Extraction of Classes Through the Application of Formal Concept Analysis

Decius Pereira, Luis Zárate and Mark Song
Departamento de Ciência da Computação, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, Brazil

Keywords: FCA, Formal Concept Analysis, Object-oriented, Class Hierarchy Engineering.

Abstract: The class hierarchy is one of the most important activities of the object-oriented software development. The class design and its hierarchy is a difficult task especially when what is sought is an extensive and complex modeling. Some problems are difficult to understand even when modeled using a methodology. The precise construction of a class hierarchy requires deep understanding of the problem, a correct identification of attributes and methods, their similarities, dependencies and specializations. An inaccurate or incomplete class hierarchy entails manufacturing defects of the software, making it difficult to maintain or make corrections. The Formal Concept Analysis provides a theory which enables troubleshoot hierarchy of classes to accomplish the maximum factoring of classes while preserving the relationships of specialization. This paper presents an approach to the application of Formal Concept Analysis theory in class factoring to simplify the design stages of new classes. A framework was developed to support experiments.

1 INTRODUCTION

The design and maintenance of a hierarchy is recognized as a difficult problem (Joshi and Joshi 2009). This difficulty increases with the number of classes involved and possible evolution of requirements which may demand the incorporation of changes in the hierarchical model.

Some problem domains are difficult to understand even when modeled using a methodology. The precise construction of a class hierarchy requires deep understanding of the problem, a correct identification of attributes and methods, their similarities, dependencies and specializations. An inaccurate or incomplete class hierarchy entails manufacturing defects of the software, making it difficult to maintain or make corrections.

Software Engineering has emerged as a systematic and disciplined approach to software development (Glinz 2007), establishing a set of activities to be followed by analysts, designers, developers and partners. The stage of software design became more complete and accurate with an application of universal language, such as UML, and use of Object-Oriented theory. However, even with the evolution in the development process occurred in recent years, it is evident the need to streamline the design steps.

The correct application of the concepts of object-oriented enables the reuse of software components, and the development with higher quality, easier maintenance, adaptations and extensions.

The Formal Concept Analysis (FCA) (Arévalo, Ducasse et al 2010) is a field of mathematics presented in the early 1980s.

The main FCA goal is the classification of objects based on their attributes. In the FCA commonly a problem domain is modeled as a cross table, called Formal Context, where the rows correspond to objects and the columns to the attributes.

The FCA theory can be applied to class model during the object-oriented design resulting in a deeper review of the model and ensuring the desirable qualities.

Much of this paper is focused on solving the problem of factoring classes and generating a new class hierarchy by maximizing the concept of inheritance through the application of FCA and a set of heuristics, whose goal is speeding stages of software design which is applied to various fields.

The software design in diverse areas of knowledge such as engineering, natural sciences, human sciences, and many others, usually requires technical expertise of the designer, which makes it

Pereira D., Zárate L. and Song M..
Extraction of Classes Through the Application of Formal Concept Analysis.
DOI: 10.5220/0004892302750282
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 275-282
ISBN: 978-989-758-028-4

more difficult for the designer the task of modeling the class structure of such systems. This paper provides guidance for class hierarchy generation for any type of systems or even the information generation in a database schema.

This paper is organized as follows: The next section presents the related work. Section III briefly describes the theoretical aspects of the theory of Formal Concept Analysis. Section IV discusses the proposal that is presented. Section V describes the experimental results using the framework. Section VI provides the final conclusions and suggestions for future work.

2 RELATED WORKS

The class hierarchy and its factoring has been reported by other authors in various development scenarios, such as the construction of the hierarchy of its starting point through objects and specifications of classes (Arévalo, Ducasse et al 2003); the evolution of the class hierarchy in order to accommodate new requirements through the addition of unlimited classes (Godin and Mili 1993) or by adding limited compatibility with a prior hierarchy or existing objects (Rapicault and Napoli 2001); reengineering of an existing class hierarchy from the relationship between classes and their attributes and methods (Godin and Chau 2000), using code analysis tools by applying refactoring (Snelting and Tip 2000) and in reengineering procedural code in the environment of objects (Moha, Hacene et al 2008).

