

# Evaluating and Optimizing Factorization in Inheritance Hierarchies

Michel Dao<sup>1</sup>, Marianne Huchard<sup>2</sup>, Thérèse Libourel<sup>2</sup>, and Cyril Roume<sup>2</sup>

<sup>1</sup> France Télécom R&D, DAC/OAT, 38-40 av. Général Leclerc,  
92794 Issy-les-Moulineaux cedex 9, France  
michel.dao@rd.francetelecom.com

<sup>2</sup> LIRMM, 161 rue Ada,  
34392 Montpellier Cedex 5, France  
{name}@lirmm.fr

**Abstract.** Inheritance hierarchies often constitute the backbone of object-oriented systems. Their quality is therefore quite crucial. We present here our work the objective of which is to help designers to improve and to measure the quality of their inheritance hierarchies. The quality we deal with is the degree of factorization of information. Our work is based on the mathematical structure of Galois (concept) lattice which ensures, when applied to inheritance hierarchies, a maximal factorization of properties (methods and attributes) along with a minimal number of classes. Furthermore, the links between Galois lattice concepts have a direct correspondence with the inclusion relationships of class property sets in inheritance hierarchies.\*

## 1 Introduction

Inheritance may be defined as the mechanism used in object-oriented languages to implement specialization-generalization relationships between classes. In particular, it offers a way to share (factorize) properties (methods and attributes) between classes. An inheritance (or class) hierarchy usually constitutes the backbone of an object-oriented system and its quality is therefore very important. Though the initial version of a class hierarchy is often built with care, it may contain some flaws regarding the factorization of information. Furthermore, as the hierarchy is modified through corrective or evolutive maintenance operations, more flaws may be introduced that may reduce both system performance and ease of maintenance. For instance, multiple occurrences of a method along an inheritance path may add a time overhead to the invocation mechanism due to the resolution of name conflicts; if an attribute is defined in several sub-classes while it should only be present in one super-class, modifications of the attribute must be applied to all occurrences which is time consuming and error-prone. In both cases, multiple unnecessary declarations of properties make the class hierarchy less easy to understand and to use.

Our work is based on the assumption that a hierarchy is “ideally” factorized if the number of classes is minimal and the factorization of the properties is maximal. We therefore believe that object-oriented system designers should be supplied with tools to help them ensure that the class hierarchy on which they built up their systems contains as few flaws as possible. Those tools should both give them a means to measure the “quality” of their class hierarchies and to improve it.

The sequel of this paper presents three aspects of our work:

- the mathematical structure that supports our definition of the ideal factorization;
- some algorithms used to reorganize a hierarchy so that it conforms to the ideal factorization;
- a set of metrics that we have designed in order to measure the distance between the ideally factorized hierarchy and the existing one.

We conclude by discussing the links between the evaluation and optimization approaches.

---

\* This work was supported by France Telecom Research and Development center (contract 97 1B 602) and the french national network on software technologies RNTL. The framework is MACAO, a common project of France Telecom R&D, SOFTEAM and LIRMM (CNRS and University Montpellier 2.)

## 2 The Underlying Mathematical Structure

Our approach is based on the mathematical structure of Galois lattices [BM70]. Galois lattices (or concept lattices [Wil82,DP90]) are used in several domains such as knowledge representation, machine learning (conceptual clustering), data mining, classification and software engineering [GMM<sup>+</sup>98]. A Galois lattice is the result of the formation of *concepts* associated with a binary relation between *entities* (in a broad sense) and their *features*. A *concept* is a pair  $(X, Y)$ , where  $X$  is a set of entities and  $Y$  a set of features, such that all entities of  $X$  share all (and no more) features of  $Y$ , and symmetrically all features of  $Y$  belong to all (and no more) entities of  $X$ .  $X$  is usually called the extent and  $Y$  the intent of the concept. Concepts are arranged in a partial order, deduced from the inclusion of extents (symm. reverse inclusion of intents) that can be interpreted as an inheritance or specialization/generalization relation. In the pragmatic approach introduced by [GM93], only concepts carrying informations are kept, leading to a noteworthy suborder called here the Galois sub-hierarchy. The Galois sub-hierarchy can be defined by a simplification of the Galois lattice as proposed in [GM93,GMM<sup>+</sup>98].

In the case of object-oriented class hierarchies, concepts correspond to classes and features to properties (methods and attributes). Left part of Figure 1 shows both a class hierarchy ( $H$ ) and the corresponding binary relation. There exists several ways to build the binary relation from the class hierarchy. The most straightforward consists in considering that there exist as many features as syntactically different occurrences of properties. We have followed this assumption during this first part of our work, in particular for the design of the CERES algorithm (see Section 3). This is also how we have dealt with attributes in our example where **weight**, **tax** and **discount** are three different properties. In that case, we consider that a class is in relation with every attribute that it declares or inherits.

