TOWARDS PRACTICAL TOOLS FOR MINING ABSTRACTIONS IN UML MODELS

M. Dao
France Télécom R&D/MAPS/AMS
38-40 rue du général Leclerc – 92794 Issy Moulineaux Cedex 9, France
michel.dao@francetelecom.com

M. Huchard
LIRMM, CNRS et Université Montpellier II, UM 5506
161 rue Ada – 34392 Montpellier cedex 5, France
huchard@lirmm.fr

M. Rouane Hacène, C. Roume, P. Valtchev
DIRO, Université de Montréal
C.P. 6128, Succ. "Centre-Ville" – Montréal, Québec, Canada, H3C 3J7
{*rouanehm, roume, valtchev*}*@iro.umontreal.ca*

Keywords: Formal concept analysis, UML class diagram, Galois lattice.

Abstract: We present an experience of applying an extension of Formal Concept Analysis to UML class model restructuring. The Relational Concept Analysis (RCA) mines potentially useful abstractions from UML classes, attributes, operations and associations and therefore outscores competing restructuring techniques which usually focus exclusively on classes. Nevertheless, the complexity and the size of the RCA output require interactive tools to assist the human designers in comprehending the corresponding class model. We discuss the benefits of using RCA-based techniques in the light of an initial set of tools that were devised to ease the navigation and the visual analysis of the results of the restructuring process.

1 INTRODUCTION

Current trends in object-oriented software construction, namely MDA[1]-based approaches (Object Management Group, 2003; Mellor et al., 2004), promote designing high-level models concerned with domain and application concepts ("Platform Independent Models") which are belatedly mapped to the target implementation platform ("Platform Specific Models"). This emphasizes the importance of the model construction activity and, hence, the design of tools to support it.

One of the highly desirable properties of a class model is the right use of relevant abstractions, allowing readability and maximal factorization of the underlying hierarchical structure, i.e., the optimal use of inheritance to avoid duplication of specifications/code. Many automated approaches have been proposed in the literature for the design of a well factorized hierarchy out of a flat set of classes, for the restructuring of an existing one in order to improve its factorization level or for the composition of class hierarchies (Moore, 1996; Casais, 1995; Snelting and Tip, 2002).

Our work is based on Formal Concept Analysis (FCA) which is a mathematical approach towards clustering and categorization (Ganter and Wille, 1999). FCA discovers useful abstractions from (individuals $\times$ properties) data expressed as binary tables, the formal contexts. The abstractions, called (formal) concepts, are hierarchically organized into a lattice, the concept lattice of the context[2]. FCA has been successfully applied to the refactoring of class hierarchies, with classes and class members substituted to individuals and properties, respectively (Godin and Mili, 1993). Nevertheless, taking full advantage of the expressive power of UML (Rational Software Corporation, 1999; Object Management Group, 2004) requires relational information — e.g., associations that relate two or more classes, attributes whose type is a class, operations whose parameter and return types are classes — to be successfully processed. However, this sort of knowledge about entities is far

[1] Model-Driven Architecture.

[2] An excellent introduction to lattice theory is provided in (Davey and Priestley, 2002).

beyond the scope of the classical FCA framework.

Taking into account all the structural knowledge encoded into a class model was the motivation behind the relational extension of FCA (RCA) as described in (Valtchev et al., 2003b; Dao et al., 2004). RCA relies on a data model comparable to E-R as it is made of a set of individual types, each introduced by its own context, and a set of inter-context relations. The underlying algorithmic method, called ICG, structures the individuals of each type into a concept lattice where the concepts are described both by the shared properties of their member individuals and by the links to concepts from related contexts. The key advantage of RCA is that it allows the abstraction knowledge to "flow" among related contexts: the discovery of a new concept on one context triggers the formation of new concepts on every related context. In the software engineering translation of RCA, the individuals are instances of relevant UML meta model (meta)classes. For the sake of simplicity, the theoretical description is restricted in this paper to classes and associations: the set of classes and of associations are assigned a type each while a collection of relations express the various aspects of the incidence between a class and an association. The whole framework, applied in the experience, also considers abstractions of properties and operations.

