ORIENTAL JOURNAL OF
COMPUTER SCIENCE & TECHNOLOGY
An International Open Free Access, Peer Reviewed Research Journal
Published By: **Oriental Scientific Publishing Co., India.**
www.computerscijournal.org

ISSN: 0974-6471
June 2012,
Vol. 5, No. (1):
Pgs. 155-159

Using Conceptual Analysis in Java Environment for the X-Ray view on a Class

ZAHEER ASLAM, MOHAMMAD KAMRAN KHAN, SHAHZAD RIZWAN, FAWAD ALI KHAN, ZAHID HAROON and NOOR ZAMAN

CECOS University Hayatabad Peshawar, Pakistan.
Sarhad University KPK Peshawar, Pakistan.
Agriculture University KPK Peshawar, Pakistan.

(Received: February 12, 2012; Accepted: June 04, 2012)

ABSTRACT

Modularity is one of the very vital Methods in Software Engineering and a requirement for every reliable Software. As the design space of Software is normally pretty large, it is important to offer automatic way to assist modularizing it. An automatic way for Software Modularization is Object Oriented Concept Analysis (OOCA). X-Ray view of the Class is one of the features of this Object Oriented Concept Analysis. We will utilize this design in a Java Environment.

Keywords: Modularity, Object Oriented Concept Analysis views, Relyencies, Classes, Methods and Attributes.

INTRODUCTION

A branch of Lattice theory i.e a Concept Analysis (CA) let us to discover significant Groupings of essentials that has familiar characteristic and is referred as Attributes in Concept Analysis literature. These Groupings are described as Concept and capture resemblance among a set of Elements founded on their known Properties.

In the particular case of Software reengineering, the system are Composed of a big amount of dissimilar Entities (Classes, Methods, modules, subsystems) and there are dissimilar kinds of relationships among them. It also signifies Relyencies among the Classes or entities. X-Ray views —a Technique based on Concept Analysis - which reveal the internal relationships among groups of Methods and Attributes of a Class. X-Ray views are Composed out of basic cooperation among Attributes and Methods and assist the Engineer to assemble a mental model of how a Class workings within inside.

Existing Design

Within Object Oriented Software, the smallest unit of progress and testing is a Class. Typically, a Class is Composed of Instance Variables utilized to signify the state, and Methods utilized to signify the performance of the Classes. Then, understanding and discovers several aspects anyhow a class works:

- How the Methods are interacting mutually (Coupling among Methods).
- How the Instance Variables are functioning (or not) mutually in the Methods (Coupling among Instance Variables).
- Which Methods are using (or not) the State of the Class.
- If there are Methods that form a cluster and describe mutually a accurate performance of the Class.
- Which Methods are measured as Interfaces
- Which Methods are utilized as entry points (Methods that are measured as Interfaces and communicate with other Methods defined in the Class).
- Which Methods and Instance Variables signify the core of the Class.
- Which Methods are using all the State of the Class.

In paper [1], the authors have given an design of Concept Analysis. Mathematically, Concepts are maximal collections of Elements distribution familiar Properties. To use the CA Technique, one only needs to identify the Properties of interest on each Element, and do not need to consider about all possible combination of these Properties, As these Groupings are made automatically by the CONCEPT ANALYSIS algorithm. The possibility of capturing resemblances of Elements in groups (Concepts) - based on the specification of simple Properties permit to discover familiar features of the Elements. When we are capable to distinguish the Entities in terms of Properties, and we can detect if these distinctiveness are constant in the system, then we can decrease the amount of Information to examine and we can have an abstraction of the dissimilar parts of a system. These abstractions assist us to start to see how the parts are working, how they are defined and how they are connected to other parts of the system .The Elements are the Instance Variables and the Methods defined in the Class, and the Properties are how they are related among themselves. If we have the set of Instance Variables {• , B}, and the set of Methods {P, Q, X, Y} defined in a Class, the Properties we use are:

B is utilized by P means that the Method P is accessing directly or through an access or / mutator to the Instance Variable B. Q is described in P means that the Method Q is described in the Method P via a self-call. It also shows indirect Relyencies among Elements if exists. They have also revealed dissimilar types of relations and Relyencies through some Notations.

