Pragmatic Issues in Multilevel Modeling

Insights from Modeling the MULTI-Warehouse Challenge in MLM-USE

Mira Balaban, Azzam Maraee, Arnon Sturm

אוניברסיטת בן-גוריון בנגב
جامعة بن غوريون في النقب
Ben-Gurion University of the Negev

ACHVA ACADEMIC COLLEGE

Pragmatics

- *Pragmatics* is the study of *language usage within a **context***

- Language *categories*:
  - **Syntax:** Defines valid combinations;
  - **Semantics:** Meaning of valid combinations;
  - **Pragmatics:** *Usage* of meaningful combinations;

- *Pragmatics* studies the contribution and influence of *context* on meaning:
  - implied meanings – based on *shared knowledge, social context*, and *social rules;*
  - impact of *nonverbal communication* – tone of voice, body language (gestures), speech acts;
  - how human language is utilized in *social interactions;*

Pragmatics in Programming

- Practical aspects of ***using language constructs***:
  - Build **effective programs**;
  - **How to use** language constructs;
  - **Impact** of software **context**;
  - **Implementation** details;
  - **Software development** impact;

- *Best practices* in code writing:
  - **Clean code:** Manageable; flexible; stable; robust; efficient; understandable;
  - **Design principles:** Duplication; orthogonality (components); inheritance vs. dispatch;
  - **Code idioms:** Best practices in writing commonly-recurring construct in a particular programming language;
  - **Code smells:** Heuristics for identifying software weaknesses;

Pragmatics in Software Modleing

Best practices in construction of software models:

- *General Modeling principles (contrasting):*
  - Low coupling
  - High cohesion
- *Context-depended modeling idioms:*
  - Attributes vs. associations;
  - Attributes (fields) vs. classes (objects);
  - Physical vs. virtual
  - *type-of* relations;
  - Event-based concepts (like meetings, appointments)
- *Model structuring:*
  - *Hierarchy constructs; parallel hierarchies*;
  - Inheritance vs. dispatch; covariance in inheritance overriding;
  - Indirect interaction;
  - Association cycles;
- *Modeling patterns:*
  - Model interaction
  - Design patterns
  - Analysis patterns
  - Enterprise patterns

Pragmatics in MultiLevel Modeling

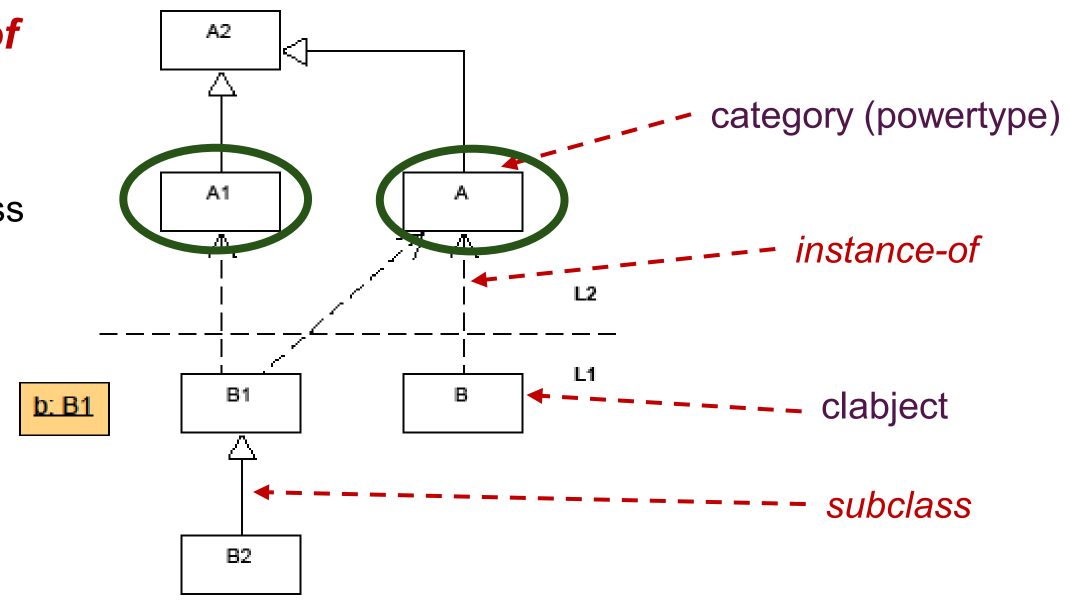
- The ***multi* *vs.* *two* *level*** argument:
  - *Subclass* vs. *type-of* association vs. *instance-of* relation;

