Extending the Notation for Specialization/Generalization
Proposal for a Discussion Group

M. Dao
France Télécom R&D, DAC/OAT, 38-40 av. Général Leclerc,
92794 Issy-les-Moulineaux cedex 9, France
michel.dao@francetelecom.com

M. Huchard, T. Libourel
LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France
{huchard,libourel}@lirmm.fr

A. Pons
Dept Informatique, Université du Québec à Montréal
C.P. 8888, Succursale centre ville,
Montréal Québec H3C 3P8, Canada
pons.anne@uqam.ca

1 Objective

This paper does not present research results: our motivation is rather to propose a topic for discussion at the MASPEGHI 2003 workshop. It is inspired by several papers of the previous MASPEGHI workshop and by conclusions of our research work about automatic class hierarchy construction and transformation.

The question we would like to ask is as follows: do we think current proposal (UML) for standard specialization/generalization notation is appropriate and sufficient for common *versus* specific use in analysis and design? If this is not the case, which improvements can be proposed?

A first need for precise description of specialization/generalization in diagrams clearly is to help humans, including domain experts, analysts, designers and programmers in elaboration and understandability of models. Besides, in the perspective of the MDA (Model Driven Architecture) approach, which encourages automatic model transformation and mapping operations, accurate annotations on diagrams seem us crucial to avoid misinterpretation of the models by automatic tools and production of meaningless results.

The outline of this position paper is then as follows. First we recall UML notation for specialization/generalization (Section 2). Then further annotations are proposed (Section 3) and we show in Section 4 a possible application of these annotations in the context of class hierarchy transformation (from multiple inheritance towards single inheritance). Directions for discussion at the workshop are given in Section 5.

2 Current Notation for Specialization/Generalization in UML

We first recall the possibilities offered by UML for describing specialization/generalization relationships. They are illustrated on examples inspired by a famous OMT reference [RBP+91] (see Fig. 1 and 2).

In UML 1.4, specialization/generalization links (arrows going from the most specialized concept towards the most general one) can be given with two types of indications: discriminators and constraints on class extension.

Discriminators are associated with specialization links and describe a specialization criterion. In Figure 1[1], class `Employee` is specialized according to two criteria, `status` (related to the salary payment), and `pension` (vested *versus* unvested). Discriminators involve a partition of the specialization links coming in a parent class: in our example this partition has two elements, the set of the links labelled with the discriminator `status` from one side, the set of the links labelled with the discriminator `pension` from the other side.

[1] This figure shows the same class hierarchy expressed in UML 1.4 (two upper parts) and in UML 2.0 (lower part).