{E•,..,E•} • {M•, ..,M•} means that the Entities{E•,..,E•} rely Exclusively on {M•, ..,M•}. This means that {M•, ..,M•} are the only Entities that are related through the property • to {E•, ..,E•}. {E•, ..,E•} • {M•, ..,M•} means that the Entities{E•,..,E•} do not rely Exclusively on {M•, ..,M•}.{E•, ..,E•} R*{M•, ..,M•} means that the entities{E•,..,E•} rely Exclusively and transitively on {M•, ..,M•}. This means that {M•, ..,M•} are the only ones that are related to {E•, ..,E•} through the property • and R•, where •• is an intermediary property, as there is a set {N1,..,Nk} such that: {E•,..,E•} • {N1, ..,Nk} •• {M•, ..,M•}. {E•, ..,E•} • {M•, ..,M•} means that the Entities{E•,..,E•} do not rely completely but transitively on {M•,..,M•}. This means that {M•, ..,M•} are not the only ones that are connected to {E•, ..,E•} through the property • and ••, where •• is an intermediate property, as there is a set {N1, ..,Nk} such that: {E•, ..,E•} • {N1, ..,Nk} •• {M•, ..,M•}. A special case: {E•, ..,E•} ¬• {M•, ..,M•} means that the Entity {E•, ..,E•} has any Relyencies on {M•, ..,M•}. In paper [2], the authors have discussed dissimilar types of X-Ray views, which will be useful for our future work. In paper [3] there is a Concept on Modularization using the Conceptual analysis on Object Oriented Environment.

Application

Our design is now to use the above said Concepts in the Environment of Java and to way out the Modularization in Java programs. Modularization as well helps in Software reengineering. For the present reason, let us have an example of Java coding. We have applied the planned design in dissimilar Properties of Java programming each of which are illustrated underneath.

Polymorphism

Polymorphisms deal with of dissimilar forms of a Method where limits are dissimilar according to the forms of the Methods. Polymorphisms can too happen in constructors.

```
Class Overload {
int •;
Void test (int x) {
•=x;
System.out.println (“•: “ + ¡);
}
void test(int x , int y) {
•= x;
int b= y;
System.out.println (“• •nd b: “ + • + “,” + b);
}
}
Class Method Overloading {
Public Static void main (String args[ ] ) {
Overload overload = new Overload( );
Overload. test(10);
Overload. test(10, 20);
}
}
```

Instance Variable a is not commonly associated to test(int x) or
test(int x, int y).

{ a } • { test(intx), test(int x, int y) } ——————1

{ b } • { test(intx, inty) }————————————
——2

So, where 2 relationships are found and we can say that these two relations will create two Concepts.

Overriding

In a class of hierarchy, when a method in a subclass has the same name and the same type signature as a method in that of a superclass, then the method in a subclass is called an override the technique in a superclass. As we know that overriding ia a runtime polymorphism. The Methods are similar in syntax. It is certain in the run time, which Method is to be invoked.

Method overriding.

```
Class   {
Int i, j;
 ( int •, int b ) {
i = •;
j = b;
}
// display i and j
Void show( )
{
System.out.println (“i and j: “ + i + “ “ + j );
}
}
Class B {
intk;
B( intc) {
k = c;
}
void show( ){ // display k – this override show( ) in
•
System.out.println (“k: “ + k);
}
}
Class Override {
Public Static void main(String args[ ] ) {
B subOb = new B(1 );
subOb.show( ); // this calls show( ) in B
}
}
```

• ( ) accessing the Instance Variables i , j
 . Show( ) [show( ) of Class •] accessing the Instance

Variables directly for both • ( ) and . show( ) the Relation will come like this :

{i, j} • {• ( ), . show( ) } —————————— 1

Similarly B( ) andB. show( ) absolutely related to

Variable k. so we can say that the relationships are like this:

{B( ), B. Show( )} • {k} ———————————— 2

So , two relations are generating two Concepts . Now as the
Method show( ) is overridden, we shall consider the .
show( ) and B. show ( ) as a single Entity say show( ).

•s we are allowing for here only the property of Overriding we shall pay no attention to the other Methods and we can decrease the relationships or Relyencies like:

show( ) • { i, j, k } and as a result creating a module.

