

Fixing generalization defects in UML use case diagrams

Xavier Dolques

INRIA, Centre Inria Rennes - Bretagne Atlantique, Campus universitaire de Beaulieu, 35042 Rennes, France
xavier.dolques@inria.fr

Marianne Huchard, Clémentine Nebut and Philippe Reitz

LIRMM, CNRS and Université Montpellier II, Montpellier, France
first.last@lirmm.fr

Abstract. Use case diagrams appear early within a UML-based development, structured over the concepts of actors and use cases to capture the users requirements of an application. Good modeling practices suggest that use case diagrams need to be simple and easy-to-read, two goal that can be achieved by introducing relevant generalizations of actors and use cases. Using this method, the approach presented in this paper allows, using Formal Concept Analysis and one of its variant, Relational Concept Analysis, to refactor a use case diagram as a whole in order to make it clearer while keeping in mind to respect the semantic of the original diagram. The relevancy of this approach has been confirmed by its implementation as a tool and the results obtained from its application on several significant diagrams.

Keywords: Formal Concept Analysis, UML use case diagrams

1. Introduction

Within a UML-based development, the very first model to be designed is the use case diagram, that is later used as a base reference all over the modeling process. For example, in the Rational Unified Process [18], the development process is driven by the use cases: use cases are continuously used to inject further functionalities in static models, to guide the testing plan, etc. Thus even if the use case diagram is probably the furthest one from implementation, it takes a large place in the design of many artifacts. Use case diagrams model the boundary of a system, the actors interacting with the system, and the use cases. A use case is a functionality offered by the system, it is linked to the actors interacting with it and to other use cases by three kinds of relation: inclusion, extension and generalization.

As mentioned in the UML user guide [4], “Organizing your use cases by extracting common behavior (through include relationships) and distinguishing variants (through extend relationships) is an important part of creating a simple, balanced, and understandable set of use cases for your system.” A good practice for use case diagrams is to make them simple to read [2], so as to catch at first glance the boundaries of the system, its main actors and main functionalities. Thus a tangled use case diagram (*e.g.* with numerous and tangled relations linking use cases to actors and to other use cases) will brake the development since a designer, who will regularly consult it as a reference, will have a delay to understand (again) the use case diagram.

In this paper, we investigate an approach to make use case diagrams clearer by introducing generalized actors and use cases to factorize relations. Our approach is based on several refactoring patterns (*e.g.* when two use cases include the same third one, they can be generalized by a new use case including this third use case), that are designed to introduce new abstractions in order to factorize common behavior while keeping the semantic of the initial diagram. Those patterns are globally combined using Formal Concept Analysis (FCA) and one of its variant Relational Concept Analysis (RCA) instead of using a local approach that would reveal to be less efficient. The approach is implemented using model driven engineering facilities in an Eclipse-integrated tool that takes as input a UML diagram, encodes it into FCA (or RCA) contexts, generates the corresponding concept lattices, and finally produces as output the refactored use case diagram. We studied the results of this approach on several examples, comparing the pairs of input diagram/output diagram, and FCA versus RCA.

The following of the paper is structured as follows. We first illustrate in Section 2 the use case diagram refactoring with an example that will be used in the following to illustrate our approach. Section 3 details the used refactoring patterns and difficulties presented by their application. Section 4 explains how they are globally applied using FCA or RCA. Details on the implementation and results of our experiments on examples are given in Section 5. Section 6 compares our approach w.r.t. related work.

2. A motivating example

This section presents a motivating example. We briefly recall the main elements involved in UML use case diagrams, and then illustrate the use case generalization defects. The application we are interested in should allow various users to buy products or apply for internships. Figure 1 shows a use case diagram for this application. Actors represent the role(s) played by external persons or systems that interact with the modelled system. They are depicted using a “stick man” icon (or a box stereotyped `<<actor>>` with the name of the actor). Here, three roles are identified: a company, a registered client, and a registered company. Use cases, depicted by named ellipses, model large functionalities. In our example, we find the use case Apply for Multi-Period Internship, modelling the functionality corresponding to the possibility to apply for internships organized over several periods (for example a day per month). Use cases are associated (graphically by a continuous line) to the actors involved in them: actors to which the functionality is dedicated or secondary actors that interact with the system during the achievement of the functionality. In our example, the use case Apply for Multi-Period Internship is linked to the Registered Client and the Registered Company actors, since both of those actors can apply for such internships (whereas non-registered companies cannot). A use case may include another one. Graphically, it is shown by a dashed oriented arrow stereotyped by `<<include>>`. Semantically, that means that the including use case explicitly incorporates the behaviour of the included use case. For ex-

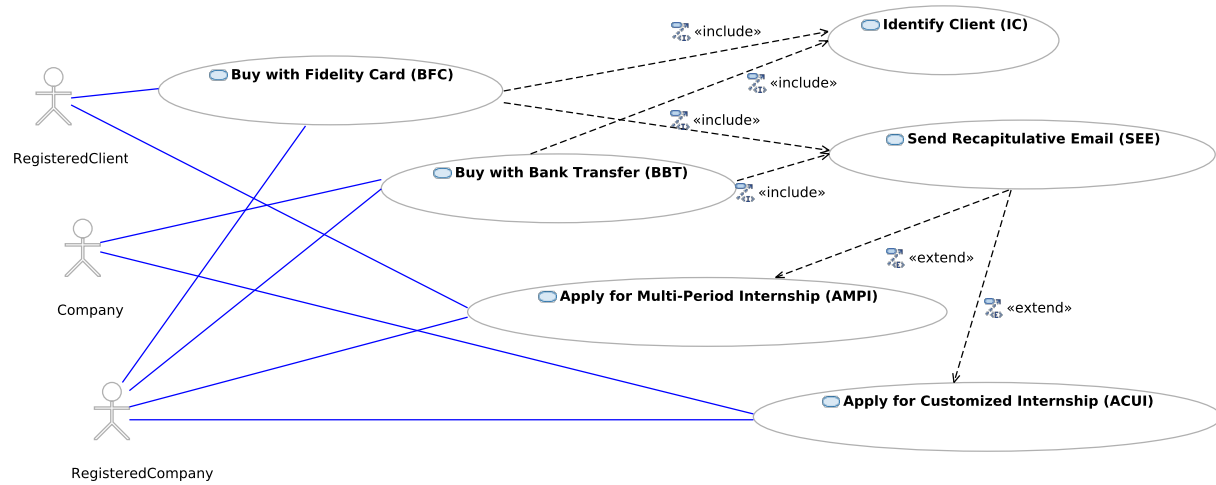


Figure 1. A use case diagram for the internship example

ample, the use case Buy with Bank Transfer includes the use case Send Recapitulative Email. A use case may be extended by another one. Graphically, it is shown by a dashed oriented arrow stereotyped by `<<extend>>`. Semantically, that means that the extended use case *may* include the behaviour of the extending use case. In our example, a recapitulative email may be sent while realizing an appliance for a multi-period internship. Conditions for the extension and extension points can complete the extension relationship, as discussed later in the paper. UML provides two generalization mechanisms in use case diagrams: for actors and for use cases. Figure 2 shows a refactored use case diagram for the internship example, in which those two mechanisms appear. A use case may generalize another one:

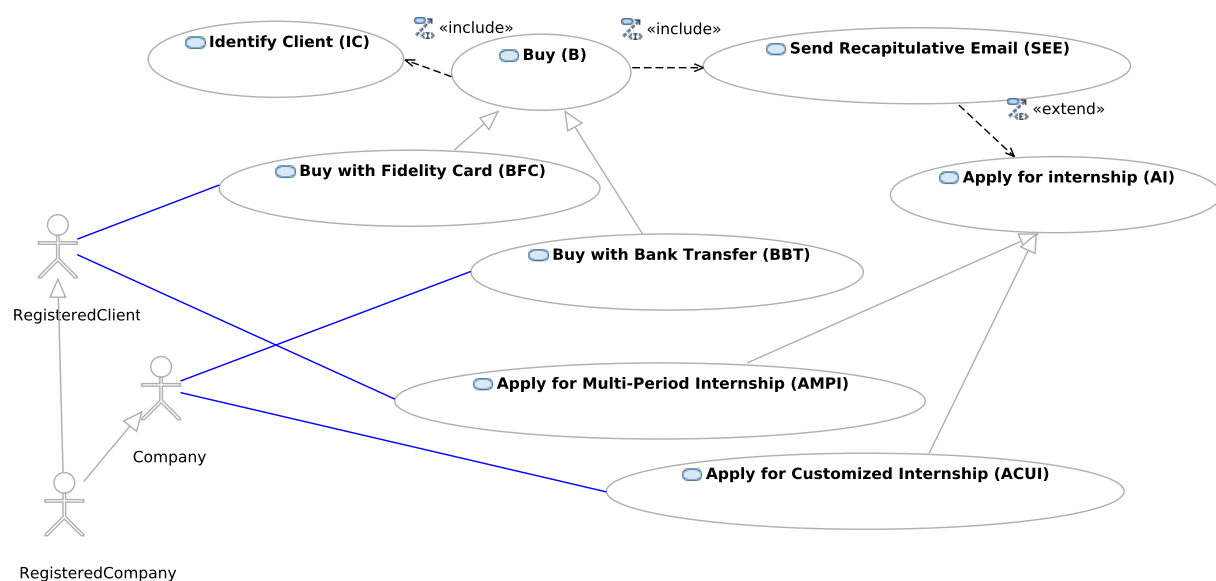


Figure 2. A refactored use case diagram for the internship application using FCA

the child use case inherits the behaviour of the parent use case, as well as its meaning. It is graphically represented by an edge ended by a triangle, like for class inheritance. In the refactored example of Figure 2, the use case Buy generalizes the use cases Buy with Bank Transfer and Buy with Fidelity Card. An actor may generalize another one: the child actor inherits the roles of the parent actor, *i.e.* its interactions with use cases. In Figure 2, the actor RegisteredCompany specializes the actor Company.

