



# **FROM OBJECT-ORIENTED TO COMPONENTS: Generating Component-Based Software from UML Diagrams**



:: Submitted by::

**Hind Al-Hakami**

:: Supervisors::

**Professor Ghazy Assassa and Dr. Amir Tourir**

**KING SAUD UNIVERSITY  
COLLEGE OF COMPUTER AND INFORMATION SCIENCE  
DEPARTMENT OF COMPUTER SCIENCE**

**IN THE NAME OF ALLAH  
THE COMPASSIONATE  
THE MERCIFUL**

## **ACKNOWLEDGMENTS**

To start with, I want to thank the Almighty Allah for his blessings and benevolence. They ensured the completion of this task in a satisfactory manner. I was able to overcome the arduous challenges of writing a research paper only because of the wonderfully warm and supportive environment at home. My parents made sure that I was able to focus on this paper without the usual worries of life. To them, I owe everything. I was very fortunate to also have the strong and unwavering support of my teachers. First and foremost, I owe a huge debt of gratitude to Professor Hassan Mathkour. He helped me conceptualize the project and encouraged me to keep going through all the rough patches along the journey. My supervisors, Professor Ghazy Assassa and Dr. Amir Tourir, went beyond the call of duty to not only guide me at every stage of this project but also motivated me to strive for excellence. Under the guidance of these three great teachers, I learnt more than any book could have taught me.

# TABLE OF CONTENTS

<b>ACKNOWLEDGMENTS .....</b>	<b>II</b>
<b>TABLE OF CONTENTS.....</b>	<b>III</b>
<b>LIST OF TABLES .....</b>	<b>V</b>
<b>LIST OF FIGURES.....</b>	<b>VI</b>
<b>LIST OF FIGURES.....</b>	<b>VI</b>
<b>ABSTRACT .....</b>	<b>1</b>
<b>INTRODUCTION.....</b>	<b>2</b>
Project Goal .....	4
Project Objectives .....	4
Methodology .....	4
Overview.....	5
<b>1 REVIEW OF RELEVANT AND RELATED LITERATURE .....</b>	<b>6</b>
1.1 Legacy System Renovation.....	6
Motivation: The Business Case for Renovation.....	6
1.1.1 Renovate From Source Code.....	6
1.1.2 Renovate From Design: .....	8
1.2 Unified Modeling Language (UML) .....	9
Why Use UML.....	10
UML Diagrams .....	10
Structural diagrams .....	10
Class diagram .....	10
Component diagram.....	10
Xml Meta-model Interchange (XMI) .....	10
1.3 Design Metrics .....	11
Coupling.....	11
Cohesion.....	11
Complexity .....	12
1.4 Reverse Engineering .....	12
1.5 Clustering Technique .....	13
Partitional Clustering.....	13
Hierarchical Clustering.....	13
1.6 Forward Engineering .....	14
1.7 System Reengineering .....	14

<b>2</b>	<b>TOOLS.....</b>	<b>15</b>
	SDMetrics.....	15
	Java Universal Network / Graph (JUNG).....	15
	ArgoUML.....	15
	Stylus Studio.....	16
	Sourceforge.....	16
<b>3</b>	<b>METHODOLOGY.....</b>	<b>17</b>
	Project Phases.....	18
	3.1 Reverse Engineering.....	18
	3.2 Read and Analyze XMI file.....	18
	Weights Calculations.....	18
	3.3 Create Weighted Directed Graph.....	23
	3.4 Cluster the Graph.....	23
	3.5 Generate GraphML file.....	24
	3.6 Transfer GraphML to XMI file.....	27
	3.7 Forward Engineering.....	29
<b>4</b>	<b>SYSTEM DESIGN.....</b>	<b>30</b>
	4.1 Class Diagram Level # 0:.....	30
	4.2 Input/Output Package.....	31
	4.3 XMI2GraphML Package.....	32
	4.4 CLUSTER Package.....	36
	4.5 Graphical User Interface Package.....	38
	4.6 Overall Class Diagram.....	42
<b>5</b>	<b>CASE STUDY.....</b>	<b>43</b>
<b>6</b>	<b>CONCLUSION AND FUTURE WORK.....</b>	<b>57</b>
	<b>REFERENCES.....</b>	<b>58</b>

## LIST OF TABLES

Table 1: Dependency Weigh Value Equations .....	22
Table 2: Analytic of Design Elements .....	43
Table 3: Analytic of Relations .....	44
Table 4: Analytic of Relations .....	55

## LIST OF FIGURES

Figure 1: Project Phases.....	17
Figure 2: Directed Weighted Graph.....	23
Figure 3: Clustered Graph.....	24
Figure 4: GraphML Schema .....	26
Figure 5: XSLT - GraphML2XMI.xsl .....	29
Figure 6: Class Diagram Level # 0 .....	30
Figure 7: Input / Output Package .....	31
Figure 8: XMI 2 GraphML Package.....	32
Figure 9: XMI 2 GraphML Package Dependencies Level # 0 .....	33
Figure 10: XMI 2 GraphML Package Dependencies Level # 1 .....	34
Figure 11: Cluster Package .....	36
Figure 12: Cluster Package Dependencies Level # 0.....	36
Figure 13: Cluster Package Dependencies Level # 1.....	37
Figure 14: Graphical User Interface Package .....	38
Figure 15: GUI Package Dependencies Level # 0.....	39
Figure 16: GUI Package Dependencies Level # 1 .....	40
Figure 17: Overall Class Diagram .....	42
Figure 18: Window#1 – Graph Representation .....	43
Figure 19: Forum Case Study GraphML File .....	55
Figure 20: Window#2 – Components Manager.....	56

## **ABSTRACT**

Demand for component-based software development is increasing exponentially due to highly dynamic environments confronting software systems in today's world. This is a result of factors such as frequent changes in business (user) requirements and challenging development schedules. Many of these challenges are indeed directly related to rapid changes in software and hardware technologies. To deal with this environment, this project develops a tool to generate component-based software from object-oriented design. It is expected that this tool will facilitate the transformation of object-oriented legacy system to component-based system by using their design parameters or build a new component-based system.



