Towards a Metric Suite for Evaluating Factorization and Generalization in Class Hierarchies

M. Dao*, M. Huchard**, H. Leblanc**, T. Libourel**, C. Roume**

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

** *LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France, email: {name}@lirmm.fr*

May 18, 2001

Abstract

In the context of object-oriented development, several software quality criteria (e.g. reusability, easy maintenance and evolution) are influenced by factorization and generalization. This proposal is a first advance towards a metric collection entirely dedicated to feature and class generalization measurement. These metrics are classified into four levels: feature, generic feature, class and hierarchy. We show that these measurements are difficult to do because of a semantic aspect which is not entirely captured by the syntactic constructs of a program or of a class diagram. We present and discuss preliminary experimental results.

This work is part of MACAO[1], a joint project of France Télécom[2], SOFTEAM[3] and LIRMM[4], supported by the french department of research and industry (RNTL).

1 Introduction

Two of the main benefits of object-oriented approaches are reusability and extensibility that facilitate design, development and maintenance. This is closely related to the use of a generalization process that is mainly based on feature factorization. For example, it is well known that code duplication is an obstacle to maintenance and evolution. Moreover, finding reusable classes is often based on the emergence of classes and features that generalize the initial classes.

We are currently working on tools that help programmers and designers to build class hierarchies [1, 2, 3]. These hierarchies may be design hierarchies (UML [4]) or implemented hierarchies (Java, C++, Eiffel, etc.). The techniques we use in these tools follow a general model (based on Galois lattice) introduced by [5] in the framework of class hierarchy construction. The main principle is building class hierarchies maximally factorized, while introducing as few classes as possible for feature factorization.

Besides providing construction tools, we think essential to propose evaluation tools. In particular, they may allow to quantify the improvements carried out by a reconstruction tool: as we know that the resulting hierarchy is maximally factorized, a first problem is to measure the level of factorization in the original one, a second problem is to evaluate the quality of concepts (represented by new classes) stemming from the maximal factorization. More generally, these measurements may highlight design defects connected with feature factorization, such as

[1] http://www.lirmm.fr/~macao/
[2] http://www.francetelecom.fr/
[3] http://www.softeam.fr/
[4] CNRS et Université Montpellier 2, http://www.lirmm.fr/

features which are obviously redundant, classes useless with regard to feature factorization and generalization. Such informations are really interesting to guide more local improvements.

In a preliminary bibliographical study about metrics, we have found some metrics more or less connected to the problems of factorization and generalization, e.g. those counting attributes and methods overridden, the specialization index [6], the pure inheritance index [7] or the method/attribute inheritance factor [8]. Nevertheless, we could not find a metric collection entirely dedicated to the factorization and generalization measurement. This paper is a first attempt to define such a metric collection.

Section 2 introduces several useful notations and vocabulary. Then we propose metrics that apply to different elements: features (Section 3), sets of features connected by specialization (Section 4), classes (Section 5), and then hierarchies (Section 6). In each category, we give some examples of the metrics we have defined. These metrics are gathered in a table at the end of the each section. Section 7 describes first experiments. Section 8 gives some perspectives of this work.

2 Definitions and Notations

We define a class hierarchy as an oriented acyclic graph $GH = (\mathcal{C}, \mathcal{U})$ where $\mathcal{C}$ is the set of classes, and $\mathcal{U}$ the set of inheritance links. GH induces an order denoted by $H = (\mathcal{C}, <_H)$. We will rather use H to denote the hierarchy. We will use also the following notations:

- $feat(C)$, set of the features of a class $C \in \mathcal{C}$; attributes, methods and association ends (in the case of UML class diagrams) are considered as potential features. In this feature set, we count all the inherited features, even if they are overridden (this could be a variant),
- $inherit(C)$, set of the features inherited by a class $C \in \mathcal{C}$,
- $declar(C)$, set of the features syntactically declared in a class C; note that we may have $inherit(C) \cap declar(C) \neq \emptyset$,
- $Feat(H)$, set of all the features that appear in the hierarchy; $Feat(H) = \bigcup_{C \in \mathcal{C}} feat(C)$,
- $\mathcal{R}_{HF}$ denotes the binary relation between classes and features (Has the Feature), $(C, f) \in \mathcal{R}_{HF}$ if $C \in \mathcal{C}$, $f \in Feat(H)$ and $f \in feat(C)$,
- a *feature occurrence* is a pair $o = (f, C)$, where C is a class which declares the feature f.

3 Feature factorization

Metrics are defined having in mind what is a maximal feature factorization.

Definition 1 (Maximal Feature Factorization —MFF) *A feature must have only one occurrence, therefore it must be declared by only one class.*

Note that, for a given relation $\mathcal{R}_{HF}$ between a class set and a feature set, there are several class hierarchies that verify the maximal feature factorization property.

In this section, except the last metric, which is dedicated to a specific feature occurrence, the others take into account the whole set of occurrences of a feature.

The first two metrics are based on the *redundancy number* (RN) of a feature f, that is the number of classes which declare f (or the number of occurrences of f), *less one*, because we consider that at least one occurrence is necessary to preserve the services provided by the hierarchy. In Figure 1, $RN(p) = 3$, $RN(q) = 1$, $RN(x) = 0$ because x is maximally factorized.