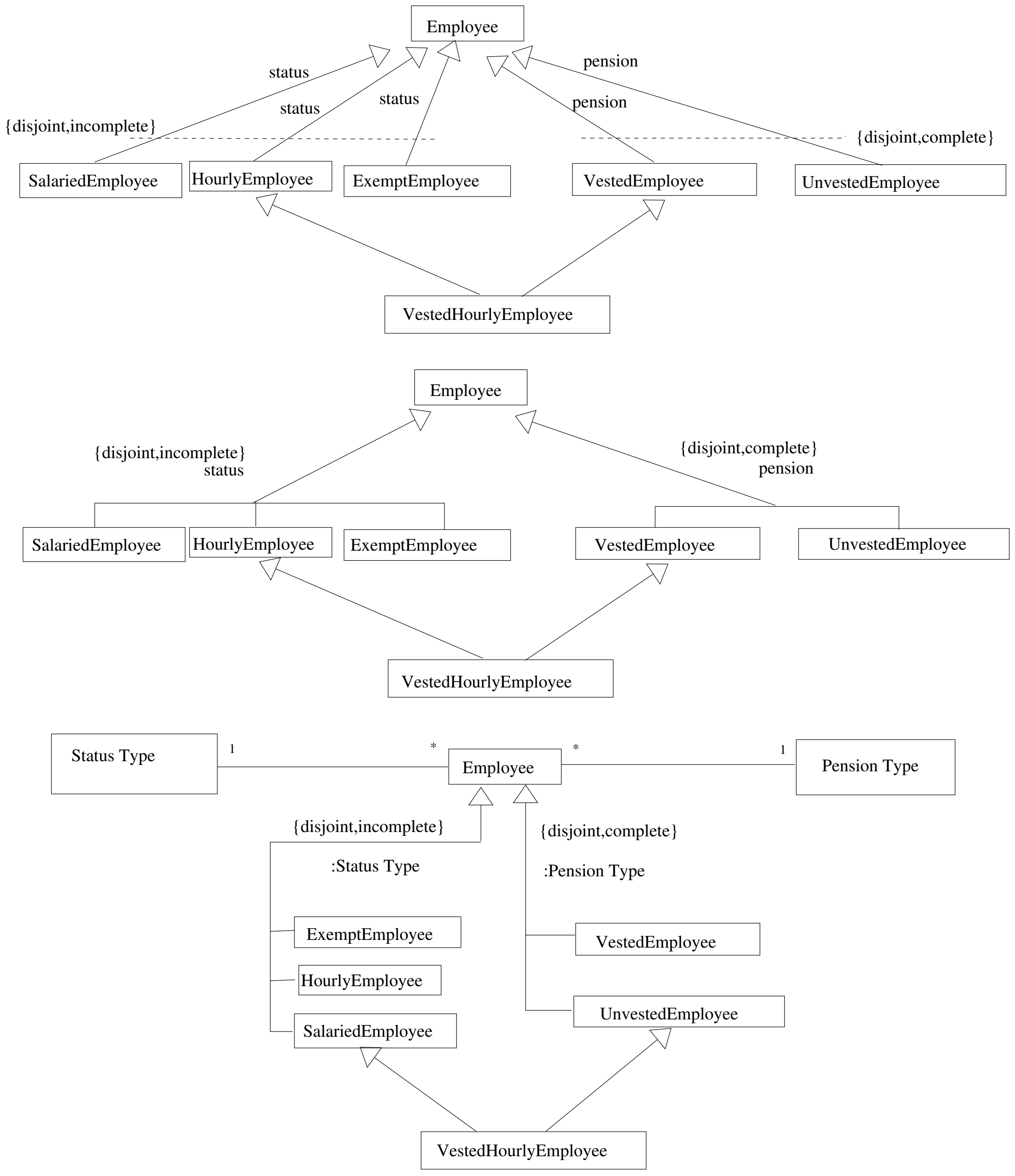

Figure 1. Employee hierarchy with discriminators and constraints (with the different UML possible notations)

The partition can be reduced to one element, as in the case of another example from [RBP+91]. In Figure 2, the only discriminator (`venue`) indicates that the specialization criterion corresponds to the type of venue where the vehicle can move.

A second kind of indications appears as *constraints about the extension* of a class C and of its subclasses. These constraints are placed next to a dotted line which crosses the concerned specialization links. We denote by E the set of the direct subclasses which are gathered by this line. Four constraints are predefined:

- `overlapping` an instance of C can simultaneously be instance of several classes of E. For example (Figure 2), with $E = \{ LandVehicle, WaterVehicle \}$, an instance of $Vehicle$ which would also be instance of $AmphibianVehicle$, is instance of two classes of E.

- `disjoint` an instance C is instance of at most one class of E. This is the case in Figure 1 for classes of $E = \{ VestedEmployee, UnvestedEmployee \}$: an employee will never simultaneously be instance of these two classes.

- `complete` elements of E are origins of links annotated by a same discriminator; any instance of C is instance of one of the elements of E: an employee pension status is necessarily vested or unvested[2].

- `incomplete` the classes of E are origins of links annotated by a same discriminator; an instance of C is not necessarily instance of one of the classes of E; for example, other status can exist for employees.

Version 2.0 of UML specification extends the modeling of specialization/generalization as follows. Firstly, the class `GeneralizationSet` is introduced in the metamodel. It roughly corresponds to the reification of *discriminator*. For instance, in Figure 2, two GeneralizationSet's would be created: one for the *status* regrouping classes `SalariedEmployee`, `HourlyEmployee` and `ExemptEmployee` and another for the *pension type* regrouping classes `VestedEmployee` and `UnvestedEmployee`. The constraints associated to a `GeneralizationSet` remain the same and correspond to two boolean variables `isCovering` and `isDisjoint` declared in class `GeneralizationSet`.

