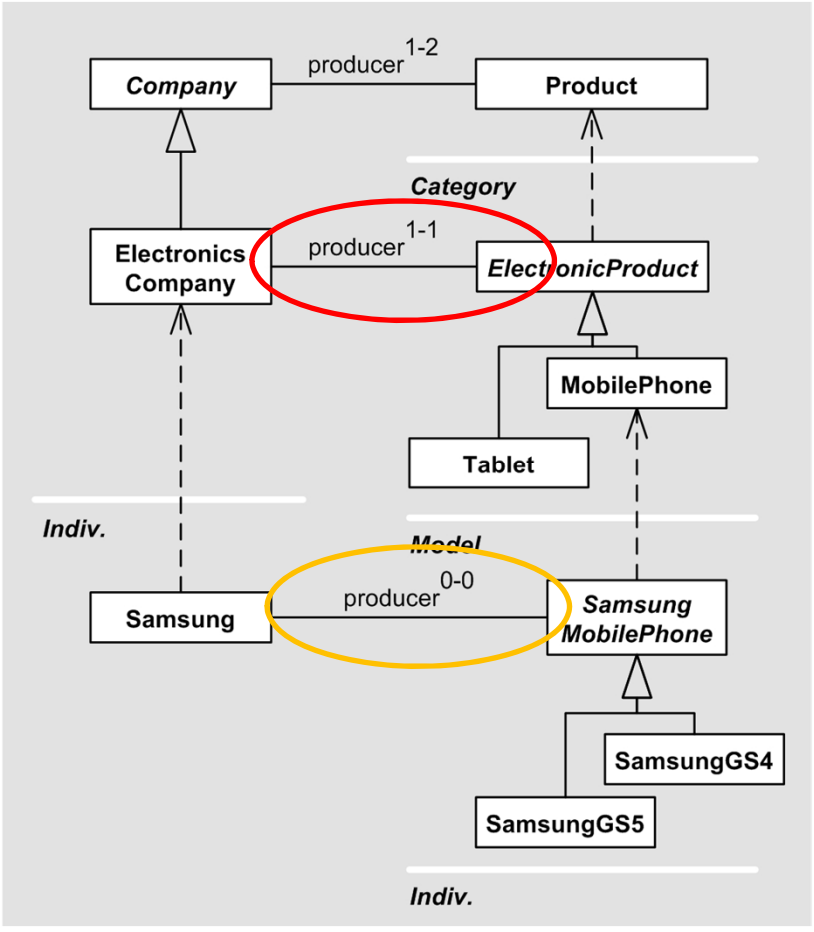




Objects may be combined into an abstract super-object.