In many cases the proposed approach is based on techniques that produce hierarchies that are not readily comprehensible for developers who need to spend a good amount of effort to interpret them.

The Formal Concept Analysis, in contrast, provides a theoretical framework that can be applied to the design and maintenance of class hierarchy in object-oriented environments whose comprehension is more natural. Several researches have adopted the Formal Concept Analysis in solving this problem (Bhatti, Anquetil et al 2012), (Arévalo, Falleri et al 2006), (Huchard, Dicky et al 2000) and (Falleri, Huchard et al 2008).

In (Bhatti, Anquetil et al 2012) a catalogue of patterns in concept lattices were generated with the purpose to allow automating the task of lattice interpretation helping the designer to concentrate on the task of reengineering rather than understanding a complex lattice. It is not aim of (Bhatti, Anquetil et al 2012) the hierarchization of classes from the concept lattice generation.

The abstraction of concepts and relationships for a specific domain were automated by techniques based on application of FCA in a model-driven context as proposed by (Arévalo, Falleri et al 2006). However this work does not address the semantics of the attributes or simplifies the concept lattice through their pruning.

In (Huchard, Dicky et al 2000) algorithms were developed for the building class hierarchies by different frameworks showing the advantages and drawbacks of using the Galois lattice and sub-hierarchy as models of class hierarchies. An inconvenience of (Huchard, Dicky et al 2000) consists in the generation of multiple inheritance, requiring adjustements for languages that have only single inheritance.

In (Falleri, Huchard et al 2008) was presented a generic approach implemented in a tool capable of dealing with any language described by a meta-model, that helps software architects designing and improving their class models. This work showed the Relational Concept Analysis technique (RCA), as an extension of FCA (Dao, Huchard et al 2004), (Huchard, Hacene et al 2007). Although (Falleri, Huchard et al 2008) has contributed a theory capable to normalize class models based on different metamodels, it does not address the semantics of the attributes such as (Arévalo, Falleri et al 2006).

Unlike the surveys previously presented this paper shows how to simplify the lattice concepts through heuristic pruning, dealing the semantics of the class attributes and supports the concept of multiple inheritance in hierarchies generated.

3 THEORETICAL ANALYSIS OF FORMAL CONCEPTS

The representation of the FCA enables to obtain the relationship between the set of objects or instances of domain from the list of attributes that describe its characteristics, thus resulting in the Formal Concept (Nilander and Zárate 2011). In Table 1, called Formal Context, an example for a hypothetical domain is presented.

Table 1 represents a structure that defines objects (rows), the attributes (columns) and their respective relationship of incidence. A Formal Context (G,M,I) consists of two sets G and M, and a binary relation I between these sets. The elements of G are called objects, while M are called attributes. If an object g

has a relation I with an attribute m, this ratio is expressed as gIm ou (g,m) ∈ I. This is interpreted as the "object g has the attribute m".

Table 1: Context Formal example.

	a	b	c	d	e	f	g	h
1		×		×				
2	×	×	×	×				
3		×		×	×		×	×
4		×		×	×		×	
5		×		×		×		
6	×		×					

For a set A ⊆ G of objects (called extension) is defined

$$A' := \{m \in M \mid gIm \forall g \in A\}$$

as the set of attributes common to the objects in A.

In correspondence, the set B ⊆ M (called intent) of attributes is defined

$$B' := \{g \in G \mid (gIm \forall g \in B\}$$

as the set of attributes common to the objects in B. Thus, a Formal Concept of a context (G,M,I) consists of an ordered pair (A,B) where the following property applies:

A ⊆ G, B ⊆ M, A' = B e B' = A

In simplified form, the set of objects of formal concept is called extension and attributes intention. Each element of the extension has all the intention and vice versa.

