Empirical comparison of two class model normalization techniques: Obstacles and questions

Jean-Rémy Falleri, Marianne Huchard, Clémentine Nebut

Empirical comparison of two class model normalization techniques *Obstacles and questions* ⋆

J.-R. Falleri[1], M. Huchard[1], and C. Nebut[1]

LIRMM, CNRS and Université de Montpellier 2,
161, rue Ada, 34392 Montpellier cedex 5, France
{falleri, huchard, nebut}@lirmm.fr

Abstract. Designing accurate models is a true challenge for model driven engineering approach. We are currently exploring techniques derived from Formal Concept Analysis (FCA) theory for finding possible class, association, attribute or method generalizations in models with the aim of improving their abstraction level. Using four models, we compare classical FCA approach to Relational Concept Analysis (RCA) which allows to discover more subtle generalizations. Interestingly, expected combinatorial explosion does occur in all cases when using RCA, making it a feasible solution in a special range of models. The study highlights several difficulties, including the need for costly and subjective human intervention in assessing or filtering the results.

1 Introduction

We are involved for 11 years in several projects with an industrial partner, France Télécom R&D, dealing with the general problem of assessing and improving the quality of the abstraction level of a class model. By class model we refer to UML structural class models, Ecore models as well as Java or C++ programs. A small part of our work is dedicated to abstraction metrics [1] while the main effort is put on developing theory [2,3], methodology [4,5], algorithms [6] and software tools [4,5] for improving abstraction level. We are currently involved in a project which is concerned with Model Driven Engineering (MDE) approach in two ways: we use the MDE spirit and technologies for developing a generic tool, based on data input/output metamodels and on a configuration metamodel; the purpose of the tool, which is improving abstraction level of models, deals with the core material of MDE, namely models. Our approach is based on Formal Concept Analysis (FCA) and a derived data analysis method called Relational Concept Analysis (RCA). RCA helps discovering more accurate generalizations in models, unattainable by FCA, but the counterpart is that many non relevant generalizations can be found at the same time. In this paper, we study four models, mining generalizations using classical FCA as well as RCA. Results show a

⋆ France Télécom R&D has supported this work (CPRE 5326).

combinatorial explosion for RCA applied to the two Ecore models, but a reasonable result size for Java models. In the last case, the gain in obtaining relevant, more subtle, generalizations is not ruined by the necessity of mining these interesting generalizations into a huge amount of artifacts. In Section 2, we describe the research problem and the studied solutions. Section 3 presents the empirical comparison which was conducted, as well as the difficulties encountered. We conclude by a discussion in Section 4.

2 Clustering techniques for Class Model Normalization

Problem description One effect of the lack of abstraction in class models is the introduction of duplicated elements (e.g. attributes, parts of methods). This occurs because of the iterative way of building software and its constant evolution. Table 1 gives an insight of the number of duplicated attributes names (identifiers) in four class models that will be used in the case study. UML2 and Docbook are two metamodels designed with Ecore, Apache Common Collections (ACC) and Minjava are written in Java. Those name duplications do not necessarily imply redundant declarations since two attributes can have the same name and different meanings. However, it gives an indication on the actual number of duplicated attributes. More generally, we would like to improve the level of abstraction of

	Docbook	UML2	Minjava	ACC
#Classes	40	246	29	250
#Attributes	183	615	340	544
#Attrib. name duplications	161	319	63	373

For a given identifier `I` duplicated `n` times, we count `n` duplications (and not one).

Table 1. Attribute name (identifier) duplications in four class models

class models: adding generalizations of operations factoring out common code in their body, adding generalizations of attributes because of common or close name and compatible types (with a common semantically close super-type) or adding generalizations of associations in UML because their ends and part of their description can be generalized. As a consequence of these generalizations, new classes are introduced, and they often highlight new abstract concepts useful in next steps of evolution, for reusing and easy maintenance. Among existing proposals for finding generalizations, we study more precisely derived solutions from Formal Concept Analysis field.

Studied solutions Formal Concept Analysis [7] is a clustering method that classifies a set of entities described by attributes. More formally, let $K = (E, A, R)$ be a formal context. E is a set of entities, A is a set of attributes and R the `own` relation, with $R \subseteq E \times A$. A sample formal context is shown at the right of Figure 1.