Secondly, the classes belonging to a `GeneralizationSet` must all be instances of the same meta-class (called a `PowerType`) which is represented on the diagrams as shown on Figure 1.

[2]The definition of *complete* in versions 1.4 of UML is unclear whether there may be other subclasses besides those that are part of the complete constraint. UML 2.0 removes this ambiguity and clearly states that the complete constraint does not gather all the subclasses.

Though the expressiveness regarding specialization/generalization is not extended in UML 2.0, the new modeling features presented above may allow to introduce new notations more easily.

3 A few proposals for extending UML notation

Most of these proposals are based on general expertise in modeling. Another source of inspiration is research works on hierarchy classification [MC02] or on meaning of the hierarchical links [Gir91, Cas94, CL02].

alternative this tag may be set on a link connecting a class to its direct super-classes the characteristics of which are used alternatively (and exclusively), as in the case of universal electric machines, input/output flows or amphibian vehicles (see Figure 3).

roles/states this indication states if the specialization relies on states or roles (see Figure 4). It may be completed by precisions on those states or roles as follows:

- concurrent (special case of overlapping for roles and states): father/husband;

- successive (special case of disjoint with a temporal scheduling):
chrysalis/caterpillar/butterfly, child/teenager/adult;

- exclusive (special case of disjoint for roles and states): empty/full for a stack, married/single for a person.

More complex informations on the states may be taken into account by state charts. For instance, a conference paper may be successively submitted, read, evaluated, accepted or rejected (but not both simultaneously), then, if accepted, it may be submitted as a final version, etc. Such a complex state diagram may obviously not be described using the previous tags, but could be used to complete one of them. It should be noted that this type of specialization is a kind of dynamic classification and that class hierarchy specialization/generalization may not be appropriate. Furthermore, it implies to associate several instances to represent one entity.

This point of view may be compared to the one of [MC02], which derives from Aristotle, and distinguishes accidental features of an object from essential ones.

repeated it may be useful to spot the cases of repeated inheritance: a class, such as `Sportsman`, is specialized along different branches that end up meeting, for instance in a class of fusion (`SkaterAndChessPlayer`), as shown in Figure 5. A semantic for this class may be to consider