In this paper we describe an industrial experience of applying RCA to software engineering tasks. The experience was carried out within the MACAO project[3] and consisted of the restructuring of a set of class diagrams of different UML models. The key challenge here was not that much the processing of the relevant knowledge and the extraction of abstractions as the sorting of the newly created elements in the final diagram to select only relevant and useful ones. To assist the user in inspecting these elements, we provided a set of tools that postprocess the results of RCA on a UML diagram and that allow the designer to browse those results more easily.

Section 2 is a brief introduction to both the classical FCA and its relational version, RCA. Section 3 presents the way ICG was applied in our experiment whereas section 4 discusses the issues related to the analysis of the obtained results.

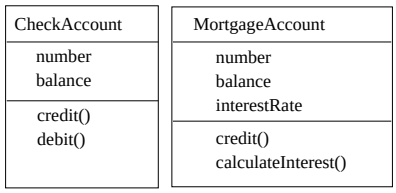
[3]A joint project of France Télécom, SOFTEAM and LIRMM, supported by the french department of research and industry (RNTL).

2 ICG: A NEW METHOD FOR MINING ABSTRACTIONS IN RELATIONAL DATA

The method ICG is rooted on FCA techniques. Classical FCA aims at mining concepts in a set of entities described by properties. It has been successfully used in several software enginering applications for class hierarchy analysis (Snelting and Tip, 2000; Arévalo and Mens, 2002) or construction (Godin and Mili, 1993; Dicky et al., 1996; Godin et al., 1998; Huchard et al., 2000).

However, as detailed below, classical FCA is unable to process interrelated entity descriptions like those extracted from a UML class diagram. This was the key motivation behind the design of RCA and its main algorithmic method, ICG.

2.1 Classical Formal Concept Analysis



Entities and their description

E \ P	nb number	b balance	i interestRate	cr credit()	d debit()	ci calculateInterest()
CA CheckAccount	X	X		X	X	
MA MortgageAccount	X	X	X	X		X

Formal context

Figure 1: A formal context describing classes.

In classical FCA, a set of concepts provided with a specialization order (the concept or Galois lattice) emphasizes commonalities in descriptions (by property sets) of entities. Concepts emerge from a formal context $\mathcal{K} = (E, P, I)$ where E is the entity set, P the property set and I associates an entity with its properties: $(e, p) \in I$ when entity e owns property p. Figure 1 provides an example of context, where entities are classes and properties are their attributes and opera-

tions (in the table representing I, abbreviations are proposed to be used below).

Any entity set $X \subseteq E$ has an image in P defined by $X' = \{p \in P \mid \forall e \in X,\ (e,p) \in I\}$. Symmetrically, any property set $Y \subseteq P$ has an image in E defined by $Y' = \{e \in E \mid \forall p \in Y,\ (e,p) \in I\}$. In the example, let $Y = \{nb, b\}$, we have $Y' = \{CA, MA\}$, while for $X = \{CA, MA\}$, $X' = \{nb, b, cr\}$.

A concept is a pair (X, Y) where $X \subseteq E$, $Y \subseteq P$, $X' = Y$ and $Y' = X$. In Figure 1, $\{\{CA, MA\}, \{nb, b, cr\}\}$ is a concept. X (resp. Y) is usually called the extent (resp. intent) of the concept.

The specialization order between concepts corresponds to extent inclusion (or intent containment). In Figure 1, the concept $\{\{CA\}, \{nb, b, cr, d\}\}$ specializes the concept $\{\{CA, MA\}, \{nb, b, cr\}\}$. Figure 2 shows the whole concept lattice (top) and an interpretation of concepts as UML classes (bottom). In this UML interpretation, a new $Account$ class appears, which corresponds to the top concept of the lattice. This can be considered as a learned abstraction. Class hierarchies constructed using this technique have several strong properties including maximal property factorization and conformity between inheritance links and property set inclusion.

