Improving Generalization Level in UML Models
Iterative Cross Generalization in Practice

M. Dao[1], M. Huchard[2], M. Rouane Hacène[3], C. Roume[2], and P. Valtchev[3]

[1] michel.dao@francetelecom.com,
France Télécom R&D, DAC/OAT, 38-40 av. Général Leclerc,
92794 Issy-les-Moulineaux cedex 9, France
[2] {huchard,roume}@lirmm.fr,
LIRMM, UMR 5506, 161 rue Ada, 34392 Montpellier cedex 5, France
[3] {rouanehm,valtchev}@iro.umontreal.ca
DIRO, Université de Montréal, C.P. 6128, Succ. "Centre-Ville", Montréal, Canada, H3C 3J7

Abstract. FCA has been successfully applied to software engineering tasks such as source code analysis and class hierarchy re-organization. Most notably, FCA puts mathematics behind the mechanism of abstracting from a set of concrete software artifacts. A key limitation of current FCA-based methods is the lack of support for relational information (e.g., associations between classes of a hierarchy): the focus is exclusively on artifact properties whereas inter-artifact relationships may encode crucial information. Consequently, feeding-in relations into the abstraction process may substantially improve its precision and thus open the access to qualitatively new generalizations. In this paper, we elaborate on ICG, an FCA-based methodology for extracting generic parts out of software models that are described as UML class diagrams. The components of ICG are located within the wider map of an FCA framework for relational data. A few experimental results drawn from an industrial project are also reflected on.

1 Introduction

Current trends in object-oriented software construction, namely MDA (Model-Driven Architecture)-based approaches, promote designing high-level models that represent domain and application concepts ("Platform Independent Models"). These models, typically described in UML (Unified Modeling Language), are further on mapped to the target implementation platform ("Platform Specific Models"). Modeling has thus become a key activity within the software process whereas large efforts are currently spent in developing automated tools to assist it.

Formal Concept Analysis (FCA) has already been successfully applied to the analysis [1] and restructuring [2–7] of conceptual class models: it helps reach optimal hierarchical organization of the initial classes by discovering relevant new abstractions. However, providing far-reaching abstraction mechanisms requires the whole feature set of UML to be covered, inclusive those encoding relational information (e.g., UML associations), whereas such features clearly outgrow the scope of standard FCA.

Making FCA work on UML models is the global aim of our study. Here, we propose a new relationally-aware abstraction technique, ICG (*Iterative Cross Generalization*),

which works on several mutually related formal contexts that jointly encode a UML class diagram. It performs simultaneous analysis tasks on the set of contexts where inter-context links are used to propagate knowledge about the abstractions from a context into its related contexts (and thus broaden the discovery horizon on those contexts).

The paper recalls the basics of FCA (Section 2) before providing a motivating example (Section 3). Our recent FCA-based framework for processing relational data is presented in Section 4. In Section 5 we specify ICG while emphasizing the role UML meta-model plays in data description within ICG. Experiments done in the framework of industrial projects are then reported (Section 6) with a discussion of benefits and difficulties in applying ICG.

2 FCA and class hierarchy restructuring

Formal concept analysis (FCA) [8] studies the way conceptual structures emerge out of observations. Basic FCA considers an *incidence* relation I over a pair of sets O (*objects*, further denoted by numbers) and A (*attributes*, denoted by lower-case letters). Binary relations are introduced as formal contexts $\mathcal{K} = (O, A, I)$. An example of a context, `Foo`, is provided in Figure 1 on the left, where a cross in i-th line / j-th column means that the object i has the attribute j.

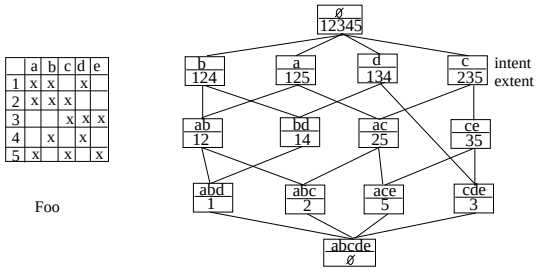


Fig. 1. A sample context and the Hasse diagram of its concept lattice

A pair of derivation operators, both denoted by $'$, map sets of elements between A and O by performing intersections on the corresponding sets of rows/columns. For instance, $\{1,2\}' = \{a,b\}$ and $\{a,c\}' = \{2,5\}$. The $'$ operators define a *Galois connection* [9] between 2^A and 2^O, whereby the component operators $''$ satisfy the closure properties. The underlying sub-families of closed sets are bijectively mapped to each other by $'$ with pairs of mutually corresponding closed sets termed *(formal) concepts*. More precisely, a concept is a pair (X, Y) from $2^O \times 2^A$ with $X = Y'$ and $X' = Y$, where X is the *extent* and Y is the *intent*. The set $\mathcal{C}_\mathcal{K}$ of all concepts from $\mathcal{K}$ is partially ordered by the inclusion of extents, while the resulting ordered structure $\mathcal{L}_\mathcal{K} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ is a complete lattice with joins and meets based on intersection of concept intents and extents, respectively. The lattice of the `Foo` context is drawn in Figure 1 on the right (as a Hasse diagram).