Through Formal Context is possible to generate the Concept Lattice. The Concept Lattice is a directed graph whose nodes represent objects or entities modeled, or just an association of concepts. Coupled to the nodes are the properties or attributes of the model and/or methods. The lattice allows the extraction of concepts in various applications, such as database design or the class design in an object-oriented approach. Figure 1 illustrates the Concept Lattice for Context Formal of Table 1.

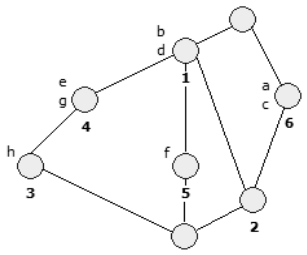

Figure 1: Concept Lattice for Context Formal.

In one lattice, if A is a concept above a concept B, and the two are connected, the concept A can be considered a more general concept than B and, as such, loads the common attributes between A and B. As a consequence, it is true that if B happens, A is also present, suggesting a binding logic. The lattice not only describes a hierarchy of concepts, but also the whole set of binary relations between these concepts. This causes the visual analysis of the object which can be obtained by searching in a class hierarchy.

In Figure 1, each node in the graph is a concept. If two objects were placed on the same node (concept), they have the same attributes and are therefore instances of the same class of objects that have that attribute set.

FCA thus provides a tool for formal recognition of groups of elements that share common properties and methods which reveal implicit and explicit dependencies, enabling a better understanding of the concepts.

4 EXTRACTION OF CLASSES THROUGH THE APPLICATION OF FCA

When using Formal Concept Analysis for the design of the class hierarchy, the set of formal objects G is a set of software artefacts, in other words, classes, objects or programs variables, which are used as a starting point in the search by appropriate class hierarchy.

The set of formal attributes M correspond to properties of classes or objects. The properties that are relevant include the attributes (instance variables) and methods (body and/or the method signature). In this paper, what is considered as a starting point is the set of specifications of classes - G is a set of objects or model entities. It is still only factoring attributes of classes, whose implementation is extended to methods.

An important aspect of this work is to minimize redundancy and to create subclasses via specializations. Regarding the idea of redundancy minimization is the factoring of classes reducing inconsistencies and minimizing future redundancy code.

For the subclass it is also used a factoring of classes as a means for identifying the hierarchy by setting an identification of type and subtype.

To obtain the maximum factoring of classes and a new hierarchical model this work proposes executing the following steps iteratively: 1. Mapping Model Entities for a Context Table; 2. Concept Lattice Generation; 3. Eliminating Multiple

Inheritance in cases where the target language is not supported; 4. Removal inconsistent classes and, 5. Segmenting class where common attributes have different semantic.

A. Mapping Model Entities for a Context Table

First consider that the software designer has the option of choosing the entities/objects from the class model in its original hierarchy or in the same level hierarchical, without their relationships of specialization or association. For the entities/objects from the model chosen the designer lists the properties that characterize them. For illustration purposes, due to the space occupied by the figures, consider that the software designer has selected the entities/objects from the class model without their relationships. Since objects are instances of classes, it may be assumed that the entities found can be considered as an initial class model (or a set of concrete classes).

Based on what was previously stated, consider the mapping of the attributes from the model in a Context Table. The following example illustrates this basic idea. Suppose the following specification of attributes for a set of four concrete classes as illustrated in Figure 2. The specification could be interpreted as the exact set of concrete classes that the hierarchy should contain, in other words, these classes are the only ones to produce objects in an application.

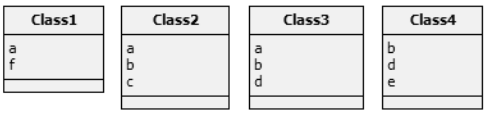

Figure 2: Concrete classes.