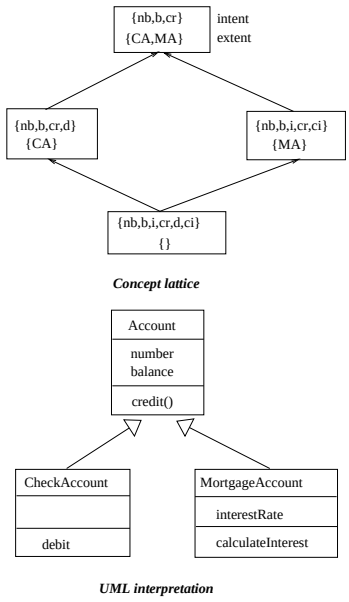


Figure 2: The concept lattice for account concepts, interpretation in UML.

2.2 Relational Concept Analysis

Although powerful, classical FCA fails taking into account realistic UML diagrams that contain interrelated entities. Consider for example the very simple extension of our first example shown in Figure 3. Classes $CheckAccount$ and $MortgageAccount$ are involved in associations, called $owns$ with similar role names. Description of classes now also depends on other entities, namely associations, as classes are involved in association ends. Similarly, considering associations as entities, among their properties, we can find their names, role names, and the classes they link. Attributes and operations can also be considered like first-class entities, described by properties including their name, type, parameter type list, etc. The types being classes, this description here also depends on entities of another sort.

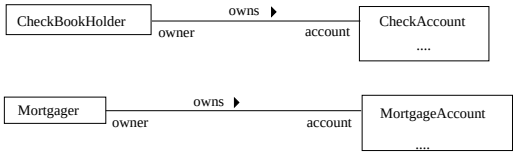


Figure 3: Extending the account example.

In RCA, this sort of data is introduced as a Relational Context Family (RCF), a family of contexts (one for each sort of entity) provided with inter-context relations linking entities of two sorts. In the case of UML diagrams, a basic RCF includes a context for classes and another one for associations, where a set of relational properties model the incidence between classes and associations. For instance, a relation will link a class to an association if objects at the end of the association are instances of the class. ICG is a method for extracting formal concepts out of a RCF. It performs iterative lattice construction on all contexts of the RCF doubled by an enrichment of entity descriptions. Thus, at each step, the contexts are extended to incorporate the conceptual knowledge learned at the previous step. Figure 4 shows a RCF for the extended account example which is composed of a class context (top) and an association context (bottom). Relational properties like $originOf$ or $origType$ have been included in the binary tables (in italics).

At the first construction step, class context is processed, producing the new $Account$ class seen before (Figure 2 bottom) but no concept (then no class at interpretation step) for generalizing M and CBH which have nothing common. Before processing the association context, the fact that a new class has been constructed is integrated. In Figure 5 it is shown how the association context is extended by a new column $destType = Account$ in order to precise that, as instances at the end of own associations links are (respectively) from classes CA and MA, they are also, thanks to class specialization, instances of the new class $Account$. Constructing the concept lattice will produce a concept then interpreted as a new associa-

$P_{classes}$ / $E_{classes}$	nb number	b balance	i interestRate	cr credit()	d debit()	ci calculateInterest()	*originOf(ownsCBH−CA)*	*originOf(ownsM−MA)*	*destOf(ownsCBH−CA)*	*destOf(ownsM−MA)*
CA CheckAccount	X	X		X	X				X	
MA MortgageAccount	X	X	X	X		X				X
CBH CheckBookHolder							X			
M Mortgager								X		

Relational properties

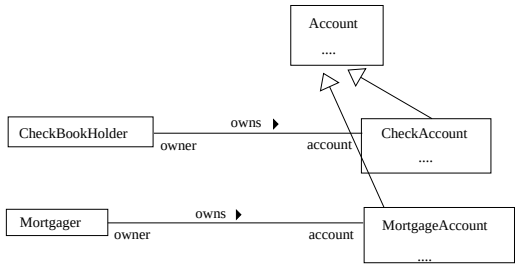
Formal context for classes

P_{assoc} / E_{assoc}	name=owns	originRole=owner	destRole=account	*origType=CBH*	*origType=M*	*destType=CA*	*destType=MA*
ownsCBH−CA	X	X	X	X		X	
ownsM−MA	X	X	X		X		X

Relational properties

Formal context for associations

Figure 4: A partial relational context family for the extended account example.