## Introduction

Change is the only constant in the contemporary world. While this is true for most aspects of our lives today, it is particularly so for software development. This field is responding to forces of change from many directions: software development life cycle is getting exponentially shorter, software application domains are expanding rapidly, pressures on the software development costs are higher than ever before, and there is an ever-increasing need for integration of different domains. In addition, there are additional pressures to change resulting from new technological advances in both software and hardware.

The famous scientist Charles Darwin once wrote, “It is not the strongest of the species that survive nor the most intelligent, but the ones who are most responsive to change.” His words are as relevant to the software companies as they are to any other specie. Thus, the future success and survival of software companies will depend critically on their ability to manage complexity and rapidly adapt to change. More specifically, the ability to reuse system components already developed will be a key success factor to survive and respond to the rapidly changing requirements of the market. This implies that the development process must change its focus from programming-intensive activities to greater emphasis on reuse, integration, standards, management of complex and flexible structures, finding proper solutions, tradeoff analysis and marketing survey.

Component recovery and re-modularization were strategies used to get a firm grip on large and complex legacy systems suffering from *ad-hoc* changes. This was done by recovering logical components and restructuring the physical components accordingly to decrease coupling among components and thereby increase the cohesion within components. The idea that software should be componentized, built from prefabricated *components*, is not a new one. It was first published in Douglas McIlroy's address at the NATO conference on software engineering in Garmisch, Germany, 1968 titled “*Mass Produced Software Components*.” This conference set out to counter the so-called software crisis. With the increase in the speed of change, the relevance and popular acceptance of this idea has also grown.

It is now estimated that the largest portion of traditional software development cost goes to maintenance process. According to some estimates, almost 80 percent of the applications budget is typically allocated to software maintenance, as many legacy systems tend to be very hard to adapt and, over time, suffer from lack of proper documentation. In addition, those systems find it harder to keep pace with changes required by the evolving business needs and advances in technology. Beside all this, the growing need for intra-organization collaboration and knowledge sharing makes it imperative for the companies to modernize for their survival.

There are many options for transforming legacy systems into component based systems: it can be done semi-automatically or automatically. In this paper we deal with a subset of these approaches.

Software systems from the past (“legacy” systems) did not anticipate the pace of change in the modern world. Thus they suffer from several drawbacks:

- They tend to be very hard to adapt.
- It is hard to train people on legacy systems as often there is absence of relevant knowledge regarding these old systems and new users are not familiar with the old methodologies and concepts. This makes it hard to maintain them.
- These systems cannot keep up with the rapid adaptation demanded by realities of fast-paced modern businesses and technology.

The rapid growth of horizontally and vertically integrated supply chains requires integration of systems both inside and between organizations. The speed of technology changes, coupled with the growing requirement for intra-organization software collaboration, makes modernization of software a must for survival. Transformation enables migration to new platforms, and often includes translation to new programming languages. Evolution is necessary to cope with endless new software releases and manage hardware and software obsolescence. To succeed, the methodology used for managing evolution must include component-based software engineering (CBSE).

It can not be denied that legacy software systems provide valuable functionalities that have been proven in practice. These legacy systems often embody valuable intellectual assets, representing

successful business practices and procedures. It is in the interest of the enterprise to reuse and reconvert these intellectual assets and not discard them. This is where retrieving useful components from the legacy systems can prove to be much more effective than building them from scratch.

## **Project Goal**

Generate component-based software from object-oriented design using UML class diagrams.

## **Project Objectives**

- a. Develop a system recovery tool that converts an object-oriented structural design into component-based software.
- b. Test it on java programming language.
- c. Ensure that it can be used for any programming language.
- d. Find an optimum method to define relation importance weights to give satisfactory results.
- e. Suggest future modifications to improve the system recovery tool.

## **Methodology**

This system takes UML class diagrams via XMI file as inputs. Analysis class diagrams and set weight for each edge according to type of relation it represent. Take this weighted graph and cluster it into highly connected clusters then generate the XMI file so that each cluster represents a component. Generate fully deployable components using one of the Forward Engineering tools.

With continued acceleration in the speed of change, it will be important to preserve what is valuable and develop new programs for the evolving business needs. In this context, the component-based approach is likely to acquire even greater relevance in the future.

## **Overview**

The remainder of this paper is divided into 5 sections. In sections 1, we start by a review of the relevant and related literature. Section 2 provides a description of various tools and techniques utilized in this project. Sections 3 briefly describe the project plan and phases and describe in some details procedures and methodology. A case study is overviewed in section 4. And finally, conclusion and future work are stated in section 5.

# **1 Review of Relevant and Related Literature**

## **1.1 Legacy System Renovation**

### **Motivation: The Business Case for Renovation**

As mentioned earlier in the introductory section, in today's business environment, there is a constant need for updating and renovating business-critical software systems for a variety of reasons. Some of these reasons are: business requirements change rapidly and frequently, technological infrastructure is modernized at exponential speeds, changes in government laws. For these reasons, the fields of reverse engineering and system renovation are becoming increasingly important. The interest in such subjects originates from the difficulties that organizations encounter when attempting to maintain extremely large software systems developed in the past. Such software systems are often referred to as "legacy systems," since they are a legacy of many different people that have developed and maintained them. It is not very difficult to see that is very challenging—if not impossible—to maintain them.

Before the actual renovation can start it is necessary to make an inventory of the specification and the documentation of the system to be renovated. Also at this point there is a challenge for software engineers since the old systems lack mostly these sources of information. Experience shows that either there is no documentation at all, or the original programmers that could possibly explain the functionality of parts of the system have left, or both. The only documentation that is left is the source code itself. Thus, since the vital information of the software is solely accessible via the source code it will be necessary to develop tools to facilitate the renovation--a task for software engineers.