Ratio Incidence I of the formal context K represents a formal set of four classes and their instance variables is presented in Figure 3. Context is designed as a table - rows and columns, with the classes identified by whole numbers and variables by letters.

	a	b	c	d	e	f
1	x					x
2	x	x	x			
3	x	x		x		
4		x		x	x	

Figure 3: Formal Context.

B. Concept Lattice Generation

Since the problem is to organize these classes in a hierarchy, a Concept Lattice is used as a guide for the design. Each formal concept is interpreted as a class in the hierarchy and the links between classes are viewed as relations specialization. In Figure 4 is presented a Concept Lattice. The labels assigned to the concepts indicate that an attribute class in particular should be stated. For example, the declared attributes a and b are two general classes that are located immediately below the root of the class hierarchy. In the class hierarchy, the concept defined by the bottom node is ignored since it is of no use, because it does not represent information of classes.

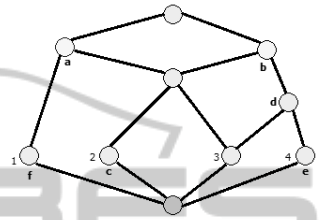

Figure 4: Concept Lattice for Formal Context of Figure 3.

Figure 5 shows the hierarchy in the form of the lattice attributes factored corresponding to its interpretation of Concept Lattice. The four initial classes remain in the hierarchy but there are fewer attributes declared in these classes due to factoring produced by Concept Lattice.

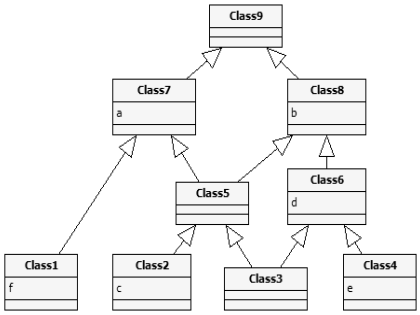

Figure 5: Class Hierarchy from Concept Lattice of Figure 4.

New classes (nodes 5-9) are added because of the factoring of common attributes. These are empty classes because instances are created only for the four initial classes. The nature of the reduction in labeling of Concept Lattice guarantees that each attribute appears exactly once in the hierarchy. The object attributes in the initial concrete classes remain unchanged. However, some of them are now inherited by some new classes. Generally all subclasses are specializations that inherit the attributes of parent classes without any exception.

From the software designer viewpoint, using this hierarchy produce the same effect as if the early four

classes were used. So the hierarchy generated can be interpreted as a refactoring of the specifications of the four initial classes.

C. Elimination of Multiple Inheritance

There is a large number of designs that enable minimize redundancy. The Concept Lattice achieves this goal by minimizing the number of classes and multiple inheritances (for target languages that do not support multiple inheritance).

This is achieved by grouping classes whenever possible, as illustrated in Figure 6.

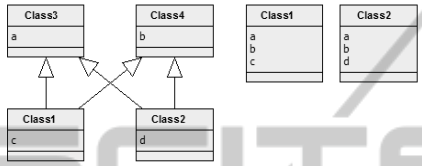

Figure 6: Elimination of Multiple Inheritance.

The design class presented in Figure 6, on the left, factors out the common attributes a and b but is more complex since it contains four classes, one for each attribute, capturing the classes 1 and 2 in a model of multiple inheritance. In contrast, the design presented on the right of Figure 6 is simpler and provides the same quality criteria to avoid redundancy and conformance to specialization in a model of single inheritance.

The transformation of a model that contains multiple inheritance in a model that contains single inheritance consists only on the copy of the attributes of their classes ascendants in their descendant classes, where there is the relationship of multiple inheritance. The ascendants classes thus cease to exist in the model after completion of copies, case there is not other binding with other classes.

D. Removal Inconsistencies

The Concept Lattice is a representation of similarities among a set of concrete classes. As its size grows quickly one can think of ignoring some of its nodes in order to maintain its structure manageable. Thus a first idea could be the removal of classes that do not declare any property or method. These classes commonly called empty classes could be removed without violating the quality criteria, in other words, without redundancy and specialization. In the example of Figure 5, the empty classes Class5 and Class9 could be omitted. Although Class3 does not declare any attributes it is kept because it is a bottom class.