In this context, entities (UML classes) are the rows and attributes (UML properties) are the columns. A *concept* is a set of entities that share several attributes. It can be considered as an abstraction of these entities. More formally, a concept is a pair (X, Y) with $X \subseteq E$, $Y \subseteq A$ and $X = \{e \in E | \forall y \in Y, (e, y) \in R\}$ is the extent (covered entities), $Y = \{a \in A | \forall x \in X, (x, a) \in R\}$ is the intent (shared attributes). These definitions ensure maximal factorization of attributes, and in the context of class model, avoid property and method duplications.

The concepts can be organized in a specialization lattice: a concept c_1 is lower than a concept c_2 if the intent of c_2 is included in the intent of c_1. The specialization lattice ensures, in the context of class model normalization, that inheritance or specialization links respect property/method sets inclusion and refinement. A sample lattice corresponding to the context of Figure 1 is shown at the left of Figure 2 (intents are the upper part of the labels, extents are the lower part).

Three steps are required to apply formal concept analysis on a class model. First, the class model is converted into a formal context (Figure 1), which encodes the ownership (by contrast with [8] approach that mainly encodes access in formal contexts). Second, a concept lattice is built, according to the formal

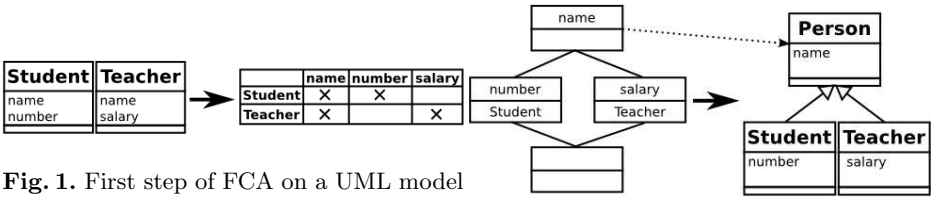

Fig. 1. First step of FCA on a UML model [9]

Fig. 2. Second step of FCA on a UML model

context. This concept lattice will contain concepts that represent the existing entities (and thus the classes) of the formal context, and new concepts that will lead to the creation of new classes. The last step (Figure 2-right) is to build a class model according to the concept lattice. It is clear that the output class model is normalized whereas the input class model was not. This normal form is called *attribute lattice factored form* in [10].

Formal Concept Analysis is powerful to distribute attributes in a class hierarchy, but is unable to deal with relational descriptions. As an example, let us consider the class model in the left of Figure 3. The same conversion and application of FCA on this model, as previously described, would lead to the creation of the model shown in the right of Figure 3. The resulting model, even if it is in normal form, could still be improved with a new property with type Person, introduced in the class Person, and redefined by the *friends* and *colleagues* properties.

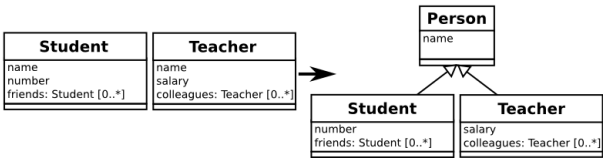

Fig. 3. Limitations of FCA on a UML model

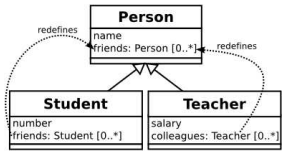

Fig. 4. RCA result on a UML model

Relational Concept Analysis [2,3] is an extension of FCA. It is designed to take into account entities described by attributes and by inter-entity links. In RCA, instead of having just one formal context, there is one formal context for each kind of entities. Then these formal contexts are filled out with other contexts that show relations between entities coming from one context and entities coming from another context (which can be the same). More formally, a Relational Context Family (RCF) is a pair $F = (K, L)$ where K is a set of formal contexts where $K_i = (E_i, A_i, R_i)$ and L a set of relational contexts, $L_i = (E_a, E_b, R_i)$ with $R_i \subseteq E_a \times E_b$. Figure 5 shows the relational context family corresponding to the class model at the left of Figure 3.