- ***Best practices*** in construction of MLM models**:**
  1. ***Feature inheritance*** mechanisms in MLM;
  2. ***MLM idioms***:
     - *Clabject-category* interaction;
     - *Subclass – instance-of* interaction;
  3. ***Constraints*** in MLM*:*
     - Constraint *languages*;
     - *Multiplicity constraints* on associations;
  4. ***MLM structuring***:
     - *Status of the **level** concept*: ***Syntax; semantics, computed***; [not discussed in this talk]
     - General *level decisions*;
     - Level of *non-category classes*;

- ***Based (depend) on semantics of basic elements***

Elements of an MLM Model and Intended Semantics

- Basic relations between classes: ***subclass, instance-of***
- ***Clabject:*** A class which is an *instance-of* a class
- ***Category (powertype):*** A class that has an *instance-of* class



Intuitively intended ***set-based semantics:***

- *subclass* = set inclusion ⊆ :

  All *members (instances, objects) of B2 are members of B1*

- *instance-of* = set membership ∈ :

  b is a member (instance, object) of B1, B1 is a member of A1 and of A

❑ What are these ***category*** classes? Their denotation? Objects?

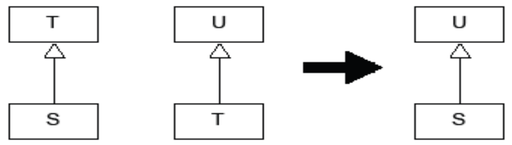
- Are the clabjects of a category-class in an MLM the only instances of that class?
- Are all instances of a category-class clabjects?
- Does a category-class have instances of its own, i.e., not of its clabjects?

❑ What are the implications to ***feature-management!***

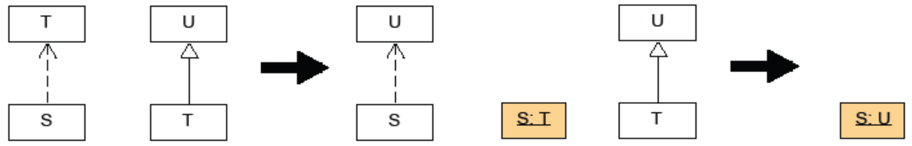
Semantics of Category-classes in MLM

Set axioms for subset and membership:

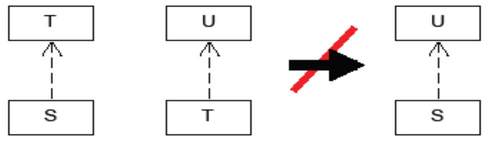
Subset transitivity: $S \subseteq T, T \subseteq U \Rightarrow S \subseteq U$



Membership distributivity over subset: $S \in T, T \subseteq U \Rightarrow S \in U$



Membership non-transitivity: $S \in T, T \in U \nRightarrow S \in U$



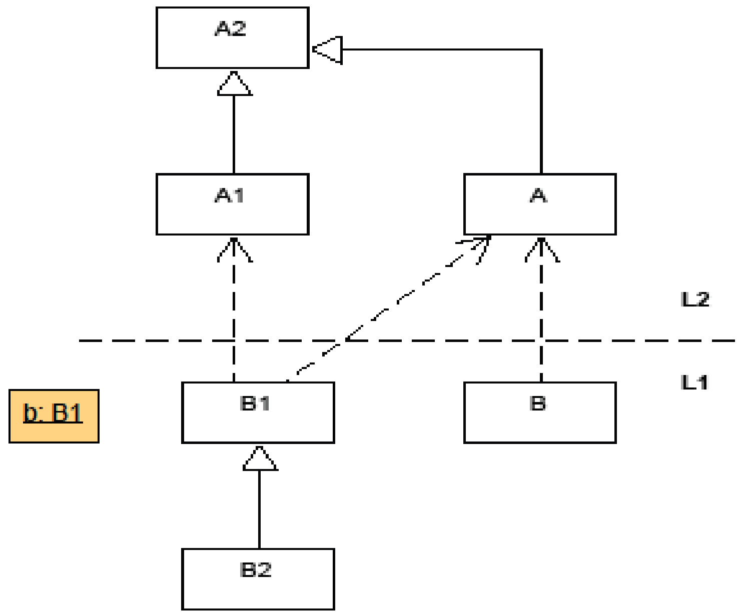
⟹ **Implication for feature inheritance:**

subclass* as subset**: → **Complete *feature inheritance ⋎

instance-of* as membership**: $\cancel{\rightarrow}$ ***No implications for feature inheritance $\cancel{⋎}$

Semantics of Category-classes in MLM

- Members of *B1* are not necessarily members of *A1*:
  - ~~$B1.allInstance() \subseteq A1.allInstance()$~~
  - *b is not necessarily a member of $A1$*
- ***No justification*** for class *B1,* ***to inherit*** features from class *A1*
- *b* ***does not have features*** that are abstracted by class *A1*