The structure resulting from the removal of all empty classes is called Galois sub-hierarchy (Snelting and Tip 2000) and corresponds to the simplified set of all concepts of attributes and objects from Concept Lattice. Figure 7 depicts the new lattice which was pruned.

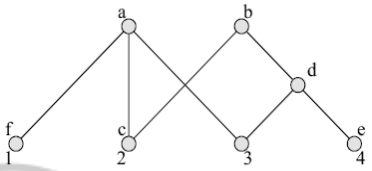

Figure 7: Concept Lattice resultant of removal empty classes.

Considering that the programming language supports multiple inheritance, the resulting new class model is the structure presented in Figure 8.

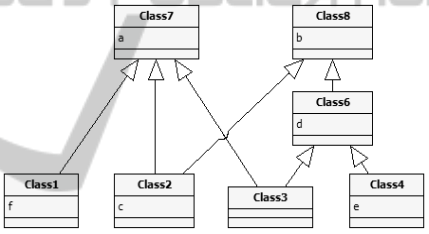

Figure 8: Class model resulting after application of factoring steps.

The class depicted in Figure 8 is the result of mapping the original classes of Figure 2 in a Context Table which in turn was converted into a Concept Lattice and disposed empty classes. However, if the initial definition of the design was foreseen that one of the prerequisites was modeling classes without multiple inheritance support, the resulting class diagram would be modeled according to Figure 9.

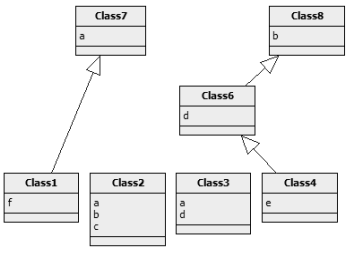

Figure 9: Resulting class model without multiple inheritance support.

E. Segmenting Class Where Common Attributes Have Different Semantics

The class attributes defined by the software designer may have identifying labels identical. However their properties can be different, making them semantically distinct. This fact implies that although they have the same name, they does not share the same characteristics. In this way they should not be summarized as a single attribute belonging to a new generation ancestor class created in the Concept Lattice.

Figure 10 illustrates two tables which exemplify the mapping of the attributes from existing classes. Figure 10, on the left, describes the attributes of each class, and on the right, the common attributes highlighted in bold. In this example, it was considered the class model of Figure 8, which supports the multiple inheritance concept.

Class1	**a,f**
Class2	**a,b1,c**
Class3	**a,b2,d**
Class4	**b2,d,e**
Class6	**b2,d**
Class7	**a**
Class8	**b1**

	a	*b1*	*b2*	*c*	*d*	*e*	*f*
Class1	**x**						**x**
Class2	**x**	**x**		**x**			
Class3	**x**		**x**		**x**		
Class4			**x**		**x**	**x**	
Class6			**x**		**x**		
Class7	**x**						
Class8		**x**					

Figure 10: Mapping of the factored class attributes on the left and common attributes on the right.

Class1	**a,f**
Class2	**a,b,c**
Class3	**a,b,d**
Class4	**b,d,e**
Class6	**b,d**
Class7	**a**
Class8	**b**

	a	*b*	*c*	*d*	*e*	*f*
Class1	**x**					**x**
Class2	**x**	**x**	**x**			
Class3	**x**	**x**		**x**		
Class4		**x**		**x**	**x**	
Class6		**x**		**x**		
Class7	**x**					
Class8		**x**				

Figure 11: Segmentation of common attributes and their new labels.

In this stage of the factoring process the software designer must intervene in order to identify the attributes that have the same label and whose properties are distinct. This step is semi-automatic. The result is shown in Figure 11.