The *redundancy ratio* (RR) is the number of redundancies of f divided by the number of classes that own f: it is useful to distinguish for example a feature declared 5 times but owned by 50 classes ($RR = 4/50 = 0.08$) from a feature declared 4 times but owned by 4 classes

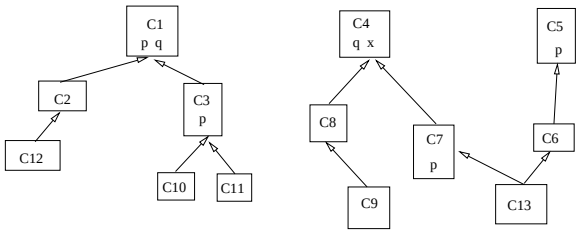


Figure 1: Hierarchy illustrating metrics at the feature level

($RR = 3/4 = 0.75$). The former is rather well factorized, while the latter is not. If a feature is maximally factorized, RR is always 0. For a feature never inherited, the more it has occurrences, the more the redundancy ratio is close to 1, this bound being never reached. In Figure 1, $RR(p) = 3/10 = 0.3$, $RR(q) = 1/11 = 0.09$, $RR(x) = 0/1 = 0$. q is better factorized than p.

Two other metrics (MON and its associated ratio MOR) are a variant where only *maximal occurrences* of a feature f are considered. This number is interesting because it provides an upper bound for the number of links to be added in $<_H$ in order to factorize f, while $RN(f)$ counts also occurrences that only need to be removed to obtain a maximal factorization of f.

DEFINITION 2 (MAXIMAL OCCURRENCE) *A feature occurrence $o = (f, C)$ is maximal if none of the super-classes of C also declares f.*

In Figure 1, $MON(p) = 3$, $MOR(p) = 3/10$, due to the maximal occurrences $(p, C1)$, $(p, C7)$ and $(p, C5)$; $MON(q) = 2$, $MOR(q) = 2/11$, due to the maximal occurrences $(q, C1)$, $(q, C4)$; $MON(x) = 1$, x has only one maximal occurrence $(x, C4)$. In these metrics, p is no more penalized by the occurrence $(p, C3)$. MOR seems to be not significant when $MON = 1$.

A metric LFF, dual of RR, intends to capture in which proportion a feature is factorized. The definition of the metric is based on the fact that a feature is factorized every time it is inherited (and not redeclared). In Figure 1, $LFF(p) = 7/10$, $LFF(q) = 10/11$, $LFF(x) = 5/5 = 1$. Once again, we see that x is maximally factorized, proportionally q is better factorized than p.

The last metric we present in this part tries to highlight the importance of an occurrence $o = (C, f)$ in the subclasses of C. This importance is measured by the number of subclasses of C that inherit f only from C, divided by the number of subclasses of C. In Figure 1, $IOD(o_1 = (p, C1)) = 2/5$, $IOD(o_2 = (p, C5)) = 1/2$ ($C13$ is excluded because it also inherits p from $C7$), $IOD(o_3 = (q, C4)) = 4/4$. o_2 is (slightly) more important in its descent[5] than o_1, while o_3 is really important. The more IOD is close to 1, the more important it is in its subclasses.

RN	Redundancy Number $RN(f) = \lvert\{C \in H / f \in declar(C)\}\rvert - 1$
RR	Redundancy Ratio $RR(f) = \frac{RN(f)}{\lvert\{C \in H / f \in feat(C)\}\rvert}$
MON	Maximal Occurrence Number $MON(f) = \lvert\{C \in H / f \in declar(C),\ \ and\ \forall C' >_H C,\ \ f \notin declar(C')\}\rvert$
MOR	Maximal Occurrence Ratio $MOR(f) = \frac{MON(f)}{\lvert\{C \in H / f \in feat(C)\}\rvert}$
LFF	Level of Factorization for a Feature $LFF(f) = \frac{\lvert\{C / f \in inherit(C) \setminus declar(C)\}\rvert + 1}{\lvert\{C / f \in feat(C)\}\rvert}$
IOD	Importance of an occurrence in its descent $IOD(o = (f, C)) = \frac{\lvert\{C' <_H C / \nexists C'' \neq C, C'' \in \mathcal{C}, C'' >_H C'\ with\ f \in declar(C'')\}\rvert}{\lvert\{C' <_H C\}\rvert}$

[5] We call descent of a class the set of all its subclasses.