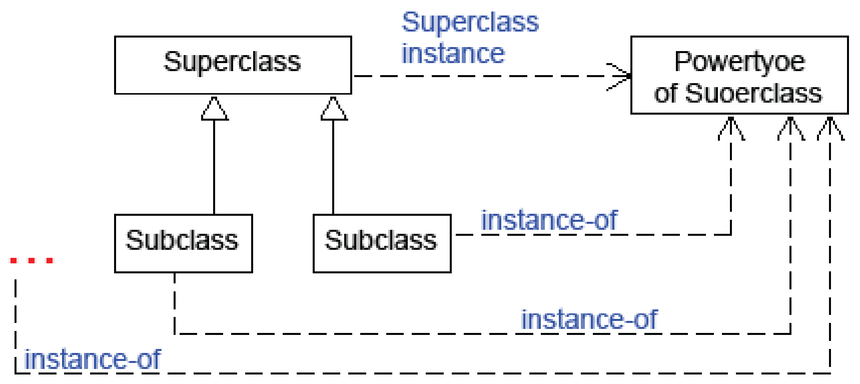


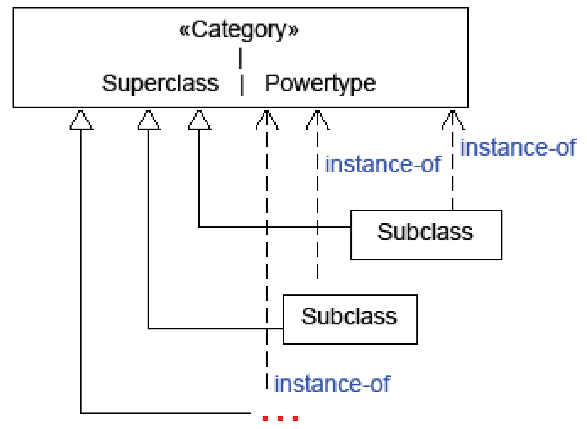
Actual Dual Semantics of category-classes in MLM

Observation: All MLM applications implement feature inheritance along instance-of.

- Adopt ***SOME subclass-based feature inheritance*** along *instance-of*.
- ***Category-classes*** behave ***"to a greater or lesser extent"*** as ***superclasses***!

Double view of category classes: *Category = Superclass view ∪ Powertype view*

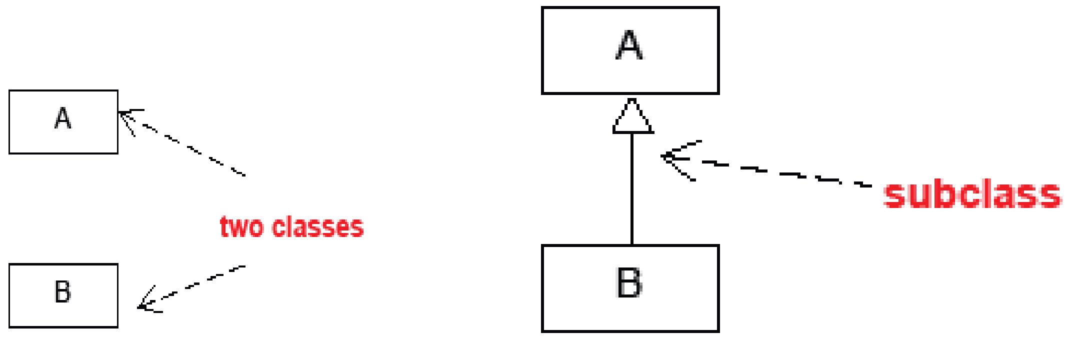




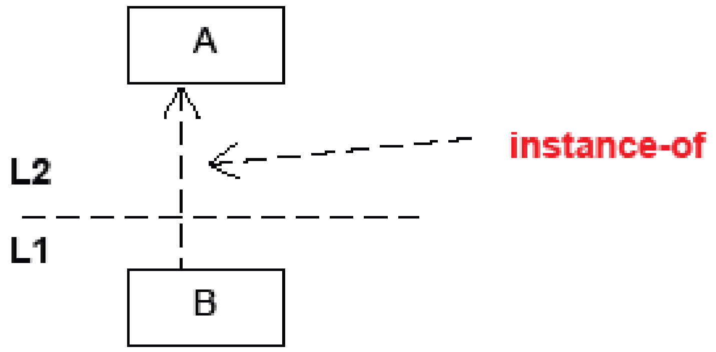
Category class functioning: Superclass + Powertype class

subclass ***or*** *instance-of* ***– THAT is the question***

THE fundamental MLM structuring criterion:



or



Complete feature inheritance

B inherits all types of A

Partial feature inheritance

A has its own objects

THE fundamental pragmatic structuring MLM principle!