The identification of common attributes demands the labels to be changed, as shown on the right of Figure 11. It creates segmented classes, as shown on the left of the same Figure.

A new class model is obtained, where Class2 inherits the renamed attribute b1 belonging to Class8, and a new class, named Class9, which contains the renamed attribute b2. Both attributes have as origin the attribute b, coming from of the previous class model. Figure 12 presents the new class hierarchy generated.

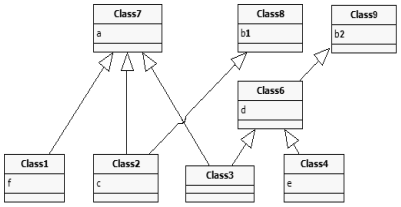

Figure 12: New class model resulting from segmentation of common attributes.

5 EXPERIMENTAL RESULTS

In order to automate the factoring process of an initial class model, a framework was developed in order to test the theoretical aspects explained in this paper. Initially two basic premises were established: 1. The initial class model is described in the UML standard language; 2. There is an integration with a tool, such as Conexp (Yevtushenko 2000) or Galicia (Valtchev, Grosser et al 2003), to interpret the FCA XMI format.

Figure 13 illustrates the operating steps of the framework.

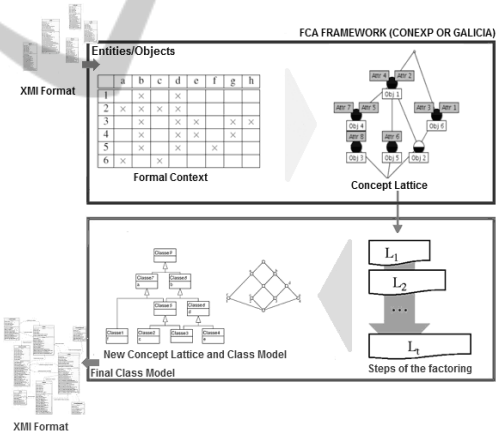

Figure 13: FCA Framework for factoring of classes.

The framework works as follows: the reading of a XMI file obtained by exporting a UML class diagram is performed. The application of a FCA tool captures the descriptions translating them into a Context Table and a Concept Lattice. A sequence of steps (section 4) is then applied to eliminate the empty nodes and to treat redundant classes on the lattice - for example, when it is not desired to have multiple inheritance in the model.

The examples of Figures 14 to 18 illustrate the application of the approach to a class model of the some modules of the Enterprise Resource Planning

Software of the Regional Council of Pharmacy of Minas Gerais State – Brazil, named SIGCRF. This example has been simplified due to the space occupied by the figures. For illustration purposes the initial association/specialization relationships between the classes have been also removed, however they could be maintained without prejudice to the framework, as described in section 4.

Step 1: Initial Class Diagram - As illustrated in Figure 14 the initial class diagram is designed as a set of concrete classes in the same hierarchical level. The results obtained for this approach or for the class model which contains its original hierarchy are both discussed later. In class model designed of Figure 14 the number of participating entities are fifty-nine, among them the classes Student, Teacher, Monitor, Trainee and Employee, which belong to the module of Training Center of the company. For illustration purposes only this module is discussed, however the final results are presented for the entire software.

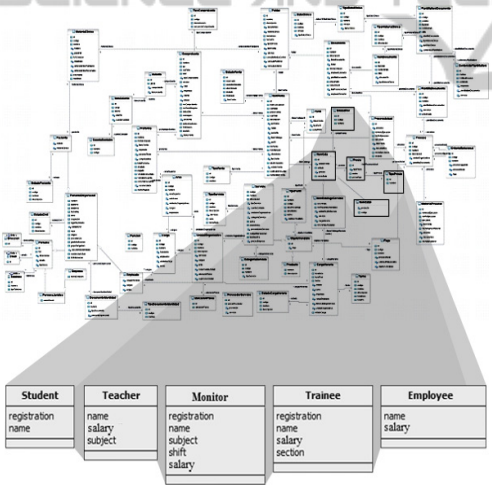

Figure 14: Initial Design of the Class Diagram.