The initial diagram of Figure 1 suffers from the large number of crossed links from actors to use cases, and to a lesser extent, from the large number of links from use cases to use cases. The actor RegisteredCompany is linked to four use cases. Introducing a generalization from this actor to the actors Company and RegisteredClient is semantically correct and factorizes the four links: in the refactored diagram proposed in Figure 2, the actor RegisteredCompany inherits its links to the use cases from its two parent actors. Similarly, the two inclusions of the use case Client Identification (resp. Send Recapitulative Email) by the use cases Buy with Fidelity Card and Buy with Bank Transfer can be factorized introducing the sound use case Buy. Last, the two extensions can be factorized introducing the sound Apply For Internship use case.

The approach we propose aims at detecting the possible relevant generalizations in a use case diagram, and introduces them in order to obtain a simpler use case diagram. For that, we first identify (in refactoring patterns) situations for which introducing a generalization may be useful. Then to automatically apply these patterns, we use Formal Concept Analysis (FCA) or one of its variants Relational Concept Analysis (RCA). The refactoring patterns are presented in the next section while the FCA/RCA application is detailed in Section 4.

3. Refactoring patterns

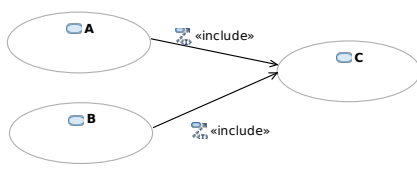
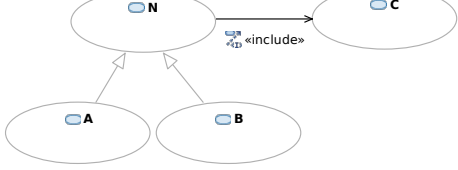
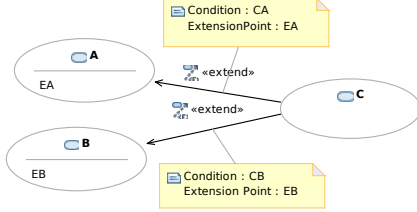
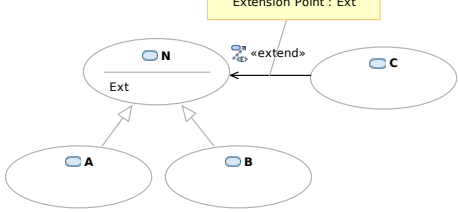
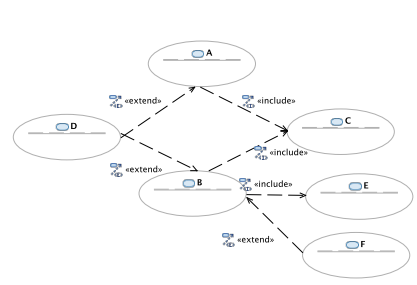
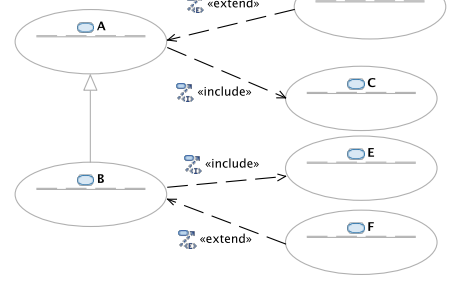
To correct generalization defaults, we propose a systematic approach that introduces more general elements (actors or use cases) based on the detection of shared relations. However, this does not guarantee the relevance of factorizing the relations using more general use cases or actors. In this section, we study several refactoring proposals. We consider here a simple set-based semantics, associating to a use case diagram the set of its actors and for each actor the feasible sequences of leaf use cases. The use cases having specializations are supposed to be abstract, *i.e.* replaced by one of their specializations during a concrete usage of the system.

3.1. The local refactoring patterns

Refactorings between use cases. Table 1 presents the refactoring patterns for use cases. The pattern I considers the situation of two use cases A and B including the same use case C. In this situation, all the sequences that can be performed by an actor and that contain A (resp. B) will also contain C. We propose to refactor the two inclusions as seen in the refactored pattern: a sequence that can be performed by an actor and containing A will still contain C, since A specializes N that contains C. In situation II, when the two conditions of extension CA and CB are identical and that the extension points (that roughly specify when the extension should occur) are also identical, then a generalization can be introduced. It frequently occurs that extension points and conditions are not specified for the extensions, in this case we propose to apply the refactoring, whereas if the extension conditions or the extension points are different, this refactoring does not apply. The situation III introduces a specialization refactoring. The source pattern owns a use case A for which the sets of outgoing include relations and incoming extend relations are

strictly included in those of another use case B. The proposed refactoring derives B from A and removes from B the inherited relations. Situations to factorize incoming include relations or outgoing extend relations were studied but rejected since they were not pertinent.

Table 1. Refactoring patterns between use cases

	source pattern	refactored pattern
I	 <p>outgoing include</p>	
II	 <p>incoming extend</p>	
III	 <p>shared extend and include</p>	

Refactorings between actors. Table 2 presents two refactoring patterns between use cases and actors. The first one (I) introduces a specialization relation between actors when the set of associations of one actor is strictly included in the associations of another actor, and then removes the inherited associations. The second one (II) adds an abstract actor when the sets of associations of two actors have a non-empty intersection but each actor has associations that the other one does not have.

Two additional refactorings were studied but rejected, they are shown in Table 3.

The first rejected pattern aims at introducing a generalizing use case N to two use cases A and B, when the use cases A and B are included by the same use case C. The use case C then includes the use case N. The problem with this refactoring is that it does not preserve the operational semantics of the initial diagram. Indeed, in the source pattern, the use case C includes the use cases A and B, whereas in the refactored pattern, the use case C includes the use case N, i.e. whether the use case A or the use case B.

Table 2. Refactoring patterns between actors and use cases

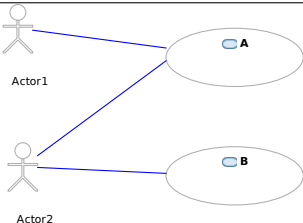
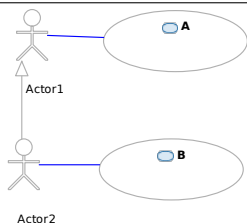
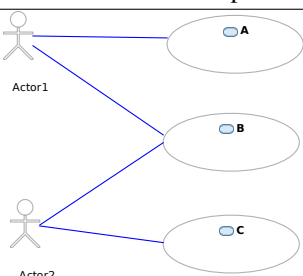
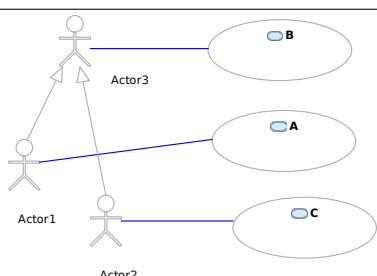
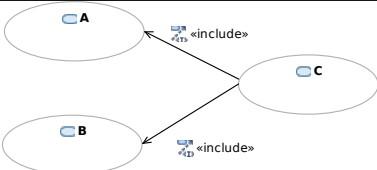
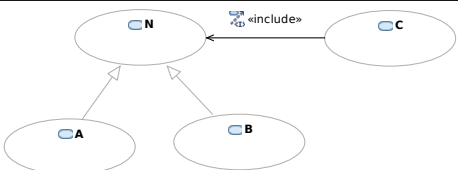
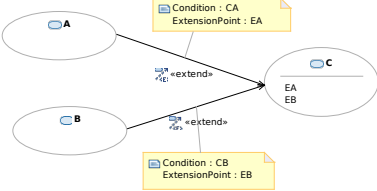
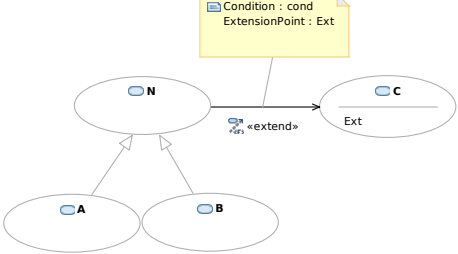
	source pattern	refactored pattern
I	 <p>Actors and use cases with specialization</p>	
II	 <p>Actors and use cases with factoring</p>	

Table 3. Non pertinent studied refactoring patterns

	source pattern	refactored pattern
I	 <p>incoming include</p>	
II	 <p>outgoing extend</p>	

The second rejected pattern aims at introducing a generalizing use case N to two use cases A and B, when the use cases A and B extend the same use case C with the same extension condition. The use case C is then extended by the use case N. However, this refactoring makes no sense. Indeed, it makes no sense to generalize two use cases just because they may extend another use case. Use cases A and B could be two optional use cases extending C, with no other common characteristic than just extending C. In addition, the attached extension conditions would probably be very different, so that this refactoring could not be applied. Last, introducing the refactoring in a diagram does not simplify the reading at all, the use case N does not help in having an abstract view of the system.