MLM-USE: *Explicit dual semantics for Category Classes*

- ***MLM-USE:*** A ***MedMLM*** component built on top of the ***USE*** modeler
- ***MedMLM:*** Mediation-based MLM theory
  - ***Formal:*** Syntax + semantics
  - Semantics defines the ***legal object*** (instance) models of a MedMLM model

- ***Dual semantics*** for ***category classes:***
  - ***Category view:***
    - Data objects: Have category features
    - *Instance* classes – the ***clabjects as objects***
      - Used for defining syntactically *well-defined model*
  - ***Superclass view:***
    - Data objects: Have plain object features
    - Follows standard OO subclass inheritance typing

Warehouse in MLM-USE: Category dual semantics

Category view: *taxRate, soldProductCopyPrice, warehousePspecType, taxRateValue*
Superclass view: *pSpecTypeLabel, currency*

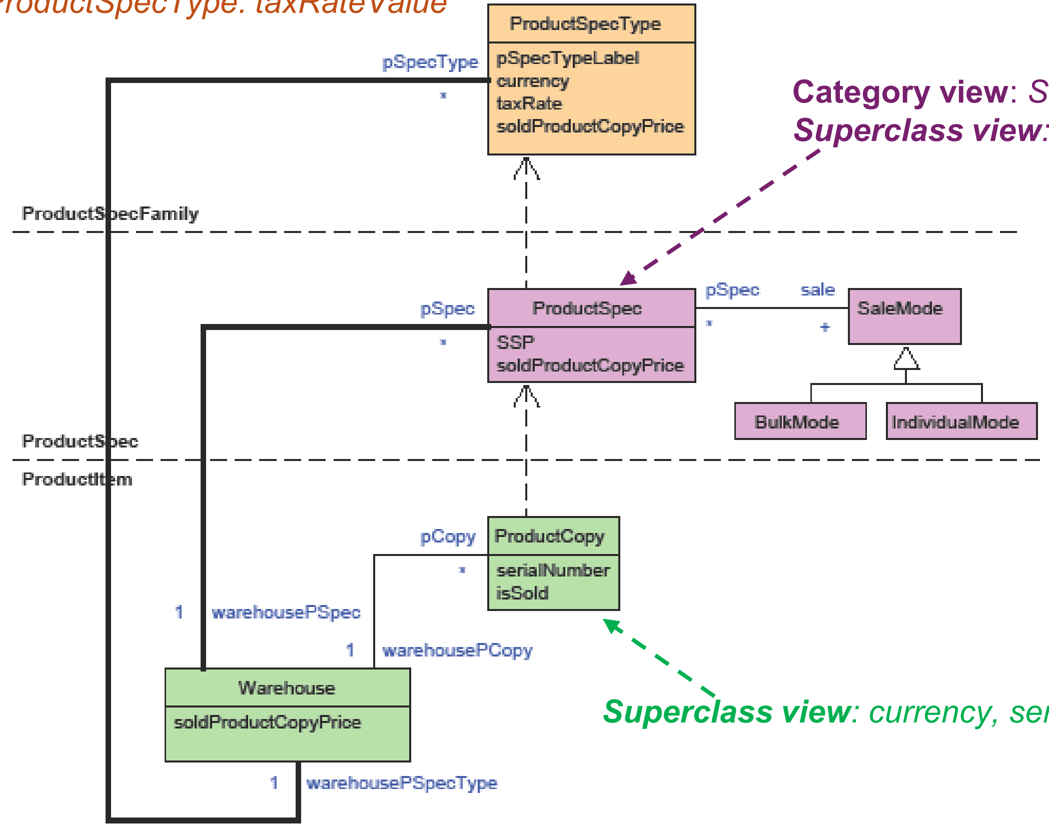
constraint:
ProductSpecType: taxRateValue