Research on applications of FCA has yielded a set of meaningful substructures of the concept lattice. For instance, in object-oriented software engineering, the assignments of specifications/code to classes within a class hierarchy is easily modeled through a context, and applying FCA to a particular hierarchy may reveal crucial flaws in factorization [2] and therefore in maintainability. The dedicated substructure that specifies a maximally factorized class hierarchy of minimal size is called the Galois sub-hierarchy (GSH) of the corresponding context. Mathematically speaking, the GSH is made out of all the extremal concepts that contain an object/attribute in their extents/intents: $\{(o'', o') | o \in O\} \cup \{(a', a'') | a \in A\}$.

Moreover, as practical applications of FCA may involve processing of non-binary data, many-valued contexts have been introduced in FCA. In a many-valued context $\mathcal{K} = (O, A, V, J)$, each object o is described by a set of attribute - value pairs (a, v), meaning that J is a ternary relation that binds the objects from O, the attributes from A and the values from V. The construction of a lattice on top of a many-valued context requires a pre-processing step, called scaling, which basically amounts to encoding each non-binary attribute by a set of binary ones.

3 Improving UML models: a motivating example

A highly simplified example introduces the problem domain. Consider the UML model in Figure 2. A class `Diary` is associated to a class `Date` through the association `orderedBy`. Class `Date` has three attributes (or variables) `day`, `month` and `year` and two methods including `isLeapYear()` and a comparison method `<(Date)`. Another class `Clock` is linked to `Time` class via the association `shows`. Class `Time` is described by the three attributes `hour`, `min` and `sec`, and by a method `<(Time)` which aims at comparing times.

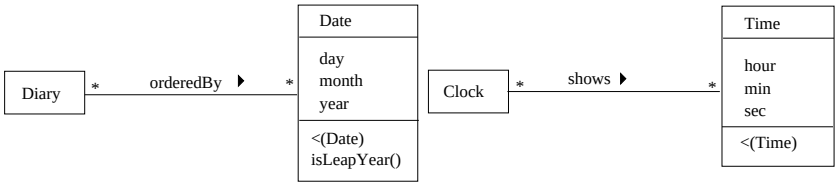


Fig. 2. Diary and clock

Current approaches for applying formal context analysis to this UML model would lead to the formal context of Figure 3: classes are the formal objects while UML attributes, methods and association ends are the formal attributes (names have been reduced to their first letters). This formal context does not reveal any new concept, although comparison methods $<$ indicate that a *magnitude* concept is underlying the model and that diaries and clocks are devices which manipulate magnitudes.

To infer a more elaborate UML model, we apply an approach that may be summarized as follows. On the one hand, we process various sorts of UML entities such

	d	m	y	h	mn	s	<(D)	<(T)	isLeapYear()	origin orderedBy	origin shows	destination orderedBy	destination shows
Diary										x			
Clock											x		
Date	x	x	x				x		x			x	
Time				x	x	x		x					x

Fig. 3. Formal context for diary and clock

as attributes, methods and associations, as first-class formal objects and assign a formal context to each entity sort. Moreover, we use *relational attributes* to express links between entities and model them as inter-context binary relations.

On the other hand, we use a repeated scaling along the relational attributes to propagate the knowledge about possible generalizations between related contexts. Thus, the concept construction process amounts to alternating scaling and proper construction until stability in concept structures is reached.

In Figure 4, three many-valued formal contexts describe classes, associations and methods as first-level formal objects, respectively. Here, UML class attributes are not processed as objects for simplicity sake, but in the general case they are. Note that some formal attributes (*e.g.* `originType`) are relational ones while others are not (*e.g.* `originMultiplicity` or `name`). Figure 5 shows the main relational attributes of the example.

Class Context

	isDescribedBy	*has*	*isOrigin*	*isDestination*
Diary			{orderedBy}	
Clock			{shows}	
Date	{d,m,y}	{<(D),isLeapY}		{orderedBy}
Time	{h,mn,s}	{<(T)}		{shows}

Method Context

	name	*typeOfParam(1)*
<(Date)	{<}	{Date}
<(Time)	{<}	{Time}
isLeapY()	{isLeapY}	{}

	originType	*destType*	*originMultiplicity*	*destMultiplicity*
orderedBy	{Diary}	{Date}	*	*
shows	{Clock}	{Time}	*	*

Association Context

Fig. 4. Formal contexts for classes, methods and associations

Scaling techniques are used to transform many-valued contexts into binary ones. Values of each many-valued attribute are represented as the objects of scale context where the formal attributes are important properties of these values. In Figure 6, values of `typeOfParam(1)` are scaled considering the specialization order (inheritance) on classes: a value (a class C) for `typeOfParam(1)` is associated in the scale with all super-classes of C and C (`All` represents the top of the class hierarchy). Note that the available class organization is replicated in the scale lattice (which basically represents a nominal scale). The concept lattice of the scaled method context (see Figure 7) contains four concepts: `m4` represents `isLeapYear()`; `m1` and `m2` represent both initial