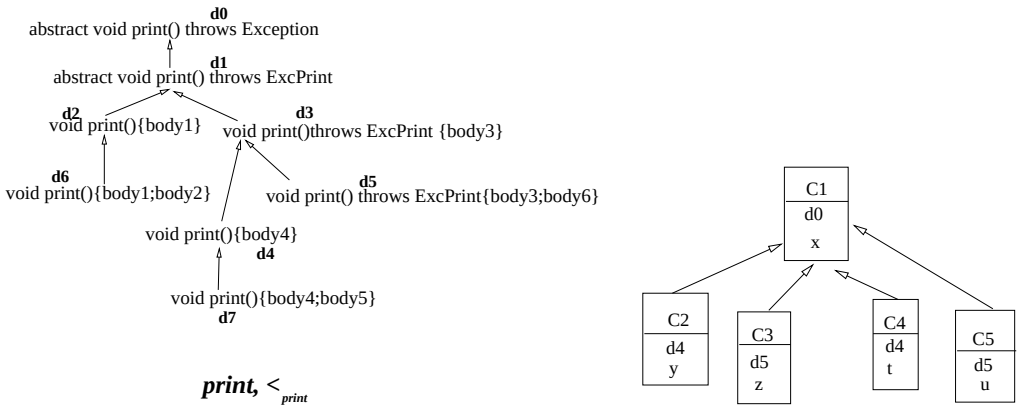


Figure 2: *left* a generic feature *print*; *right* an initial hierarchy

4 Feature generalization

4.1 Generic Features

To discuss this point, we introduce the notion of generic feature.

DEFINITION 3 *A generic feature $\mathcal{F}$ is a set of features semantically connected and partially ordered by specialization.*

For example, a generic method in Java is the set of all methods that have the same name, return type and parameter list. The specialization order may consider:

- an abstract form of the method is more general than a concrete form,
- exception classes that are declared in a method header can be specialized or can disappear when the method is specialized,
- the `super` keyword used in a body can lead to interpret that the overriding method (that contains `super`) is a specialization of the overridden method (called through `super`).

Figure 2 (left) shows an example of a specialization order for a generic feature `print`. The reader is invited to imagine that `super.print()` was replaced by the code called that way; for example, `body1` in method $d6$ corresponds to the replacement of `super.print()`.

Such generic features are difficult to detect in practice. Some aspects can be automatically deduced; in the case of Java, for example, a program can extract from a hierarchy all the methods having a same name, return type and parameter list, and organize these methods through a specialization order deduced from the rules given in the previous paragraph (about generic methods). But the whole semantics of a design is rarely captured by such syntactic aspects, and we think that at this step, a designer may come in to change the specialization order $\mathcal{F}$ being deduced (adding, removing features and links) with respect to usual rules of design or programming. A frequent change would be to add abstract forms of methods when they are missing. In general, several specialization orders can be considered for a given generic feature $\mathcal{F}$, and each of these specialization orders can be a basis for measurement: it gives a theoretical model of the generic feature validated by a human designer.

4.2 Metrics based on generic features

Then, in order to measure the quality of the factorization of a generic feature $\mathcal{F}$ (and for a generic feature, the factorization is also made through specializations and generalizations), three points are considered:

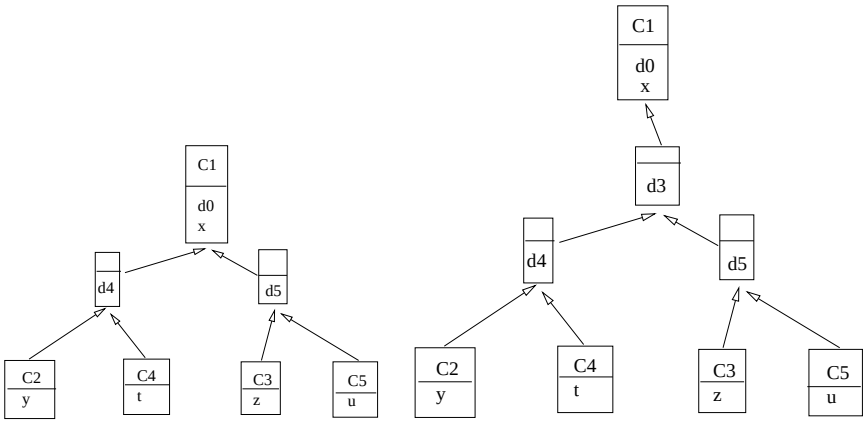


Figure 3: *left* a maximal feature factorization; *right* a least maximal generic feature factorization

- the position of the elements of $\mathcal{F}$ in H,
- a specialization order on $\mathcal{F}$,
- an extension of the definition of the maximal factorization in the case of a generic feature, which is used as an ideal reference.

We thought of two different definitions of the maximal generic feature factorization. We develop here only the one that we call *the least maximal generic feature factorization.* It is basically defined considering that for any two occurrences of a generic feature, the least common generalization of these features should appear (in a right place) in the hierarchy.

Definition 4 (Least Maximal Generic Feature Factorization (LMGFF)) *Let $\mathcal{F}$ be a generic feature, and let GH be a class hierarchy, $\forall f_1, f_2 \in \mathcal{F}$, f_1 and f_2 are each declared in a unique class (resp. C_1 and C_2), and every least upper bound of f_1 and f_2 in $<_{\mathcal{F}}$ is declared in one and only one common superclass of C_1 and C_2.*

Figure 3 shows, for the initial hierarchy of Figure 2, a maximal feature factorization (left) and then a LMGFF (right) deduced from $print, <_{print}$. The difference to note between the two factorizations, is that, in order to obtain a LMGFF, the feature `d3` has been added since it is the least upper bound of `d4` and `d5`.

In a hierarchy under construction, the degree of satisfaction of two criteria may be measured. Firstly, the generalization of the occurrences of the generic features; secondly, the accordance of the order $<_{\mathcal{F}}$ with the order $<_H$. The ultimate goal is that those two criteria are fully satisfied.