3.2. Local refactoring versus global refactoring

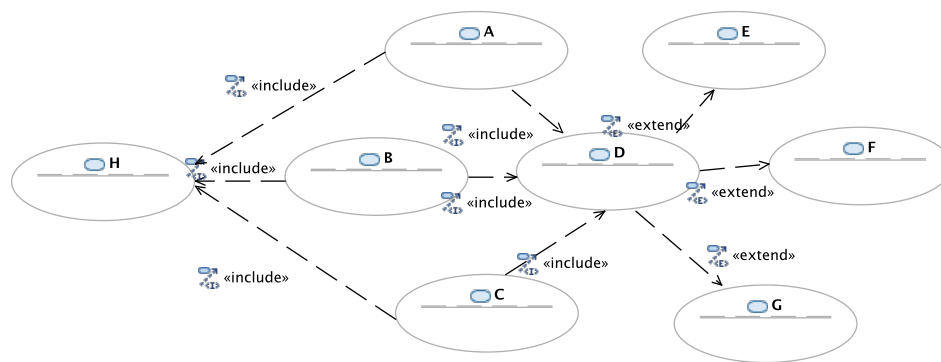


Figure 3. A use case diagram to refactor

The refactoring schemas presented in the previous section do not give a unique solution to deal with a use case diagram as a whole.

Let us consider a simple example (presented in Figure 3), in which the use cases D and H are included by three other use cases A, B and C, and in which D extends the three use cases E, F and G. Applying the refactoring patterns with a limited viewpoint on the diagram can lead a tool or a designer to propose a refactoring solution (given in Figure 4) in which the inclusion of D and H by A and B is considered independantly from the inclusion of D and H by B and C, thus leading to create two new use cases N1 and N2. A symmetrical refactoring solution can be applied to the use cases that the use case D extends. Though this solution may be pertinent in certain particular cases, it rather makes the refactored diagram more complex than the initial one.

However, there is a minimal solution, presented in Figure 5, in which a global refactoring vision leads to introduce a single generalization (the use case N1) for the three use cases that include D and H, and a single generalization (the use case N2) for the three use cases that the use case D extends. Those situations can be much more complex, for example when three use cases have features in common, and among these three use cases, two use cases have features in common not shared with the third one, etc.

Table 4. Contexts for FCA refactoring (use case names have been shorten)

			IsInvolvedIn				
			BFC	AMPI	BBT	ACUI	
			RegisteredClient	×	×		
			Company			×	×
RegisteredCompany	×	×	×	×			

	Includes		IsExtendedBy	Name					
	SEE	IC	SEE	BFC	SEE	AMPI	BBT	ACUI	IC
BFC	×	×		×					
SEE					×				
AMPI			×			×			
BBT	×	×					×		
ACUI			×					×	
IC									×

Definition 4.2. (Concept)

Let be a formal context $K = (O, A, I)$, $X \subseteq O$ and $Y \subseteq A$. (X, Y) is a concept if and only if $X = \{o \in O | \forall y \in Y, (o, y) \in I\}$ and $Y = \{a \in A | \forall x \in X, (x, a) \in I\}$. X is called the extent of the concept and Y is called its intent.

Using the order relation $(X_1, Y_1) \leq (X_2, Y_2) \iff X_1 \subseteq X_2$, the concepts can be represented under the form of a lattice. The representation of extents (resp. intents) may be simplified without losing information by keeping only the objects from an extent (resp. attributes from an intent) that do not appear in the lower concepts (resp. upper concepts).

The lattices with simplified extents and intents resulting from the application of FCA on the contexts from Table 4 are presented in Figures 6 and 7. Each concept is presented as a box with three parts, the top one is the name generated automatically to identify the concept, the middle part is the simplified intent of the concept and the bottom part is the simplified intent of concept.

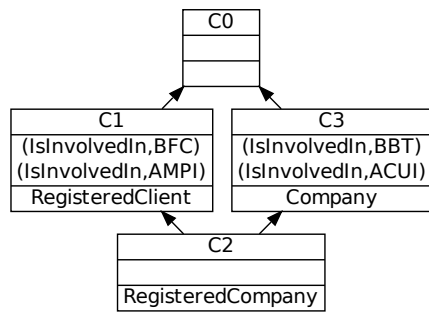


Figure 6. The lattice of actors built using FCA

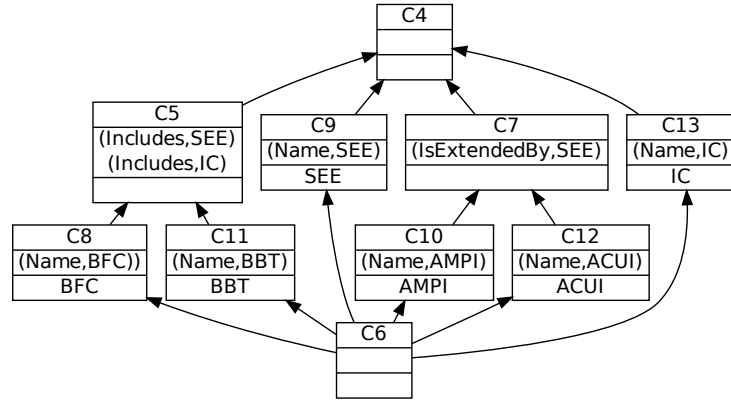


Figure 7. The lattice of use cases built using FCA

In the lattice of actors (Fig. 6), a generalization relationship is introduced: C2, which represents RegisteredCompany, is indeed a subconcept of C1 (RegisteredClient) and C3 (Company). This is an application of the first actor-use case refactoring. The lattice of use cases (Fig. 7) highlights two refactorings. The outgoing-include refactoring appears on the left: C5 generalizes the concepts that introduce Buy with Fidelity Card (C8) and Buy with Bank Transfer (C11) because these two use cases

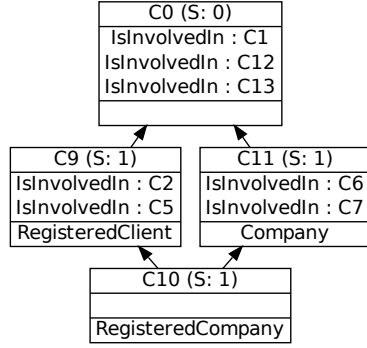


Figure 8. The lattice of actors built using RCA

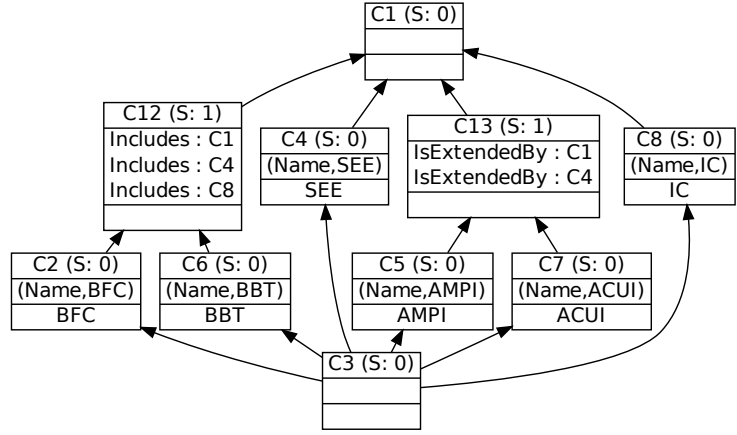


Figure 9. The lattice of use cases built using RCA

include Send Recapitulative Email and Identify Client. C7 shows an opportunity to apply the incoming extend refactoring (II): it factorizes the characteristic of being extended by Send Recapitulative Email, which is shared by Apply for Multi-Period Internship (C10) and Apply for Customized Internship (C12). The final use case diagram (Fig. 2) is deduced from the two lattices. The (non-trivial) concepts are interpreted as actors or use cases and the lattice partial order as generalization/specialization as illustrated just before.

In our example, no new actor is created, but it happens in other diagrams of our benchmark. Two new use case factorizations are introduced: Buy (from C5) and Apply for internship (from C7) that appear in the final diagram and increase its abstraction level and reusability.

4.2. Refining the refactoring with RCA

For further refining of the result, we applied Relational Concept Analysis (RCA) [15], one of the extensions of Formal Concept Analysis that takes into account relationships between formal objects in the classification process. The input of Relational Concept Analysis is a set of tables (a relational context family or RCF) such that some tables represent object-attribute relations and some tables capture object-object relations. In the encoding we choose (Table 5), the non-relational part of the RCF is composed of two object-attribute tables: actors with no description (empty attribute set), and use cases described by their names. The relational part of the RCF is composed of three object-object tables: Includes relation, IsExtendedBy relation and IsInvolvedIn relation. The choice of these relations is guided by the refactoring patterns that we have identified as relevant, and by preliminary experiments that highlighted cases where combinatorial explosion occurred.