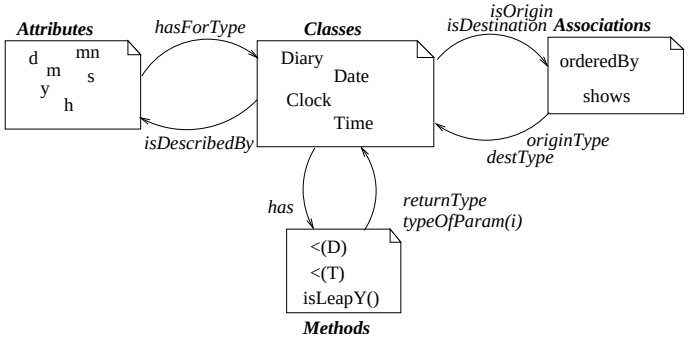


Fig. 5. Relations between the formal contexts

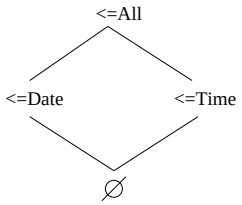
comparison methods, respectively, and `m3` introduces a generalized method $<$. Bottom and top are skipped since useless here.

	name	*typeOfParam(1)*
<(Date)	{<}	{Date}
<(Time)	{<}	{Time}

Many−valued Method Context

	<=Date	<=Time	<=All
Date	x		x
Time		x	x

Scale Context for typeOfParam(1)



Scale for typeOfParam(1)

	name:<	*typeOfParam(1):Date*	*typeOfParam(1):Time*	*typeOfParam(1):All*
<(Date)	x	x		x
<(Time)	x		x	x

Scaled Method Context

Fig. 6. Scaling the method context

The method lattice (Figure 7) is now used as a scale for the formal attribute `has` owned by classes. Thus, if a class has a method $meth$ in the initial many-valued context, then it owns all the formal attributes `has:m` in the scaled class context where `m` stands for a method concept whose extent contains the formal object representing $meth$. The resulting scaled class context and its lattice (with top and bottom dropped) are shown in Figure 8. The lattice includes a new concept `c1` which obviously represents comparable objects, hence it could be called `Magnitude`.

Our knowledge about the concept structure on classes has thus grown and the new abstractions can be used as descriptors that could, whenever shared, induce potential abstractions on related contexts. For example, the method context could be fed with the knowledge about `Magnitude` thus prompting a re-consideration of its con-

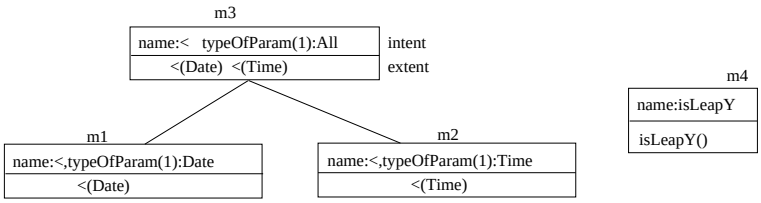
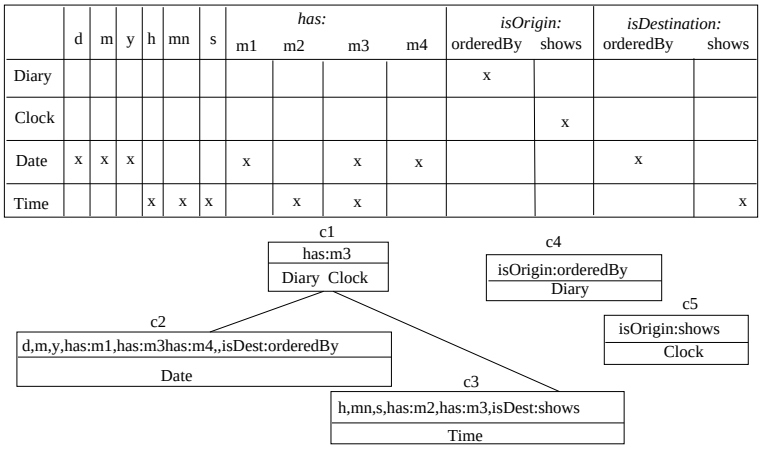


Fig. 7. The first concept lattice on methods

ceptual structure. Thus, three new binary attributes `typeOfParam(1):c2`, `typeOfParam(1):c3` and `typeOfParam(1):c1` replace the initial ones (`typeOfParam(1):All`, `typeOfParam(1):Date` and `typeOfParam(1):Time`). The resulting concept lattice remains isomorphic to that of Figure 7, however its concepts are explicitly related to existing concepts on classes, e.g., the top concept intent is bound to `c1` via `typeOfParam(1):c1`.



	d	m	y	h	mn	s	*has:* m1	m2	m3	m4	*isOrigin:* orderedBy	shows	*isDestination:* orderedBy	shows
Diary											x			
Clock												x		
Date	x	x	x				x		x	x			x	
Time				x	x	x		x	x					x