attributes

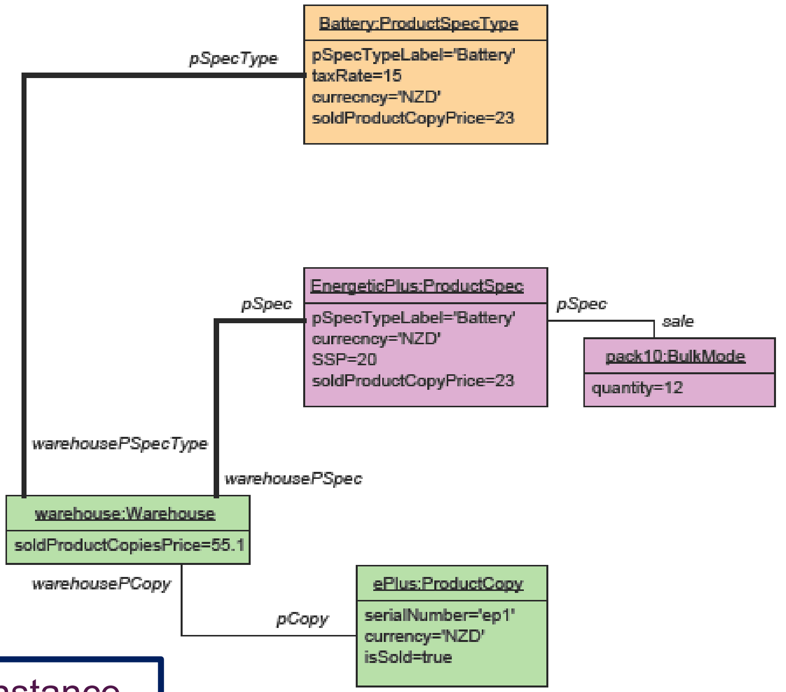
roles

constraint

Category view: *SSP, soldProductCopyPrice, pSpecTypeLabel, sale, warehousePSpec*
Superclass view: *currency*



Superclass view: *currency, serialNumber, isSold*



Part of the warehouse MLM-USE model

Legal instance

Feature inheritance mechanisms in MLM

Inheritance parameters:

- Inheritance range – how inheritance spots are determined:
  - **All** levels, with possible nuances. ← Slicer, Melanee, FMML^{X} , MetaDepth for constraints
  - **Selected** spots. ← Jump potency, MLM-USE

- Inheritance mechanisms :
  - Default inheritance unless overridden ← Slicer, MetaDepth for constraints, MLM-USE
  - Intentional inheritance instructions: Potency marking ← Melanee, FMML^{X} , MetaDepth

- Inheritance forms – what kind of inheritance is implemented :
  - As is – for attributes ← Slicer, Melanee, FMML^{X} , MetaDepth; MLM-USE
  - Value, type restriction – for attributes ← Slicer, Melanee, FMML^{X} , MetaDepth
  - Value finalization – for attributes ← Slicer, Melanee, FMML^{X} , MetaDepth
  - As is – for roles ← MLM-USE
  - As is – for constraints ← MLM-USE; Slicer, MetaDepth

- Kinds of inherited elements:
  - Attributes ← Melanee, FMML^{X}; Slicer , MetaDepth; MLM-USE
  - Roles ← MLM-USE
  - Constraints ← Slicer , MetaDepth; MLM-USE
  - Operations ← MLM-USE

usage: **comparison; evaluation**

- **all levels inheritance →** ***superclass semantics***
  * descendants inherit all features
  * OO type inheritance
- **selected inheritance →** ***combined category-supertype***
  * ancestor – descendent carry different information

Value finalization implies:
- ***Continued inheritance?***
- ***Stopped inheritance?***

MLM modeling idioms

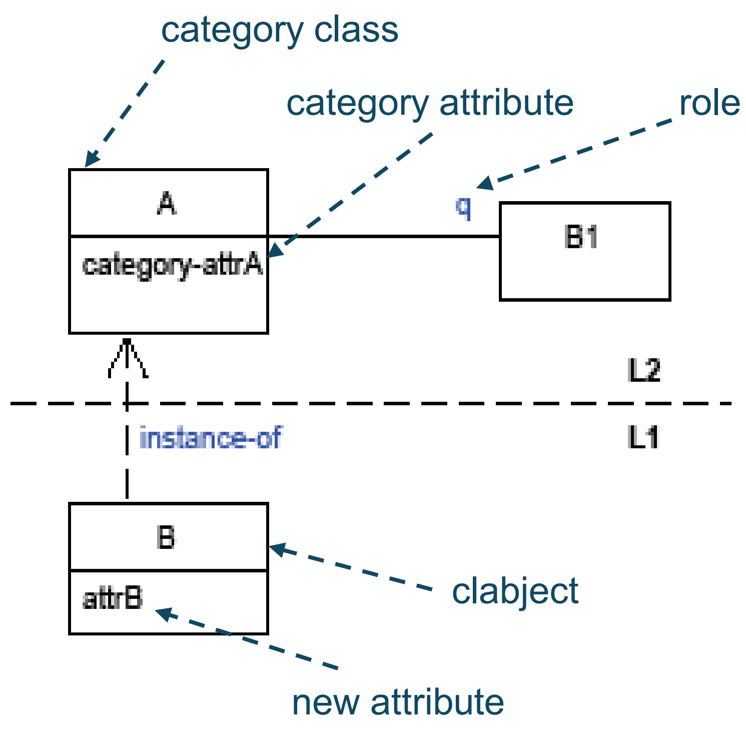
Typical and frequently recurring modeling patterns

Clabject-category interaction idiom:

