

R. Godin¹, M. Huchard², C. Roume², and P. Valtchev³

¹ Département d'Informatique, UQAM, C.P. 8888, succ. "Centre Ville", Montréal, Québec, Canada, H3C 3P8

² LIRMM, 161 rue Ada, 34392 Montpellier cedex 5, France

³ DIRO, Université de Montréal, C.P. 6128, Succ. "Centre-Ville", Montréal, Québec, Canada, H3C 3J7

1 Introduction

In object-oriented applications, the core of systems is a class hierarchy, that organizes concepts of the application domain or software artifacts useful in the development. Design, implementation and maintenance of specialization/generalization hierarchies are difficult tasks [45, 7], due to the size of the hierarchies, the numerous and often conflicting criteria (see for example [33, 20]) which can be used in the specialization/generalization process, as well as the natural evolution in the domain and in the knowledge about that domain, which has to be reflected by the hierarchy structure.

Real challenges for modern object-oriented CASE tools include the automated support for class hierarchy manipulations at any stage of the development life-cycle (domain modeling, analysis, design, implementation, maintenance and re-engineering).

In this paper we present a short survey of existing work on automation of class hierarchy manipulation in both OOP and OODBMS. Our study represents an extension and an update of previous studies on the domain reported in [12] and in [31] which have been carried out from a perspective different from ours. In the same time, the careful examination of the relevant aspects in the concurrent approaches helps us highlight some directions for future research.

The paper starts by a list of common scenarios that involve various reworking tasks for class hierarchies (Section 2). The external parameters of the proposed methods, such as the input and output data format, are discussed in Section 3. A special attention is paid to the quality criteria that are used to guide the hierarchy manipulations (Section 4). Finally, the algorithmic aspects of the examined methods are outlined (Section 5). To conclude, we draw some perspectives for future work (Section 6).

2 Hierarchy manipulation scenarios

The automated tools can clearly bring added value during the key processes that mark the class hierarchy life-cycle, in particular its initial design or its complete re-engineering. However, there is a wide range of other situations in which the structure of the class hierarchy undergoes changes of smaller scope, that can also benefit from automation.

In the literature, the following manipulation scenarios have been considered for complete or partial automation:

- **construction of the hierarchy from scratch:** based on a set of objects of some of the target classes as in [36, 35] or based on high-level specifications of the classes as in [25] or in [19],
- **evolution of the current hierarchy** through a set of restructuring operations:
 - unconstrained, class-addition-driven restructuring [10, 27, 17, 18],
 - under compatibility constraints, for example backward compatibility (with a previous version of a class library [42]) or compatibility with existing instances [28, 8],
 - prompted by the discovery of significant usage patterns [34],
 - aimed at improving the values of a set of software metrics [1, 47].
- **reconstruction of the entire class hierarchy** based on a sub-set of the information encoded in the current hierarchy:
 - using a set of existing classes and the binary relation "owns" that relates them to the properties (attributes and/or methods) they admit [9, 15],
 - under constraints of preserving a fixed sub-set of classes (for example the set of all non-abstract classes [25]),
 - constraint by a set of the known uses of the hierarchy in applications [48],

- **merge of already existing partial hierarchies**, for example during the aspect inter-weaving task in aspect-oriented development [49].

The above list could have been extended by further works on hierarchy manipulations that rather put the emphasis on the definition of a set of atomic operations such as add/drop/move of classes/properties/inheritance links, split of a class into several classes or merge of a set of classes into a new class:

- for purposes of code refactoring [40, 22],
- for purposes of hierarchy evolution in OOP and OODB [11, 4],
- for purposes of hierarchy optimization [5] (under the constraint of preserving the semantics of instanciable classes).

Although these operations are integrated into environments assisting hierarchy restructuring (e.g., "Refactoring Browser" [43]), and their execution is completely automated, the decision about their invocation is left to the user of the environment.

In the following, we examine the existing approaches and shed light on their similarities and distinctive features. For our study, we selected a set of key aspects that enable a meaningful comparison.

3 External parameters of the approaches

The context in which the hierarchy manipulations have been applied is of high importance to our study as it embodies a set of initial hypotheses that are not always explicitly stated and that often limit the scope of the proposed algorithmic solutions.