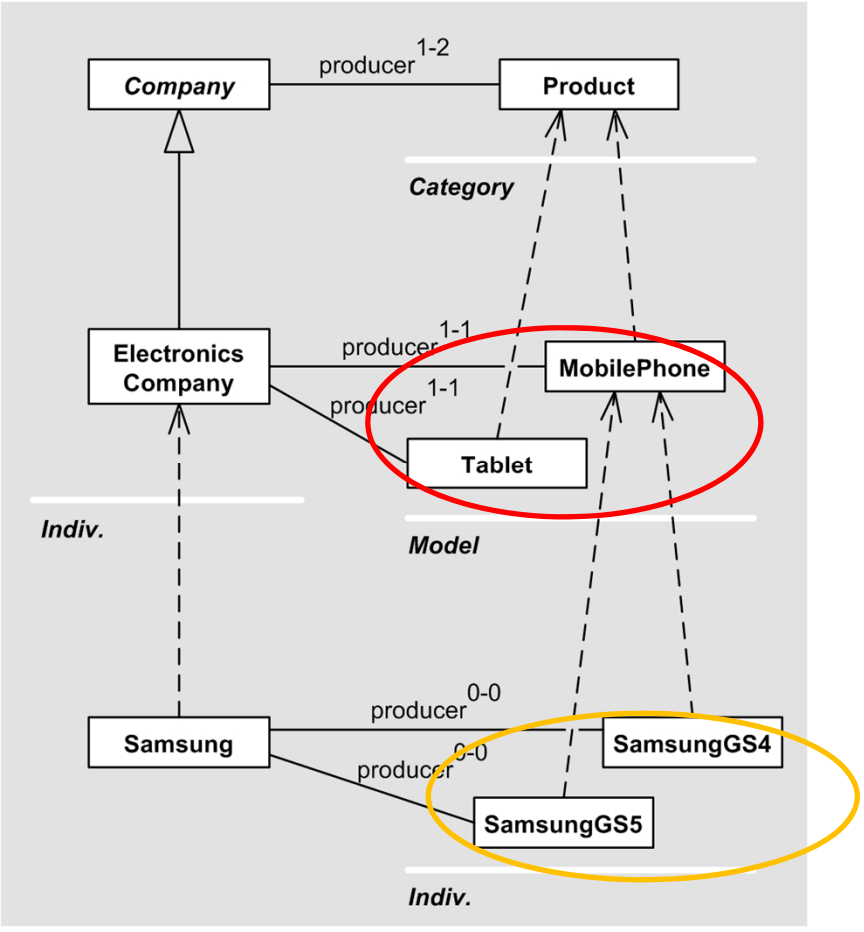
Extending DDI with Generalization

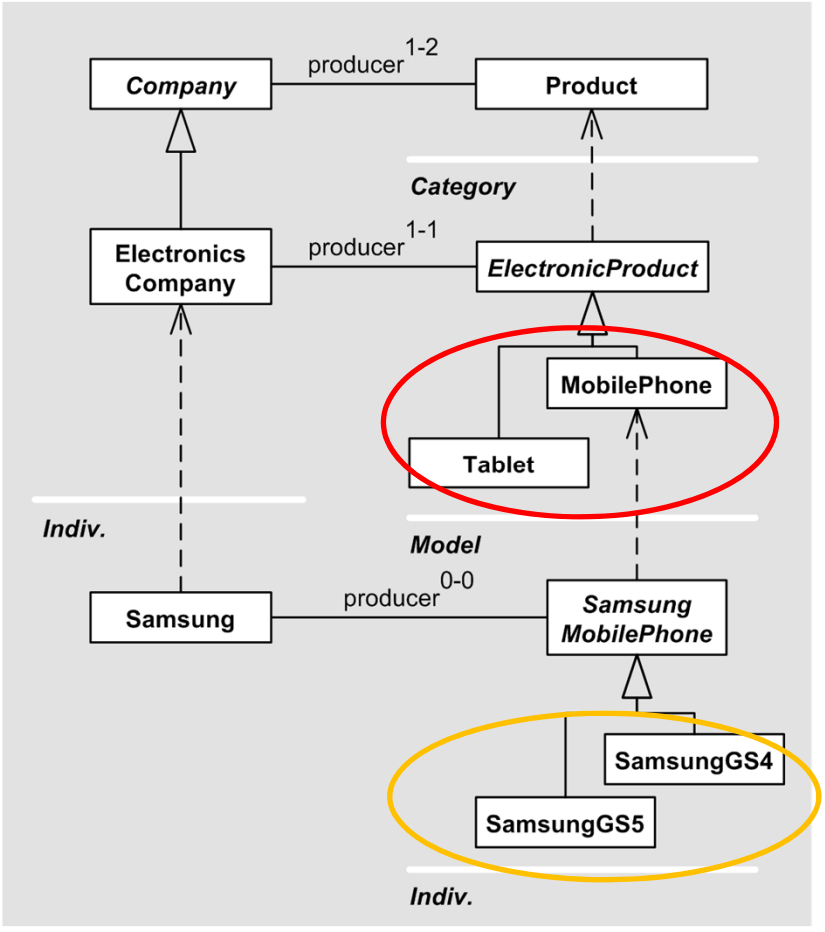




Common properties of objects are combined to the abstract super-object

Extending DDI with Generalization