RCA is an iterative process and at the end of it one lattice is built for each object-attribute table, in our case a lattice for actors and a lattice for use cases. The process, as illustrated by Table 6, starts by applying FCA on each object-attribute table, generating lattices that only takes into account relations between object and attribute (see step0). Thus the lattice obtained for the use cases contains, in addition to the top concept grouping all the objects and their common attributes and the bottom concept grouping all the objects that share all the attributes, one concept for each use case with the identifying name as

attribute. This guarantees that at the end of the whole process all the use cases will be contained by a concept that contains no other object. The lattice of the actors contains only one concept as no attributes has been defined. This will allow the merge of actors in particular situations.

RCA iterates on two successive steps : retrieving of the concepts from the newly created lattices to extend the tables using a scaling operation and building of lattices with classical FCA on the new tables obtained. It stops when the FCA steps doesn't introduce new concepts.

An object-attribute table $K_i = (O_i, A_i, I_i)$ will be extended using all the object-object table $R_l = (O_i, O_j, J_l)$ and considering the lattice $L_j \subseteq \mathcal{P}(O_j)$ from the object-attribute table $K_j = (O_j, A_j, I_j)$. For each R_l an object-attribute table $K_{i,l} = (o_i, L_j, M_l)$ will be created with $M_{jl} = \{(o, c) | o \in O_i, c \in L_j, S(o, c, J)\}$ where $S(o, c, J)$ is a scaling operator. The scaling operator used here is what we call the existential operator that is true if $\{o\} \times c \cap J \neq \emptyset$.

Hence, in step 1 the use case table is extended using the `includes` and `isExtendedBy` tables from Table 5. For the `includes` table, a pair object-concept is added to the binary relation when the object includes an object that exists in the concept.

RCA iterates on two successive steps: building of lattices with classical FCA framework (on each object-attribute table concatenated with the corresponding object-object tables) and integration of the concepts discovered at the current iteration in the relational part (to be used at the next iteration). The integration of the concepts discovered at the current iteration uses scaling operators, here the existential operator has been used. For the integration, an object-object relation (*e.g.* `Includes`) is transformed by replacing the objects which are in the columns by the concepts of the lattice built at the current iteration on these objects. Then a relation is established in the new, existentially scaled, table, between an object and a concept, when the object is in relation with at least one object in the extent of the concept. For example, `RegisteredClient` `IsInvolvedIn` `BFC` in the original relation. Then, when at an iteration `BFC` is in the extent of a concept, say `C12`, in the scaled `IsInvolvedIn` relation, `RegisteredClient` is considered as participating in one of the use cases grouped by `C12`. A pair (`RegisteredClient`, `C12`) is added in the scaled table `IsInvolvedIn`. This pair enriches the description of the three actors and at the next FCA step, it is factored out in `C0`, top of the actor lattice. That highlights the opportunity to define a more generalized actor, this opportunity did not appear in the one-step FCA building.

At each step, new concepts can appear, as well as new pairs relating these new concepts. The process iterates until no new concept or description emerge. Figures 8 and 9 show the final lattices. Lattices are interpreted as in the FCA analysis, adding an interpretation of the references between the two lattices. The result of RCA improves the result of FCA on this example by adding the new, more general actor (Fig. 10). Furthermore, the interpretation of the concepts is used to interpret the references between lattices. For example, the characteristic `isInvolvedIn:C12` which appears in `C0` of the actor lattice signals that the actor which will be created to interpret `C0` will participate in the use case resulting from the interpretation of the use case `C12`, that is the general use case `Buy`.

Discussion on the application of the global application of the refactoring patterns

During the refactoring process, abstraction of use cases or relations may appear that are useful to understand the global behavior of the system, but that should be correctly annotated in order to respect the use case operational semantics.

Let us take the example of Figure 10, obtained through an RCA process. The introduced use case `Concept_12` is interesting, since it represents the `Buy` use case. Similarly, the introduced actor

Table 5. Relational Context Family for RCA refactoring

use case	Name					
	BFC	SEE	AMPI	BBT	ACUI	IC
BFC	×					
SEE		×				
AMPI			×			
BBT				×		
ACUI					×	
IC						×

Includes	BFC	SEE	AMPI	BBT	ACUI	IC
BFC		×				×
SEE						
AMPI						
BBT		×				×
ACUI						
IC						

actor
RegisteredClient
Company
RegisteredCompany

IsExtendedBy	BFC	SEE	AMPI	BBT	ACUI	IC
BFC						
SEE						
AMPI		×				
BBT						
ACUI		×				
IC						

IsInvolvedIn	BFC	SEE	AMPI	BBT	ACUI	IC
RegisteredClient	×		×			
Company				×	×	
RegisteredCompany	×		×	×	×	

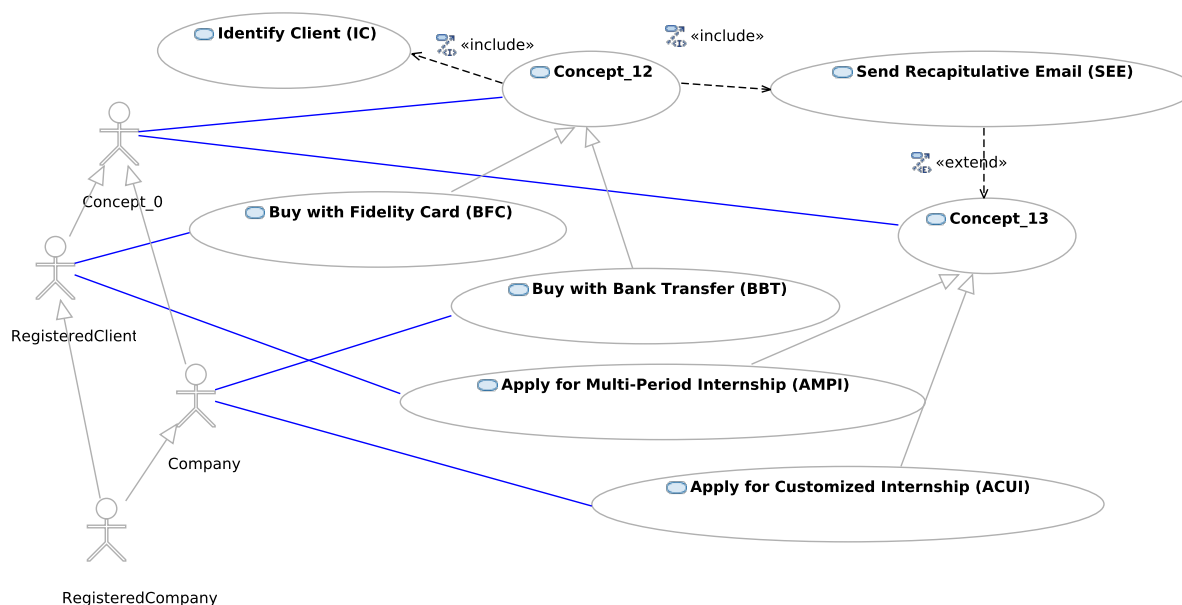
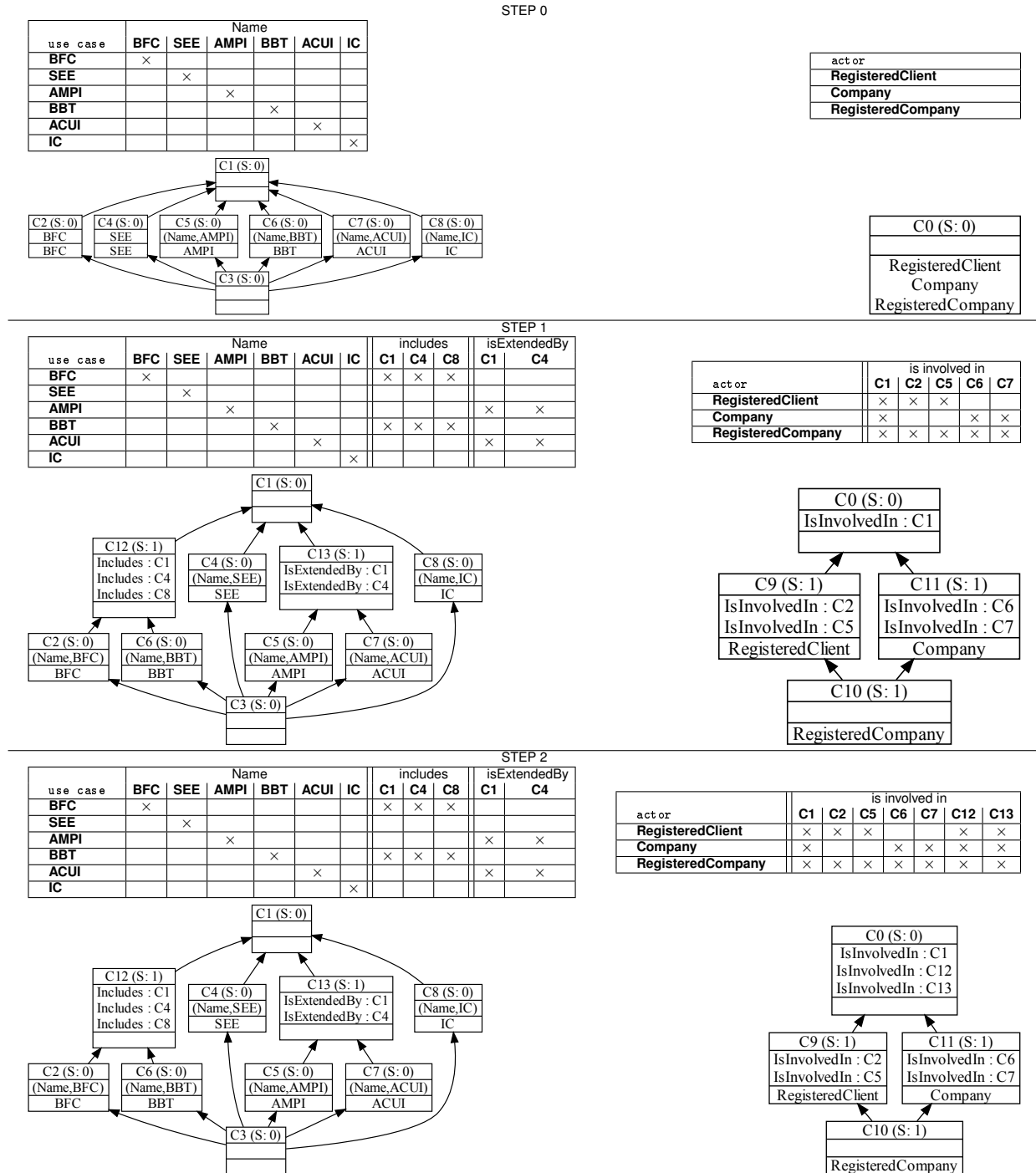