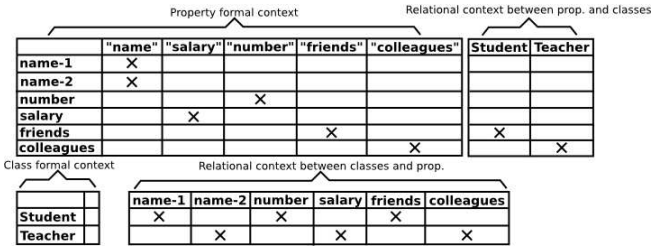

Fig. 5. Applying RCA on a UML model: produced contexts

An iterative lattice construction is applied on the relational context family. A concept lattice is built for each formal context K_i of the Relational Context Family. The discovered concepts of these lattices are injected as new entities in the RCF, and new lattices are built. This iterative construction stops whenever for each category of entities, the lattices built while performing two successive steps are isomorphic. The set of lattices produced after each step of the process is called a Concept Lattice Family (CLF). The class model in Figure 4 has been produced from the contexts of Figure 5.

3 An empirical comparison of Class Normalization techniques

In this section, we develop our view on an empirical study we began to carry out. We examine four facets: how a precise hypothesis we would like to check is formulated, how the experiment is prepared, how it is conducted and finally how results are evaluated.

3.1 Formulating hypothesis

As presented on small examples, FCA and RCA are very attractive techniques. From a theoretical point of view, it can be shown that RCA finds relevant generalizations (abstractions) that are not obtained by FCA. From a practical point of view, RCA is intrinsically much more combinatorial than FCA and it seems more difficult to fine-tune and control the huge set of produced generalizations. When the initial model contains long paths of associations, some inferred generalizations can have poor semantics and often mean something like (for a class) "class whose instances are linked to instances that are linked to instances that are linked .... to instances of" a given initial class of the model.

The question we would like to answer is formulated as follows: "Comparing generalizations produced by RCA versus those produced by FCA, and considering the effort needed for parameterizing and using results of FCA/RCA, is RCA an interesting improvement in practice?". This question is still too general, as several parameterizations are possible for FCA and RCA, depending of what we decide to encode in the tables (formal contexts for FCA and relational context family for RCA). This can be considered as part of the preparation of the experiment (reducing the hypothesis).

3.2 Preparation of the experiment

Preparing the experiment in our context involved choosing class models on which we could test, choosing some procedures to make the data usable or more relevant, choosing the part of models to consider because many elements can be abstracted, and choosing the gauges for evaluating results.

In previous experiments with industrial models from France Telecom R&D, data were confidential and it was impossible even for us to see them and we just could access to partial informations: partial examples of built abstractions and partial results. Now, after several years, these models are no more confidential but people who designed the models are no more in charge of the projects and have no time to devote to new experiments. The problem we face here is the obsolescence of data.

Choosing data To evaluate our class model normalization approach, we carried out an experiment on four open source class models. Two of them, UML [11] and Docbook [12], are design models written in Ecore. The two others, Apache Commons Collections (ACC) [13] and Minjava [14] (author: J.R. Falleri), are implementation models, obtained by reverse-engineering on Java code. UML stands for the UML 2.0 meta-model. Docbook is a meta-model of the Docbook

language. Apache Commons Collections is a Java library that extends the Java collections. Minjava is a Java reverse engineering tool that analyses Java bytecode and produces an Ecore compliant Java model conforming to a simple Java meta-model. Open source class models have the advantage of being available by everyone. Designers may change or may be too busy to discuss with experimentalists about the models, but at last, in industrial context this is often the same situation.

Choosing configurations We restricted the experiment, for a first study, to parts of models composed of classes and attributes (properties in UML terms):

1. Basic FCA configuration (*FCA1*): it corresponds to the one in [10], that generates a class and a property context and analyses the attribute ownership to discover super-classes, based on attribute names.
2. Enhanced FCA configuration (*FCA2*): same as the previous configuration, but using information specific to the input language (*e.g.* static keyword in Java, cardinality in Ecore) to avoid incorrect generalizations.
3. Enhanced Properties configuration (*RCA*): a RCA configuration that generates a class and a property context and analyses the attribute ownership and the attribute type to discover super-classes and redefined properties.