The metric $NMGL$ counts the number of generalizations that are missing with respect to a LMGFF. The set $MGL(\mathcal{F})$ of missing generalizations is the fix point of the following series of sets (lub stands for "set of least upper bounds").

$$MGL_0(\mathcal{F}) = \{f_i / f_i \in \mathcal{F}, f_i \notin Feat(H) et\ \exists f_1, f_2 \in Feat(H)\ with f_i \in lub(f_1, f_2)\} \quad (1)$$

$$MGL_j(\mathcal{F}) = \{f_i / f_i \in \mathcal{F}, f_i \notin Feat(H) et\ \exists f_1, f_2 \in MGL_{j-1} \cup Feat(H)\ with f_i \in lub(f_1, f_2)\} \quad (2)$$

A ratio, not detailed here, could be obtained by dividing this number by the number of features. For example, in the original hierarchy of Figure 2 (right) $d4$ and $d5$ have a missing least upper bound, which is $d3$, and we have $NMGL(print) = 1$.

The lack of accordance between a generic feature provided with its specialization order and $<_H$ is measured by two metrics: the first ($NIHR$) counts the number of relationships in H that break $<_{\mathcal{F}}$, the second (NMR) counts the number of relationships of $<_{\mathcal{F}}$ that are missing in H.

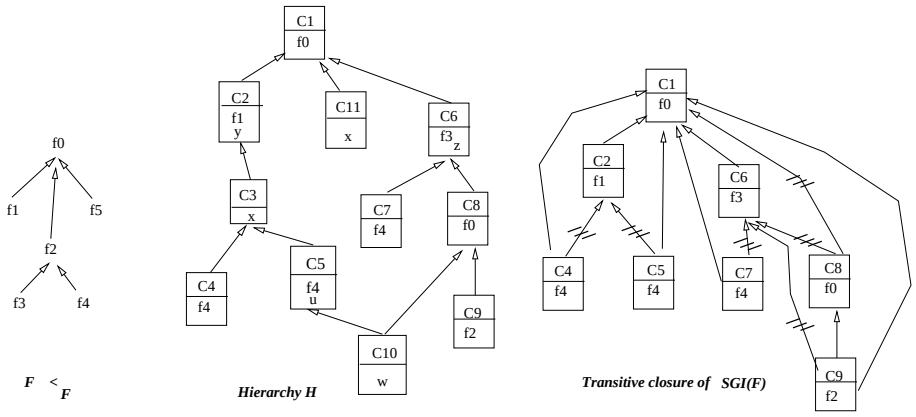


Figure 4: Counting the discrepancies between $<_{\mathcal{F}}$ and $SGI(\mathcal{F})$

Let $SGI(\mathcal{F})$ be the subgraph induced by the occurrences of $\mathcal{F}$ in GH. To obtain ratios, on the one hand, $NIHR(\mathcal{F})$ is divided by the number of edges of the transitive closure of $SGI(\mathcal{F})$: this gives the metric $IHRR$; on the other hand, we divide NMR by the number of relations of $<_{\mathcal{F}}$ that should appear in $SGI(\mathcal{F})$: this gives the metric RMR.

In Figure 4, $NIHR(\mathcal{F}) = 6$ (the edges are ticked on the figure), $IHRR(\mathcal{F}) = 6/12 = 0.5$ (half of the edges are incorrect), $NMR(\mathcal{F}) = 5$, because the edges $(f3, f0)$, $(f4, f0)$, $(f1, f0)$, $(f4, f2)$, and $(f3, f2)$ of $<_{\mathcal{F}}$ are missing at least one time in $<_H$. $RMR(\mathcal{F}) = 5/6$, revealing that nearly all the relations of $<_{\mathcal{F}}$ are not respected at least one time in the hierarchy.

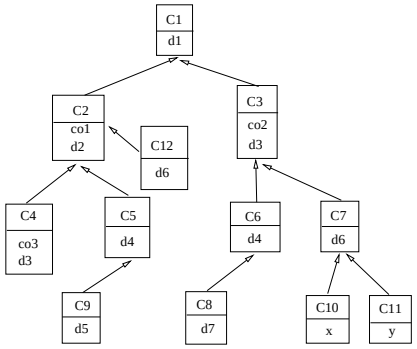
We also have define variants of these metrics that capture the number of times a generalization is missing, or a link is incorrect.