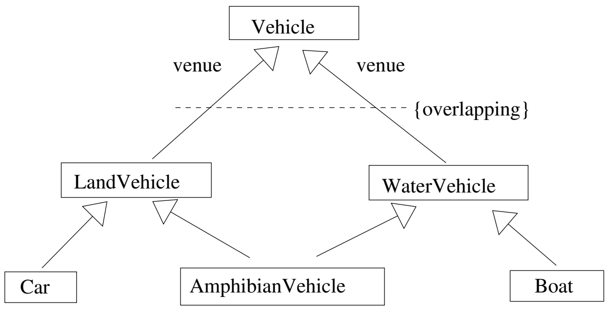

Figure 2. Vehicles

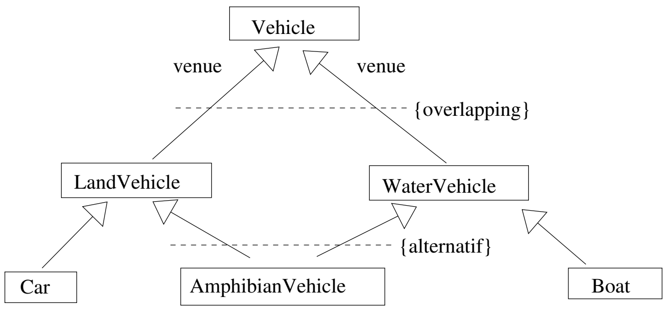

Figure 3. Annotation `alternative`

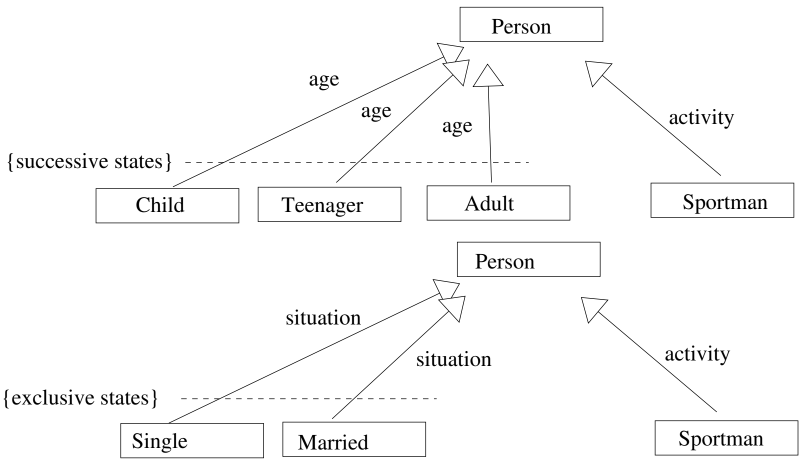

Figure 4. Annotation `roles/states`

that an instance of `SkaterAndChessPlayer` owns as many variables `card` as the number of paths to the class that declares this variable [CG90].

partition coverage when a class is directly specialized according to several criteria (as `Employee` is specialized according to status and pension), an instance may be constrained to belong to at least one class of each element of the partition (each employee must have a status and a pension type). The notation is illustrated in Figure 6.

disjoint partition conversely, an instance may be constrained to belong to only one element of a partition. For example, let us consider a class `Animal` specialized using two criteria: `kindOfSkin` and `kindOfShell`. Suppose that subclasses corresponding to the first specialization criterion are `AnimalWithHair` and `AnimalWithFeather`. The second criterion leads to three subclasses `FixedShell`, `MobileShell`, `Bivalve`.

An animal can simultaneously be instance of `FixedShell` and instance of `Bivalve`, which are two classes of the same element of the partition; this is the case of mussels. Nevertheless an animal cannot be simultaneously instance of a class obtained by `kindOfShell` specialization and of a class obtained by `kindOfSkin` specialization.

The `disjoint` constraint is not adapted to simply express this idea as in our case, it should be placed on several group of subclasses *e.g.* pairs composed of one class from `kindOfSkin` specialization and one class from `kindOfShell` specialization.

domain/implementation in numerous examples of multiple inheritance in the programming area, a class (for instance `Rectangle`) ends up deriving from superclasses that it specializes either to represent the classification in the modeled domain (for instance it specializes `Parallelogram`), or for implementation needs (for instance, it derives from `Serializable` or `Displayable`). If `Rectangle` must be disconnected from some of its super-classes, the link to `Parallelogram` is obviously the one to keep.

4 An application to automatic hierarchy transformation (multiple towards single)

In this section, we show how such notations could be useful for automatic tools which would help designers in model elaboration. We consider the case of tools that would help a designer in transforming multiple inheritance into single inheritance. Several works have yet been published about this subject [CMR02, Rou02, TKH99, Mal01, TB98] which concentrate on transformation techniques. Our approach here is rather to consider the reasons for choosing a transformation or another and we expect that additional informations on hierarchical links can help the process.

The discussion is based on the same example of employees [RBP+91] (see Fig. 7), where a partition expresses a special category of roles for the classifier Employee.

According to the proposals of Section 3 and the existing notations in UML, we analyze the different combinations of constraints that could lead to multiple inheritance.

- For a two-element partition of specialization links (two discriminators), as in the example, multiple inheritance can arise if c is not *disjoint*.
- For a one-element partition, such as the partition elements annotated by c' or c'': if the constraint contains the annotation *overlapping* (or *concurrent* that is a special case for *overlapping*), a multiple inheritance can arise when the constraint is
  - *complete, overlapping or concurrent* - any instance of Employee is instance of 1 or more classes of the partition.
  - *incomplete, overlapping or concurrent* - an instance of Employee is instance of 0, 1 or more classes of the partition.

  Otherwise the constraint is *disjoint* (the special cases for *disjoint* are *exclusive* and *successive*).