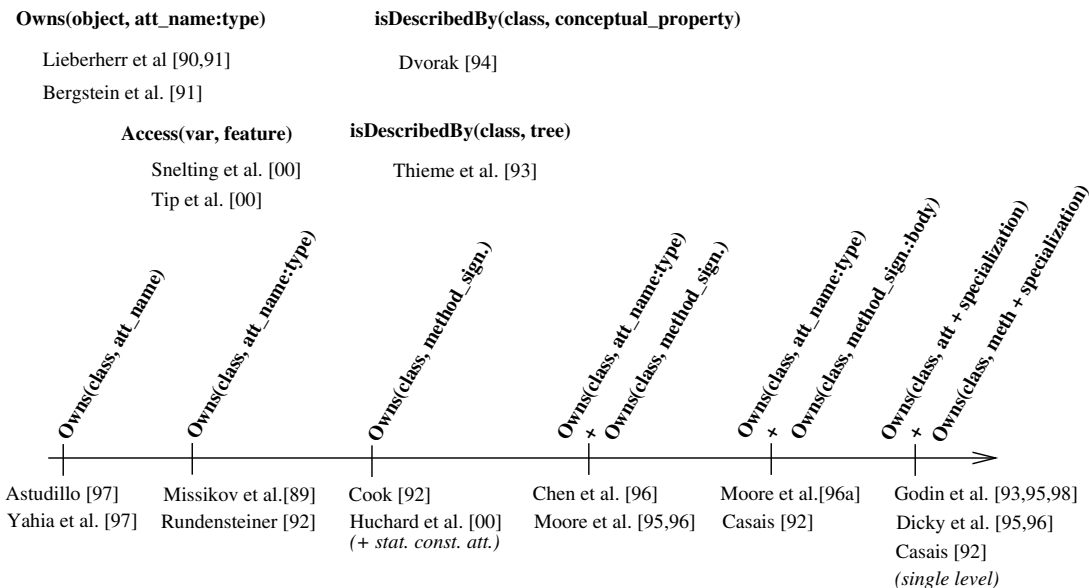


Fig. 1. Classification of approaches with respect to the data description formalisms used.

In this sense, the *input data format* (or the data description formalisms) of the methods constitutes a key distinction criterion since it has a strong impact on both the precision of the obtained hierarchy and the complexity of the algorithmic methods. Whereas most of the methods consider class descriptions, which typically represent lists of **class-owns-property** expressions, some of the advanced approaches focus on objects. Moreover, the former group splits in several subgroups with respect to the kinds of properties admitted: pure attributes vs pure methods vs both attributes and methods. The granularity of the property descriptions discriminates even further the approaches (e.g., methods described by their signatures or by signature and body). Finally, specialization relationships between properties are taken into account by some of the methods.

In Figure 1, a tentative classification of the approaches with respect to the data description formalisms, is given. In its lower part, a set of class-based approaches are situated on an axis that models the precision

of class descriptions with the methods using least precise descriptions drawn on the left. The 'outliers', i.e., methods using different kinds of input data, are situated above the axis. Curiously enough, these are the methods which explore the information beyond the immediate class description, i.e., the set of pairs of the *owns* relation between classes and properties. All these methods study associations between classes and/or links between objects, although for varying purposes.

The context of the study, e.g., the intended programming language, may impose further constraints on both the input and the output of the proposed methods. For instance, the multiple inheritance that usually characterizes the obtained class hierarchies may not fit to the chosen programming framework (e.g., Java or SmallTalk). Thus, the approaches described in [25, 48] include indications for transforming the output hierarchy that may contain multiple inheritance cases into a single-inheritance one.

Another characteristic of the output is the distinction among the concepts of the target hierarchy according to their nature: whereas most of the methods deal with classes and interfaces in an indiscriminate manner, others focus on interfaces [15, 2, 30].

4 Hypotheses and quality criteria

The examined studies are founded by a set of ideas about what a target hierarchy should be, most often (but not always) expressed as explicit external quality criteria [37]: simplicity, comprehensibility, usability (easy to use), good performance (size and time complexity), reliability, reusability, extensibility, maintainability, low development cost, etc.