Step 2: Concept Lattice Generation, Identifying the Inconsistent Classes, Potential Classes and Multiple Inheritance - After the conversion of the class diagram to the XMI format, it is performed the reading of the XMI file by the FCA Framework. The XMI file is interpreted and the Concept Lattice is generated. The lattice nodes are classified by the framework as concrete, potential or inconsistent classes in the model. The edges are also classified and can be interpreted as generalizations, whose relationship between classes is achieved through single or multiple inheritance as illustrated in Figure 15.

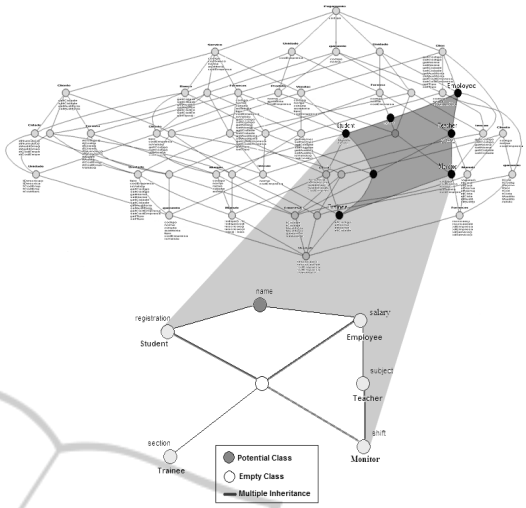

Figure 15: Concept Lattice Generation.

Step 3: Elimination of Inconsistent Classes - The inconsistent classes, or empty classes, turn the understanding of the model more difficult and they are eliminated by the FCA Framework. The empty node presented in Figure 15, which represents an empty class, has been removed from the model and a new Concept Lattice generated as illustrated by Figure 16.

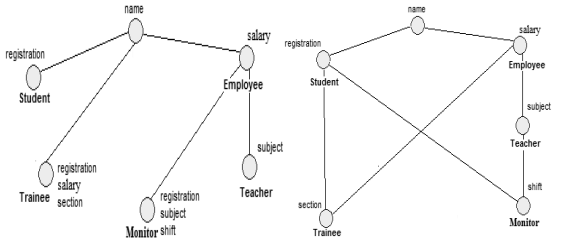

Figure 16: Concept Lattice Generation with Multiple and Simple Inheritance support.

Step 4: Class Segmentation Where Common Attributes Have Different Semantics – The fourth step of the FCA Framework consists on identifying the semantic of the attributes to correctly associate them into the respective classes. In this stage the framework makes available to the software designer, a list of classes and attributes for he/she chooses what attributes belong to which classes.

The software designer, in the example, identified that the salary attribute does not make sense for the Monitor class and thus segmented this attribute into two new classes, so that a new attribute, whose suitable name is scholarship, appears in Monitor

class. Figure 17 illustrates what was explained here.

Student	name, registration,section
Teacher	name,salary, subject
Monitor	name, registration, subject, shift, scholarship
Trainee	name, registration, salary, section
Employee	name, shift, salary

	name	*registration*	*subject*	*shift*	*salary*	*scholarship*	*section*
Student	x	x					x
Teacher	x		x		x		
Monitor	x	x	x	x		x	
Trainee	x	x			x		x
Employee	x			x	x		

Figure 17: Segmentation of common attributes and their new labels.

Step 5: Generation of a New Class Diagram - The last step of the FCA Framework consists in the generation of a new class model, which results from the application of the previous steps. If the software designer has defined that your model supports multiple inheritance, the resultant class diagram obtained is presented in the Figure 18, on the left. Otherwise, if the programming language does not support multiple inheritance concept, the new class diagram obtained is presented on the right of the same figure. In this new model the potential class Person was created to make it comprehensible, and some bindings between the classes were removed due to application of the Framework FCA iteratively.