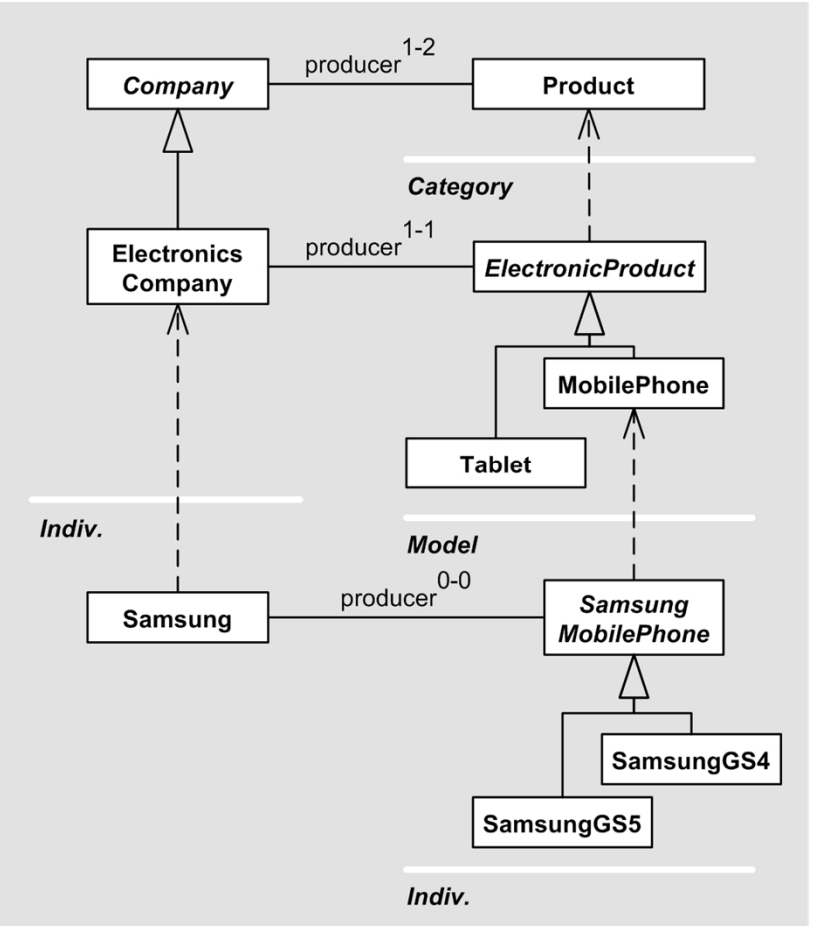




The number of concrete objects remains stable.

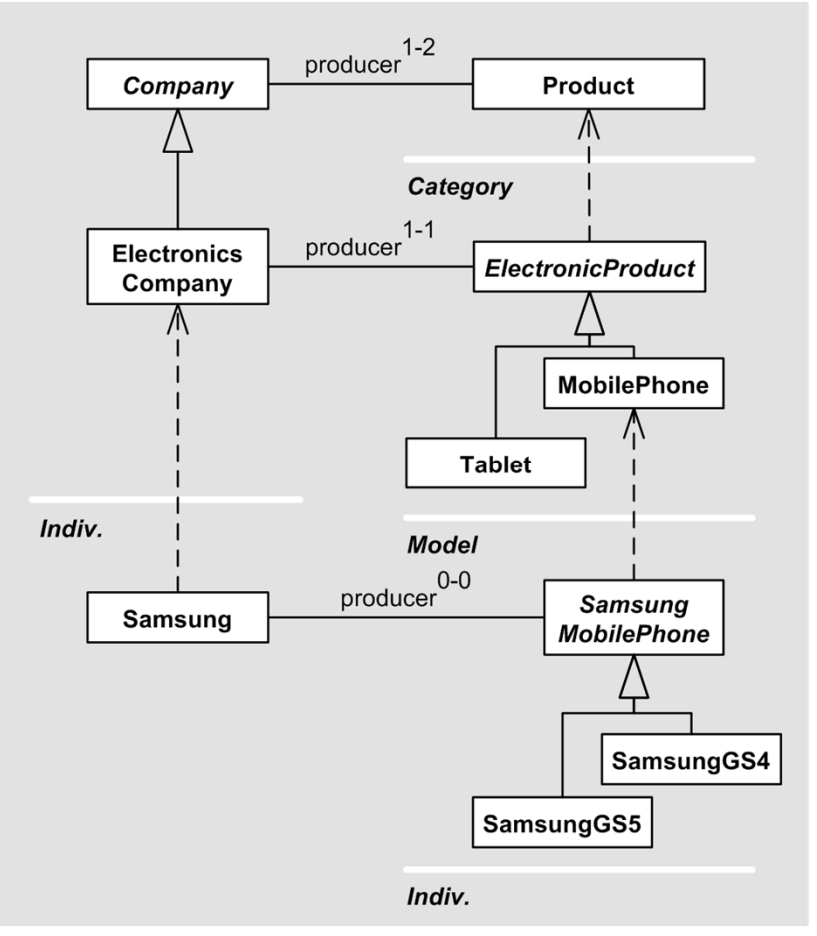
Extending DDI with Generalization
Abstract vs Concrete Objects