Evolution of classes

P_{assoc} / E_{assoc}	name=owns	originRole=owner	destRole=account	*origType=CBH*	*origType=M*	*destType=CA*	*destType=MA*	***destType=Account***
ownsCBH−CA	X	X	X	X		X		X
ownsM−MA	X	X	X		X		X	X

Relational properties

Enhancement of the association context

Figure 5: Integrating $Account$ class in the association context.

tion, refered afterwards as $NEWowns$ that generalizes $ownCBH - CA$ and $ownM - MA$ (see Figure 6).

Returning to the class context, this new association $NEWown$ will be "shared" by classes M, CBH, MA and CA in two ways. Firstly, instances of M and CBH can be origins of $ownCBH-CA$ or $ownM-MA$ links then they can also be origins of links of an association that represents their disjunction. This appears in Figure 7 (top) where M and CBH own the property $origOf(NEWown)$. This results in a new class, we call it $Client$, which generalizes M and CBH. Secondly, classes MA and CA now share the newly added property $destOf(NEWown)$, which then will be integrated in the definition of class $Account$. The process stops there because no new generalization appears in next steps.

For a formal definition of RCA and ICG the reader is referred to (Huchard et al., 2002; Valtchev et al., 2003b; Dao et al., 2004).

3 APPLYING ICG ALGORITHM TO REAL WORLD MODELS

The ICG procedure has been implemented in the Java-based GALICIA platform (Valtchev et al., 2003a). The implementation has been connected to the UML CASE tool Objecteering[4] (part of the MACAO project) thus enabling the application of ICG to UML class diagrams designed within Objecteering. Thus, for a given diagram, derived relational contexts are exported[5] in a format which is readable by GALICIA. ICG within the platform is run and its results are imported back in Objecteering in order to create a new class diagram which is then studied and compared to the original one.

We present here some results of the application of ICG to several medium-sized projects of France Télécom. The projects pertained to different application domains: information systems, intranet software and user data model for telecommunication services. Moreover, among the corresponding UML class diagrams there were both analysis-level and design-level models.

Once constructed, the resulting class diagrams were shown to the designers of the initial models who gave an appreciation as to the pertinence of the proposed restructuring with respect to underlying domain semantics. Class hierarchies of those projects consist of several dozens of classes while the number of new UML elements created by ICG (attributes, operations, classes, inheritance links) may vary from a few to several hundreds in some cases. Many new factorization classes or associations proposed by ICG were found absolutely useful by the class diagram designers.

[4] `http://www.objecteering.com`.

[5] A limited parametrization of the constitution of the relational contexts is possible within Objecteering.

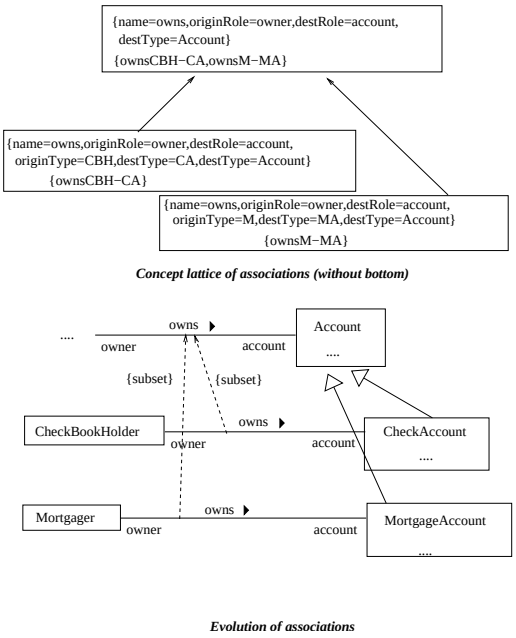


Figure 6: A new own association appears.