Figure 10. The internship example refactored using RCA

Concept_0 represents an abstraction of client, that is useful to show that all the clients can buy and apply for internships (use case Concept_13). However, if we look carefully at this diagram, we can see that

Table 6. Detailed steps of Relational Concept Analysis



the association between the actor Concept_0 and the use case Concept_12 (interpreted as : a client can

buy) can lead to understand that a `RegisteredClient`, being a kind of actor `Concept_0`, can buy with bank transfer (since the actor `Actor_0` participates in the use case `Concept_12` that has for specialization `BuyWithBankTransfer`). To prevent that, constraints should be added on the associations linking the actors to the use cases the following way. Let `buy` be the name of the `Concept_12`-side extremity of the association from `Actor_0` to the use case `Concept_12`. The extremity `buy` should be enhanced with the constraint `union`, while the `BFC` (resp. `BBT`) extremity of the interaction from `RegisteredClient` (resp. `Company`) to `BFC` (resp. `BBT`) should be constrained so as to specify that it is subsets the `buy` extremity.

Another situation that may require to add constraints is illustrated in Figure 11. Let us suppose that the generalizations from use case 5 and use case 6 to use case 4 were created by a designer, as well as the two bottom include associations respectively from use cases 2 to 5 and 3 to 6. Our refactoring process will create the use case 1, and the includes association from use case 1 to use case 4. Indeed, an includes association will first be created from usecase2 to usecase4, and from usecase3 to usecase4. Then the use case 1 will be introduced, that factorize those two include associations. If shown this way, it can be understood from this diagram that for example use case 2 includes use case 6, since it is a generalization of use case 1 that includes a generalization of use case 6. So as to avoid this interpretation, constraints should be added so as to specify that the bottom include associations hide the upper one, as illustrated in Figure 12.

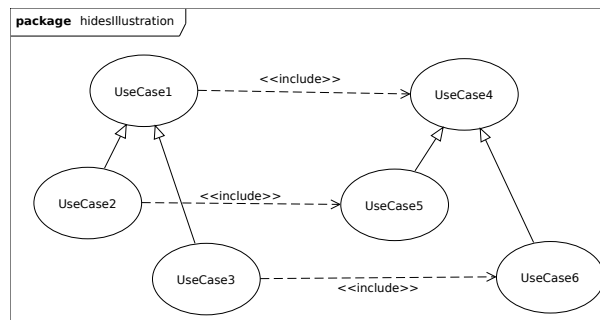


Figure 11. Illustration of includes relations that should be hidden

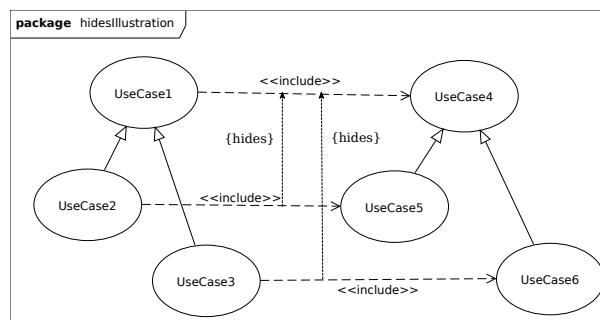


Figure 12. Illustration of hidden includes relations

5. Case Study

We present in this section the current implementation of our approach and we evaluate the results obtained on different kinds of data.

5.1. Tool

The Eclipse Modelling Framework (EMF) is a facility from the Eclipse IDE to implement modelling languages and generate tools to manipulate instances of those languages from programs. We use in our tool two Eclipse plugins based on EMF: UML2¹ and eRCA². UML2 is an implementation of UML and therefore allows us to load, modify and save use case diagrams. eRCA is an implementation of FCA and RCA algorithms which uses EMF to implement contexts and lattices. We have developed a tool implementing both the FCA and RCA approaches for use case refactoring. For both of them we developed the same three steps described in Section 4 and illustrated in Figure 13: first the tool loads a use case diagram using the UML2 plugin and encodes it as contexts in eRCA format; second it uses the eRCA tool to generate lattices in eRCA format; finally lattices are automatically analyzed to create back a use case diagram in the UML2 plugin format and to save it.

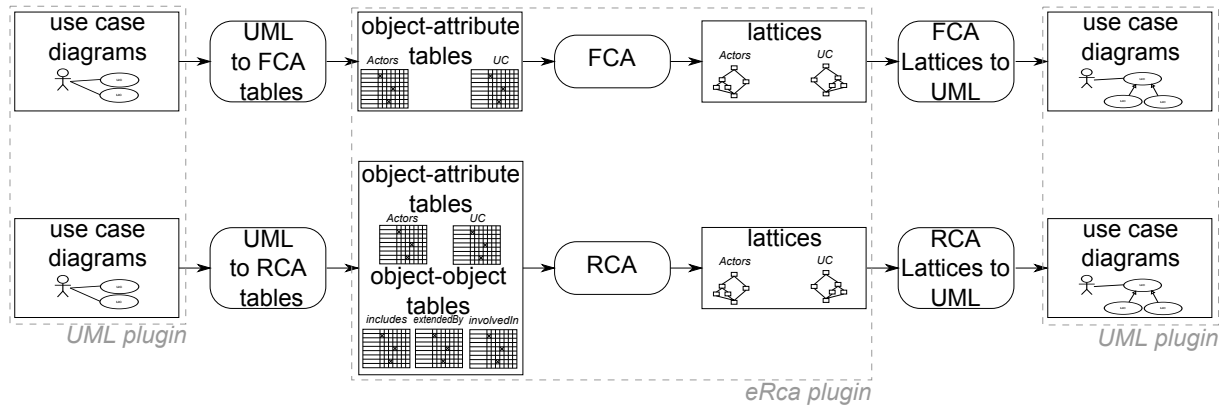


Figure 13. Presentation of the tool.