### **System Re-factoring Techniques**

#### **1.1.1 Renovate From Source Code**

Most of methodologies in this category include the following typical set of steps: (i) restructuring the legacy code, (ii) extracting reusable business models corresponding to candidate components,

and (iii) wrapping them into deployable components. And since most of legacy systems were written in COBOL there was a lot of researches dealing with reengineering COBOL codes as in [28]. Errickson-Connor in [11] provides an excellent guide for source code renovation processes through a four stage process consisting of cleaning, restructuring, transformation, and managing. In [8] they analyze the legacy code before restructuring rather than cleaning. This approach of analyzing the legacy code thoroughly before cleaning it appears to be a more efficient. In fact, the cleaning is part of analyzing processes and can even be automated. As stated by them, there are several starting points to analyze legacy systems. They include: interviewing users, maintainers and, if possible, the designers of the legacy system. In addition, we can use persistent data stores as a starting point and track down the procedures or classes that can be a candidate of components. Similarly, as done in [23], we can use legacy system features of the systems along with the regression testing techniques to define fine-grained components. This technique allows each component to accomplish one feature while not all features have to be wrapped into a component. Another starting point for the analysis of the legacy systems can be the system structure or the use cases as described in [3]. They used PERFORM graph and the dominance relation of the calling structure between procedures to generate dominance tree to identify reusable components. As we will see later, these tools are more commonly used in the second category of methodologies which renovate from design.

Following are examples of some system analysis and component mining methods:

#### **1.1.1.1 Design Patterns**

This approach seeks to identify instances of Design Patterns based on the idea that they can be characterized as a group of classes sharing mutual relations. “Object-Oriented Reengineering Patterns” Book by Demeyer et al describes a number of recurring solutions that experts apply while reengineering and maintaining object-oriented systems. The principles and techniques described in this book have been observed and validated in a number of industrial projects, and reflect best practice in object-oriented reengineering.

Reverse engineering patterns help to extract models from existing applications and source code, and reengineering patterns help to identify and resolve problems in legacy code.

### **1.1.1.2 System Features Approach:**

Using regression testing documentation which is an important resource tell what features the system accomplish and by using any existing code-profiling tool to trace the code implementing a certain feature; [23] found three cases were the feature implementation needs refinement. First case is when a feature implementation spread among several functions, Second case is a function implements several features, the last case is a combination of both preceding cases. Matching one of these cases the refactor the code then create a fine-grained components.

### **1.1.2 Renovate From Design:**

Renovating from design has an advantage that its platform independent, can readily verify dead codes.

#### **a) Clustering Approaches**

##### **1. Divisive Clustering (Top-Down Approach)**

Graph clustering analysis [5] [21] processes goes as follow, first abstract the legacy system into undirected graph, calculate the edge strength which is the core step, then cut all the edge smaller than some threshold, and at last vary the value of the threshold to test the best gained candidates. Edge strength was calculated using 3-cycle and 4-cycle edge density.

##### **2. Agglomerative Clustering (Bottom-Up Approach)**

In Graph iterative analysis [21] you have to predefine the size of out-coming graph, the size of sub-graphs to experiment, and the independency threshold then iteratively apply the independence metrics to identify the most independence sub-graphs until the number of clusters meets the predetermined size of graph.

The independency metric is calculated using this equation  $[IM = cohesion / (coupling * subgraph\ size)]$ . Where cohesion is the sum of weights of inner relations, within the subgraph, coupling is sum of outer relations and the size is determined by the number of sub-graph's vertices.

Component Identification Method with coupling and cohesion [19]:

Highly demands on well formed architecture that is divided into well defined subsystems.

In their methodology to identify components, they first identify the core classes and then use agglomerative clustering for each core class group the subsidiary class that relate to the core class by a relation with a weight beyond the threshold value, the relation weight is calculated as the multiplicity of cohesion\* interaction coupling\* static coupling.

While in [15] Obtain from object oriented analysis phase the structural view from UML class diagrams. And all of use case diagrams and sequence diagrams to represent the dynamic view. Calculate the relation strength which is equal to the relative importance of static weight \* static weight + relative importance of Static weight \* Dynamic weight, where the summation of the both importance weight must equals exactly one. Then binary merges cluster is iteratively done until reach a cut-off depending on a predefined threshold to identify the components candidates also a further refinement are done to maintains high cohesion within a component and low coupling between components some examples of these refinement is retain inherited class and it's parents in the same component to keep low coupling. The user also can add, move, exchange or delete some of the component classes.

Two stages approach provided in [18] to generate the components from the Legacy source code first stage is to create basic components and then refine components. First stage is accomplish in four steps, group classes related to each other by a composition relationship into a component, followed by cleaning the unused classes, After that create a component that groups the hierarchical related classes if the parent class is abstract add a copy of it to the component in the other hand if parent class was concrete move it to the component at last clean unused classes again. Second stage use agglomerative clustering starting with basic components and considering other classes as components then gradually group these components into a bigger component based on coupling and cohesion metrics and using the component complexity threshold to manage the granularity of the created components.

## **1.2 Unified Modeling Language (UML)**

“Unified Modeling language is an industry standard language for specifying, visualization, constructing, and documenting the artifacts of software systems standardized by the”.



One way to use UML is as blueprint which is a relatively detailed design diagram used in reverse and forward engineering.

## **Why Use UML**

UML is mainly used due to three main reasons:

1. UML is independent from programming languages and development process.
2. Supports component methodology.
3. Provides an XML based interchange language which is XMI.

## **UML Diagrams**

Currently many of UML tools support all thirteen diagrams provided by UML, and are classified into three categories; first category includes six structural diagrams known as static diagrams; second category includes three behavioral diagrams known as dynamic diagrams, and the third category includes four interaction diagrams.

In this project we will consider only class diagrams and component diagrams from the structural view.