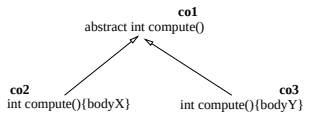
NMGL	Number of Missing Generalizations w.r.t. a LMGFF $NMGL(\mathcal{F}) = \lvert MGL(\mathcal{F})\rvert$
NIHR	Number of Incorrect Hierarchy Relations $NIHR(\mathcal{F}) = \lvert\{(Ci, Cj)$ with Ci subclass of Cj $(Ci <_H Cj)$ that contradict $<_{\mathcal{F}}$: $\exists oi = (fi, Ci)$ and $oj = (fj, Cj)$ with fj is not a generalization of fi $(fi \not\leq_{\mathcal{F}} fj)\}\rvert$
IHRR	Incorrect Hierarchy Relation Ratio $IHRR(\mathcal{F}) = \frac{NIHR(\mathcal{F})}{\lvert\{(Ci,Cj)/Ci<_H Cj, Ci \text{ and } Cj \in SGI(\mathcal{F})\}\rvert}$
NMR	Number of Missing Relations of $<_{\mathcal{F}}$ $NMR(\mathcal{F}) = \lvert\{(fi, fj) s.t. fi <_{\mathcal{F}} fj$, but there are two occurrences $oi = (fi, Ci)$ and $oj = (fj, Cj)$ with $Ci \not\leq_H Cj\}\rvert$
RMR	Ratio of Missing Relations $RMR(\mathcal{F}) = \frac{NMR(\mathcal{F})}{\lvert\{(fi,fj)/fi,fj \in Feat(H), \text{ and } fi <_{\mathcal{F}} fj\}\rvert}$

5 Class impact in factorization and generalization

The first metric ($RCFF$) reports the role of a class C in the factorization of a feature f. It is defined as the quotient of the number of subclasses of C (plus one for C) by the number of classes that own f. In Figure 1, $RCFF(C1, p) = 6/10$, that is $C1$ factorizes about half of the times the feature p is owned.

The second metric ($RCFGF$) reports the role of a class in the factorization of a *generic feature*. In this case, we consider that a class C that declares f is useful to factorize the features of the subclasses of C that specialize f. In Figure 5, $RCFGF(C2, print) = 2/3$, since $C2$





	d1	d2	d3	d4	d5	d6	d7	co1	co2	co3	x	y
C1	x											
C2		x						x				
C3	x		x						x			
C4	x	x	x							x		
C5	x	x		x				x				
C6	x		x	x					x			
C7	x		x			x			x			
C8	x		x	x			x		x			
C9	x	x		x	x			x				
C10	x		x			x			x		x	
C11	x		x			x			x			x
C12	x	x				x		x				

Figure 5: Hierarchy illustrating class impact metrics (using also the previous generic method `print`)

generalizes $(d2, C2)$ and $(d6, C12)$, but not $(d6, C7)$. $RCFGF(C3, print) = 3/5$, since $C3$ generalizes $(d3, C3)$, $(d4, C6)$ and $(d7, C8)$, but not $(d4, C5)$ and $(d5, C9)$. $C2$ and $C3$ are roughly equivalent in the factorization of *print*.

A first variant consists in considering all the owned features rather than the declared ones. A dual metric could consist in counting the features of $\mathcal{F}$ in the subclasses C that do not specialize f. In fact, this is close to compute $NIHR$ on the subgraph induced by C and its subclasses.

Then, to have a more general idea about the role of a class C in the factorization of the features it declares or owns, we have followed two directions. The first, not developed here is based on an average of the previous metrics weighted by the respective importance of each feature in the whole hierarchy. A bad factorization is indeed less serious for a generic feature that has very few occurrences. A second direction (metric RCF), more global and perhaps less precise, is to consider that C is responsible for its subclasses for all the features C owns. In Figure 5, $RCF(C2) = 3 \times 3/42 = 0.2$, $RCF(C3) = 3 \times 5/42 = 0.3$; as a result $C3$ is slightly more important in factorization than $C2$.

RCFF	Role of a Class in the Factorization of a Feature f $RCFF(C, f) = \frac{\lvert\{C'/C' \leq_H C\}\rvert}{\lvert\{C'/f \in feat(C')\}\rvert}$
RCFGF	Role of a Class in the Factorization of a Generic Feature $\mathcal{F}$ For a class C with $f \in declar(C)$, and $f \in \mathcal{F}$, $RCFGF(C, \mathcal{F}) = \frac{\lvert\{C'/C' \leq_H C \text{ and } \exists f' \leq_{\mathcal{F}} f,\ f' \in declar(C')\}\rvert}{\lvert\{C'/\exists f' \leq_{\mathcal{F}} f,\ f' \in declar(C')\}\rvert}$
RCF	Role of a Class C in the Factorization $RCF(C) = \frac{\lvert feat(C)\rvert \times \lvert\{C'/C' \leq_H C\}\rvert}{\lvert\mathcal{R}_{HF}\rvert}$

6 Factorization and generalization level of a class hierarchy

6.1 Direct measurements in the hierarchy

A first way to measure the level of factorization in the whole hierarchy is based on several averages on the previous metrics. We do not detail here the metrics we deduced that way.