During the development of our research, we have been led to consider that beyond syntactical identity, semantically close properties should be grouped into units that we call « generic properties » and that those structures should be used to build the class/property binary relation (see Section 3). The construction of the generic properties may be carried out by heuristics algorithms or by the designer of the class hierarchy (both ways may be combined). We have used the generic property approach for methods in our example.

We also make the assumption that we work at a conceptual level which in particular means that if a property is declared twice in the initial hierarchy, both occurrences will be represented by the same unique property in the Galois sub-hierarchy (see attribute **weight** for an example of this situation). We are aware that such a situation is not possible in some object-oriented languages (like SmallTalk) whereas other accept multiple declarations of the same attribute along an inheritance path (Java or C++ for instance). This has two consequences:

- initial conceptual class hierarchies coming from OO languages such as SmallTalk will not contain multiple declarations, but those coming from other languages or UML will;
- the Galois sub-hierarchy will certainly not have any duplicates. In case a designer should want such duplicates, he should give different names to them, possibly grouping them inside a common generic property.

## 3 Class Hierarchy Reorganization

We have designed several algorithms based on the Galois sub-hierarchy to propose different levels of reorganization.

The first one, ARES [DDHL96], deals with the insertion of a new class into a class hierarchy: it can be viewed as an incremental inserting algorithm in a Galois sub-hierarchy.

The second algorithm CERES [HL00,DHL<sup>+</sup>b] is an original global algorithm that, given a set of classes and properties and their relations as described in Section 2, computes the corresponding Galois sub-hierarchy (see Figure 1). In other words, this algorithm transforms any class hierarchy into one where properties are maximally factorized with a minimum number of classes and in which inheritance links directly correspond to specialization/generalization relationships between concepts. We are currently working on a more general global algorithm that will enhance CERES in two ways.

Firstly, instead of taking into account properties (methods and attributes) as isolated syntactical units (in the same way as we deal with attributes in our example of Section 2), they will be grouped