## **Structural diagrams**

### **Class diagram**

UML class diagram is a “Pictorial representation of a detailed system design” [36] consisting of classes, interfaces, and relations among them. It represents a “*static view*” of the system.

Relations are of different types, associations, aggregation, generalization and dependencies.

### **Component diagram**

UML Component diagram shows system components and the dependencies between them.

## **Xml Meta-model Interchange (XMI)**

XMI is an OMG standard for exchanging metadata information via XML. The most common use of XMI is as an Interchanging format for UML models. It is supported by almost all UML tools, the most popular versions are 1.0, 1.2, 2.0, and now 2.1 was released. Since versions less than

2.0 don't support visualization of diagrams and because we are using version 1.2 we can't represent the generated component diagram. We use version 1.2 and not a newer version because currently most of the forward engineering tools support version 1.2.

### **1.3 Design Metrics**

Design Metrics provide measurement of several aspects of quality of design properties. With component development, metrics help us to measure the cohesive within a component's constituents, coupling between components, and the granularity of components.

#### **Coupling**

Since the coupling is a measure of how strongly a component is connected to, has knowledge of, or relies on other components, the aim is to maintain the coupling as low as possible, but because the "components are for composition" [38] we can not eliminate coupling between classes. Despite this fact, high coupling to stable or preservative elements is acceptable. Benefits of Low Coupling are:

- a. Increase reusability.
- b. Increase maintainability.
- c. Increase understandability.

#### **Cohesion**

A measure of how strongly related and focused the responsibilities of component's constituent are [25] i.e. component should implement a single function or a highly related functions.

##### **Very Low Cohesion**

Component constituents are responsible for many things in many different functional areas.

##### **Low Cohesion**

Component is very complex and hard to manage even though it serves one functional area.

### **High Cohesion**

Component has moderate complexity, serves one functional area, but depends on other components.

### **Moderate Cohesion**

Component has sole responsibilities in a few different areas that are logically highly related to each other. Going down the level of cohesion is enhanced.

### **Complexity**

Class diagram complexity measures the degree of connectivity between elements of a design unit and can be measured by measuring the complexity of both classes and relations composing the diagram.

#### **Class Complexity**

Many factors should be considered while measuring class complexity such as class size, Inheritance, the number of public operators and attributes [16], and the number of method invocations among the methods within the classes.

#### **Relations complexity**

Not only different kinds of relations have different degrees of complexity even different types of each kind have different degrees of complexity.

## **1.4 Reverse Engineering**

Reverse engineering has its origins in hardware technology and denotes the process of obtaining the specification of complex hardware systems. And in Software has been defined as “The process of analyzing software with the objective of recovering its design and specification” [37].

Often, over time, the design documentations are lost. Even if exist, they are not updated during the process of updating or maintenance of the original software system. Hence, they do not reveal the actual current design. Reverse engineering is used to recapture the high level design of the system. In our approach we used reverse engineering to obtain the system structure via UML

class diagrams of the current updated system that will be renovated. Reverse engineering restricts itself to *investigating* a system without any adaptations or modifications.

## 1.5 Clustering Technique

Clustering is the process of grouping some related data according to similarity of measurements. In our context similarity is represented by the level of dependency between classes.

### Partitional Clustering

Considers dataset as a whole cluster and divide into non-overlapping clusters. In partitional clustering a number of clusters to be produced or size of clusters must be predefined. And usually the cluster's elements are chosen arbitrarily. One of the famous partitional clustering is K-means clustering.

### Hierarchical Clustering

Hierarchical Clustering provides a set of nested clusters that are organized as a tree.

#### Agglomerative Clustering (Bottom-Up)

Starts with each element as a cluster and iteratively do a successful merges into a larger clusters.

**Min-Max cut:** The Min-Max cut main principle is the association between two sub-graphs is minimized, while association within each sub-graph is maximized [10].

#### Divisive Clustering (Top-Down)

Treats all dataset as a whole cluster and repeatedly break them into smaller clusters.

**Average Similarity:** The average similarity within a cluster must be maximized and minimized between clusters.

## **1.6 Forward Engineering**

As defined in [37] “Forward Engineering is the set of engineering activities that consume the products and artifacts derived from legacy software and new requirements to produce a new target system”.

Generating code in forward engineering process can be done in several ways, one way is model driven Architecture (MDA) based forward engineering which uses templates mostly written in Velocity scripting language to generate the code. Other way is to use design patterns.

## **1.7 System Reengineering**

System Reengineering is the examination and alteration of an existing subject system to reconstitute in a new form. This process encompasses a combination of sub-processes such as reverse engineering, restructuring, re-documentation, forward engineering, and retargeting.

## **2 Tools**

To accomplish this work we utilized some tools.

### **SDMetrics**

SDMetrics is a stand alone tool that measures the structural properties of UML designs using object-oriented measurement of size, complexity, coupling, and cohesion. It provide a statistical analysis of the design that can exported as an excel spreadsheet.

### **Java Universal Network / Graph (JUNG)**

JUNG framework is a free open source library written in java to manipulate, visualize, and analyze graphs and networks data. This library has many useful features; it supports many kinds of graphs including directed graph which is needed in this project, it provide a mechanism to attach metadata for the graph, nodes, and relations as labels and weights. Moreover, it supports some of the graph methodologies including filtering, decomposition, statistical analysis, and clustering such as Edge betweenness clustering. And the most important thing is the capability of importing and exporting graphML, an XML based file describes a graph, that facilitate the mission of generating the new components-based XMI file using a style sheet that converts the original file XMI based on the rules of the graphML.

### **ArgoUML**

ArgoUML is a free UML design open source tool. In this project we used ArgoUML-0.20.BETA\_2 that can reverse engineer a java codes and generate XMI1.2 file. At the beginning a plan to plug this tool into our system was in mind, but due to the long time the reverse engineering process takes which may extend to several hours or may not be accomplished at all. This will affect the efficiency of the system, so we will leave the user to use his discretion in deciding to use any UML tool that support generating XMI version 1.2. Moreover they use a simple code generation rather than full forward engineering.