5.2. Data

We present now the results obtained using data gathered from different sources such as student projects or UML courses taken from the internet³. The Table 7 lists the diagrams. We encoded the 24 diagrams using the UML2 plugin in Eclipse and we applied the FCA and the RCA processes. The five diagrams called `test1` to `test5` are used to test specific situations. The other diagrams are representative use case diagrams, dedicated to ordinary topics for software including internship registration, data analysis, or various business systems. Use case diagrams are usually small. Compared to the common usage,

¹<http://www.eclipse.org/uml2/>

²<http://code.google.com/p/erca/>

³all data available at <http://www.lirmm.fr/%7Edolques/publications/data/cia10>

Name	Actors	Use Cases	Associations	Includes	Extends	Generalizations	Actor Generaliz.	UC Generaliz.
useCaseStudentRegistration	7	18	21	0	3	0	0	0
InternshipSystem2	8	28	38	15	13	5	2	3
Order	4	11	17	3	2	0	0	0
dataAnalysis	1	7	3	5	3	0	0	0
TCMS	4	22	17	1	8	0	0	0
balevChocolate	4	25	14	18	3	2	0	2
accountTest	1	7	4	5	2	0	0	0
CDsales	3	4	5	0	0	0	0	0
ecommerce	3	22	18	8	0	4	0	4
neuralDevelopmentKit	1	6	6	6	0	0	0	0
Simple-forum	2	8	4	1	1	3	1	2
JambetHotel	2	13	5	13	4	2	0	2
Internship	4	8	10	6	3	0	0	0
test	1	6	1	4	5	0	0	0
test3	2	5	2	4	0	0	0	0
Patient	2	9	9	3	1	0	0	0
ProjectManage	1	8	3	2	4	0	0	0
account	1	6	4	4	2	0	0	0
test4	1	9	2	8	0	0	0	0
test2	1	5	2	4	0	0	0	0
Invoice	17	18	22	0	4	14	5	9
chocolate factory	4	23	11	11	7	2	0	2
HEapplicants	5	6	13	0	0	0	0	0
Internship-article	3	6	8	4	2	0	0	0
total	82.00	280.00	239.00	125.00	67.00	32.00	8.00	24.00
average	3.42	11.67	9.96	5.21	2.79	1.33	0.33	1.00
min	1.00	4.00	1.00	0.00	0.00	0.00	0.00	0.00
max	82.00	280.00	239.00	125.00	67.00	32.00	8.00	24.00
standard deviation	3.31	6.55	6.07	4.36	2.26	3.01	1.05	2.06

Table 7. Size of the models

Invoice and InternshipSystem2 diagrams are relatively big use case diagrams that a good designer would divide into several subparts. A software project thus includes a (sometimes large) set of use case diagrams, and refactoring operations on a software system have to be considered at this level.

5.3. Evaluation of the results

Results are analyzed from two points of view. Firstly we study the feasibility of the FCA and RCA analyses, and if diagrams are simplified in terms of structure (structure analysis). Secondly, we study if the actors and use cases introduced for improving the diagram structure and its abstraction level are relevant (qualitative study).

5.3.1. Structure analysis

In the quantitative study, we consider a use case diagram as a graph $G = (V, E)$. Vertices are the actors and the use cases. Edges are include relationship, extend relationship, involvement relationship (associations) and specialization-generalization relationship.

We study three measures on input and output use case diagrams:

- Density $D(G)$. The density gives a measure of how many edges are in E compared to the maximum possible number of edges between vertices of V . We calculated the density in a directed graph as $D(G) = |E|/|V|^2$.

- Average degree $AvD(G)$. The average degree gives indication about the connectivity in the graph and also compares the edge number and the vertex number. The average degree is calculated considering the graph as undirected $AvD(G) = 2|E|/|V|$: this is justified by the fact that each edge is incident to two vertices and is used to calculate the degree of both vertices.
- Maximal degree $\Delta(G)$. The maximal degree $\Delta(G)$, where the degree $d(v)$ of a vertex v is the number of edges incident to v , $d(v) = |\{(x, v) | (x, v) \in E\} \cup \{(v, x) | (x, v) \in E\}|$, and $\Delta(G) = \max\{d(v), v \in V\}$.

We assume that the lower density, average degree and maximal degree we have, the clearer the diagram is. We calculated the three, because each of these measures gives its own point of view on the graph structure.

We present in Figure 14 how the density can be changed by our approach on all our data. We see that in most of the diagrams the density is lowered by using FCA and RCA, but with better results for FCA. Density goes down due to the creation of new use cases or actors and to the factorization of the relationships. An inverse situation can occur if the number of generalization links created is higher than the number of links removed by factorization. This is why FCA seems to be better here than RCA, because RCA creates more concepts. Nevertheless, the density for RCA usually remains better than the original density. We also see that density is not improved for the diagram *Invoice*. Indeed, we assume that each actor involved in a use case has the same role, so they can be factored into the same concept. But in the case of this diagram, many actors are involved in the same use cases but with different roles. This may be allowed depending on the interpretation of the UML specification, but results in the merging of many actors into one actor concept. By removing those actors (merged in one single actor) and thus not adding new abstractions among actors, the density is then increased. This can be partly solved by adding an identifier for each actor when creating the contexts, but this prevents us to find new generalizations of actors since none of them would share in their intent the same identifiers. But the problem is also due to the fact that roles usually are not mentioned in the real use case diagram. We will talk more about this issue later in the qualitative analysis.

A problem with the density measure is that if the number of vertices grows (with a constant number of edges), the density can decrease but keeping some parts of the graph with a high density. Considering the degrees is an alternative way of measuring the graph complexity, because when there are few edges incident to a vertex, the graph is easier to understand. Figures 15 and 16 show respectively the average degree and the maximal degree in initial diagrams and then in diagrams refactored using FCA and RCA. The maximal degree gives a bound, while the average is less impacted by the vertex number. In 16 diagrams, the average degree is lowered by using FCA and RCA. In two configurations that we created (test3 and test4) RCA increases strongly the average degree, but in most of the cases, there is not a big difference between FCA and RCA. RCA always increases the average degree. The maximum degree is also lowered (or remains equal) in 17 diagrams.

We can synthesize this quantitative analysis by the remarks that follow:

- Modified diagrams (RCA or FCA) are better in density and degree than original diagrams in most of the cases.
- FCA method gives better results in size than the RCA method, this was expected, because the RCA process we study is an iteration of the FCA process.

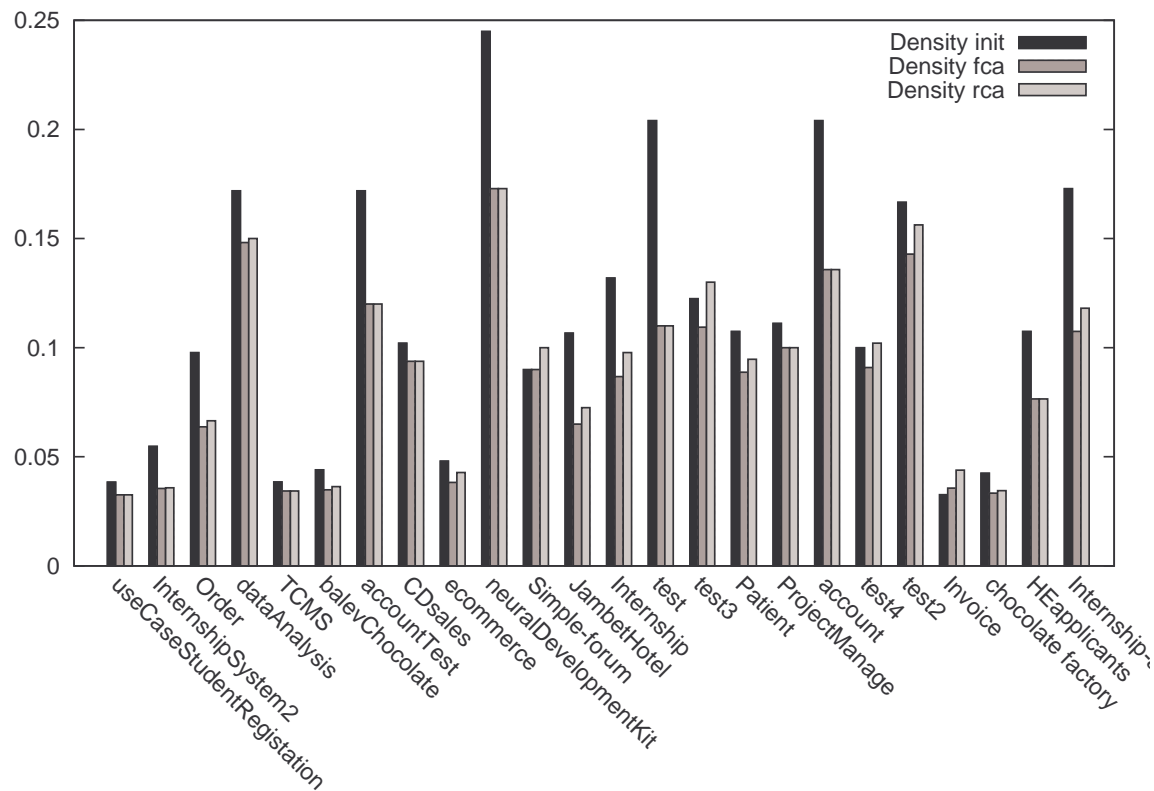


Figure 14. Density

- RCA remains feasible in size, even if in terms of theoretical complexity RCA risks to produce many more abstractions than FCA. This can be explained by the fact that combinatorial explosion in RCA is often triggered by paths linking objects along which generalizations accumulate. In most of the use case diagrams, such paths are short, due to the size but also due to the configuration of the diagram.