Fig. 8. Scaling the class context (up), class concept lattice (down) without the top and the bottom

The same procedure can be applied for scaling the association context, revealing that the two formal objects can be generalized by a new association which ends into the `c1` concept. The scaling of `isOrigin` and `isDestination` from the class context, using the augmented association lattice, introduces a new generalization of `Diary` and `Clock` (representing devices which manipulate magnitudes). The resulting set of abstractions, re-interpreted in UML, is shown in Figure 9. The associations `orderedBy` and `shows` are linked to the new association `manipulate` by the constraint `subset` which indicates a specialization relationship. Because of this constraint, their names are now prefixed by the symbol "/", used in UML for highlighting elements that derive from others.

To sum up, we may claim that the apparent commonalties between the classes `Time` and `Date` have led to the constitution of common superclass, `Magnitude`. The discovery of this class has been propagated to both method and association contexts where new abstractions have been created to reflect the existence of `Magnitude`. Finally, these new abstractions reflected reversely on classes where a superclass of `Diary` and `Clock` emerged. New generalizations are especially useful in design: *e.g.* new general classes given with their abstract methods can serve as type for writing generic code; new general associations clarify the model as specializations like `orderedBy` and `shows` can disappear in overviews of the model; new classes can factorize attributes, methods, associations added in further development, etc.

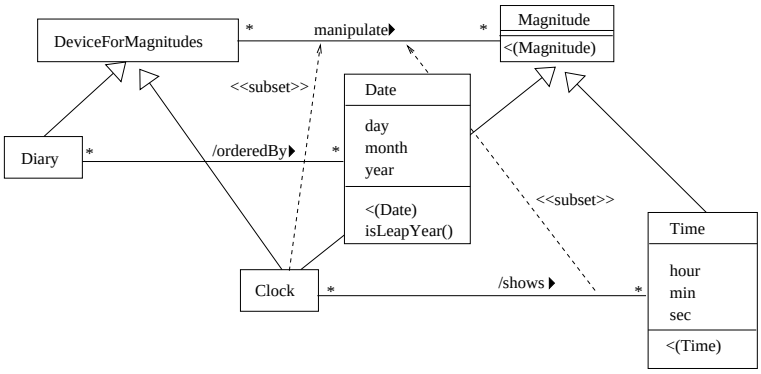


Fig. 9. Diary and clock after iterative cross generalization

4 Bringing relational concepts to core FCA

In the following, we summarize the key elements of our relational FCA framework. A detailed description could be found in [10].

As in classical FCA, heterogeneous datasets, i.e., ones made out of several sorts of individuals, are introduced through a family of contexts, one per sort of formal objects. Here, a set of binary relations (or set-valued functions) is added to data description, which map objects from a context to sets of objects from another one.

Definition 1 (Relational context family).
A relational context family $\mathcal{R}^s$ *is a pair* $(\mathbf{K}_\mathcal{R}, \mathcal{A}_\mathcal{R})$ *where* $\mathbf{K}_\mathcal{R}$ *is a set of* s *multi-valued contexts* $\mathcal{K}_i = (O_i, A_i, V_i, J_i)$ *and* $\mathcal{A}_\mathcal{R}$ *is a set of* p *relational attributes (set-valued functions)* α_j *such that for each* j, $1 \leq j \leq p$ *there exist* r *and* q *in* $[1, s]$ *with* $\alpha_j : O_r \rightarrow 2^{O_q}$.

For instance, the data in our running example (see Section 3) constitute a RCF with four contexts and nine relations.

Conceptual scaling [11] is a FCA technique that transforms a many-valued context $\mathcal{K} = (O, A, V, J)$ into binary one $\mathcal{K}^d = (O, A^d, I^d)$ by replacing non-binary attributes from A by a set of binary ones, called *scale attributes*. Scale attributes basically describe meaningful features of the values of the initial attribute, say a, and therefore induce a lattice of concepts, called the *scale lattice*, on top of the value set $V(a)$. By bringing those attributes to the objects from O, conceptual scaling allows new concepts to occur in which the members of the extent share abstractions of the initial values rather than values themselves. Clearly, the choice of scale attributes has a direct impact on the structure of the target concept lattice: different attribute sets may lead to different lattices.

The same principle may be applied to the processing of relations which are basically object-valued attributes: given a relation $\alpha : O_r \rightarrow 2^{O_q}$ and $o \in O_r$ the set $\alpha(o)$ could be replaced by a collection of binary attributes that characterize it. As the entire process is ultimately aimed at detecting commonalties in the abstractions that conceptually describe the target objects, the scaling binds scale attributes to existing concepts on the co-domain context rather than to formal attributes of this context (see the attributes `typeOfParam(1):cX` from Section 3). Moreover, as we argued in [10], the most natural choice for the scale lattice of α is the lattice of the context $\mathcal{K}_q$ since it embeds the most precise knowledge about the meaningful abstractions on the set O_q. However, in specific situations smaller structures, such as the GSH, may be more appropriate.