## **Stylus Studio**

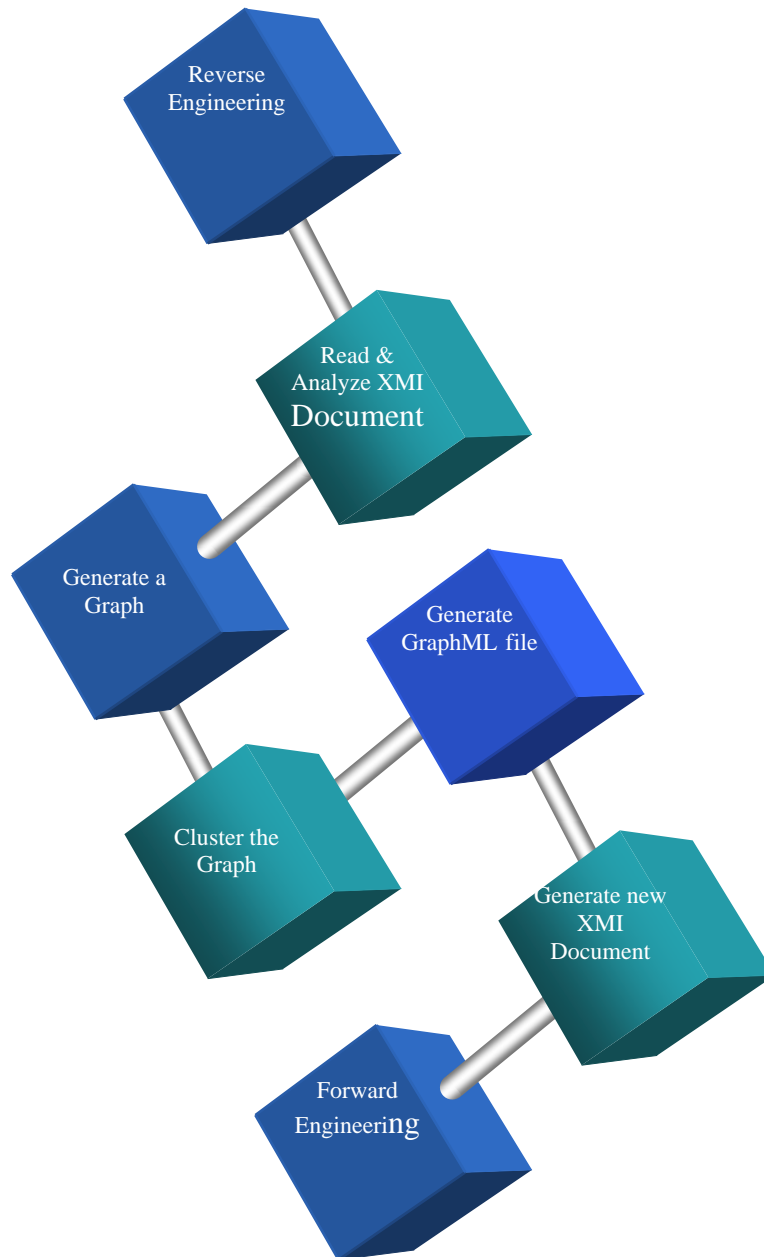
Stylus Studio is a commercial XML tool. I used “XSLT Mapper” to write the style sheet mentioned earlier.

## **Sourceforge**

It considered as one of the largest open source libraries I benefit from this site in taking the test cases java codes from there also to seek for an appropriate reverse engineering tool, since it has a lot of these tools with variety of specifications. All what you have to do is search for “XMI”

### 3 Methodology

The following diagram depicts the methodology graphically. It is followed by a description of individual phases.



**Figure 1: Project Phases**



## **Project Phases**

### **3.1 Reverse Engineering**

Generate UML Models from Java codes and then exports XMI file by using any UML tools that support exporting XMI v.1.2, This System is tested through ArgoUML and most of the UML tools can accomplish this task.

### **3.2 Read and Analyze XMI file**

Read class diagram's design elements and relations from XMI file, UML design elements are classes, interfaces, components, and packages which consist of combination of previous elements. Classes, interfaces, components, and the contents of packages are read. You will ask why the contents of the packages and not the packages themselves?? And this is to obtain more cohered clusters sense you cannot confirm that a package doesn't consists of any loosely coupled elements. Relations were also read and then coupling and complexity metrics values were extracted.

#### **Weights Calculations**

Assigning weights for several kinds of relations so that each kind of relation has a strength weight associated to it. But not all the edges representing a relation from the same kind has the same weight because we take into account the complexity of the supplier and/or client design element.

#### **Relations Types**

As stated in [16] they sorted relations according to its dependency weight importance. Also they stated that dependency relation is the most common relation

#### **Complexity**

Complexity of a design model is calculated using the in/out degree of node i.e. number of relations where the node is the client or supplier.

## **Coupling**

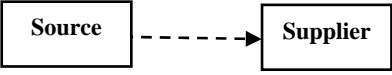
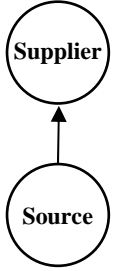
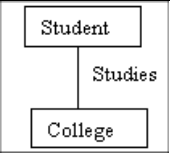
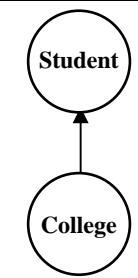
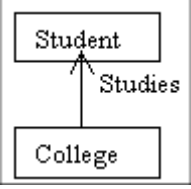
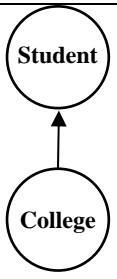
We can classify coupling metrics depending on their domain into:

### **Domain: Class**

- a. Export Coupling for Attributes (EC\_Attr): The number of times the class is externally used as attribute type. This is the number of attributes in other classes that have this class as their type.
- b. Import Coupling for Attributes (IC\_Attr): The number of attributes in the class having another class or interface as their type.
- c. Export Coupling for Parameters (EC\_Par): The number of times the class is externally used as parameter type. This is the number of parameters defined outside this class that have this class as their type.
- d. Import Coupling for Parameters (IC\_Par): The number of parameters in the class having another class or interface as their type.