A Category class and its clabjects might carry different information

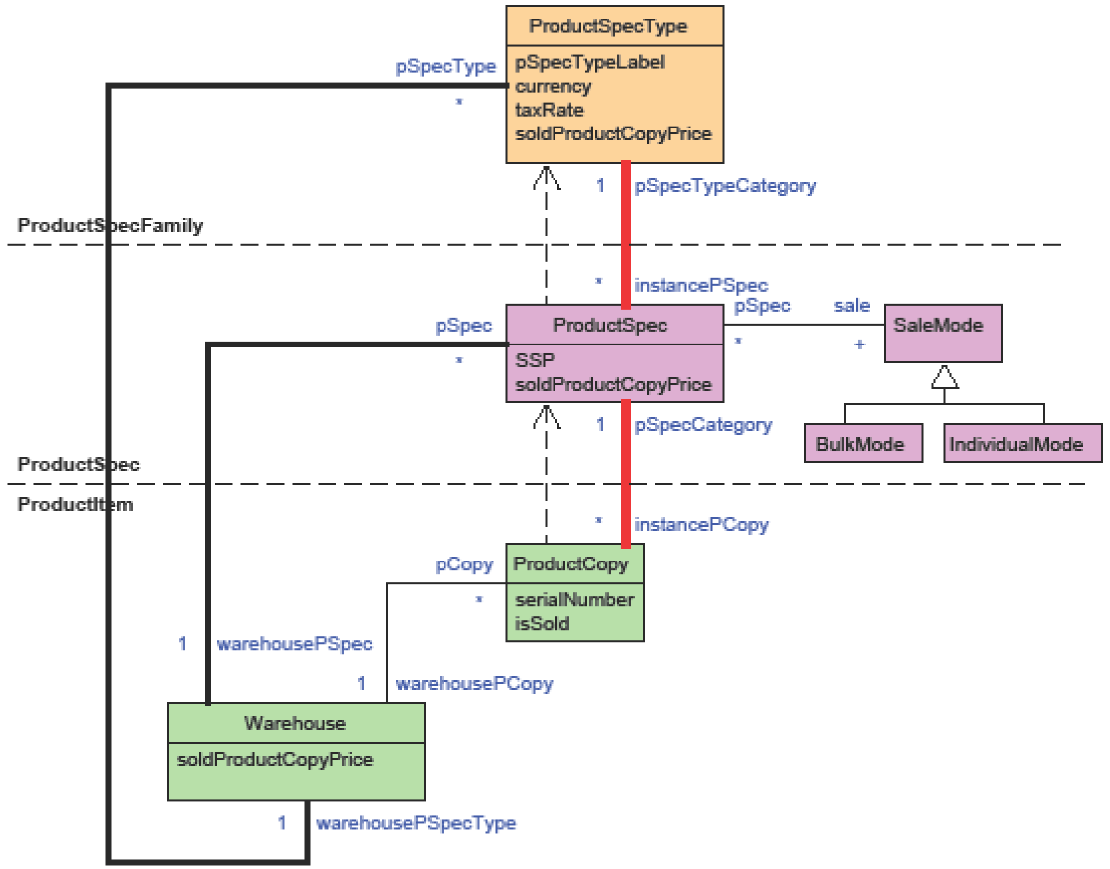
- a *B* object asks for the *A .category-attrA,* or *A.q* of its category object

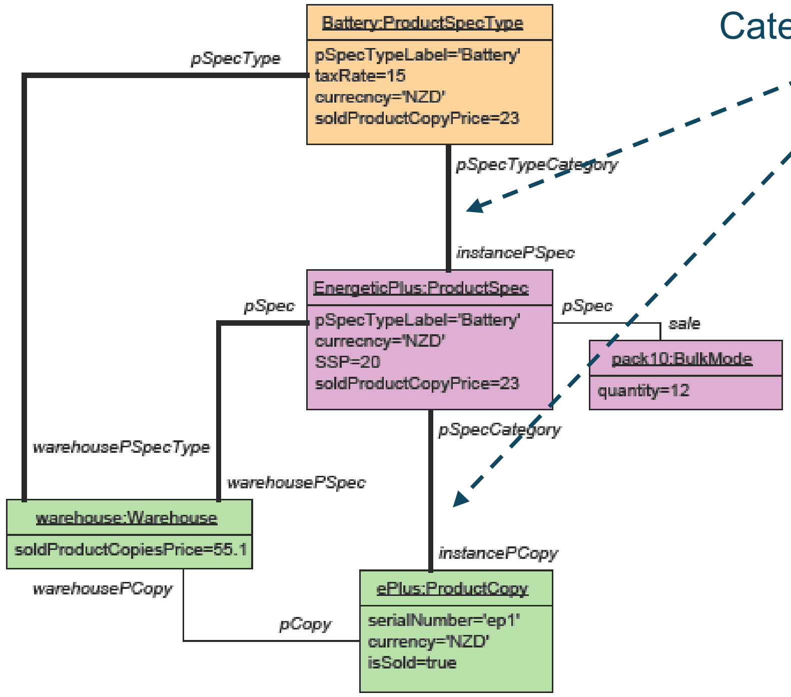
An MLM application should provide an interaction means