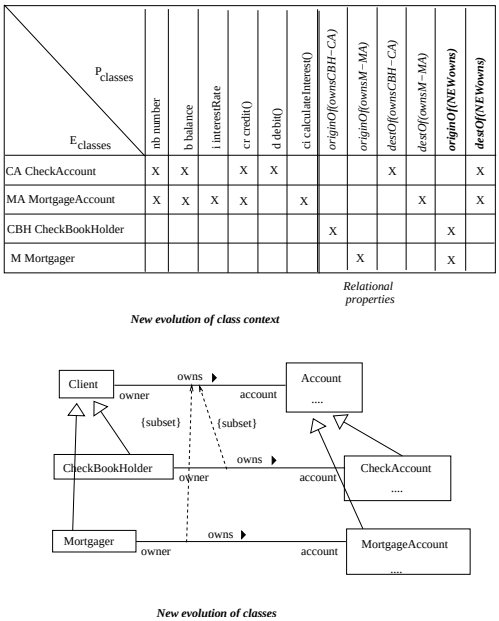


Figure 7: A new $Client$ class appears.

For instance, the diagram of Figure 8 shows the intial state of a small part of a class hierarchy. The gray background of the three right-hand-side classes indicates that they are in a different package than those on the left-hand-side. The ICG algorithm was applied once exclusively on the package containing the four target classes and once on both packages. In the first case (upper part of Figure 9), the associations have not been taken into account, hence the merge of the two classes `Subscribing client`[6] and `Owning client` that declare only the attribute `civil status`. Class `Paying client` inherits from this class and declares the additional attributes that it declared in the initial diagram. In the second case (lower part of Figure 9), a new class `Fact217` has been created by the algorithm to factorize the attribute `civil status` of classes `Paying client`, `Subscribing client` and `Owning client`. It is noteworthy that the associations ending on those three classes (`has >`, `sends >`, `owns >`) ensure that classes `Subscribing client` and `Owning client` remain present in the new class hierarchy. Indeed, `civil status` should be declared in the initial class `Client` but the ICG algorithm has no way to detect such a fact: `Client` does not appear in the modified diagram as it does not declare any specific attribute or operation. This is where the knowledge of the UML diagram designer gets essential.

[6]In the restructured hierarchy, classes corresponding to initial classes have the same name prefixed by an `@`.

Though ICG has proved useful for the improvement of the project class diagrams, it also brought to the spot some problems or shortcomings, most of which admitted straightforward solutions. First, the present parameter settings of the ICG algorithm may result in the creation of numerous classes whose only function is to factorize some very general information. For instance, in the whole ICG framework, including attribute generalization, a class could be created to factorize the existence of an attribute of type integer (it declares only `newAttrib: integer;`). The same goes for all types of attributes used in the class diagram, for operations and for associations. For the time being, we have chosen to remove those "information-less" classes in the CASE tool, once the final UML hierarchy has been created. It should also be possible to directly modify the ICG algorithm but this would also impact other uses of the GALICIA environment.

FCA and, as a result, RCA approaches rely essentially on the name of the entities and properties on which they operate. This may lead to the situation where two attributes (or any model element, for that matter) of the same name are factorized by the ICG algorithm into a single one, although they correspond to two different pieces of data. Therefore, ICG may provide some support in detecting naming problems.

Last, the amount of new information (new classes, new inheritance links, moving of properties, etc.) generated by the algorithm may be difficult to analyze by the designer of the class diagram. The next section describes a set of tools that have been developped in

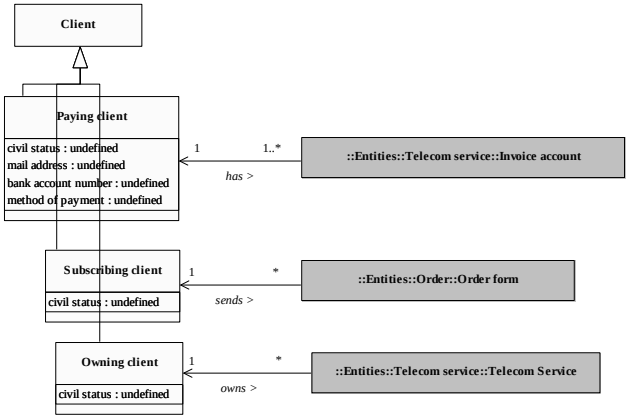


Figure 8: Initial class diagram.