### **Domain: Interface**

- a. Export Coupling for Attributes (EC\_Attr): The number of times the interface is used as attribute type.
- b. Export Coupling for Parameters (EC\_Par): The number of times the interface is used as parameter type.
- c. Import Coupling for Parameters (IC\_Par): The number of parameters in the interface having an interface or class as their type

DC <sup>1</sup>	RELATION	SYMBOL	WCDG SYMBOL	DWV OF RELATION
[1]	Dependency			$DW = DC * C(\text{Supplier})$  $C() = \text{complexity of the class}$
[2]	Association			$DW1 = DC * C(\text{Student}) * C(\text{College})$  $n, m = \text{max destination multiplicity,}$ if $n = 0$ let $1/n = 0$
	Directed Association			$DW = DC * C(\text{Student})$

<sup>1</sup> DC = Degree of Complexity.

DC <sup>1</sup>	RELATION	SYMBOL	WCDG SYMBOL	DWV OF RELATION
	Association Class	<pre> classDiagram     class Company     class Person     class Employment     Company -.- .. Employment     Person -.- .. Employment </pre>		$DW1 = DC * C(\text{Employment}) * C(\text{Person}) * C(\text{Company})$  $n = 1$ , if edge starting from Association class.
[3]	Aggregation	<pre> classDiagram     class Whole     class Part     Whole o-- Part </pre>		$DW1 = DC * C(\text{Whole}) * C(\text{Part})$
[4]	Composition	<pre> classDiagram     class Whole     class Part     Whole *-- Part </pre>		$DW1 = DC * C(\text{Whole}) * C(\text{Part})$

DC <sup>1</sup>	RELATION	SYMBOL	WCDG SYMBOL	DWV OF RELATION
[5]	Binding			$DW = DC * C(\text{Student})$
[6]	Inheritance/ Generalization			$DW1 = DC * C(\text{Language}) * C(\text{Java}) * C(\text{C++})$
[7]	Realization			$DW = DC * C(\text{Parser})$

**Table 1: Dependency Weigh Value Equations**

### 3.3 Create Weighted Directed Graph

Generate a Graph such that each vertex represents a design element and each edge represents the relationship between those elements. And the weight of edge is calculated as illustrated in Table 1:

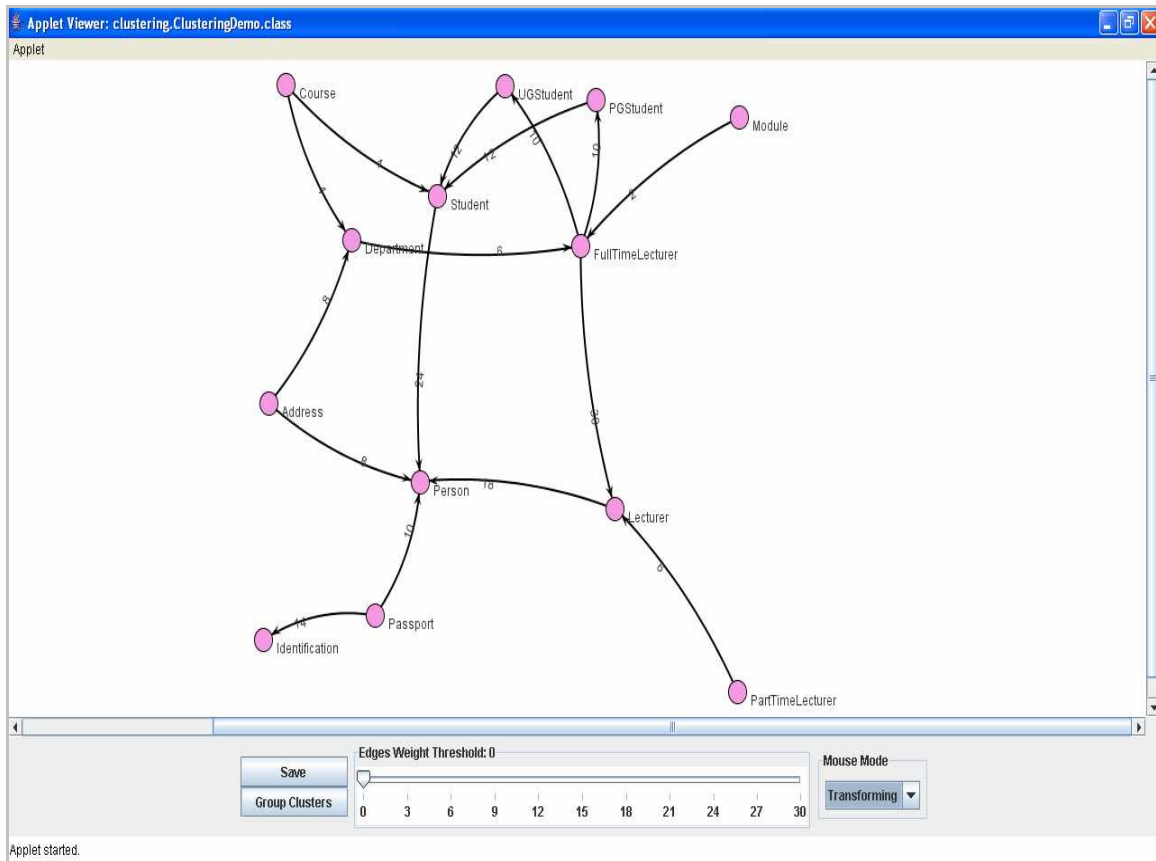
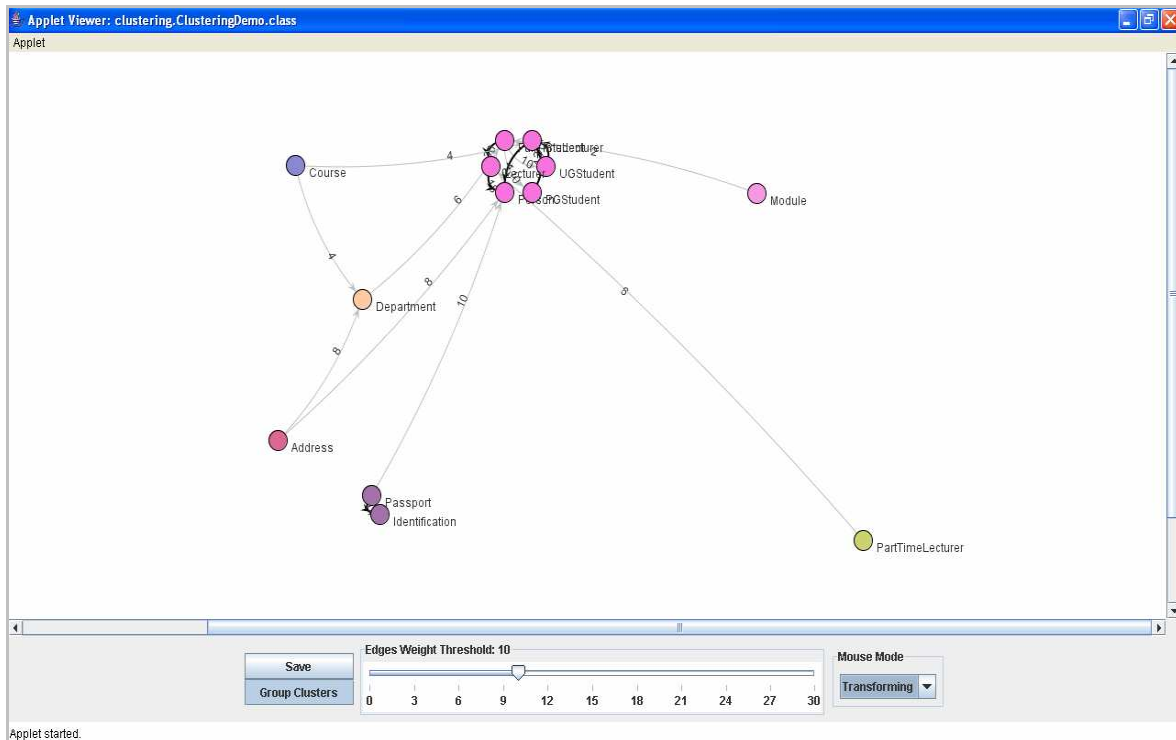