Consider an object o_i from $\mathcal{K}_r$ and its encoding in terms of concepts from $\mathcal{K}_q$. Thus, given a concept c_j from $\mathcal{L}_q$, the corresponding binary attribute $\langle \alpha : c_j \rangle$ will be incident to o_i depending on the way the image set of objects $\alpha(o_i)$ compares to $Extent(c_j)$. Two different encoding schemes are possible, i.e., o_i gets $\langle \alpha : c_j \rangle$ whenever: 1) $\alpha(o_i) \subseteq Extent(c_j)$ ("narrow") or 2) $\alpha(o_i) \cap Extent(c_j) \neq \emptyset$ ("wide"). The "narrow" scheme clearly fits lattice-shaped scales whereas the "wide" one suits also less extensive concept structures.

To sum up, the encoding by concepts rather than by formal attributes from the destination context $\mathcal{K}_q$ eases the interpretation of the formal concepts discovered in the source context $\mathcal{K}_r$. Moreover, such an encoding fits a step-wise discovery of the scale concepts as illustrated in Section 3: the formation of some new concepts within $\mathcal{K}_q$, e.g., through refining of the descriptions of the context objects, results in the addition of new scale attributes in the encoding of O_r along α.

Given a relational context family $\mathcal{R}^s$, our aim will be to construct s lattices of formal concepts $\mathcal{L}_i$, $(0 \leq i \leq s)$, one per context $\mathcal{K}_i$, such that the concepts reflect both shared attributes and similarities in object relations, i.e., common concepts in the co-domain context. Obviously the relational scaling helps to reduce the lattice construction on relational data to the binary case so that the same algorithmic procedures could be applied. However, unlike conventional constructions, some RCF may require a step-wise construction process due to the mutual dependencies between contexts, as it was shown in the UML model analysis. Indeed, having aligned scales with actual concept hierarchies on the destination contexts, an apparent deadlock occurs whenever two contexts are connected both ways by a pair of relational attributes (or chains of such attributes). For instance, in Figure 5, the class context is doubly connected to the method one by the initial attribute pair (`typeOfParam(i)`, `has`).

To resolve the deadlocks resulting from circularity in the relational structure of a RCF, we apply a classical fixed-point computation mechanism that proceeds stepwise. Its grounding principle lies in the gradual incorporation of new knowledge gained through scaling: the computation starts with uniformly nominal scales for all relational attributes and at each subsequent step uses the previously discovered concept structures within the respective co-domain contexts as new and richer scales.

Technically speaking, the global lattice extraction process associated with a RCF alternates between relational scaling and lattice construction (see Algorithm 1). At the initial step, relations are ignored (line 5), hence the lattices at this stage (line 6) are not impacted by the relational information and rather reflect common non-relational attributes. At the following step these lattices are used as new scales for a first-class relational scaling thus providing new possibilities for generalizations (lines 10-11). The scaling (line 10) / construction (line 11) steps go on until the global set of concepts stabilizes, i.e., for each context $\mathcal{K}_j$ the lattice $\mathbf{L}^i[j]$ at the i-th step is isomorphic to the one of the $i-1$-th, $\mathbf{L}^{i-1}[j]$ where $\mathbf{L}^i$ denotes the array of contexts at step i. Stabilization of the process can be deduced from the fact that the formal objects do not change over the steps; the concept number of the lattice associated with $\mathcal{K} = (O, A, V, J)$ is bounded by $2^{min(|O|,|A\times V|)}$, which gives a bound to the scaling of relational attributes.

```
1: proc MULTI-FCA( In: R^s = (K_R, A_R) a RCF,
2:                 Out: L array of lattices )
3: i ← 0 ; halt ← false
4: for j from 1 to s do
5:   K_j^d ← SCALE-BIN(K_j)
6:   L^i[j] ← FCA(K_j^d)
7: while not halt do
8:   i++
9:   for j from 1 to s do
10:    K_j^d ← EXTEND-REL(K_j^d, L^{i-1})
11:    L^i[j] ← FCA(K_j^d)
12:  halt ← ⋀_{j=1,s}(L^i[j] = L^{i-1}[j])
```

Algorithm 1: Construction of the set of concept lattices corresponding to a RCF.

Once the lattices of all contexts are available, a post-processing step clarifies the links between concepts induced by relational scale attributes. In fact, many concept intents will present redundancies: a concept c from $\mathcal{K}_r$ related, via $\langle\alpha : c_1\rangle$, to c_1 from $\mathcal{K}_q$ will necessarily be related to all the super-concepts of c_1. Thus, for each super-concept c_2 of c_1, c will possess the attribute $\langle\alpha : c_2\rangle$. As the latter reference does not add new information with respect to $\langle\alpha : c_1\rangle$ to c_1, it may be deleted. This means that in the intent of c, among all the binary attributes $\langle\alpha : c_i\rangle$, only those corresponding to minimal c_i will be preserved. For instance, in Figure 8, the attribute `has:m3` is redundant in the intent of class concept `c2` since `c2` also owns `has:m1`, whereas `m3` is a super-concept of `m1` in the method lattice.