Clabject-category interaction in MLM-USE

MLM-USE inserts: Application specific ***Category-Clabject inter-association***





Category-Clabject links

MLM modeling idioms

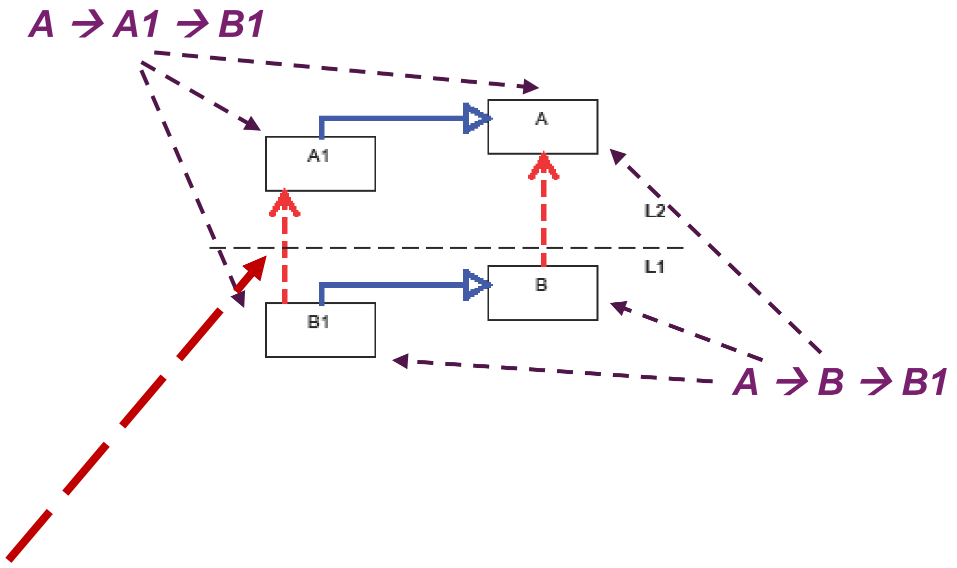
Parallel subclass – instance-of idiom:

MLM applications have two built-in relations between classes that carry feature inheritance:

- *subclass* – superclass-subclass
- *instance-of* – category-Clabject

- Parallel *subclass – instance-of* structures are typical
  - Like parallel inheritance structures in plain OO models
  - ***Multiple inheritance*** problem:
    - ***B1*** inherits ***A*** features in two ways:

      A → B → B1; ***A → A1 → B1***
  - Possible ***inheritance conflicts***:
    - ***B*** and ***B1*** might modify a feature in different ways
  - MLM-USE solution:
    - ***cancel*** the *instance-of* ***inheritance*** between the ***subclasses***