Figure 2: Directed Weighted Graph

### 3.4 Cluster the Graph

Using Hierarchical divisive clustering methodology and a Threshold placed by the user the weak edges are temporarily removed to generate clusters and replace them again.



**Figure 3: Clustered Graph**

### 3.5 Generate GraphML file

Generate a temporary graphML file, and because it is an xml based format we extended it to meets our needs, following is the schema of the modified graphML.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="http://graphml.graphdrawing.org/xmlns/graphml "
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:graphml="http://graphml.graphdrawing.org/xmlns/graphml ">
  <xs:import namespace="http://www.w3.org/2001/XMLSchema-
instance" schemaLocation="xsi.xsd"/>
  <xs:element name="graphml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="graphml:graph"/>

```

```

    </xs:sequence>
    <xs:attribute ref="xsi:schemaLocation" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="graph">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="graphml:clusters"/>
      <xs:element ref="graphml:edges"/>
    </xs:sequence>
    <xs:attribute name="Clustering_Weight_Key" use="required"/>
    <xs:attribute name="Edge_Weight_Key" use="required"/>
    <xs:attribute name="StringLabeller.LabelDefaultKey"
use="required"/>
    <xs:attribute name="edgedefault" use="required"
type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="clusters">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="graphml:cluster"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="cluster">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="graphml:node"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:integer"/>
    <xs:attribute name="name" use="required" type="xs:NCName"/>
  </xs:complexType>
</xs:element>

```



```

<xs:element name="node">
  <xs:complexType>
    <xs:attribute name="id" use="required" type="xs:integer"/>
    <xs:attribute name="name" use="required" type="xs:NCName"/>
    <xs:attribute name="xmi.id" use="required"
type="xs:NMTOKEN"/>
  </xs:complexType>
</xs:element>
<xs:element name="edges">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="graphml:edge"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="edge">
  <xs:complexType>
    <xs:attribute name="RelType" use="required"
type="xs:NCName"/>
    <xs:attribute name="directed" use="required"
type="xs:boolean"/>
    <xs:attribute name="edge_weight" use="required"
type="xs:decimal"/>
    <xs:attribute name="isRemoved" use="required"
type="xs:NCName"/>
    <xs:attribute name="source" use="required"
type="xs:integer"/>
    <xs:attribute name="target" use="required"
type="xs:integer"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