The intended improvement with respect to the above criteria is measured as the gain in some internal, and more objective, characteristics of the hierarchy that are believed to contribute to their respective achievement. We tend to distinguish two types of characteristics: the first one focusing on the structural aspects of the hierarchy, and the second one on intended purposes of the hierarchy.

Most frequently used structural characteristics include:

- degree of *factorization*: high levels of factorization clearly favor maintainability and extensibility; they are also believed to contribute to better usability (elements of the hierarchy are easily retrieved [26]) and reusability (as abstraction results in a set of classes of broader applicability scope [11]); some authors argue that in certain cases good factorization also brings better comprehensibility [29];
- *minimizing class/property number*: redundancy elimination based on the use experience [52, 48], unused feature deletion [51].
- provided that multiple inheritance is supported by the environment, some specific characteristics may be explored:
 - minimizing multiple inheritance [36, 35, 5], or conversely, admitting multiple inheritance as a standard, based on studies which show that conceptual entropy decreases when multiple inheritance is allowed [41] (hierarchies developed by different designers for a given application are closer when multiple inheritance is admitted);
 - promoting lattice vs. partial order [38, 46, 55, 54].
- provided that rejection of inherited features is admitted (like in Eiffel), the number of such rejections could be used as an indicator for the satisfaction of some external criteria:
 - enhancement of reusability (in terms of coding effort reduction) through controlled use of rejection [3],
 - total avoidance of rejection [10].

Semantic criteria relate to various interpretations of the inheritance relationship:

- from the implementation point of view, classes are like “code repositories” and inheritance is a way of sharing pieces of code;
- from the modeling point of view, the class hierarchy is supposed to reflect concepts and conceptual specialization of the application domain;
- sub-typing (in the usual sense of type theory [13]) is the third usual semantics.

Many have adopted the concept (Galois) lattice as a framework for the design of class hierarchies ([25, 55, 18]) because it is a natural structure that systematically factors out commonalties while preserving specialization relationships between classes. In fact, the lattice is typically defined upon a binary relation between two sets, currently qualified as objects and attributes (in our context these correspond to the sets of initial classes and of all available properties, respectively). A formal concept in this model associates a

(closed) set of objects and the set of attributes they share, whereas the set of all concepts forms a complete lattice when provided with set inclusion between object components. By analogy with normalization for database design, the concept lattice can be considered as a kind of normal form for the design of class hierarchies. The lattice shows all potentially useful generalizations. Some of the generalizations of the concept lattice are *empty* in that they do not possess their own attributes or objects: all their attributes appear in at least one super-concept (inheritance) and dually, all their objects appear in a sub-concept (extension inclusion). These concepts could be eliminated without loss of information thus leading to a structure called a Galois sub-hierarchy (see [25]) which corresponds to the union of what is called in [23] the sets of attribute concepts and object concepts of the concept lattice. This structure is not necessarily a lattice but, when interpreting its nodes as classes, it is maximally factored, consistent with specialization, while defining a minimal number of classes. It could also be considered as another kind of normal form. As for normalization, in practice, there might be reasons to introduce deviations from these ideal structures but the decision process should be made in a controlled manner by using the normal forms as a conceptual framework.