5 Specifying the Iterative Cross Generalization Process

UML class diagrams (models) in more details In class diagrams, classes are associated to structural features (attributes) and behavioral features (operations and methods). In Figure 10 (top) the main elements of attribute and method description are presented: visibility (+, - and #); attribute types *e.g.* `String`, `Point` or `Color` which can be classes; return type; parameter type list; multiplicity for many-valued attributes (like `color`), the multiplicity is a set of integer intervals restricting the value number (for color, multiplicity `1..*` expresses the fact that color has one or more value); static status (underlined feature); derived status (introduced by /).

Figure 10 (bottom) also illustrates the main aspects of UML associations. An association is composed of at least two association ends. When it has a name (for example `place_order`), the name is followed by a triangle which establishes the direction for reading this name: a person places an order and not the other way round. An association end is typically characterized by: a type (the class C involved in this end), for example `Person` and `Order` are the two end types of the association `place_order`; a visibility; a multiplicity; a navigability (shown through an arrow next to the type end); a white or black diamond which indicates an aggregation or a composition. An association end is sometimes provided with a role name which gives more accurate semantics to objects when they are involved in the link, *e.g.* role $employee$ for a person in association $manage$. When the association owns variables and methods it is considered as an association class, *e.g.* `Access` is an association class that supports the variable `passwd`.

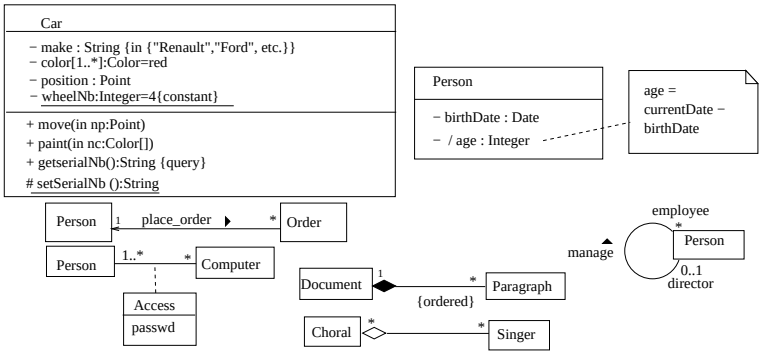


Fig. 10. Classes and associations

Using the UML meta-model to guide the context construction The definition of UML is established by the UML meta-model, that is a model that defines the language for models. The UML meta-model is described through a subset of UML, and is given with well-formedness rules in the formal language OCL (Object Constraint Language), as well as with semantics in natural language. Part of this meta-model [12] relevant

to our problem, that considers the classes and their features, is shown in Figure 11. The meta-class `Class` specializes `Classifier`, and as such, inherits from the possibility to own `Features` (`Attribute` or `Method`). An `Attribute` includes the meta-attributes `initialValue`, `multiplicity`, `visibility`, `changeable`; it has a type via the meta-association that links it to `Classifier`. A `Method` has the meta-attributes `body`, `isQuery`, `visibility`, and is composed of an ordered set of `Parameters`. An `Association` is composed of several `AssociationEnds` which have a type which is a classifier. `AssociationEnds` are described by a type (a classifier), and several meta-attributes including `isNavigable`, `isOrdered`, `aggregation` and `multiplicity`.

As a meta-description of UML, the meta-model naturally contains the good abstractions for determining the right formal contexts: meta-classes are straightly interpreted as formal objects, while meta-attributes and ends of meta-associations are their formal attributes. Nevertheless, such an approach can lead to the manipulation of many tables of data, and to the use of descriptors that generate too numerous uninteresting concepts. `Parameter` for example is preferably included in the description of methods. Associations should be described by an ordered set of association ends, but if we consider only binary and directed associations, as often suggested in modeling [13], we can avoid having a specific formal context for association end description. Conversely, if we want to inspect all possible generalizations of associations in the general case, a formal context describing association ends would be relevant.