The question is how to transform the multiple inheritance into single inheritance knowing the constraints on a partition. For the first case (overlapping), a new class representing combined classes can be added to the model (like `WidowedAndMaried` class for the civil status of a person ; or `ReadingAndWriting` class for the status of a file) at the same level than the classes making up the partition. The resulting duplication of code has not to be so important. More precise guidelines must be given to decide to apply such a transformation. For the second case where the partition has two elements (two classification criteria), the solutions given by [RBP+91] for transforming multiple inheritance into single inheritance could be applied.

- The well-known solution to hook one or both of the partition elements with an aggregation link to the most upper-class (Employee) (with a multiplicity related to the number of possible classes for one instance) instead of inheritance link to the upper-class ;
- adding new classes using nested generalization.

The aggregation solution where one partition element is in aggregation is based on the following subjective criterion:

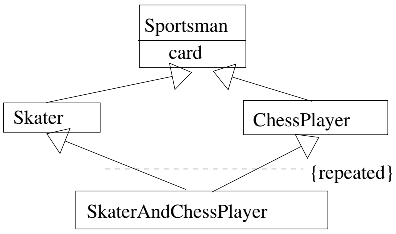

Figure 5. Sportsmen

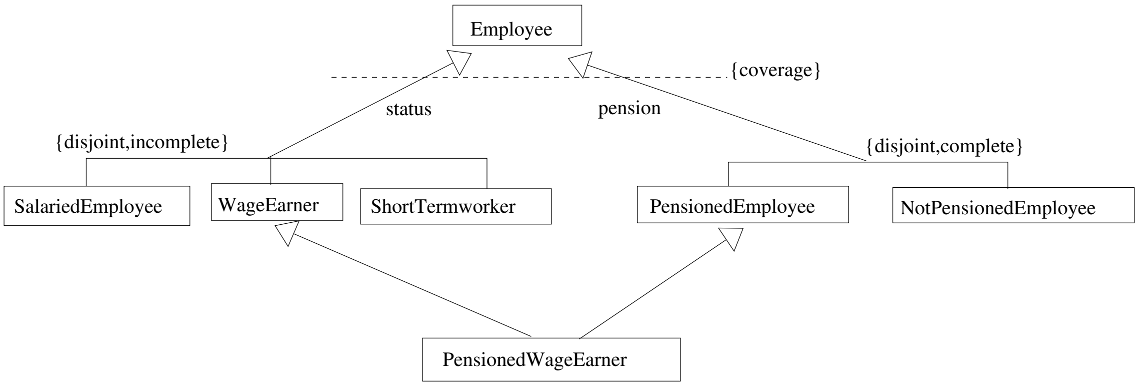

Figure 6. Partition coverage

if a partition element dominates another one, this partition element can remain in inheritance with the most upper class while the other one is transformed in aggregation. This criterion should be more precise to help an automation of a transformation. Nested generalization is also an acceptable solution under certain criteria that could be defined.

Future work is oriented toward the definition of criteria to choose the best transformation of a multiple inheritance to a single inheritance according to the existing constraints, discriminators and designer preferences.

To conclude this short presentation, we see that annotations can partially guide the choice of transformations. Other tools could benefit from such annotations such as tools that help to add a class [DDHL96] or reconstruct the hierarchy [GM93].

5 Basis for the discussion

In the perspective of a discussion during the workshop, we propose some directions for reflection where other participants could contribute; the list is not exhaustive!

- present notations coming from other existing modeling and programming languages which can help us; present other points of view about UML proposals;
- bring examples of hierarchies;
- discuss our preliminary proposals for extending notation; establish links with the literature; propose others!
- discuss possibilities for representing specialization on features (attributes, methods);
- study the semantic aspect of annotations (does there exist denotational semantics for example which found the annotations);
- consider different contexts and purposes for the use of the notation and determine which annotations are appropriate in which contexts (we have given the example of automatic transformation).

References

[Cas94] E. Casais. Automatic reorganization of object-oriented hierarchies. *Object-Oriented Systems*, 1(2):95–115, 1994.

[CG90] B. Carré and J. Geib. The Point of View notion for Multiple Inheritance. *Special issue of Sigplan Notice - Proceedings of ACM ECOOP/OOPSLA'90*, 25(10):312–321, 1990.