- Abstract objects may not be instantiated directly
- Abstract objects do not have own properties. They only describe common properties of their concrete sub-objects
- Abstract objects are ignored when counting objects

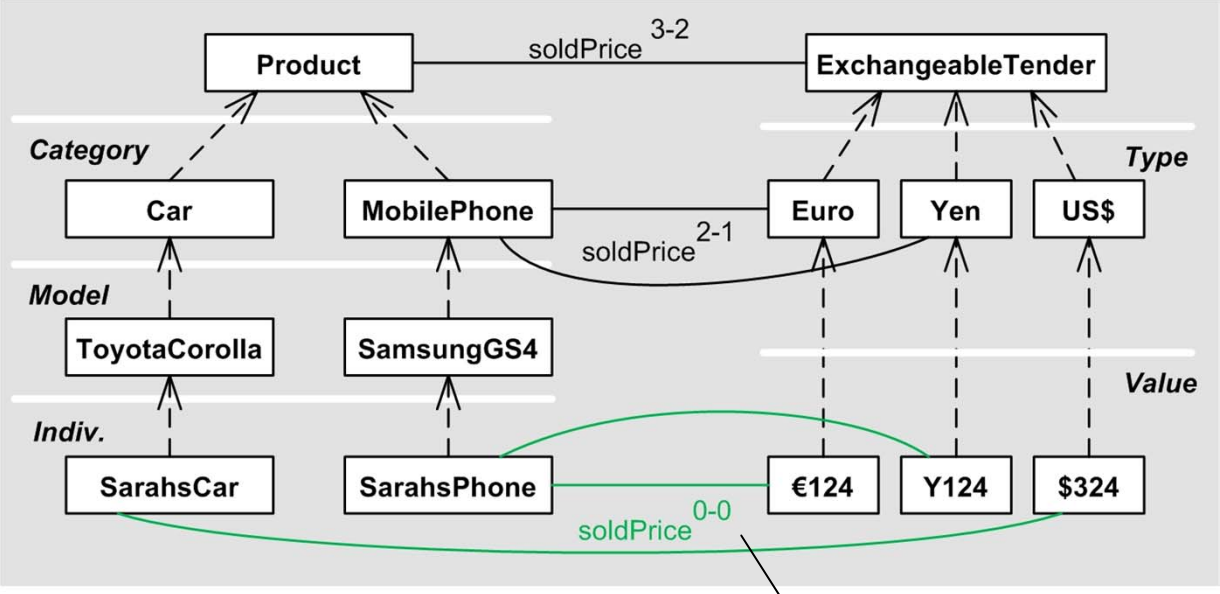


Extending DDI with Generalization Restrictions

- single inheritance
- objects that instantiate different objects may not have a common super-object
- abstract superclass rule: every super-object is abstract



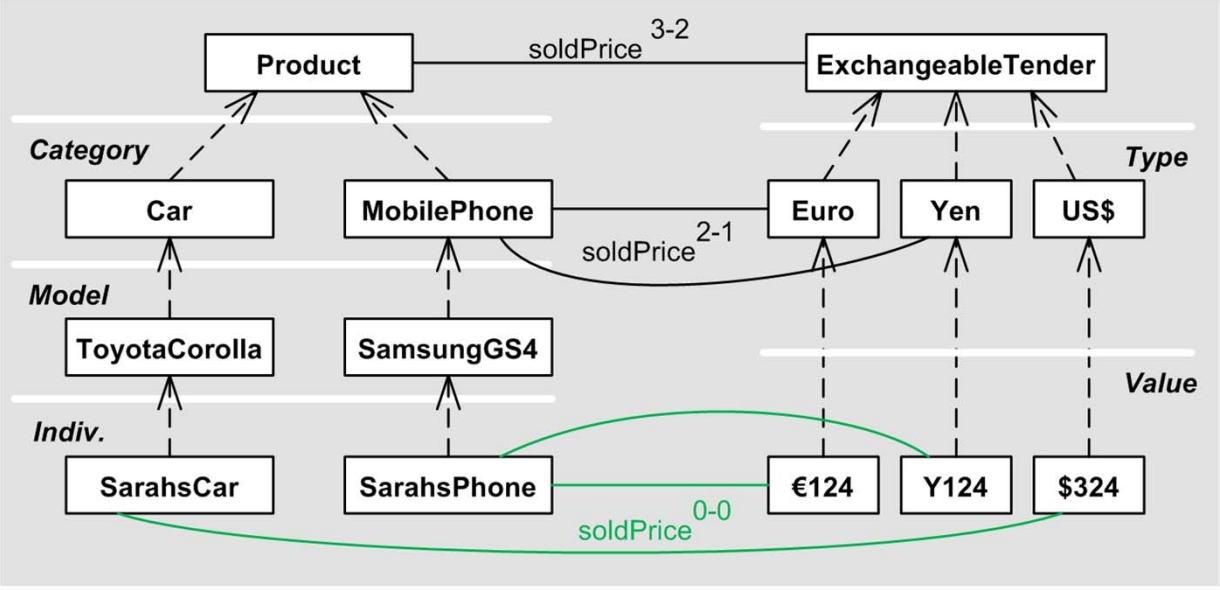
Dual Deep Instantiation - Domain/Range Constraints