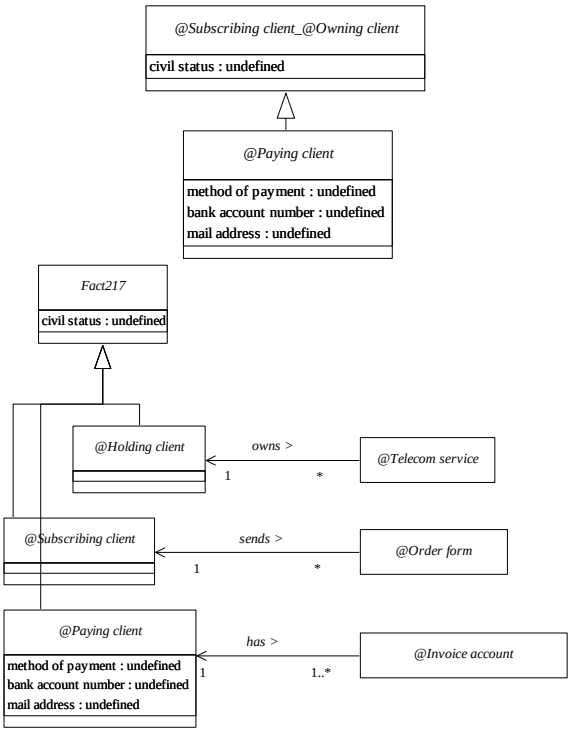


Figure 9: Different results of ICG.

order to assist the class diagram designer during the analysis of the ICG results.

4 HELPING A DESIGNER TO ANALYZE RESULTS OF THE ICG ALGORITHM

The previous section has shown the potential benefits of using the ICG algorithm on existing UML class diagrams in order to suggest improvements in the structure of the class hierarchy. Nevertheless, the experiments described have been carried out by a software engineering practitioner and not by the model designers themselves. Depending on the initial class diagram, the resulting diagram may be substantially different from the initial one: many factorization classes (classes defining properties that are inherited by other classes) may be created, an initial class may be split into several new classes, etc. Thus, it is not always straightforward for a class designer using the tool to retrieve the information present in the initial class diagram.

The main goal of this part of our work was therefore to develop a set of interactive tools that would help a designer to browse and to analyze both the initial and the modified class diagram. The ultimate goal is a tool that can be used by UML model designers on their own. We have so far implemented the following functions:

- mapping elements of the initial diagram (class, attribute, operation, association) on the modified diagram;
- mapping classes of the modified diagram to the corresponding classes of the initial diagram;
- finding in the initial diagram all the attributes or operations of a class from the modified diagram;
- finding in the modified diagram all the attributes or operations of a class from the initial diagram.

In the following sections we first briefly describe the implementation and the interface of some of the above functions and then expand on the use of two of them. We also explain the benefit that a designer may get from them.

4.1 Implementation and interface of some functions

Given a class of the initial diagram, its matching class from the modified diagram is the class that has exactly the same properties. At the end of the ICG algorithm the mapping between initial classes and the matching class is known. In order to easily retrieve the matching class, the initial class contains a tagged value which is initialized with the name of the matching class. Thus, the underlying mapping function simply consists in the output, in a new window, of the matching class.

The declaration of an attribute or an operation that was in a given class of the initial diagram may have been moved to another class in the modified diagram: this is what ICG is expected to do. A function has been implemented in the CASE tool which gathers all the classes that contribute to the definition of the

attribute or the operation in the modified diagram.

4.2 Analyzing the Factorization of Attributes and Operations

We have implemented a function that allows to show how all the attributes and operations of a class of the initial diagram have been dispatched in the modified diagram. Thus, the function displays a subgraph of the modified class diagram containing the classes where the properties of the initial class are declared. Figure 10 shows an example of such a result. The left-hand-side of the figure shows the initial class `RLink` with all its attributes and operations. The right-hand-side shows the result retrieved by the CASE tool function. For instance, the class `Fact375` has been created to factorize the attribute `comment` and the access operations `getComment` and `setComment`. The class allows the information embedded in `comment` to be shared with other classes having these properties. Likewise the class `Fact378` factorizes the operation `getNetwork` and the class `Fact380` factorizes both the operation `getTopology` and a more general definition of the operation `getSupport` that remains declared in the modified class `RLink`.