Choosing gauges To understand the results of the application of FCA (resp. RCA) to our sample models, we use the produced lattice (resp. Concept Lattices Family). We classify the concepts of these lattices into three disjoint categories. $ExistingConcepts$: for elements that were already present in the input class model; $NewConcepts$: for elements created during the RCA process; $MergeConcepts$: for the merge of existing elements from the input model.

The $ExistingConcepts$ set is not really interesting since it contains only concepts representing the input entities. The $NewConcepts$ set is very interesting. It contains the concepts that may introduce new useful elements (abstractions of existing ones) in the class model. The $MergeConcepts$ set is also interesting, since it contains the elements from the source model that have been considered similar and therefore have led to the creation of the new elements. To present the result of our case study, we choose to use the two following quantities: N, the number of new elements *i.e.* $|NewConcepts|$; M, the number of merges *i.e.* $|MergeConcepts|$.

Of course, the previous quantities show how the different configurations of the RCA process behave, but are unable to show the quality of these results. Metrics are a way of assessing quality, but they are not so easy to use: based on current inheritance metrics from [15,16], it has been shown in [17] that inheritance metrics (associated with size metrics) are useful in measuring software stability, but don't really help in detecting concrete design problems.

In [18], the case study uses a structural metric to analyze the result of FCA application on real world class hierarchies. The chosen metric, called *M2* is derived of the *M1* metric introduced in [19]. This metric measures redundancy and inheritance quality. Basically, *M2* is a weighted sum of the number of attributes

and the number of inheritance links. To defavor the use of multiple inheritance, for a given class the inheritance links count as double after the first one. The lower metric *M2* is, the better the class model is designed.

Unfortunately, this metric can lead to wrong analysis of the class model. If we imagine an output class model where a super-class has been found but is not correct (for instance because of homographs), the *M2* metric will still consider this output model as better than the input one. Moreover, this metric is not compatible with the use of redefined properties or methods. If we use the class model shown in Figure 3, the metric *M2* will be 24 for the input model, 22 for the output model without attribute redefinition and 26 for the output class model with attribute redefinition. This clearly shows that this metric is unable to correctly measure the quality of a class model design when attribute redefinition is used.

Results from FCA/RCA on class model could be assessed using recent proposals and results on specialization quality measurement [1,20]. But in this first experiment, four simple, specific metrics have been introduced based on human analysis:. *cn*: number of concepts included in the *NewConcepts* set that are considered as correct; a rate is obtained with $cnr = cn/|NewConcepts|$; *cm*: number of concepts included in the *MergeConcepts* set that are considered as correct; a rate is obtained with $cmr = cm/|MergeConcepts|$.

3.3 Experimenting

During Java model building, we faced to the presence of the standard Java API library and had to decide whether looking for abstraction in the model combined with the Java API, or inside the model only. When building our sample models, we finally chose to restrict the extraction of Java entities to the program itself, and blocked the extraction of the Java standard library (except base types). So when a Java class introduces an attribute typed by a class included in the standard Java API (for instance a *LinkedList*), the attribute appears as not typed in the resulting Java model. The output of this phase is the set of results, in the expected form if no problems for measuring have been found. For our experiments, results are presented in Figures 6, 7, 8, 9 and Tables 10, 11.