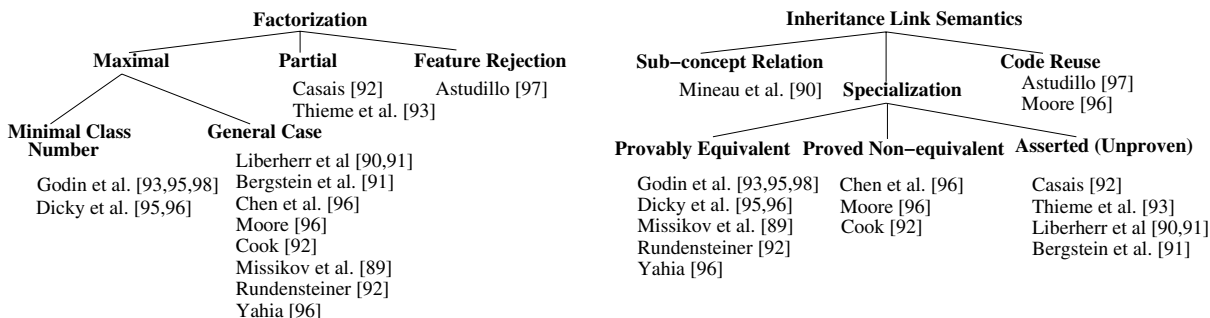


Fig. 2. Approaches classified according factorization and semantic criteria

Two main criteria are used in Fig. 2 to classify the current approaches.

5 Algorithmic aspects

Two main aspects are to be considered from an algorithmic point of view. The general approach of the algorithm is currently determined by the target structure (Fig. 3 left). When the output structure is a tree or an arbitrary DAG, it is only partially characterized, and the algorithm is rather heuristic, for example based on the minimization of the inheritance link number [35], guided by an upward search which eliminates the feature rejection [10] or based on a cladistic algorithm [3]. Conversely, if the intended structure is the Galois (concept) lattice [48] or the Galois sub-hierarchy [25], the algorithm can be proved correct as it is done for CERES in [32]. As far as the underlying model is recognized, a characterization can be achieved as it is shown in [29] for algorithms which approach a Galois sub-hierarchy [15, 39, 14].

The algorithmic form (Fig. 3 right) is noteworthy if the algorithm is to be integrated as a service in a CASE tool. Such development environments indeed require providing a wide range of actions as those proposed in Section 2. Some algorithmic forms also have specific properties it is useful to study, as *Robustness* of incremental approaches, which states that the order a set of actions are carried out does not change the final result. Algorithms AISGOOD [24], ARES [18] and the first stage in [6] share such a robustness property, just as they also ensure that the final result is achieved by an associated global algorithm (for ex. CERES for ARES). Some approaches support human interaction: for example the designer is invited to choose among several possible factorizations in [50], while Snelting et al. [48] put the constructed lattice at the center of an interactive restructuring process.

6 Conclusion

As shown in Section 5, many of the proposed methods aim at improving the factorization while preserving the specialization relationships. Furthermore, as they look for compact hierarchies, this favors the

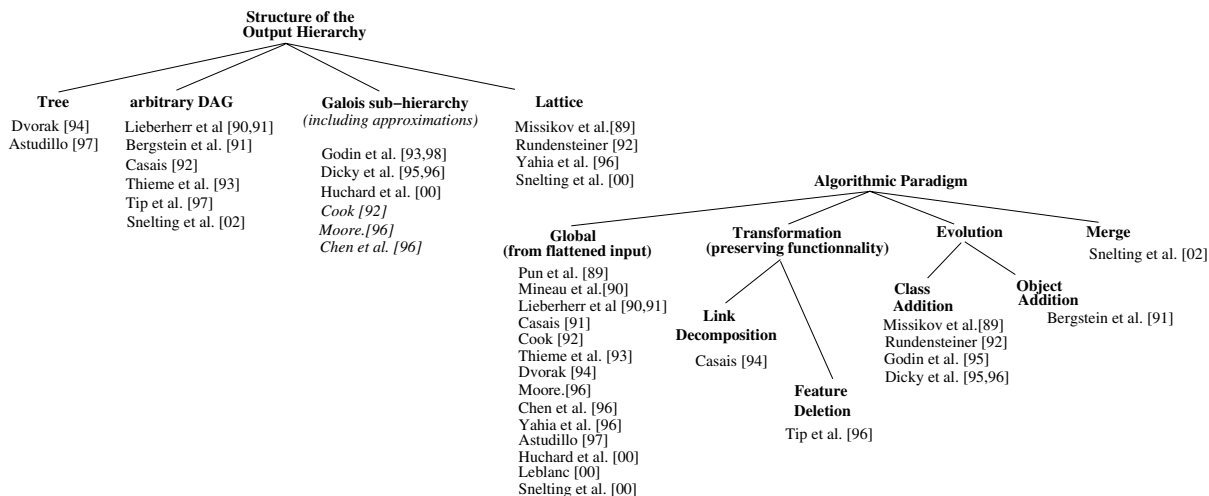


Fig. 3. Structure of the output hierarchy (left), algorithmic forms (right)

generation of structures close to the Galois sub-hierarchy or lattice. This suggests a certain convergence toward a normalized model of a class hierarchy.