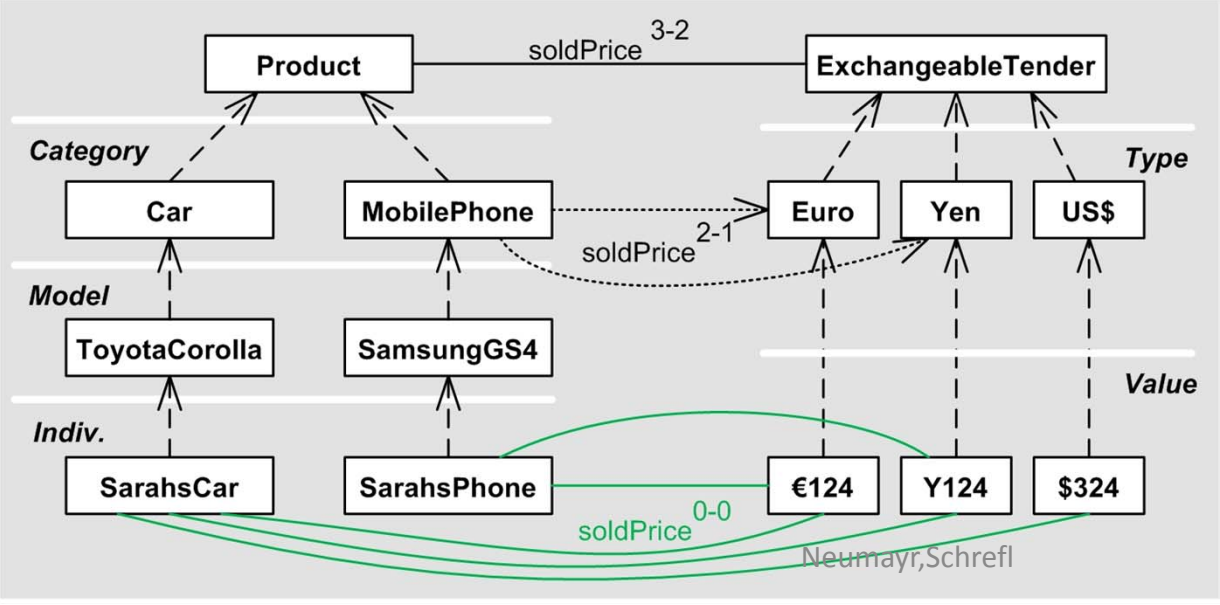


When relationships are defined partially, then implied range and domain constraints may be surprising, due to bi-directionality ...