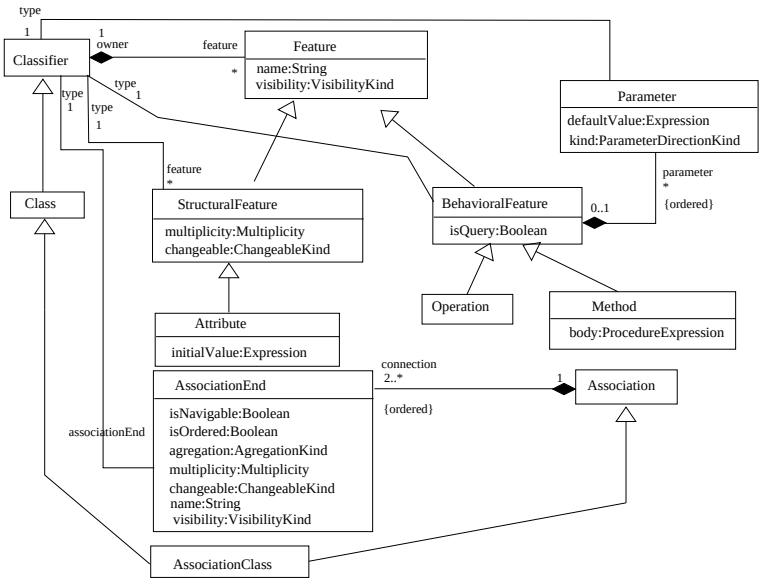


Fig. 11. Extracts from the UML meta-model

In the context of the MACAO project[4], we have considered the relational context family $(\mathcal{K}_{\mathcal{R}}, \mathcal{A}_{\mathcal{R}})$ defined as follows. $\mathcal{K}_{\mathcal{R}} = \{\mathcal{K}_C, \mathcal{K}_{At}, \mathcal{K}_M, \mathcal{K}_{As}\}$. $\mathcal{K}_C = (O_C, A_C, V_C, J_C)$ is the formal context on classes; A_C is empty in our current experiments. $\mathcal{K}_{At} = (O_{At}, A_{At}, V_{At}, J_{At})$ corresponds to the formal context on attributes. A_{At} includes the formal attributes $name$, $multiplicity$, $initialValue$ corresponding to the UML meta-attributes. V_{At} contains the possible values for these formal attributes, *i.e.* possible attribute names, unions of integer intervals, etc. $\mathcal{K}_M = (O_M, A_M, V_M, J_M)$ describes methods. A_M includes the formal attributes $name$, $body$, while V_M contains possible method names, and expressions that represent method bodies. $\mathcal{K}_{As} = (O_{As}, A_{As}, V_{As}, J_{As})$ is the formal context on associations. As we have chosen in our first experiments to consider binary directed associations, the two association ends are called $origin$ and $destination$ and their description is integrated into the formal context for associations. Formal attributes are then $name$, $nameOrigin$, $isNavOrigin$, $isOrderedOrigin$, $multOrigin$, and symmetrical attributes for destination end. $\mathcal{A}_{\mathcal{R}}$ is the set of relational attributes that relate the previous contexts. They are found using the meta-associations between meta-classes `Classifier` and `Attribute`, `Classifier` and `Association`, or `Classifier` and `Method` (going through `Parameter`). They have been presented on the edges in Figure 5. For example we have $has : O_C \rightarrow 2^{O_M}$ or $originType : O_{As} \rightarrow 2^{O_C}$.

6 Experiments

The ICG procedure has been implemented in the Java-based **Galicia**[5] platform [14] and connected to the UML CASE tool Objecteering as part of the MACAO project, thus enabling application of ICG to class diagrams designed within Objecteering. Thus, for a given UML class diagram, RCF is exported[6] in a format which is readable by ICG, which is run and its results are imported back in Objecteering in order to create a new class diagram which can then be studied and compared to the original one. We present here some results of the application of ICG on several medium sized projects of France Télécom. Three different projects have been used for these experiments [15]: *project 1* deals with the management of an information system, ICG was applied to the design model of this project (the model used for Java code generation); *project 2* concerns an intranet software that was also in its design stage; *project 3* is a prospective project regarding the elaboration of a common user data model for several telecommunication services. We have applied ICG to several class hierarchies of project 3: four class hierarchies of service-specific models and the class hierarchy of the common model being specified.

The results of the ICG implementation were shown to the designers of the class hierarchies who gave an appreciation regarding the relevance of the proposed restructuring with respect to the semantics of the underlying data model. The class hierarchies of those projects consist of a few dozens of classes and the number of new UML elements

[4] A joint project of France Télécom, SOFTEAM and LIRMM supported by the French department of research and industry (RNTL); http://www.lirmm.fr/~macao.

[5] See the web site at: `http://www.iro.umontreal.ca/~galicia`.

[6] A limited configuration of RCF is possible within Objecteering.

created by ICG (attributes, methods, classes, inheritance links) may vary from a few to several hundreds in some cases, involving a tedious work of selection and interpretation.

Several new factorization classes or associations proposed by ICG were found absolutely relevant by the class diagram designers. For instance, Figure 12 shows the