There have been several attempts to experimentally validate the proposed methods. However, most of the time, the validation methodology is limited in scope. In general, a complete and rigorous validation framework is still missing.

The normal form model mentioned above could constitute the basis for this framework, provided that an independent validation of the model is accomplished beforehand. Two different approaches may be used for this purpose. On the one hand, we could use consensual software metrics (as in [47]) to assess the impact of restructuring methods based on the normal form. On the other hand, the structural properties of the normal form could be expressed quantitatively, i.e., as metrics, structural or internal attribute ones (as in [44] and [16]). These new metrics should be validated on their own, in particular with respect to external quality criteria listed in Section 4.

In general, the combination of structure-based and metric-based reasoning deserves greater attention. Indeed, software metrics could also bring some benefits at operational level, for example as a guiding mechanism for hierarchy restructuring [1, 47]. Such a hybrid strategy could be generalized to a broader set of methods and metrics.

Another key aspect of the suggested normal form is the support of coarser-granularity operations like hierarchy slicing and merging. First results about Galois lattice merge reported in [53] provide the basis for further investigation including their adaptation to sub-hierarchy manipulation.

As mentioned by G. Booch in [7], there are apparent similarities between concerns of class hierarchy design and conceptual clustering [21]. This suggests a careful study of possible benefits from the application of established conceptual clustering algorithms to hierarchy design, in particular when intelligibility of the target hierarchies is a factor.

References

1. A. Arfi. Un outil pour la conception de hiérarchies de classes dans un modèle orienté-objet. Master's thesis, Université de Montréal, 1996.
2. H. Astudillo. *Evaluation and realization of modeling alternatives: supporting derivation and enhancement*. PhD thesis, Georgia Institute of Technology, Atlanta, USA, March 1996.
3. H. Astudillo. Maximizing object reuse with biological metaphor. *Theory and Practice of Object Systems*, 3(4):235–251, 1997.
4. J. Banerjee, W. Kim, K.J. Kim, and H. Korth. Semantics and Implementation of Schema evolution object-oriented databases. In *Proceedings of the ACM SIGMOD Conference*, San Francisco, CA, 1987.
5. P. Bergstein. Object Preserving Class Transformations. In *Proceedings of OOPSLA'91, Phoenix (AZ), USA*, special issue of ACM SIGPLAN Notices, 26(11), pages 299–313, 1991.
6. P. Bergstein and K. Lieberherr. Incremental Class Dictionary Learning and Optimization. In *Proceedings of ECOOP'91, Geneva, Switzerland*, pages 371–396, 1991.