Admissible relationships

Dual Deep Instantiation - Domain/Range Constraints

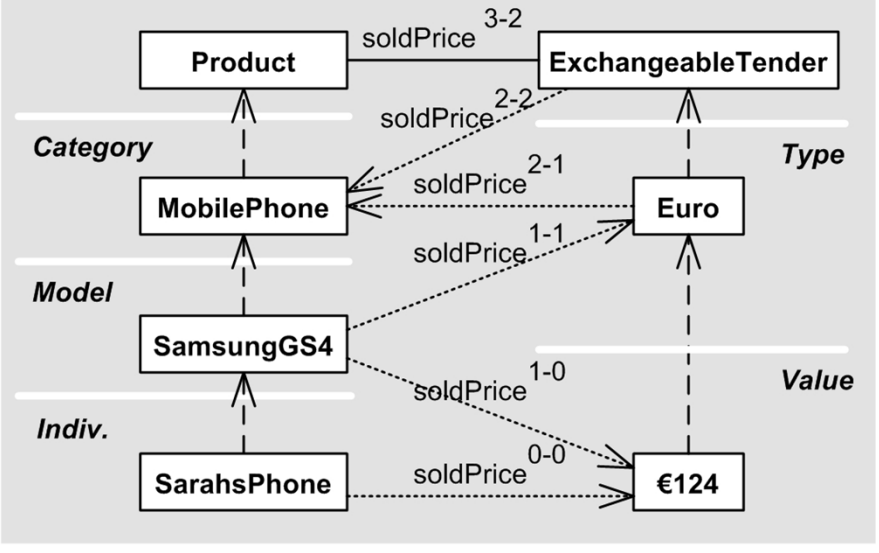




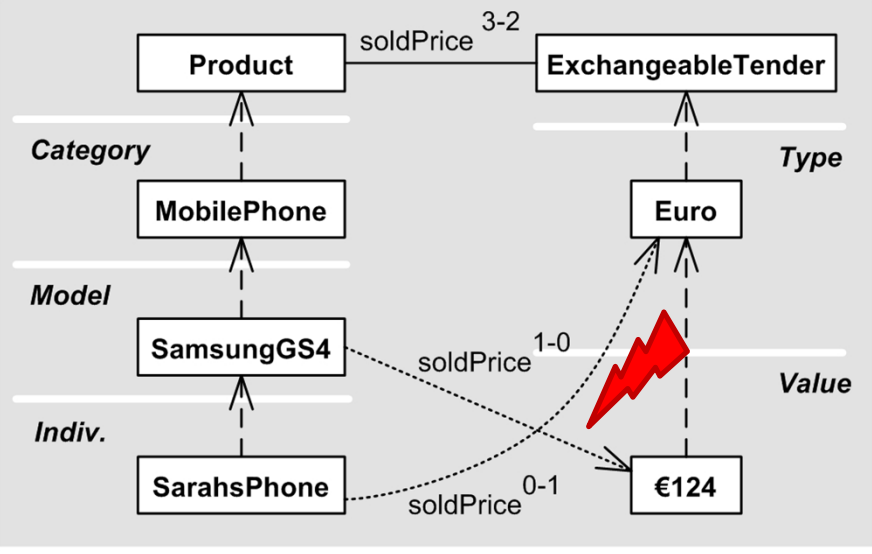
... that‘s why it‘s often prefereable to define constraints only in one direction

Restrictions of DDI
Local Stratification

- checked for relationships with the same label
- ensures that every relationship is direct instantiation of at most one other relationship
- in order to
  - reduce overall complexity of approach
  - avoids potential conflicts of domain and range constraints

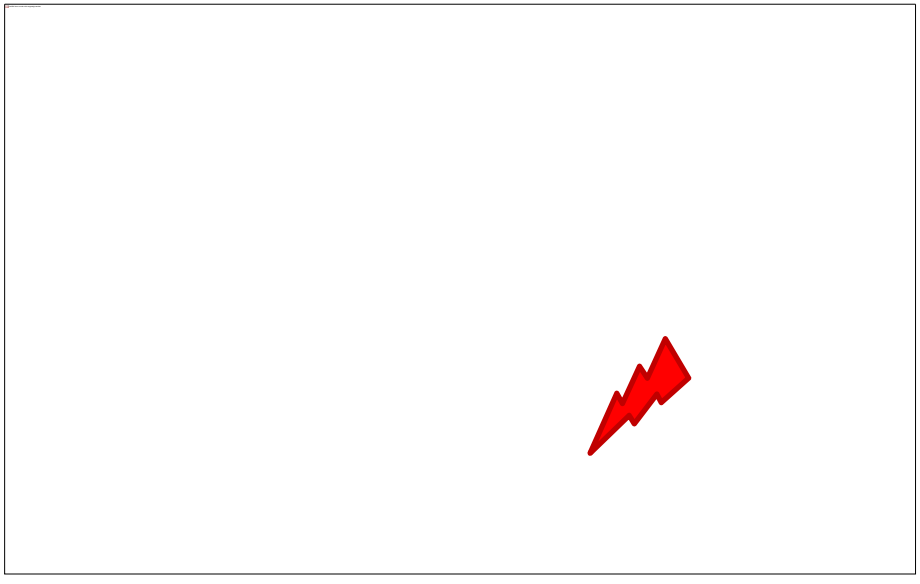


Restrictions of DDI
Local Stratification



Problem to be avoided:
a soldPrice relationship between SarahsPhone and €124 would be a direct instantiation of two relationships