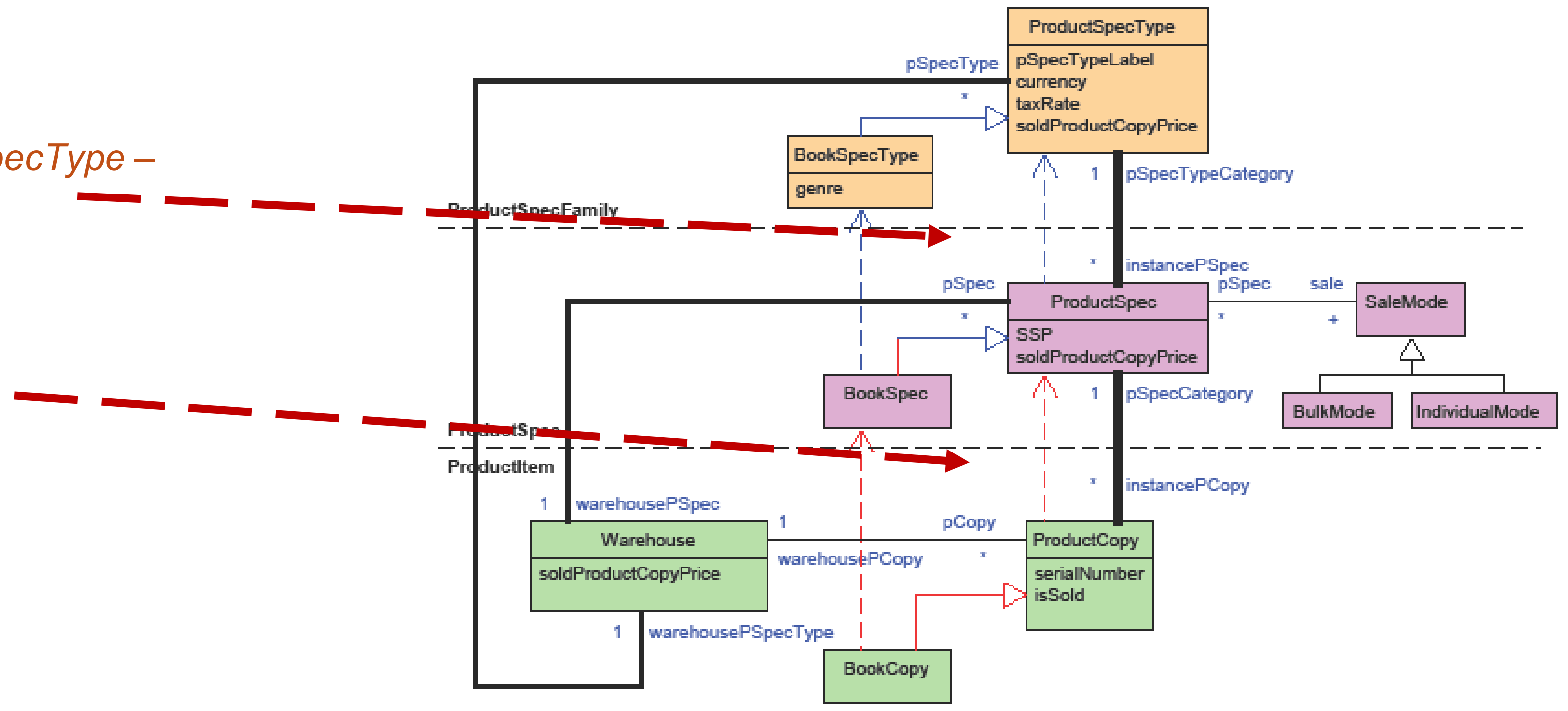
MLM modeling idioms

Parallel subclass – instance-of structures in the MLM-USE warehouse model:

Two parallel structures:

- *ProductSpecType – BookSpecType –*
  ProductSpec – BookSpec

- *ProductSpec – BookSpec –*
  ProductCopy – BookCopy

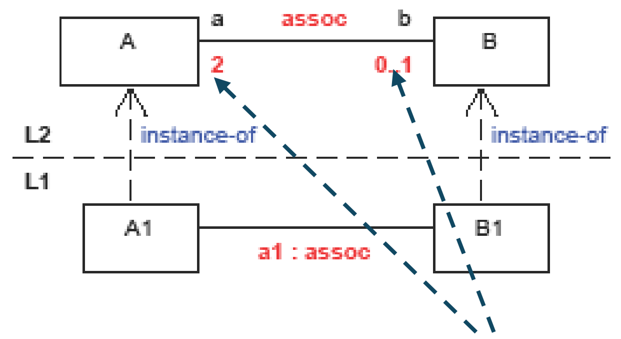


Constraints in MLM

- ***MLM specific*** relations:
  - *instance-of* relations between neighboring levels
  - Relations along levels: *off-spring* and others, e.g., *inter-level associations*
  - *Multiplicity constraints* on associations

- ***Constraint languages*** in MLM applications:
  - Constraint languages with MLM-built-ins:
    - Neighboring built-ins: e.g., XOCL of FMML^x
    - *off-spring* built-ins: e.g., DOCL of Melanee
  - General constraint languages: No MLM specific built-ins
    - MetaDepth: Inter-level global constraints in OCL
    - MLM-USE: three kinds of constraints:
      - Level-local not inherited
      - Level-local inherited
      - Inter-level

- Meaning of ***multiplicity constraints*** between categories:



  - What is the ***target*** of the multiplicity constraints?
    - Clabjects *A1, B1* as objects of categories *A, B?*
    - Data objects of *A, B?*
  - Multiplicity constraints are ***inherited?***
    - Apply to objects of *A1, B1?*
  - Status of *a1* as a ***link of*** *assoc!*
  - Variant model: *A* is ***not a category***!

MLM level structuring

***Placement of a class in an overall model*:**

Minimize interaction between levels :

- *Interlevel associations*
- *Interlevel constraints*
- *Category-view aspects – feature non-inheritance*

***Placement of a non-category class*:**

- Put the class on **lowest possible level**
  - Minimizes interactions

***Placement of a category class*:**

- **Maximize feature inheritance** along *instance-of*:
  - Gain most of OO standard inheritance theory and tools

MLM Pragmatics – Summary and Future

- ❑ ***MLM pragmatic criteria***:
  - ➢ **Category**: ***Combination*** of ***Superclass + Powertype***
    - ➢ ***Decision rule: Subclass vs. instance-of***
  - ➢ **Idiomatic** usage: ***Clabject – category interaction; parallel subclass – instance-of structures***
  - ➢ Characteristics of ***MLM constraint languages***
  - ➢ ***Level structuring*** pragmatics

➡ Set a basis for ***objective MLM benchmark (challenge) evaluation***

- ❑ ***Independent of MLM application specifics***
  - ➢ Language
  - ➢ Tool
  - ➢ Visualization