[CL02] S. Coulondre and T. Libourel. Towards a New Role Paradigm for Object-Oriented Modeling.

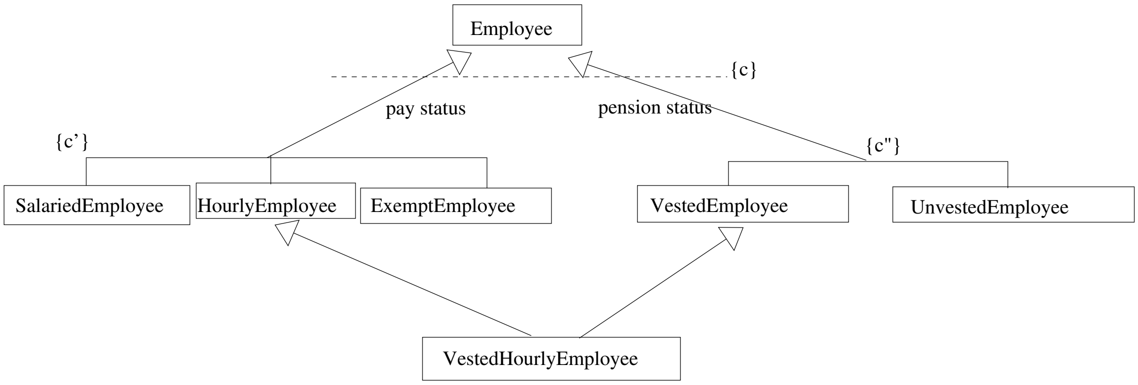

Figure 7. Constraints for employees

In J.-M. Bruel and Z. Bellahsne, editors, *Advances in Object-Oriented Information Systems - OOIS 2002 Workshops*, number 2426, pages 44–52. Springer, 2002.

[CMR02] Y. Crespo, J.-M. Marqus, and JJ Rodriguez. On the translation of Multiple Inheritance Hierarchies into Single Inheritance Hierarchies. In *Proceedings of the Inheritance Workshop at ECOOP 2002, Publications of Information Technology Research Institute, 12/2002, University of Jyvskyl, Black, Enrst, Grogono, Sakkinen (editors)*, pages 30–37, 2002.

[DDHL96] H. Dicky, C. Dony, M. Huchard, and T. Libourel. On automatic class insertion with overloading. In *Proceedings of OOPSLA'96, San Jose (CA), USA*, special issue of ACM SIGPLAN Notices, 31(10), pages 251–267, 1996.

[Gir91] X. Girod. *MECANO, une Méthode et un Environnement de Construction d'ApplicatioNs par Objets*. PhD thesis, Université Joseph Fourier, Grenoble I, 1991.

[GM93] R. Godin and H. Mili. Building and maintaining analysis-level class hierarchies using Galois lattices. In *Proceedings of OOPSLA'93, Washington (DC), USA*, special issue of ACM SIGPLAN Notices, 28(10), pages 394–410, 1993.

[Mal01] M. Malak. Simulating Multiple Inheritance. *Journal of Object-Oriented Programming*, pages 3–5, april 2001.

[MC02] E. Marcos and J. M. Cavero. Hierarchies in Object Oriented Conceptual Modeling. In J.-M. Bruel and Z. Bellahsne, editors, *Advances in Object-Oriented Information Systems - OOIS 2002 Workshops*, number 2426 in LNCS, pages 24–33. Springer, 2002.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design*. Prentice Hall Inc. Englewood Cliffs, 1991. pages 15–84.

[Rou02] C. Roume. Going from multiple to single inheritance with metrics. In F. Brito e Abreu, M. Piattini, G. Poels, and H. A. Sahaoui, editors, *Proceedings of the sixth ECOOP workshop on Quantitative Approaches in Object Oriented Software Engineering (QAOOSE 2002)*, pages 30–37, 2002.

[TB98] Ewan Tempero and Robert Biddle. Simulating multiple inheritance in java. Technical report, Victoria University of Wellington, 1998.

[TKH99] K. Thirunarayan, G. Kniesel, and H. Hampapuram. Simulating Multiple Inheritance and Generics in Java. *Computer Languages*, 25(4):189–210, 1999.