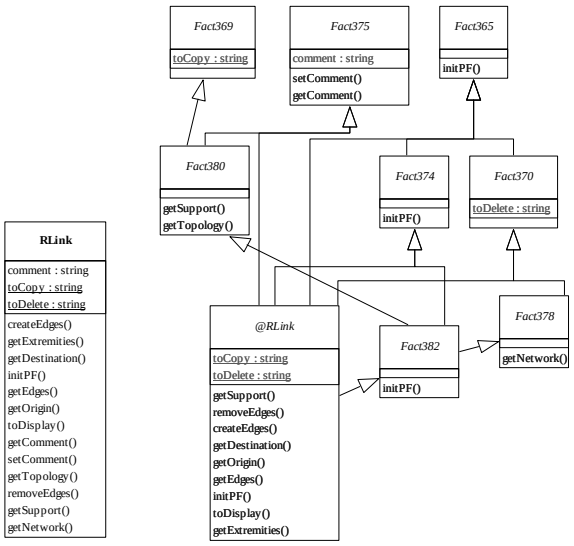


Figure 10: Example of display of all properties of an initial class.

4.3 Analyzing the Factorization of Associations

As for the classes and the properties (attributes and operations), a function allows an association of the initial diagram to be highlighted in the modified diagram. Figure 11 (upper part) shows an example of the use of this function: the association between the initial classes `RMacroLink` and `RUniLink` represents the fact that a `RMacroLink` contains several `RUniLink` and, conversely, that a `RUniLink` may belong to several `RMacroLink`. The association has been factorized to another one, connecting the new classes `Fact393` and `Fact394`.

The existence of the factorization classes points out that some other classes must share the factorized association between them. Another function allows to shows from a factorized association all the initial associations that it has factorized. This is depicted in Figure 11 (lower part) where three initial associations (inclusive the one between `RMacroLink` and `RUniLink`) are shown. This allows the designer to analyze and to question the factorization.

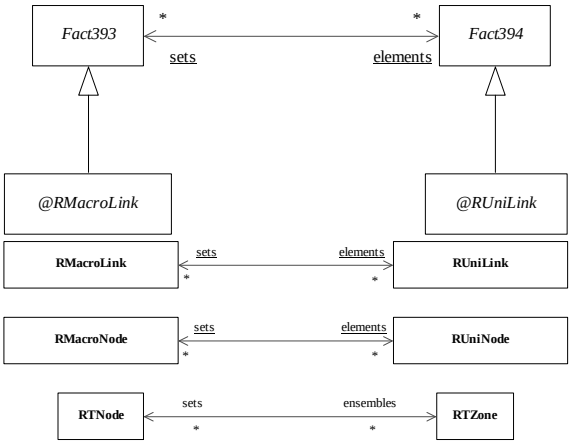


Figure 11: A factorized association and the corresponding initial associations

5 CONCLUSION

We have presented here the first results of the application of an extension of FCA to “real world” UML class diagrams. This extension allows us to take into account richer set of elements from the UML metamodel (e.g., associations, properties, operations and classes as types of model elements) thus improving the proposed factorization of the inspected class diagrams. Class diagram designers have found a large number of relevant factorization propositions among those suggested by the ICG algorithm. Even factorizations that were not immediately recognized as useful highlighted some modeling problems that the designers were not aware of. In order to help designers cope with the potential complexity of the class diagrams produced by the algorithm, we developed a few

interactive tools for browsing class sets efficiently and visually analyze them.

Future research should focus both on the ICG method itself and on the tool support for the exploration of the automated analysis results. First, it would be interesting to provide a more precise and thorough tuning of the ICG algorithm. In particular, the possiblity of ignoring part of the model elements (for instance, some attributes) in the abstraction process should be granted to the ICG user. This would allow the designer to enforce multiple declarations. As shown in the paper, the ICG algorithm may produce numerous so-called "information-less" model elements. It is desirable for the designer to be able to tune the generation of such elements. Moreover, a run-time interaction with the designer would allow the evolution of the algorithm to be influenced in a purposeful manner. Finally, the problems we met with naming conflict resolution within this work suggest that the use of natural language techniques could be helpful for terminology clarification purposes.