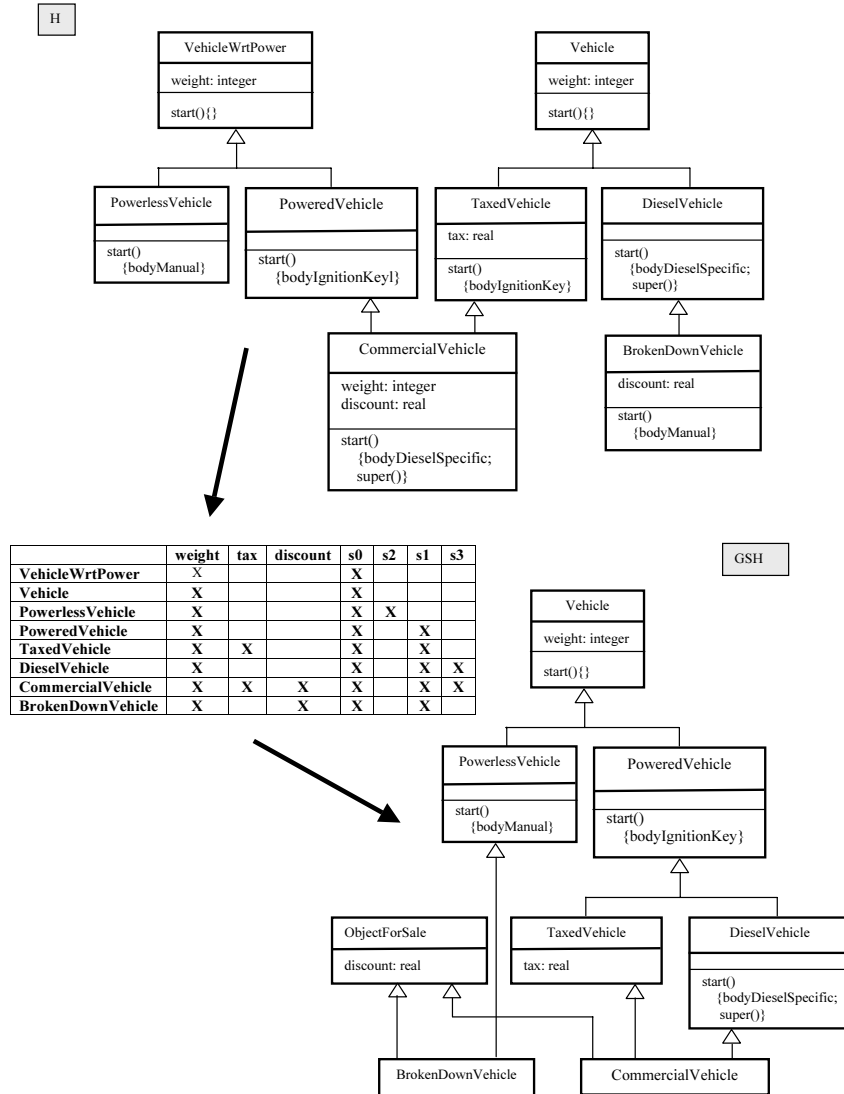


Fig. 1. Class hierarchy, binary relation and Galois sub-hierarchy

into clusters denoting a semantic unity. We call those clusters “generic properties”. Inside such a generic property, properties are ordered by a specialization relationship, such as subtyping for attributes or body specialization for methods (through the use of the `super` keyword for instance).

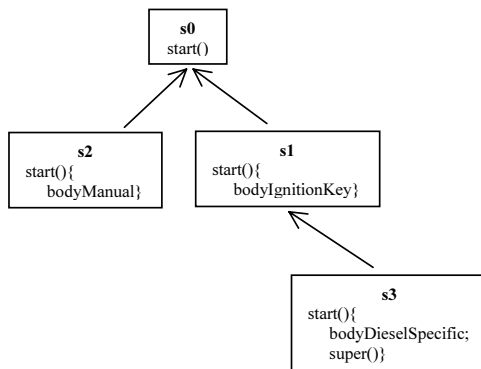
Figure 2 shows an example of such a generic property: the generic property gathering all occurrences of method `start`. In this generic property, we have named `s0`, `s1`, `s2` and `s3` the four occurrences of method `start` in the initial hierarchy and we use these names in the binary relation. We assume that the order in this generic property was computed using the following heuristics: the empty method `start` (`s0`) generalizes every other method and method `si` specializes method `sj` if `sj` can be called from `si` through the keyword `super`.

This generic property is used in the example of Figure 1: a class  $C$  is in relation with every occurrence of a method `start` that i) is declared in  $C$ , or ii) is inherited and not overridden by  $C$  or a super-class of  $C$ , or iii) is a generalization of either i) or ii) inside the generic property of `start`. For instance, as `s1` generalizes `s3`, `DieselVehicle` is in relation with `s1` through the declaration of `s3`, even if it does not declare or inherit `s1`. One should also note that, despite `BrokenDownVehicle` is a sub-class of `DieselVehicle`, it is not in relation with `s3` because it would violate the specialization order of the

generic property. Such a violation is also detected by one of our metrics (see Section 5). We are currently working on several heuristic algorithms for the construction of generic properties.

Secondly, we will modify CERES algorithm so that it takes into account UML associations. For example, the fact that the same role name is present in two different classes should impact the position of those classes in the inheritance hierarchy in a similar way as the declaration or the inheritance of properties does.

Those enhancements will hopefully lead to an algorithm that produces still better class hierarchies, that is where factorization is still better implemented.



**Fig. 2.** The generic property `start()`

## 4 Inheritance Metrics

Designers are not always inclined to use fully automatic tools to partially or completely reorganize their class hierarchies. Besides, external constraints may prevent some classes or some parts of the hierarchy to be modified. For instance, the use of a framework or of a class library may constitute such a constraint. This observation led us to consider useful to offer a set of metrics that would help a designer to evaluate to which extent the class hierarchy differs from the ideal one deriving from the Galois sub-hierarchy.

Our set of metrics may be classified as follows according to which level of system they apply (see [DHL<sup>+</sup>01,Rou02] for a complete presentation and [DHL<sup>+</sup>a] for a case study):

- property level: redundance and position of property declarations;
- generic property level: specialization/generalization of properties;
- class level: property grouping;
- hierarchy level: differences with the reference structure (Galois sub-hierarchy).

Metrics concerning some aspects of inheritance have been proposed ([LK94]), but, to our knowledge, this is the first coherent and consistent set of metrics for inheritance in object-oriented systems.

## 5 Discussion

We have developed two complementary approaches that should help a designer to improve the quality of his class hierarchies based on a common underlying mathematical structure, the Galois sub-hierarchy (*GSH*). The first one proposes algorithms to reorganize those hierarchies so that they conform to the *GSH*; the second one has defined a set of inheritance metrics that measure the distance between the hierarchy and the *GSH*. We have tested those two approaches on several parts of the Java JDK (see [DHL<sup>+</sup>a,DHL<sup>+</sup>b]). In the limited space of this article we will give only a small insight of the practical link between the two approaches.