7. G. Booch. *Object Oriented Analysis and Design with Applications, Second Edition*. Benjamin/Cummings Publishing, Reading (MA), USA, 1994.
8. A. Capouillez, R. Chignoli, P. Crescenzo, and P. Lahire. Towards a more suitable class hierarchy for persistent object management. Workshop "Objects and Classification: a natural convergence" of 14th European Conference on Object-Oriented Programming (ECOOP 2000) URL: <http://www.lirmm.fr/~huchard/WorkshopClassif.html>.
9. E. Casais. *Managing Evolution in Object Oriented Environments : An Algorithmic Approach*. PhD thesis, Université de Genève, 1991.
10. E. Casais. An incremental class reorganization approach. In O. Lehrmann Madsen, editor, *Proceedings of ECOOP'92, Utrecht, The Netherlands*, volume 615 of *Lecture Notes in Computer Science*, pages 133–152. Springer-Verlag, Berlin, Germany, 1992.
11. E. Casais. Managing class evolution in object-oriented systems. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 201–244. Prentice Hall, 1995.
12. E. Casais. Re-Engineering Object-Oriented Legacy Systems. *Journal of Object-Oriented Programming*, 10(8):45–52, January 1998.
13. G. Castagna. *Object-Oriented Programming, a Unified Foundation*. Birkhauser, 1997.
14. J.-B. Chen and S. C. Lee. Generation and reorganization of subtype hierarchies. *Journal of Object Oriented Programming*, 8(8), 1996.
15. W.R. Cook. Interfaces and Specifications for the Smalltalk-80 Collection Classes. In *Proceedings of OOP-SLA'92, Vancouver, Canada*, special issue of ACM SIGPLAN Notices, 27(10), pages 1–15, 1992.
16. M. Dao, M. Huchard, H. Leblanc, T. Libourel, and C. Roume. A New Approach to Factorization : Introducing Metrics. In *Proceedings of the International Symposium on Software Metrics - to appear -*, 2002.
17. H. Dicky, C. Dony, M. Huchard, and T. Libourel. ARES, Adding a class and REStructuring Inheritance Hierarchies. In *11 ièmes journées Bases de Données Avancées, Nancy*, pages 25–42, 1995.
18. 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.
19. J. Dvorak. Conceptual Entropy and Its Effect on Class Hierarchies. *Computer*, 27(6):59–63, 1994.
20. G. Ewing. Class Inheritance: the mechanism and its uses. Honour's thesis, Monash University, oct 1994.
21. D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.
22. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-wesley, 1999. Object Technologies Series.
23. B. Ganter and R. Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer-Verlag, 1999.
24. R. Godin and T.T. Chau. Comparaison d'algorithmes de construction de hiérarchies de classes. *L'Objet*, 5(3):321–338, 2000.
25. 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.
26. R. Godin, H. Mili, G. Mineau, and R. Missaoui. Conceptual clustering methods based on Galois lattices and applications. *Revue d'Intelligence Artificielle*, 9(2):105–137, 1995.
27. R. Godin, G. Mineau, and R. Missaoui. Incremental structuring of knowledge bases. In G. Ellis, R.A. Levinson, A. Fall, and V. Dahl, editors, *Proceedings of the 1st International Symposium on Knowledge Retrieval, Use, and Storage for Efficiency (KRUSE'95), Santa Cruz (CA), USA*, pages 179–193. Department of Computer Science, University of California at Santa Cruz, 1995.
28. M. Huchard. Classification de classes contre classification d'instances. Evolution incrémentale dans les systèmes à objets basés sur des treillis de Galois. In *Actes de LMO'99: Langages et Modèles à Objets*, pages 179–196. Hermès, 1999.
29. M. Huchard, H. Dicky, and H. Leblanc. Galois lattice as a framework to specify algorithms building class hierarchies. *Theoretical Informatics and Applications*, 34:521–548, January 2000.
30. M. Huchard and H. Leblanc. Computing Interfaces in Java. In *Proc. IEE International conference on Automated Software Engineering (ASE'2000)*, pages 317–320, 11-15 September, Grenoble, France., 2000.
31. M. Huchard, T. Libourel, and C. Dony. *Génie objet : Analyse et Conception de l'Évolution*, M. Oussalah eds., chapter Evolution de hiérarchies de classes par approches algorithmiques, pages 215–255. Hermes Science, 1999.
32. H. Leblanc. *Sous-hiérarchies de Galois : un modèle pour la construction et l'évolution des hiérarchies d'objets (Galois sub-hierarchies: a model for construction and evolution of object hierarchies)*. PhD thesis, Université Montpellier 2, 2000.
33. M. Lenzerini, D. Nardi, and M. Simi, editors. *Inheritance Hierarchies in Knowledge Representation and Programming Languages*. John Wiley & Sons, Chichester (West Sussex), UK, 1991.
34. Q. Li and D. McLeod. Conceptual Database Evolution Through Learning in Object Databases. *IEEE, Transactions on Knowledge and Data Engineering*, pages 205–224, 1994.
35. K.J. Lieberherr, P. Bergstein, and I. Silva-Lepe. From objects to classes: Algorithms for optimal object-oriented design. *Software Engineering Journal*, 6(4):205–228, 1991.