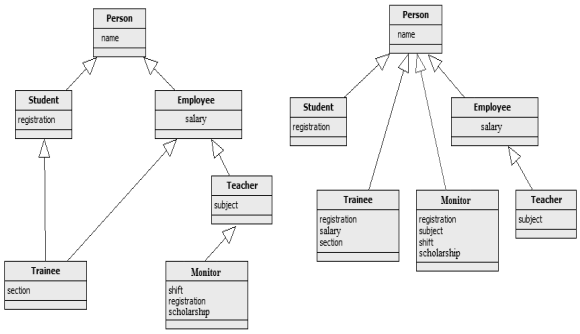

Figure 18: New Class Diagram with Multiple Inheritance support (on the left) and Single Inheritance support (on the right).

6 CONCLUSIONS AND FUTURE WORK

The paper presented the theoretical foundation and an example of the application of FCA in factoring and minimization of redundant classes in Object-Oriented Designs.

Using an appropriate framework it is possible to automate and optimize the design stage of software while maintaining the characteristics inherent to the models of object-oriented classes.

The factoring process of classes through the application of developed framework proved quite effective related to a better understanding of the problem, since it results in a reorganization of the model, approaching the most desirable characteristics of an object-oriented design.

A suggestion for future work consists in full automation of classes' segmentation where common attributes have different semantics.

REFERENCES

Arévalo G., Ducasse S. et al. Generating a catalog of unanticipated schemas in class hierarchies using FCA. *Inf. Softw. Technol.*, 52:1145–1187, November 2010.

Arévalo G., Ducasse S. et al. Understanding classes using X-Ray views. (ASE 2003), pages 2–18, Oct. 2003.

Arévalo G., Falleri J. et. al.: Building Abstractions in Class Models: FCA in MDA Models 2006: 503-527.

Dao M., Huchard M. et al. Improving generalization level in uml models. Springer, pages 336–360, 2004.

Falleri J., Huchard M. et. al. A Generic Approach for Class Model Normalization. ASE 2008.

Glinz, M.: On Non-Functional Requirements. In: *15th IEEE Conference (RE 2007)*, pages 08–26 (2007).

Godin R. and Mili H. Building and maintaining analysis-level class hierarchies. OOPSLA'93, p 374–410, 1993.

Godin R. and Chau T.T. Comparaison d'algorithmes de construction de hierarchies de classes. L'Objet, 5(3):311–338, 2000.

Huchard M., Hacene R. et al. Relational concept discovery in structured datasets. Ann. Math. 9(1-4):32–76, 2007.

Huchard M., Hervé D. et. al. Galois lattice as a framework to specify building class. ITA 34(6): 511-548 (2000).

Joshi P. and Joshi R. Concept analysis for class cohesion. *In Proceedings CSMR 2009*, pages 207–240.

Moha N., Hacene A, et al. Refactorings of design defects using relational concept analysis. *ICFCA'08*, pages 279–304. Springer-Verlag, 2008.

Nilander R. and Zárate L. Handling Large Formal Contexts with support of distributed systems, IIC 2011, p 1-15.

Rapicault P. and Napoli A. Evolution d'une hierarchie de classes par interclassement. L'Objet, 7(1-2), 2001.

Snelting G. and Tip F. Understanding Class Hierarchies Using Concept Analysis. ACM, p 540–582, 2000.

Usman B., Anquetil N. et al. A Catalog of Patterns for Concept Lattice Interpretation in Software Reengineering. SEKE 2012: 102-12.

Valtchev P., Grosser D. et al. Galicia: an open platform for lattices. In Aldo de Moor, *11th Conference on Conceptual Structures*, p. 221–254. Aachen, 2003.

Yevtushenko S. System of data analysis "Concept Explorer". *7th national conference on Artificial Intelligence KII-2000*, pages 107–134, Russia, 2000.