Inheritance

Inheritance is a characteristic in Java where the constituent of a Class becomes heir to

Properties or Attributes from its base Classes. Inheritance can be of dissimilar forms multiple, hierarchical, multistage and hybrid.

```
Class   {
Int x;
Int y;
void show x y ( )
{
System.out.println (" x and y : " + x + " " + y );
}
}
Class B extends  •
{
intz;
void show z( ){
System.out.println( "z : " + z );
}
}
Class Inheritance {
Public Static void main (String args [ ] )
{
 • = new  ( );
B b= new B( );
• . x = 5; // x of superclass  •
•.y = 5; // y of superclass  •
Show xy ( ); // show xy ( ) of Class •i.e. the superclass
x= 10; // x of subclass B as extended from  •
y= 10; // y of subclass B as extended from  •
k= 10; // k of own subclass B
b.show xy( ); // show xy of superclass • extended
by
Subclass B
b. show z( ); // own Method show z( ) of Class B
}
}
```

x, y is equally and wholly related to {show xy ( )} in case of Class • and in case of Class B too as Show xy ( ) is inherited by Class B from Class . So we can say that :

{x, y } • { show xy( ) }. Here one Concept is created. ——————————————— 1

So, the relation goes like this :

{z } • {show z( )}. Another Concept is created.——2

• s all relationships are mutually Exclusive we can get aggregation and can be write as:

{x, y, z }•{ show xy( ), show z( ) }——————————3

Exception handling

Exception handling is the property of Java by which it can invoke some work when a number of usual jobs are prohibited to implement by some defective Codes.

```
Class MyException {
Public Static void main (String args [ ] ) {
Int d, a;
try {
d= 0;
a= 42 / d;
System. Out. Println ( " This will not be printed. ");
} Catch (• rithmetic Exception e ) { // Catching of
divided by zero error
System.out.println( " This is Division by zero
creating
an Exception !!!" );
} System.out.println( " This happen when the catch
is done …." );
}
}
```

Now here we can say that, try-Catch block is in a straight-line access the Variables as every time the try block is executed then only the Arithmetic Exception e arises i.e. the Instance Variables are mutually exclusive with Exception e. Thus they are creating Concepts and as a result a module. By Notation we write that:

{ a, d} • {try-Catch ( ) }

Abstraction

Abstraction is the characteristic of Java to cover facts from the customers so that the user can deal simply with the functionalities of the Codes.

```
Class P•ly
{
// Implementation and private members hidden
P•ly (int, int);
double eval ( double );
void add ( P•ly );
Void mult (P•ly);
Public String toString ( );
}
Public Class Binomial
{
```

```
Public Static void main (String[ ] args )
{
intN = Integer. parseInt( args [ 0 ] ) ;
double p = Double. ParseDouble (args [1]) ;
P•ly y = new P•ly ( 1, 0);
P•ly t = new P•Íly ( 1, 0);
t.add ( new P•ly ( 1, 1) );
for ( inti = 0; i < N; i++ )
{
y. mult ( t ) ;
Out.println(y + “”);
}
Out.println(“value: “ + y.eval ( p ) );
}
}
```

Method add ( ) is using PÍly ( ) constructor. So, by Notation
we can write that :
{add ( ) } • * { PÍly ( ) } ———————————— (1)

Furthermore, the Method add( ) and mult ( ) by means of the total Class Methods straightforwardly or not directly as we can see from the Class definition .Then also we can write that,
{add ( ), mult ( ) } •* {Class P•ly } ——————— (2)
It means that those Methods are using the Class Methods or else.

CONCLUSION

In this paper we endeavor to implement, execute and to put in operation the basic Properties of Object Oriented model with the help of Concept Analysis Notation to generate Concepts as well as the modules. These modules will help us for reengineering as reengineering deals with change in the modules of Codes to build the obsoluted or about to obsoluting Software rework.

REFERENCES

1. Gabriela Ar´evalo, Stephan Ducasse, Oscar Nierstrasz. “X-Rayview on a Class using Conceptual Analysis” published in the Conference at University of Antwerp, p: 76-80 in 2003.
2. Gabriela Ar´evalo, Stephan Ducasse, Oscar Nierstrasz. “Understanding Classes using X-Rayviews” cited in the Proceedings of 2nd. MASPEGHI (ASE), p: 9-18 in 2003.
3. H. H. Kim, Doo-Hwan Bae. “Object-oriented Concept Analysis for Software Modularization” cited in the Proceedings of IET Software, p: 134~148 in 2008.
4. S. Demeyer, S. Ducasse, andO. Nierstrasz. Object-Oriented ReEngineering Patterns. Morgan Kaufmann, 2002.
5. B. Ganter andR. Wille. Formal Concept Analysis : Mathematical Foundations. Springer Verlag, 1999.
6. M. Fowler. Refactoring: Improving the Design of Existing Programs. Addison-Wesley, 1999.