5.3.2. Qualitative analysis

In this part of the evaluation, we evaluate the semantics of the refactoring produced by FCA and by RCA. We focus on the main elements related with the specialization-generalization as they are central for our refactoring purpose: added (or merged) use cases, added (or merged) actors, specialization-generalization links between use cases and between actors. All the links are calculated by the FCA (resp. RCA process) because they are deduced from the specialization in the lattice. Table 8 shows the results for our data. Of course, we did not analyze the four test diagrams which were introduced in our data only for structural analysis and have no meaning. Among the metrics used to evaluate the results, we compute precision. Precision is a metric well-known in information retrieval. For evaluating a set of automatically computed results it evaluates how many of these produced results are considered relevant

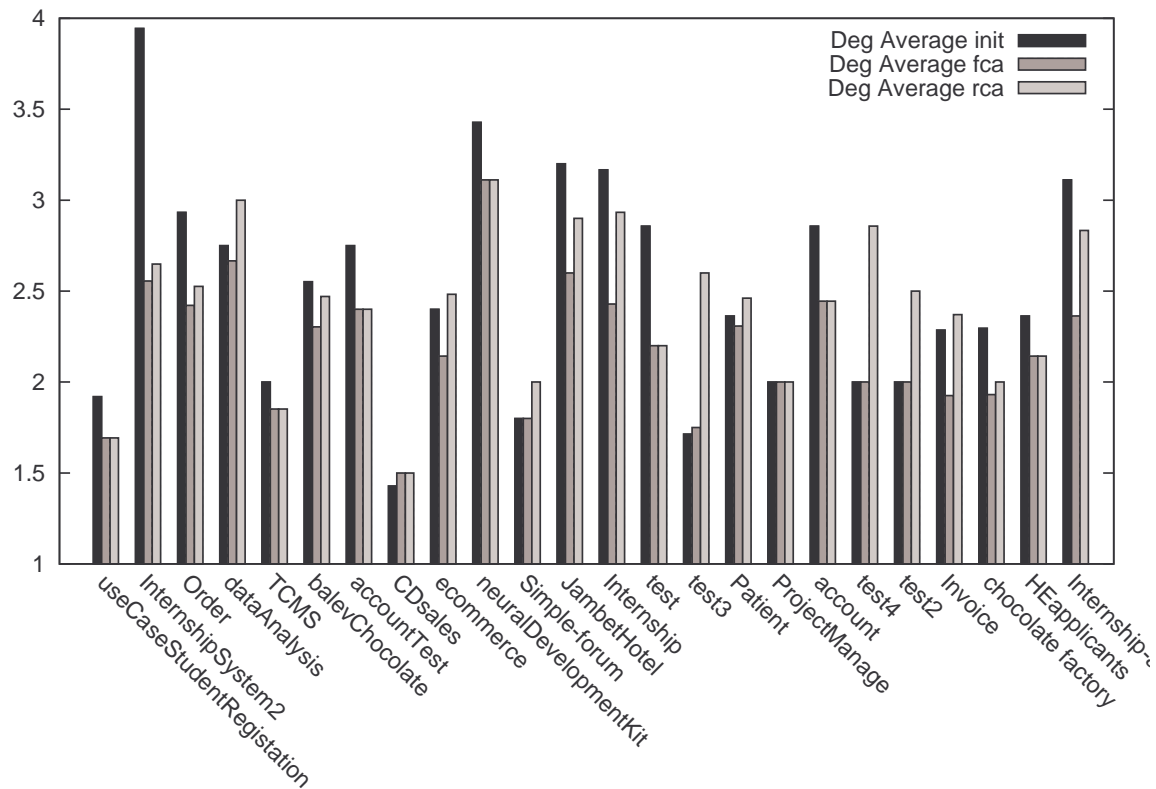


Figure 15. Average degree

by an expert. In our case, we analyzed the precision of the actors, use cases, generalizations between actor and generalizations between use cases computed thanks to FCA or RCA.

We consider the results on the set of use case diagrams, rather than only on individual use case diagrams, because an individual diagram is a small model, and the added value of our proposal has to be measured at the level of a system (which usually comprises many diagrams).

On the whole set of diagrams, between 29 (FCA) and 32 (RCA) use cases have been added, and all the use cases are connected by 84 (FCA) to 86 (RCA) specialization edges. A maximum of 4 or 5 use cases are added in each diagram, for a maximum of 15 generalization edges which is rather low. It is worth to note that having only few generalizations added by diagram makes the tool easy to use, because the designer can easily consider the generalizations and decide if they are relevant for his/her diagram. This is an advantage compared to the refactoring of class diagrams [3] where the (sometimes) huge number of abstractions built requires to filter them (asking the question of criteria to be used for that filtering) or to determine a priority order for their analysis. The precision on added use cases (greater than 0.9) measures the number of relevant use cases among the use cases added by the FCA (RCA) process. Those added use cases represent abstractions of existing initial use cases. Precision is rather high, and highlights the fact that use cases are like actions or behavior, and having common parts often reveals a common abstract behavior. The precision on added generalization edges between use cases is also good,

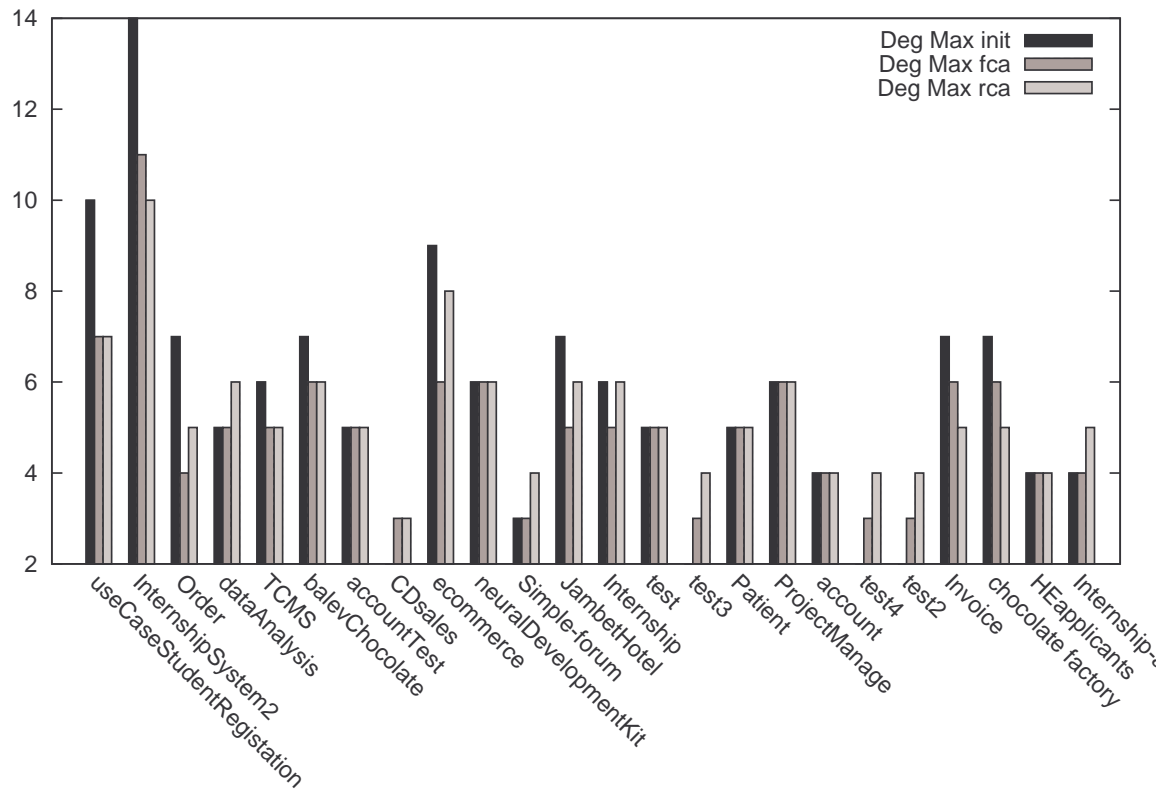


Figure 16. Maximal degree

it is of course connected to the precision on use cases.

On the set of diagrams, between 17 (FCA) and 20 (RCA) actors are added, as well as between 42 (FCA) and 50 (RCA) edges. Due to the fact that actors are not described by their names, some actors can be merged in a single concept. A maximum of 4 actors and a maximum of 9 specialization/generalization edges are added. The precision (around 0.6) is not so good compared to the one of use cases. This is mainly due to the lack of semantics on the association links between actors and use cases. Those links could be, theoretically, annotated by precise role names, indicating the function of the actor in the use case. But unfortunately designers rarely use this annotation in use case diagrams (when they use to do it in class diagrams). This leads our method to merge or to specialize actors when they participate to same uses cases, but often they play different roles in these use cases, *e.g.* student and professor in use case examination. Precision on generalization edges is better (between 0.7 and 0.6) because our method adds some forgotten specialization edges between existing initial use cases. Even if the results are not good for actors, we think they are useful because they show some opportunities for refactoring (at least half of the discovered actors) and they highlight ambiguities in diagrams and they encourage the designer to precise the roles of participants.

On 8 among our 20 diagrams (useCaseStudentRegistration, TCMS, accountTest, CDsales, neuralDevelopmentKit, ProjectManage, account, and HEApplicants), FCA and RCA produce the

	FCA	RCA
# added use cases	29	32
# relevant added use cases	28	29
# max added use cases by diagram	4	5
precision on added use cases	0.96	0.90
# added specialization edges between use cases	84	86
# relevant added specialization edges between use cases	82	79
# max added specialization edges between use cases by diagram	15	14
precision on added specialization edges between use cases	0.97	0.91
# added actors	17	20
# relevant added actors	9,5	12,5
# max added actors by diagram	4	4
precision on added actors	0.56	0,62
# added specialization edges between actors	42	50
# relevant added specialization edges between actors	30	33
# max added specialization edges between actors by diagram	9	9
precision on added specialization edges between actors	0.71	0,66

Table 8. Qualitative analysis

same results. In 4 diagrams (Order,, jambethHotel, Patient, Internship), participation links (association actor-use case) because a case generalization has been found and the participation is factorized. Three new relevant actors and only one relevant use case are added by RCA (comparatively to FCA). Relevant generalization links (between actors and use cases) are added in the same proportion.