6.2 Metrics based on a comparison with other organizations of the hierarchy

The first metric (LFH) captures the level of factorization of a hierarchy H by comparison with the flattened hierarchy deduced from H (the same classes with the same feature sets, but without

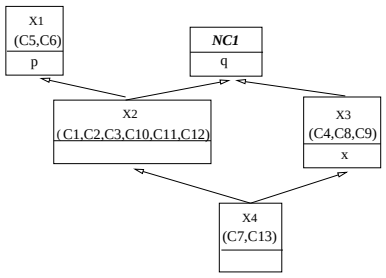


Figure 6: More compact maximal feature factorization equivalent to hierarchy Fig. 1

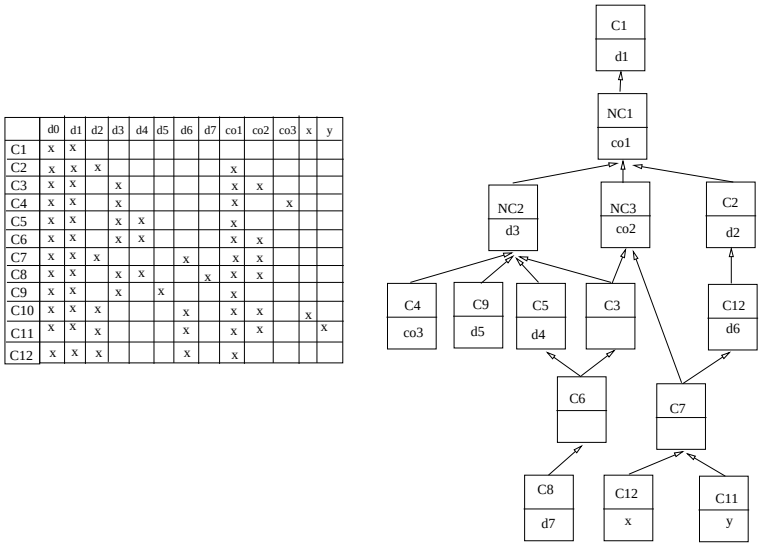


Figure 7: More compact maximal generic feature factorization equivalent to hierarchy Fig. 5

inheritance links). Here again we consider that each occurrence of feature inheritance corresponds to an instance of factorization. $|\mathcal{R}_{HF}| - |Feat(H)|$ represents the number of redundancies in the flattened hierarchy. $LFH(H) = 1$ if the hierarchy is maximally (feature) factorized, provided that the non maximal occurrences are removed. For the hierarchy of Figure 1, it is equal to 19/26. For the hierarchy of Figure 5, the value is 28/42.

We have also defined metrics based on a comparison between the hierarchy under measurement and an equivalent (in terms of services) hierarchy that should be maximally (feature or generic-feature) factorized using the less classes as possible (the most compact) [5, 1].

This second hierarchy, "ideal" in terms of factorization, is called below the Galois Sub-Hierarchy (GSH), a name that comes from its construction [1]. The GSH associated with the hierarchy of Figure 1 is given in Figure 6. Several classes of the initial hierarchy are represented by a single class of the GSH, for example, $C4$, $C8$ and $C9$ are represented by a same class $X3$.

The GSH associated with the hierarchy of Figure 5, is given in Figure 7. In order to obtain this GSH, the hierarchy is flattened and a new $\mathcal{R}_{HF}$ is built in such a way that classes keep the same semantics. Each class C now owns:

1. the features declared by C in the initial hierarchy,
2. the features inherited and not overridden by C in the initial hierarchy,
3. the features that generalize (in the associated generic feature orders) the features of 1. and 2.

We are currently investigating comparisons between the GSH and the initial hierarchy, as well as metrics that indicate the quality of concepts produced in the GSH. We give here simple examples of metrics that can be used. In the following, $Fact$ denotes the classes of the GSH that are needed only to factorize features (their feature set is not equal to any feature set of a class in the initial hierarchy). For instance, Figure 6, $Fact = \{NC1\}$, Figure 7, $Fact = \{NC1, NC2, NC3\}$. Imp denotes the classes of the GSH that are needed to represent a class of the initial hierarchy (their feature set is equal to the feature set of a class in the initial hierarchy). For instance, Figure 6, $Imp = \{X1, X2, X3, X4\}$ and Figure 7, $Imp = \{C1..C11\}$.

Firstly, the number of classes (concepts) of the two hierarchies can be compared, using the metric CNC. Figure 6, $CNC(H) = 13/5 = 2.6$ and Figure 7, $CNC(H) = 11/14 = 0.7$. In some cases the GSH reduces the number of classes (when several classes have the same set of features), while in most cases, the GSH increases the number of classes, due to the factorization. $CNC > 1$ reveals class definitions which are redundant, while $CNC < 1$ reveals feature redundancy.

Secondly, the number of features in the classes of GSH that only factorize features is very important: a factorization class of GSH which contains very few features is maybe not an interesting class. $ANFDFC$, that counts the average number of features in factorization concepts partly plays this role. Figure 6, $ANFDFC = 1/1 = 1$, and Figure 7, $ANFDFC = 4/3 = 1.33$. A variant consists in counting the average number of features (declared and inherited) in factorization concepts.

The last metric presented here (FCR) shows if many classes were needed only to factorize, relatively to the number of classes that had to be represented (without counting classes that have the same feature sets). Figure 6, $FCR = 1/4 = 0.25$, and Figure 7, $FCR = 3/11 = 0.27$. In the two cases, the number of added factorization concepts is reasonable with respect to the number of initial classes.