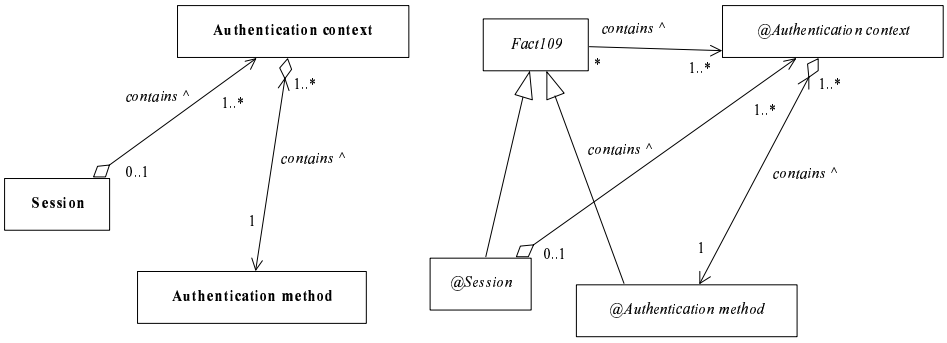


Fig. 12. Factorization of an association

factorization of an association. Left part shows the initial state of the few classes involved and the right part shows the proposed restructuring. The ICG algorithm properly proposes to factorize both associations named `contains^` through the creation of a new class `Fact109` connected to the class `@Authentication context` through a new association with the same name. Notice that the algorithm may propose to factorize role names depending on the way the designer has named the associations: association names, role names or both. This corresponds to the formal attributes *name*, *nameOrigin* and *nameDestination* of the formal context on associations. The multiplicity on the side of the class `@Authentication context` is also properly factorized into a `1..*` multiplicity. On the other hand, one may question the factorization of `0..1` and `1` multiplicities into `*` (it could have been factorized into `0..1`) but this is an internal choice of the algorithm that could be fine-tuned.

7 Conclusion

We presented a new FCA-based technique (ICG) which processes several mutually related formal contexts and sketched its application to UML class diagram restructuring. Experiments on industrial-scale projects established the feasibility of our approach (execution time and semantic relevance of the results) and highlighted the crucial role of parameter tuning and appropriate user interface. A key track of improvement is the separation of formal attributes that guide the construction of new abstractions (*e.g.* names, types of attributes, association ends, etc.) from secondary ones that only help to increase the precision (*e.g.*, multiplicity or navigability). Another current concern is the integration of a domain ontology into the ICG framework that should enable the comparison

of symbolic names used by the designer. This is crucial for any automated reconstruction technique such as our, because terms are not uniformly used over UML diagrams, many synonymy, homonymy or polysemy situations occur. Although Objecteering offers an operational user interface for ICG there is a large space for improvement. First, designers that are FCA neophytes would benefit from an automated assistance in tool fine-tuning. Second, navigation and edition tools should help make the entire ICG process more interactive and thence more purposeful, e.g., by supporting run-time filtering of the discovered abstractions.

References

1. Snelting, G., Tip, F.: Understanding class hierarchies using concept analysis. ACM Transactions on Programming Languages and Systems **22** (2000) 540–582
2. Godin, R., Mili, H.: Building and maintaining analysis-level class hierarchies using Galois lattices. In: Proceedings of OOPSLA'93, Washington (DC), USA. (1993) 394–410
3. Dicky, H., Dony, C., Huchard, M., Libourel, T.: On Automatic Class Insertion with Overloading. In: Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'96. (1996) 251–267
4. 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** (1998)
5. Huchard, M., Leblanc, H.: Computing Interfaces in Java. In: Proc. IEEE International conference on Automated Software Engineering (ASE'2000), 11-15 September, Grenoble, France. (2000) 317–320
6. Yahia, A., Lakhal, L., Cicchetti, R., Bordat, J.: iO2 - An Algorithmic Method for Building Inheritance Graphs in Object Database Design. In: Proceedings of the 15th International Conference on Conceptual Modeling ER'96. Volume 1157. (1996) 422–437
7. Yahia, A., Lakhal, L., Bordat, J.: Designing Class Hierarchies of Object Database Schemas. In: 13 ièmes journées Bases de Données Avancées. (1997) 371–390
8. Ganter, B., Wille, R.: Formal Concept Analysis, Mathematical Foundations. Springer, Berlin (1999)
9. Barbut, M., Monjardet, B.: Ordre et Classification: Algèbre et Combinatoire. Volume 2. Hachette (1970)
10. Valtchev, P., Rouane, M.H., Huchard, M., Roume, C.: Extracting Formal Concepts out of Relational Data. In SanJuan, E., Berry, A., Sigayret, A., Napoli, A., eds.: Proceedings of the 4th Intl. Conference Journées de l'Informatique Messine (JIM'03): Knowledge Discovery and Discrete Mathematics, Metz (FR), 3-6 September, INRIA (2003) 37–49
11. Ganter, B., Wille, R.: Conceptual Scaling. In: Applications of combinatorics and graph theory to the biological and social sciences. Volume 17 of The IMA volumes in Mathematics and its applications., New York (1989) 139–167
12. Rational Software Corporation: UML v 1.3, Semantics. version 1.3 edn. (1999)
13. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object Oriented Modeling and Design. Prentice Hall (1991)
14. Valtchev, P., Grosser, D., Roume, C., Hacene, M.R.: GALICIA: an open platform for lattices. In B. Ganter, A.d.M., ed.: Using Conceptual Structures: Contributions to 11th Intl. Conference on Conceptual Structures (ICCS'03), Aachen (DE), Shaker Verlag (2003) 241–254
15. Dao, M.: Validation sur de grands projets, Projet MACAO (RNTL). Technical Report sous-projet MACAO 5.1, France Télécom R&D (2003)