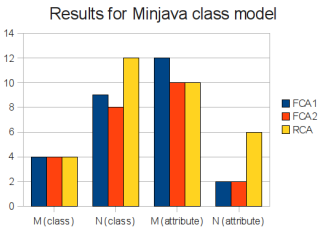

Fig. 6. Results for Minjava class model

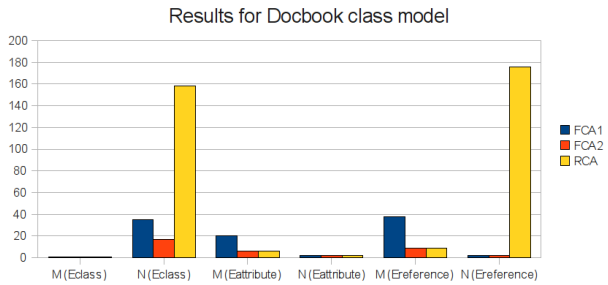

Fig. 7. Results for Docbook class model

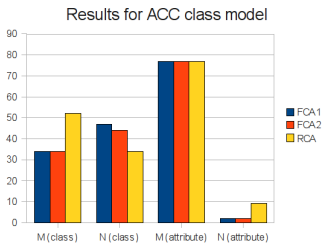

Fig. 8. Results for ACC class model

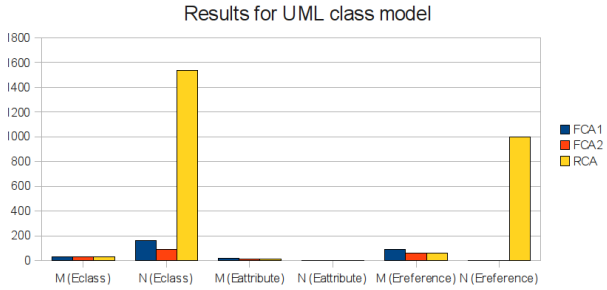

Fig. 9. Results for UML class model

	FCA1	FCA2	RCA
$\|MergeConcepts\|$	12	10	10
cm	10	10	10
cmr	0.83	1	1
$\|NewConcepts\|$	2	2	6
cn	0	0	0
cnr	0	0	0

Fig. 10. Attribute results for Minjava

	FCA1	FCA2	RCA
$\|MergeConcepts\|$	4	4	4
cm	3	3	3
cmr	0.75	0.75	0.75
$\|NewConcepts\|$	9	8	12
cn	5	5	5
cnr	0.56	0.63	0.42

Fig. 11. Class results for Minjava

3.4 Evaluating

Quantitative results shown in Figures 6, 7, 8, 9 are not sufficient to conclude on quality and relevancy of FCA/RCA, but they confirm that in some data (Java software here: ACC, MinJava) feasibility of FCA and RCA is assessed, combinatorial explosion does not occurs and results can be manually explored; in other cases (Ecore models: Docbook, UML2) the number of abstractions created by FCA remains reasonable, while abstractions created by RCA explodes, *e.g.* 1534 new classes are created by RCA for the UML2 metamodel using 246 initial classes and 615 initial properties. This last result is rather depressing and highlights the need for adapted filters. Because we face to the problem of the gold-seeker: looking for a nugget in a heap of uninteresting rocks.

Results of *cn* and *cm* metrics on Minjava are shown in Tables 10 and 11. Concept correction has been assessed by the designer of Minjava. These results confirm what was expected from the quantitative results. *FCA1* is the configuration that produces most incorrect merges and *RCA* produces most incorrect new concepts. On the other hand, new concepts produced by *RCA* could not have been created using a FCA configuration and can contain useful concepts. However it is necessary to find a way to analyse those new concepts in a semi-automatic way, because they are too numerous to be analysed by hand.

4 Discussion

We have shown several facets of an on-going empirical study about FCA and RCA techniques. Now we would like to open the discussion, based on the lessons learned during this first approach and the raised questions.

Place of humans is crucial: to decide which data are used, to evaluate quality of results. For three models we have only quantitative results which, at the end, only give indications of the number of built abstractions. For the (small-size) MinJava model, we have an evaluation which takes into account the judgment of one of us who is the designer of MinJava. This allows us to give a *precision* measure as it is commonly used in the information retrieval field: the fraction of the relevant retrieved abstractions among the retrieved abstractions. This interesting measure is difficult to compute for large models and when we do not have good knowledge of their semantics. It is also a subjective question. A second measure coming from information retrieval field is the *recall* measure which gives the fraction of the relevant retrieved abstractions among the relevant abstractions. This is even more difficult to obtain because finding all relevant abstractions in a model needs many different designers to have a kind of consensus.

Reproducibility of the study is always a challenge: in our case, it is ensured by the use of open source software MinJava for Java models, Ecore tools for Ecore models, the declarative configuration of which entities are taken into account, in available files thanks to a Model Driven Engineering approach [9].

Concerning evaluation, besides precision and recall measure, we could use comparisons between the number of duplications (Fig. 1) and the number of added abstractions. But, beyond these technical considerations, we are aware of the fact that we have (partially) evaluated a first level of quality. A more difficult, second level, of evaluation would consist of measuring: The effort achieved to build the abstractions and (manually) filtering them; What we gained in terms of software quality (readability, extensibility, maintainability).