LFH	Factorization Level of the Hierarchy $LFH(H) = \frac{\lvert\{(C,f)/\ f \in inherit(C)\}\rvert}{\lvert\mathcal{R}_{HF}\rvert - \lvert Feat(H)\rvert}$
CNC	Class Number Comparison $CNC(H) = \frac{\lvert\mathcal{C}\rvert}{\lvert\mathcal{C}_{GSH}\rvert}$
ANFDFC	Average Nb of Features Declared in Factorization Concepts $ANFDFC(H) = \frac{\sum_{C \in Fact} \lvert declar(C)\rvert}{\lvert Fact\rvert}$
FCR	Factorization Concept Ratio $FCR(H) = \frac{\lvert Fact\rvert}{\lvert Imp\rvert}$

7 Experiments under way

The experiments that we carried out relate to several packages of the JAVA language (LINUX JDK 1.1.7). In this work, we implemented and applied a relevant subset of our metrics to specification hierarchies derived from JAVA code. The introspection mechanism of this language allowed us to extract the feature signatures (except for static features, *private* features as well as features of the *inner classes*). The signatures have the following forms:

- for an attribute: `type name`
- for a method: `abstract/concrete return_type name [parametersList] [exceptionsList]`

Measurements are made on a package or on a set of packages carried out recursively. In order to take into account all inherited features, we have examined all super-classes (resp. super-interfaces) of the classes (resp. interfaces) under study.

In accordance with our proposal of metrics, the results are presented at the feature level, at the generic feature level, at the class level and at the hierarchy level.

Table 1: $MON(f)$ and $MOR(f)$

package	*property f*	MON	MOR
lang	`concrete String toString()`	1	1/81
"	`concrete double doubleValue()`	7	7/7
"	`abstract double doubleValue()`	1	1/7
io	`concrete boolean markSupported()`	2	2/22
"	`byte[] buf`	5	5/5
"	`int count`	7	7/7

7.1 Feature level measurements

We have implemented the MON and MOR metrics and we give results concerning the packages `java.lang` (basic classes of the language) and `java.io` (stream classes) in Table 1.

On the whole package `java.lang` (respectively `java.io`), we found:

- 297 features (resp. 231) having one maximal occurrence ($MON(f) = 1$) and owned by less than 2 classes ($|\{C/f \in feat(C)\}| \leq 2$),
- 29 features (resp. 50) having one maximal occurrence ($MON(f) = 1$) and owned by more than 2 classes ($|\{C/f \in feat(C)\}| > 2$),
- 41 features (resp. 117) having more than one maximal occurrence ($MON(f) > 1$) and regarded as potentially badly factorized.

More precisely, Table 1 shows that, the two methods `concrete String toString()` and `concrete boolean markSupported()` are well factorized features (`concrete String toString()` being maximally factorized).

On the other hand, the method `concrete double doubleValue()` having only maximal occurrences, is thus not factorized on the level of its specification. The code analysis of the occurrences shows that six of them are identical, which confirms our interpretation. The presence of the method `abstract double doubleValue()` which seems well factorized confirms us in the idea that a code factorization is missing for `double doubleValue()`. Attributes `byte[ ] buf` and `int count` are also not well factorized. This defect is found for many features in the package `java.io` which, in our opinion, was not designed with the objective of a complete generalization of the input-output streams.

7.2 Generic feature level measurements

A generic feature $\mathcal{F}$ is a set of features semantically connected and partially ordered by specialization. Here the semantic connection consists with the equality of reduced signatures (same `return_type name [parametersList]`), and the specialization order $<_{\mathcal{F}}$ follows the two following rules:

Let m_1 and m_2 be two occurrences in $\mathcal{F}$, $m_1 <_{\mathcal{F}} m_2$ if and only if:

1. (m_1 is abstract) $\Rightarrow$ (m_2 is abstract)
2. $\forall e_1 \in [exceptionsList](m_1)$, $\exists e_2 \in [exceptionsList](m_2)$ s.t. $e_1 = e_2$ or $e_1 <_H e_2$

Considering the way certain generic features are constituted, we thought interesting to implement $NIHR(\mathcal{F})$. The result shows that elements of the generic features `boolean equals(Object)` and `ListIterator listIterator(int)` appear in classes whose links in $<_H$ contradict the specialization orders $<_{equals}$ and $<_{listIterator}$. In both cases, this is expressed by a concrete method

Table 2: $RCFF(C, f)$ and $RCF(C)$ for `java.lang`

class c	*property f*	*RCFF*	*RCF*
`Object`			$11 \times 81/1669 = 0.533$
`Throwable`	`String getLocalizedMessage()`	46/46	
`Throwable`	`void notify()`	46/81	
`Throwable`			$17 \times 46/1669 = 0.468$
`Runnable`	`abstract void run()`	4/4	
`Runnable`			$1 \times 4/1669 = 0.023$
`Process`	`abstract int exitValue()`	2/2	
`Process`			$17 \times 2/1669 = 0.020$

Table 3: $RCFGF(C, \mathcal{F})$ for `java.util`

class C	*generic feature* $\mathcal{F}$	*RCFGF*
`Collection`	`void clear()`	$14/14 = 1$
`AbstractCollection`	`void clear()`	$6/10 = 0.6$
`Map`	`void clear()`	$5/14 = 0.357$

generalizing an abstract method. In the first case, `concrete boolean equals(Object)` declared by `Object` appears higher in $<_H$ than `abstract boolean equals(Object)` which is declared by `Permission`. In the second case, `concrete ListIterator listIterator(int)` declared by `AbsractList` appears higher in $<_H$ than `abstract ListIterator listIterator(int)` which is declared by the subclass `AbstractSequentialList`. We may question the specialization of a concrete method by an abstract method in $<_H$, perhaps to enforce overriding ?