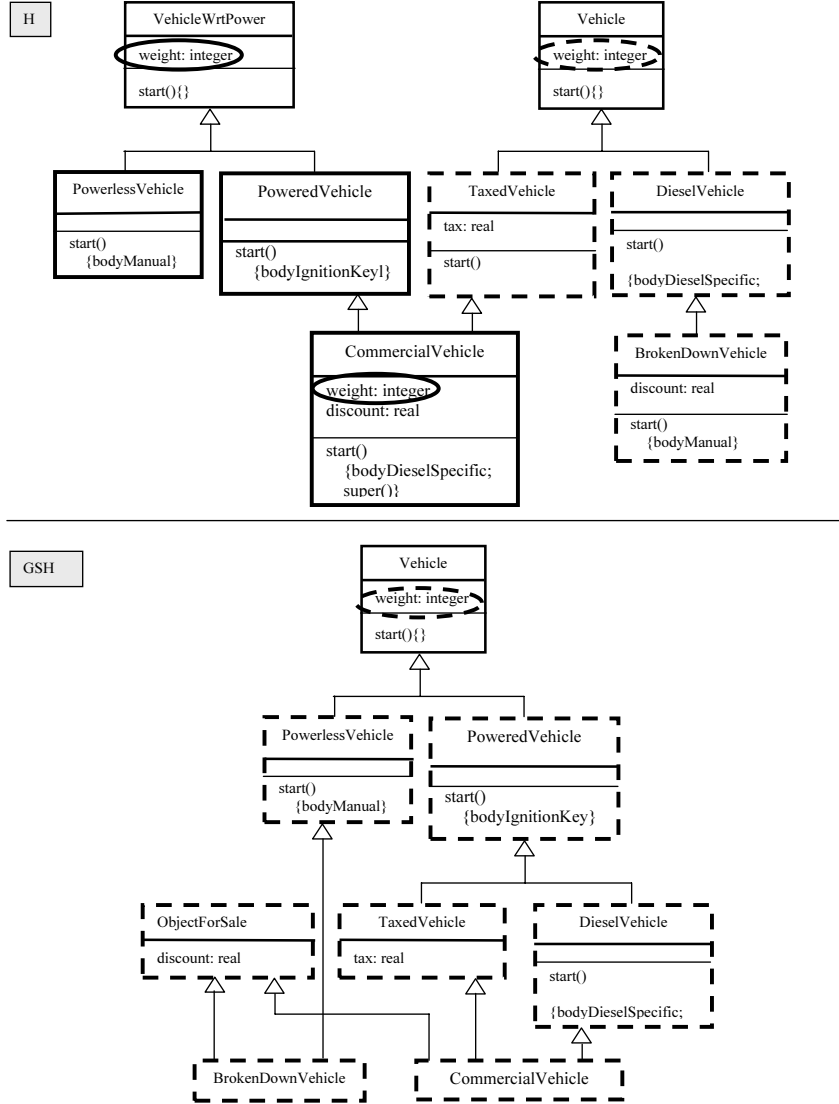


Fig. 3. Examples of the links between inheritance metrics and the GSH

One of the property level inheritance metrics that we have defined measures the *Number of Redundancies* of a property  $p$  ( $RN(p)$ ). For instance, the upper part of Figure 3 ( $H$ ) shows that attribute `weight` is declared three times hence giving a value of 2 to  $RN(\text{weight})$  metric (3 minus one useful declaration), a desirable value being zero (no redundancies). Another example is  $RN(\text{discount})$  which is equal to 1. By contrast, in the lower part ( $GSH$ ) both `weight` and `discount` are only declared once, thus yielding a value of zero for their  $RN$  metric.

A more informative metric is  $RR$  (*Redundancy Ratio*) which is defined, for a property  $p$ , as  $RN(p)$  divided by the number of classes that declare or inherit  $p$ . In our example,  $RR(\text{weight})$  is equal to .25 (2/8) and  $RR(\text{discount})$  is equal to .5 (1/2) which shows that `discount` is rather better factorized than `weight` in hierarchy  $H$ . Obviously,  $RR(\text{weight})$  and  $RR(\text{discount})$  are both equal to 0 as there is no redundancy ( $RN(\text{weight}) = RN(\text{discount}) = 0$ ) in  $GSH$ .