Discussion From the experiment, we find that FCA finds almost all interesting generalizations, and that RCA can be used in a second step in order to refine FCA results. The threats for RCA generally are: going into a combinatorial explosion which is not noticed here, finding too general abstractions, which occurs in only few cases in our experiment. This is due to the fact that paths along which the generalization process runs are not long in the diagrams (rather than on the size of diagrams). As a concluding remark, use case diagram are a good application of model refactoring based on FCA techniques, because on the one hand the impact on the diagrams is not too high and the proposed refactorings are easily handleable by designers, and on the other hand, as a software project is composed of many of such diagrams, assisting the refactoring task is really useful and save the designer time.

We also draw some advice for the designers of the use case diagrams. The lack of role names provokes the construction of non relevant actor abstractions, so it seems necessary to have an annotation step on diagrams for filling most of these role names. This would improve greatly the results.

Another opportunity to improve FCA as well as RCA analyses consists in using terms, mainly actor names and use case names either in the description of these elements, or to confirm the relevancy of

the built abstractions. In our experiment, we observed that, due to the small size of a use case diagram, vocabulary tends to be consistent: same terms are used for the same things and we don't notice neither synonymy situations, nor conflict names. This encourages us to add the possibility to use the names in the analysis.

Here we presented the approach mainly in the context of simplification of use case diagrams, with the hypothesis that creating generalizations and factorizing relations clarify the diagrams. It can be exploited in another way, by extracting the generated, more abstract, use cases, actors and their relations from the diagram, to put them in a separate diagram and give a more abstract view on the system usage. For this purpose RCA is the most appropriate as it can create relations between generated element that couldn't be found with FCA, thus allowing a higher level of abstraction.

6. Related Work

Literature dealing with designing use cases is prolific, but usually concentrates on textual use cases [21, 7], and not on UML use case diagrams. It is generally admitted that use cases should be described by quite a lot of information that does not appear in use case diagrams, such as the nominal and exceptional flows of events or scenarios corresponding to a use case, and many research activities studied how to structure each use case, in particular with dedicated templates [7] or natural language techniques [10]. Fewer work [22, 16] deal with the way to structure the relations linking uses cases to actors, or use cases to other use cases. In [22] a metamodel is proposed for use cases, extending and grouping existing metamodels, and detailing relations between use cases: composition, dependency, precedence, extension, generalization, etc. Our work focuses on the relations introduced in the use case part of the UML specification, however it should be interesting to investigate how use case models conform to the metamodel of [22] could be refactored. Issa [16] proposes refactoring the scenario part of a use case, for example by separating a use case into several use cases. Henderson-Sellers et al. [14] introduce metrics to compare various versions of a use case, mainly consisting of graph metrics.

FCA has already been applied to use cases [20, 9] but not with the objective to refactor use case diagrams. In [20] use cases written in a controlled natural language are classified into a lattice using the terms contained in the use cases, with a visualization objective. In [9] use cases are classified using their important terms, in order to detect main notions that can become classes. A preliminary version of this work can be found in the french-written reference [8]. The new contributions of this paper consist in the comparison of results obtained with RCA and FCA (while reference [8] focuses only on FCA), in the case study, and in a new tool integrating RCA and FCA.

7. Conclusion

UML use case diagrams can be seen as the functional cornerstone of UML modeling. In this paper, we proposed an approach to refactor such use case diagrams in order to make them clearer. The principle is to use FCA or RCA in order to detect new abstractions that, when introduced in the diagram, can reduce its visual complexity. Using the tool we developed to implement this approach, we carried on experiments using a set of about 20 use case diagrams to measure its efficiency in terms of density of relations in the diagram. It results that both FCA and RCA processes decrease the density of realistic-size use case diagrams, the density obtained with FCA being lower than with RCA. However, RCA discovers more

abstract use cases or actors that can be relevant. To go further in this study, we are studying accurately other metrics, both quantitative as average degree, and qualitative, as precision.

Acknowledgments. Authors would like to thank L. M. Hakik for a preliminary study, K. Bouzroud, I. Dagha, H. El Assam, M. El Asri, and J. Ruiz-Simari for their help in the implementation of the current tool, and the anonymous reviewers for their suggestions on evaluation metrics.

References

- [1] *Object-Oriented. Information Systems, 8th International Conference, OOIS 2002, Montpellier, France, September 2-5, 2002, Proceedings*, vol. 2425 of *Lecture Notes in Computer Science*, Springer, 2002, ISBN 3-540-44087-9.
- [2] Ambler, S. W.: *The Object Primer: Agile Model-Driven Development with UML 2.0*, Cambridge University Press, New York, NY, USA, 2004, ISBN 0521540186.
- [3] Arévalo, G., Falleri, J.-R., Huchard, M., Nebut, C.: Building Abstractions in Class Models: Formal Concept Analysis in a Model-Driven Approach, *MoDELS* (O. Nierstrasz, J. Whittle, D. Harel, G. Reggio, Eds.), *Lecture Notes in Computer Science*, Springer, 2006.
- [4] Booch, G., Rumbaugh, J., Jacobson, I.: *Unified Modeling Language User Guide*, chapter 16, Addison-Wesley Prof., 2005.
- [5] Cockburn, A.: Goals and Use Cases, *JOOP*, **10**(5), 1997, 35–40.
- [6] Cockburn, A.: Using Goal-Based Use Cases, *JOOP*, **10**(7), 1997, 56–62.
- [7] Cockburn, A.: *Writing Effective Use Cases*, Addison-Wesley Professional, 2000, ISBN 0201702258.
- [8] Dolques, X., Madiha Hakik, L., Huchard, M., Nebut, C., Reitz, P.: Correction des défauts de généralisation dans les diagrammes de cas d'utilisation UML, *Proc. of LMO'10 (Langages et Modèles à Objets)*, 2010, ISSN 2105-102X.
- [9] Düwel, S., Hesse, W.: Bridging the gap between use case analysis and class structure design by formal concept analysis, *Proceedings of Modellierung 2000*, Fölbach-Verlag, 2000.
- [10] Fantechi, A., Gnesi, S., Lami, G., Maccari, A.: Application of Linguistic Techniques for Use Case Analysis, *IEEE International Conference on Requirements Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, 2002, ISSN 1090-705X.
- [11] Ganter, B., Wille, R.: *Formal Concept Analysis: Mathematical Foundations*, Springer, 1999.
- [12] Genova, G., Llorens, J., Quintana, V.: Digging into Use Case Relationships, *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, Springer-Verlag, London, UK, 2002, ISBN 3-540-44254-5.
- [13] Godin, R., Mili, H.: Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices, *OOPSLA*, 1993.
- [14] Henderson-Sellers, B., Zowghi, D., Klemola, T., Parasuram, S.: Sizing Use Cases: How to Create a Standard Metrical Approach, in: *OOIS* [1], 409–421.
- [15] Huchard, M., Hacene, M. R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets, *Ann. Math. Artif. Intell.*, **49**(1-4), 2007, 39–76.
- [16] Issa, A.: Utilising Refactoring To Restructure Use-Case Models, *Proc. of the World Congress on Engineering*, 2007, LNCS, 2007, ISBN 978-988-98671-5-7, 978-988-98671-2-6.

- [17] Jacobson, I., Christerson, M., Johnsson, P., Overgaard, G.: *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional, June 1992, ISBN 0201544350.
- [18] Kruchten, P.: *The Rational Unified Process: An Introduction, Second Edition*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000, ISBN 0201707101.
- [19] OMG: *Unified Modeling Language Superstructure, version 2.2*, 2009.
- [20] Richards, D., Böttger, K., Aguilera, O.: A Controlled Language to Assist Conversion of Use Case Descriptions into Concept Lattices, *Australian Joint Conference on Artificial Intelligence*, LNCS, Springer, 2002.
- [21] Rolland, C., Achour, C. B.: Guiding the construction of textual use case specifications, *Data Knowl. Eng.*, **25**(1-2), 1998, 125–160, ISSN 0169-023X.
- [22] Rui, K., Butler, G.: Refactoring Use Case Models : The Metamodel, *ACSC'03*, 16, ACS, 2003.
- [23] Simons, A. J. H.: Use Cases Considered Harmful, *Technology of Object-Oriented Languages, International Conference on*, **0**, 1999, 194.
- [24] Tilley, T., Cole, R., 0002, P. B., Eklund, P. W.: A Survey of Formal Concept Analysis Support for Software Engineering Activities, *Formal Concept Analysis* (B. Ganter, G. Stumme, R. Wille, Eds.), Lecture Notes in Computer Science, Springer, 2005.