7.3 Class level measurements

RCF, $RCFF$, $RCFGF$ metrics were implemented; we give some results concerning the packages `java.lang` and `java.util` (abstract data types) in the tables 2 and 3.

The values of $RCF(C)$ show that the classes `Throwable` and `Object` have a strong capacity of factorization whereas the classes `Runnable` and `Process` are not of great importance for the factorization of the features. For the values of $RCFF(C, f)$, the class `Throwable` inheriting the method `void notify()` takes part for 50% in the factorization of this feature while for the method `String getLocalizedMessage()` that this class declares, its participation is 100%.

According to our experiments, it appears that the fractions (e.g. 9/14) are more interesting than the values (e.g. 0.64) for the interpretation of the metric $RCFGF(C, \mathcal{F})$. Concerning the generic property `void clear()`, the interfaces `Collection` and `Map` declare the maximal element of the order $<_{clear}$ yielding two maximal occurrences ($MON(clear) = 2$). The class `AbstractCollection` declares a feature which is declared or specialized 10 times in the whole hierarchy, and more precisely 6 times in its subclasses. It factorizes 60% of `void clear()`.

7.4 Hierarchy level measurements

CNC, $Fact$, $ANFDC$ and FCR metrics were implemented and we give some results on the packages `java.lang`, `java.util` and `java.io` in Table 4.

Table 4: $CNC(H), Fact(H), ANFDC(H), FCR(H)$ for `java.lang, java.util, java.io`

Package	`java.lang`	`java.io`	`java.util`
Total Number of Classes	84	84	25
Total Number of Features	357	398	153
Cardinal of R	1669	1981	864
$CNC(H)$	$84/47 = 1.78$	$84/112 = 0.75$	$25/36 = 0.69$
$Fact(H)$	12	44	13
$ANFDC(H)$	$25/12 = 2.08$	$103/44 = 2.34$	$19/13 = 1.46$
$FCR(H)$	$12/35 = 0.34$	$44/68 = 0.64$	$13/23 = 0.56$

These various measurements show that if we comparatively consider the packages `java.lang` and `java.io` that have about the same numbers of classes and features, it is necessary to factorize 103 features using 44 additional classes for `java.io`, for only 25 features and 12 additional classes for `java.lang`. The interpretation of CNC shows that the packages `java.io` and `java.util` could have a better feature factorization while the package `java.lang` contains a certain number of classes having the same specification. The average number of the features declared by the classes of factorization ($ANFDC$) is relatively low, the added classes of factorization will certainly have to be grouped to get more interesting concepts.

The whole set of metrics that we evaluated constitutes a coherent unit which enabled us to point out a certain number of factorization problems in the specifications that could help a designer. However, several remarks have been obtained only after code examination, which is the next stage of our work.

8 Conclusion

We have partially presented a first set of metrics dedicated to the factorization measurement. This set is not definitive, we are still discussing on variants and experiments are yet under way. We have mentioned that some of these metrics actually are not so easy to compute in practice. Problems are mainly due to the feature naming, which can cause conflicts, and to the difficulties that occur when one wants to know or compute generalization/specialization relations between features. All the feature comparisons are indeed difficult to solve using only their syntactic aspects (name, type, parameters, exceptions thrown, body, etc.): semantic aspects play an important role to decide whether a feature is a specialization of another, or whether two features with a same name are semantically connected. As we have mentioned, before any metric computation, a designer could fruitfully come in to adjust a pre-computed set of specialization relations between features. Another important point was to use the Galois sub-hierarchy as an ideal reference to evaluate a class hierarchy. This research way seems to be very promising. We also have in mind to study the cohesion [9, 10] of factorization classes to improve our construction tools.

Acknowledgments Authors would like to thank L. Nenonnen who gave them a first bibliography about metrics.

References

[1] 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.

[2] N. Chevalier, M. Dao, C. Dony, M. Huchard, H. Leblanc, and T. Libourel. An environment for building and maintaining class hierarchies. In I. Borne, editor, *ECOOP99: Workshop Object-Oriented Architectural Evolution*, Lisbonne, Portugal, 1999.

[3] 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.

[4] Object management Group. *OMG Unified Modeling Language Specification, version 1.3*. OMG, `http://www.omg.org`, March 2000.

[5] 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.

[6] M. Lorentz and J. Kidd. *Object-Oriented Software Metrics, a Practical Guide*. Prentice Hall, 1994.

[7] B. K. Miller, P. Hsia, and C. Kung. Object-oriented architecture measures. In *32nd Annual Hawaii International Conference on Systems Sciences*. IEEE Computer Soc., 1999.

[8] F. Brito e Abreu and W. Melo. Evaluating the impact of object-oriented design on software quality. In *Proc. METRICS 96, Berlin, Germany*. IEEE Computer Society, 1996.

[9] Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEE Transactions on software engineering*, 20(6):476–493, 1994.

[10] Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. Technical Report 07, ISERN, Kaiserslautern, Germany, 1998.