Some generic property level metrics aim at evaluating the discrepancies between the generic properties specialization orders and the inheritance hierarchy. For a given generic property  $P$ ,  $NIHR$  (*Number of Incorrect Hierarchy Relations*) is the number of inheritance links that violate the order of  $P$ . In our example, inheritance link between `DieselVehicle` and `BrokenDownVehicle` is in contradiction with the fact that, inside  $P$ , `s3` is a specialization of `s1`. Thus  $NIHR(\text{start}()) = 1$ . As for  $RN$ , we have defined a

metric *IHRR* (*Incorrect Hierarchy Relation Ratio*) that gives the ratio of incorrect links to correct ones. Here  $IHRR(start()) = 1/9$  which is a low value. The ideal value for both *NIHR* and *IHRR* is equal to zero as no inheritance link should violate a generic property specialization order.

A last example concerns the metrics *RCCF* which measures the role of a class in the factorization of a property. For a class *C* declaring a property *p*, *RCCF* is defined as the ratio between the number of sub-classes of *C* (including *C*) and the total number of classes that declare or inherit *p*. Thus for attribute *weight*, both class *VehicleWrtPower* and *Vehicle* have a *RCCF* value of .5 (4/8). An ideal value is 1 which is the case for the *GSH* where attribute *p* is declared once in the root class.

One should note that whereas the global reorganization algorithm (CERES) constructs the ideal *GSH*, the inheritance metrics help to evaluate different and complementary aspects of the quality of the inheritance hierarchy against the ideal *GSH*.

We have applied both reorganization algorithms and the computation of inheritance metrics to some packages of the Java JDK. Details on these experiments may be found in [DHL<sup>+</sup>01,DHL<sup>+</sup>a].

## 6 Conclusion

In this article, we have presented our work concerning the evaluation and the optimization of inheritance hierarchies. The whole of this work is based on the Galois sub-hierarchy structure which allows to produce inheritance hierarchies where the number of classes is minimal and the properties are maximally factorized. Our approach allows for two complementary ways to help maintenance and improvement of class hierarchies: one proposes reorganization algorithms that modify the structure of the hierarchy while the other offers means to measure the quality of a hierarchy as regards different aspects of factorization.

## References

- [BM70] M. Barbut and B. Monjardet. *Ordre et classification – Tome 1*. Hachette, 1970.
- [DDHL96] H. Dicky, C. Dony, M. Huchard, and T. Libourel. On automatic class insertion with overloading. *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'96.*, 31(10):251–267, 1996.
- [DHL<sup>+</sup>a] M. Dao, M. Huchard, H. Leblanc, T. Libourel, and C. Roume. A New Approach to factorization – Introducing Metrics. In *Proc. Metrics 2002 – 8th International Software Metrics Symposium*, Ottawa, Canada.
- [DHL<sup>+</sup>b] M. Dao, M. Huchard, H. Leblanc, T. Libourel, and C. Roume. CERES, Concept Hierarchy Restructuring. Application to Interface and Class Hierarchies. *Submitted to Journal of Object Technology*.
- [DHL<sup>+</sup>01] M. Dao, M. Huchard, H. Leblanc, T. Libourel, and C. Roume. Towards a metric suite for evaluating factorization and generalization in class hierarchies. In F. Brito e Abreu, B. Handerson-Sellers, M. Piattini, G. Poels, and H. A. Sahraoui, editors, *Quantitatives approaches in Object-Oriented Software Engineering (workshop ECOOP 2001)*, *Wettelijk Depot D/2001/Geert Poels, uitgever, ISBN 90-806472-1-7*, Lisbonne, Portugal, 2001.
- [DP90] B.A. Davey and H.A. Priestley, editors. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
- [GM93] R. Godin and H. Mili. Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'93*, 28(10):394–410, 1993.
- [GMM<sup>+</sup>98] R. Godin, H. Mili, G. Mineau, R. Missaoui, A. Arfi, and TT Chau. Design of Class Hierarchies Based on Concept (Galois) Lattices. *Theory And Practice of Object Systems*, 4(2), 1998.
- [HL00] M. Huchard and H. Leblanc. Computing Interfaces in Java. In *Proc. IEEE International conference on Automated Software Engineering (ASE'2000)*, pages 317–320, 11-15 September, Grenoble, France, 2000.
- [LK94] M. Lorentz and J. Kidd. *Object-Oriented Software Metrics, a Practical Guide*. Prentice Hall, 1994.
- [Rou02] C. Roume. Evaluation structurelle de la factorisation et la spécialisation au sein des hiérarchies de classes. Introduction de métriques. In *Langages et Modèles à Objets 2002*, volume 8/1-2 of *RSTI – Série L'objet*, Montpellier, France, January 2002. Hermès Science.
- [Wil82] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concept. In I. Rival, editor, *Ordered Sets*, pages 445–470. Reidel, Dordrecht/Boston, 1982.