Restrictions of DDI
Local Stratification



Local stratification does not allow to specify:

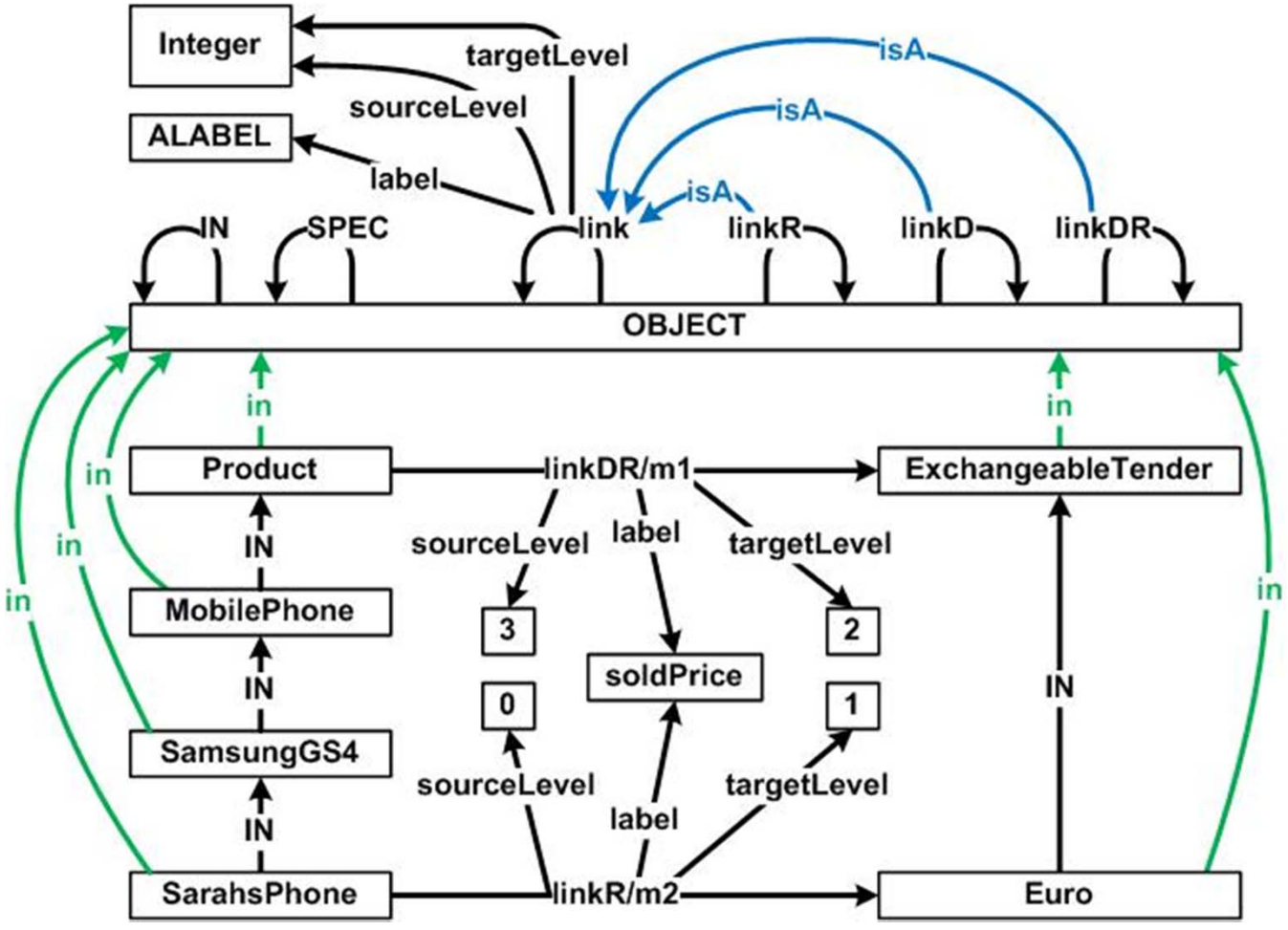
„If an iPhone5 individual is sold, then the price is €324“

and

„Sarah‘s phone is paid in Euro“

Future work: explore possibilities to further relax local stratification

ConceptBase Implementation of DDI



Telos Metamodel of DDI