36. K.J. Lieberherr, P.L. Bergstein, and I. Silva-Lepe. Abstraction of object-oriented data models. In H. Kangasalo, editor, *Proceedings of the 9th International Conference on the Entity-Relationship Approach (ER'90)*, Lausanne, Switzerland, pages 81–94, 1990.
37. B. Meyer. *Object-Oriented Software Construction, Second Edition*. The Object-Oriented Series. Prentice-Hall, Englewood Cliffs (NJ), USA, 1997.
38. M. Missikoff and M. Scholl. An Algorithm for Insertion into a Lattice: Application to Type Classification. *Proceedings of the 3rd Int. Conf. FODD'89*, pages 64–82, 1989.
39. I. Moore. Automatic Inheritance Hierarchy Restructuring and Method Refactoring. In *Proceedings of OOPSLA'96, San Jose (CA), USA*, special issue of ACM SIGPLAN Notices, 31(10), pages 235–250, 1996.
40. W.F. Opdyke and R.E. Jonhson. Creating abstract superclasses by refactoring. In S.C. Kwasny and J. F. Buck, editors, *Proceedings of the 21st Annual Conference on Computer Science, Indianapolis (IN), USA*, pages 66–72. ACM Press, New York (NY), USA, 1993.
41. H. Ouagagg and R. Godin. Étude empirique de l'influence de l'héritage multiple sur l'entropie conceptuelle : comparaison avec l'héritage simple. *Actes du Colloque Langages et Modèles à Objets (LMO'97)*, Roscoff, France, pages 161–172, 1997.
42. P. Rapicault and A. Napoli. Evolution d'une hiérarchie de classes par interclassement. *L'Objet*, 7(1-2), 2001.
43. D. Roberts, J. Brant, and R. Johnson. A Refactoring Tool for Smalltalk. *Theory and Practice of Object Systems*, 3(4):253–263, 1997.
44. C. Roume. Évaluation structurelle de la factorisation et la généralisation au sein des hiérarchies de classes : Introduction de métriques. *L'Objet*, pages 151–166, 2002.
45. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs (NJ), USA, 1991.
46. E. A. Rundensteiner. A Class Classification Algorithm For Supporting Consistent Object Views. Technical report, University of Michigan, 1992.
47. H. Sahraoui, R. Godin, and T. Miceli. Can Metrics Help to Bridge the Gap Between the Improvement of OO Design Quality and its Automations ? In *Proceedings of the International Conference on Software Maintenance*, pages 154–162, 2000.
48. G. Snelling and F. Tip. Understanding class hierarchies using concept analysis. *ACM Transactions on Programming Languages and Systems*, 22(3):540–582, May 2000.
49. G. Snelling and F. Tip. Semantics-based composition of class hierarchies. In *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP 2002)*, Lecture Notes in Computer Science, Malaga, Spain, June 2002. Springer-Verlag.
50. C. Thieme and A. Siebes. Schema Integration in Object-Oriented Databases. In *Proceedings of the 5th International Conference on Advanced Information Systems Engineering (CAiSE'93), Paris, 8-11 June*, volume 685 of *Lecture Notes in Computer Science*, pages 54–70. Springer-Verlag, 1993.
51. F. Tip, J.-D. Choi, J. Field, and G. Ramalingam. Slicing Class Hierarchies in C++. In *Proceedings of OOPSLA'96, San Jose (CA), USA*, pages 179–197, 1996.
52. F. Tip and P. Sweeney. Class hierarchy specialization. In *Proceedings of OOPSLA'97, Atlanta (GA), USA*, pages 271–285, 1997.
53. P. Valtchev, R. Missaoui, and P. Lebrun. A partition-based approach towards building Galois (concept) lattices. *to appear in Discrete Mathematics*, 2001.
54. A. Yahia, L. Lakhal, and J.P. Bordat. Designing Class Hierarchies of Object Database Schemas. In *13 ièmes journées Bases de Données Avancées*, pages 371–390, 1997.
55. A. Yahia, L. Lakhal, R. Cicchetti, and J.P. Bordat. iO2, An Algorithmic Method for Building Inheritance Graphs in Object Database Design. In *Proceedings of the 15th International Conference on Conceptual Modeling ER'96*, 1996.