To conclude, what could help us in our quest:

- sharing problems: organizing challenges like other domains do as the KDD Cup (http://www.sigkdd.org/kddcup/); This could guarantee relevancy of problems themselves;
- sharing data: stable benchmarks, different versions of models; an example of that in MDE domain is the zoo of metamodels (http://www.eclipse.org/gmt/am3/zoos/atlantEcoreZoo/); simulation (randomly generated data) can also be a solution in some special cases but is difficult to control; What is still missing is a benchmark of models, because in our experiment we had to build some of them by transforming source code, thus adding an interpretation step;
- sharing methodologies: Using metrics, using human skills, defining filtering techniques to restrict evaluation in relevant zones, etc.
- sharing results: repositories of results for given benchmarks to confront experiences; making possible the publications of negative results to avoid bias and begin again unuseful experiences.

Acknowledgments The authors would like to thank the anonymous referees for their suggestions and comments that helped to improve the paper.

References

1. Dao, M., Huchard, M., Libourel, T., Roume, C., Leblanc, H.: A New Approach to Factorization - Introducing Metrics. In: IEEE METRICS. (2002) 227–236
2. Dao, M., Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Improving Generalization Level in UML Models Iterative Cross Generalization in Practice. In: ICCS. Volume 3127 of LNCS., Springer (2004) 346–360
3. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Ann. Math. Artif. Intell. **49**(1-4) (2007) 39–76
4. Arévalo, G., Falleri, J.R., Huchard, M., Nebut, C.: Building abstractions in class models: Formal concept analysis in a model-driven approach. In: MoDELS. Volume 4199 of LNCS., Springer (2006) 513–527
5. Falleri, J.R., Huchard, M., Nebut, C., Arévalo, G.: A Model Driven Engineering approach for making generic FCA/RCA tools. In: Proceedings of the Fifth International Conference on Concept Lattices and Their Applications (CLA'07). (2007) 229–252
6. Arévalo, G., Berry, A., Huchard, M., Perrot, G., Sigayret, A.: Performances of Galois Sub-hierarchy-building Algorithms. In: ICFCA. Volume 4390 of LNCS., Springer (2007) 166–180
7. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc. Secaucus, NJ, USA (1997)
8. Snelting, G., Tip, F.: Understanding class hierarchies using concept analysis. ACM Trans. Program. Lang. Syst. **22**(3) (2000) 540–582
9. Falleri, J.R., Huchard, M., Nebut, C.: A generic approach for class model normalization. In: Proc. of ASE'08, short paper. (to appear)
10. Godin, R., Valtchev, P.: Formal concept analysis-based class hierarchy design in object-oriented software development. In: Formal Concept Analysis. Volume 3626 of LNCS., Springer (2005) 304–323
11. Eclipse: UML2 EMF Plugin. http://www.eclipse.org/modeling/mdt/?project=uml2 (2008)
12. Triskell: Docbook metamodel. http://www.kermeta.org (2008)
13. : Apache Foundation: Apache Commons Collections. http: //commons.apache.org/collections (2008)
14. Falleri, J.R.: Minjava. http://code.google.com/p/minjava/ (2008)
15. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEEE Trans. Software Eng. **20**(6) (1994) 476–493
16. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics: A Practical Guide. Prentice-Hall (1994)
17. Demeyer, S., Ducasse, S.: Metrics, Do They Really Help? In: LMO, Hermès (1999) 69–82
18. Godin, R., Mili, H., Mineau, G., Missaoui, R., Arfi, A., Chau, T.: Design of class hierarchies based on concept,(Galois) lattices. Theory and Practice of Object Systems **4**(2) (1998) 117–134
19. Lieberherr, K., Bergstein, P., Silva-Lepe, I.: From objects to classes: algorithms for optimal object-orienteddesign. Software Engineering Journal **6**(4) (1991) 205–228
20. Breesam, K.M.: Metrics for Object-Oriented Design Focusing on Class Inheritance Metrics. In: DepCoS-RELCOMEX, IEEE Computer Society (2007) 231–237