This work has shown both that the use of advanced techniques such as RCA may benefit to UML class diagram designers but that it requires high-level interactive tools to assist them in the analysis and even in the possible mining of the results.

REFERENCES

Arévalo, G. and Mens, T. (2002). Analysing Object-Oriented Application Frameworks Using Concept Analysis. In Bruel, J.-M. and Bellahsène, Z., editors, *Advances in Object-Oriented Information Systems - OOIS 2002 Workshops*, number 2426 in LNCS, pages 53–63. Springer.

Casais, E. (1995). Managing Class Evolution in Object-Oriented Systems. In O.Nierstrasz and D.Tsichritzis, editors, *Object-Oriented Software Composition*, pages 201–244. Prentice Hall.

Dao, M., Huchard, M., Rouane Hacène, M., Roume, C., and Valtchev, P. (2004). Improving Generalization Level in UML Models: Iterative Cross Generalizat ion in Practice. In Pfeiffer, H. and Wolff, K. E., editors, *Proceedings of the 12th Intl. Conference on Conceptual Structures (ICCS'04)*, volume 3127 of *Lecture Notes in Computer Science*, pages 346–360. Springer Verlag.

Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition.

Dicky, H., Dony, C., Huchard, M., and Libourel, T. (1996). On Automatic Class Insertion with Overloading. In *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'96*, pages 251–267.

Ganter, B. and Wille, R. (1999). *Formal Concept Analysis, Mathematical Foundations*. Springer, Berlin.

Godin, R. and Mili, H. (1993). Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In *Proceedings of OOPSLA'93, Washington (DC), USA*, pages 394–410.

Godin, R., Mili, H., Mineau, G., Missaoui, R., Arfi, A., and Chau, T. (1998). Design of Class Hierarchies Based on Concept (Galois) Lattices. *Theory and Practice of Object Systems*, 4(2).

Huchard, M., Dicky, H., and Leblanc, H. (2000). Galois Lattice as a Framework to Specify Algorithms Building Class Hierarchies. *Theoretical Informatics and Applications*, 34:521–548.

Huchard, M., Roume, C., and Valtchev, P. (2002). When Concepts Point at other Concepts: the Case of UML Diagram Reconstruction. In *Proceedings of the 2nd Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD)*, pages 32–43.

Mellor, S. J., Scott, K., Uhl, A., and Weise, D. (2004). *MDA Distilled – Principles of Model-Driven Architecture*. Addison Wesley Professional.

Moore, I. (1996). Automatic Inheritance Hierarchy Restructuring and Method Refactoring. In *Proceedings of OOPSLA'96, San Jose (CA), USA*, pages 235–250.

Object Management Group (2003). *MDA-Guide, V1.0.1, omg/03-06-01*.

Object Management Group (2004). *UML 2.0 Superstructure Specification. ptc/04-10-02*.

Rational Software Corporation (1999). *UML v 1.3, Semantics*, version 1.3 edition.

Snelting, G. and Tip, F. (2000). Understanding Class Hierarchies Using Concept Analysis. *ACM Transactions on Programming Languages and Systems*, 22(3):540–582.

Snelting, G. and Tip, F. (2002). Semantics-Based Composition of Class Hierarchies. In Magnusson, B., editor, *ECOOP 2002 - 16th European Conference on Object-Oriented Programming,*, volume 2374 of *LNCS*, pages 562–584. Springer.

Valtchev, P., Grosser, D., Roume, C., and Rouane Hacène, M. (2003a). GALICIA: an Open Platform for Lattices. In B. Ganter, A. d. M., editor, *Using Conceptual Structures: Contributions to 11th Intl. Conference on Conceptual Structures (ICCS'03)*, pages 241–254, Aachen (DE). Shaker Verlag.

Valtchev, P., Rouane Hacène, M., Huchard, M., and Roume, C. (2003b). Extracting Formal Concepts out of Relational Data. In SanJuan, E., Berry, A., Sigayret, A., and Napoli, A., editors, *Proceedings of the 4th Intl. Conference Journées de l'Informatique Messine (JIM'03): Knowledge Discovery and Discrete Mathematics, Metz (FR), 3-6 September*, pages 37–49. INRIA.