**Figure 4: GraphML Schema**

## 3.6 Transfer GraphML to XMI file

Transform original XMI file into a new XMI based on GraphML file according to the following algorithm:

```
<?xml version='1.0' ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:UML="org.omg.xmi.namespace.UML"
xmlns:a="http://graphml.graphdrawing.org/xmlns/graphml">
  <xsl:output method="xml" />
  <xsl:template match="/">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="UML:Namespace.ownedElement[local-
name(parent::*[1]) = 'Model']">
    <xsl:copy>
      <xsl:call-template name="graphml" />
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
```

```

<xsl:template name="graphml">
  <xsl:for-each select="//a:graphml/./a:cluster">
    <xsl:choose>
      <xsl:when test="count(child::* ) > 1">
        <xsl:variable name="cOne"
select="child::*[1]/@name" />
        <xsl:variable name="cTwo"
select="child::*[2]/@name" />
        <xsl:if test="not((count(child:*) = '2')
and ((contains($cOne, '.java') and contains($cOne, $cTwo)) or
(contains($cTwo, '.java') and contains($cTwo, $cOne))))">
          <xsl:element name="UML:Component">
            <xsl:attribute name="xmi.id">
              <xsl:text>.:000</xsl:text>
              <xsl:value-of
select="@id" />
            </xsl:attribute>
            <xsl:attribute
name="isSpecification">false</xsl:attribute>
            <xsl:attribute
name="isRoot">false</xsl:attribute>
            <xsl:attribute
name="isLeaf">false</xsl:attribute>
            <xsl:attribute
name="isAbstract">false</xsl:attribute>
            <xsl:copy-of select="@name" />
            <xsl:element
name="UML:ModelElement.clientDependency">
              <xsl:for-each
select="a:node">
                <xsl:element
name="UML:Dependency">
                  <xsl:attribute
name="xmi.idref">

```

```

        <xsl:value-of select="@xmi.id" />
                                                    </xsl:attribute>
                                                    <xsl:apply-
templates/>
                                                    </xsl:element>
        </xsl:for-each>
    </xsl:element>
</xsl:element>
</xsl:if>
</xsl:when>
</xsl:choose>
</xsl:for-each>
</xsl:template>

<xsl:template match="*[*/UML:Package/@xmi.idref]" />

<xsl:template match="UML:Package[not(@xmi.idref)]">
    <xsl:apply-templates
select="UML:Namespace.ownedElement/child::node()" />
</xsl:template>

<xsl:template match="a:graphml" />
</xsl:stylesheet>

```

**Figure 5: XSLT - GraphML2XML.xsl**

### 3.7 Forward Engineering

Using proper forward engineering tool generate type of components needed, like COM or EJB, here we test our project using Sparx – Enterprise Architect to generate EJBs.

## 4 System Design

### 4.1 Class Diagram Level # 0:

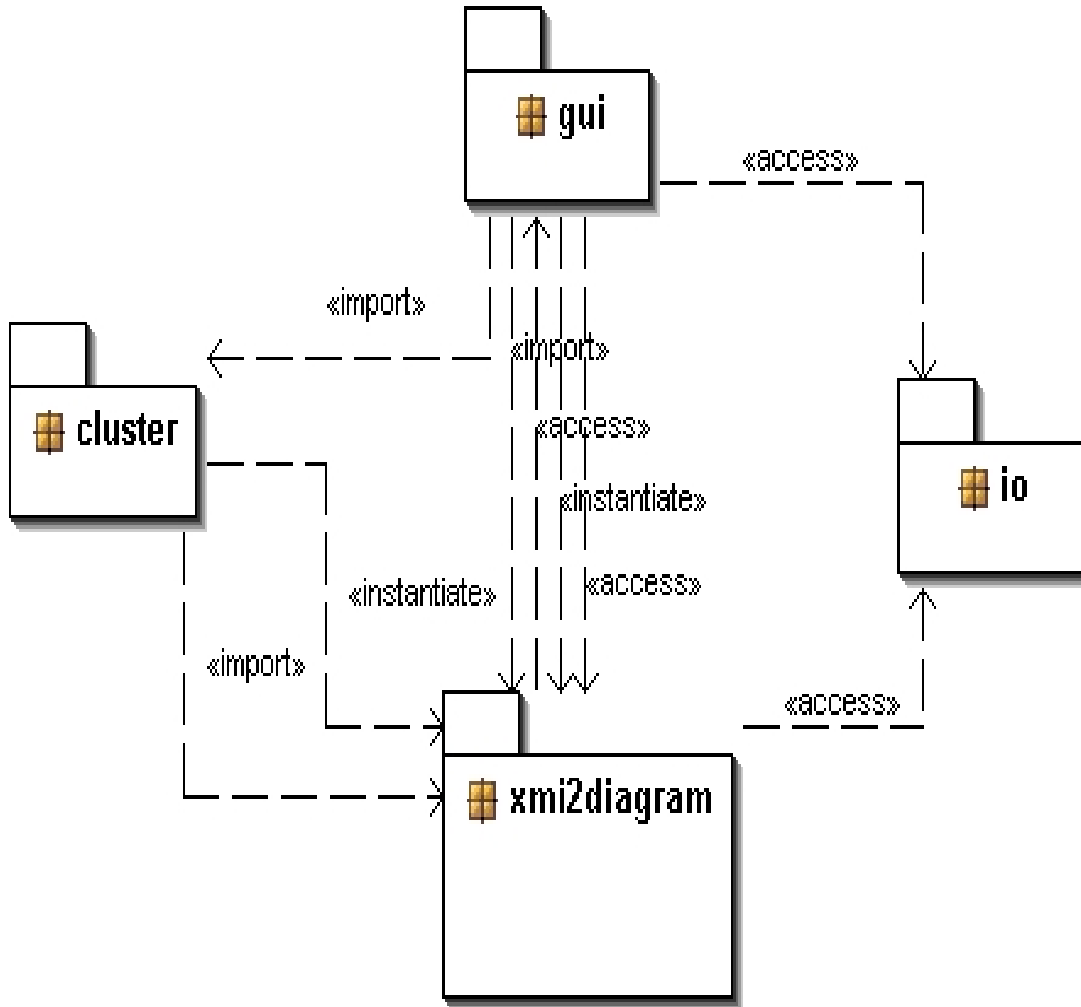


Figure 6: Class Diagram Level # 0

## 4.2 Input/Output Package



### 4.2.1 Isolated IO Package:



Figure 7: Input / Output Package

#### 4.2.1.1 Open Class

File chooser dialog to import source XMI file.

#### 4.2.1.2 Save Class

File chooser dialog to export result XMI file, which consists of the generated components.

### 4.3 XMI2GraphML Package



#### 4.3.1 Isolated XMI2GraphML

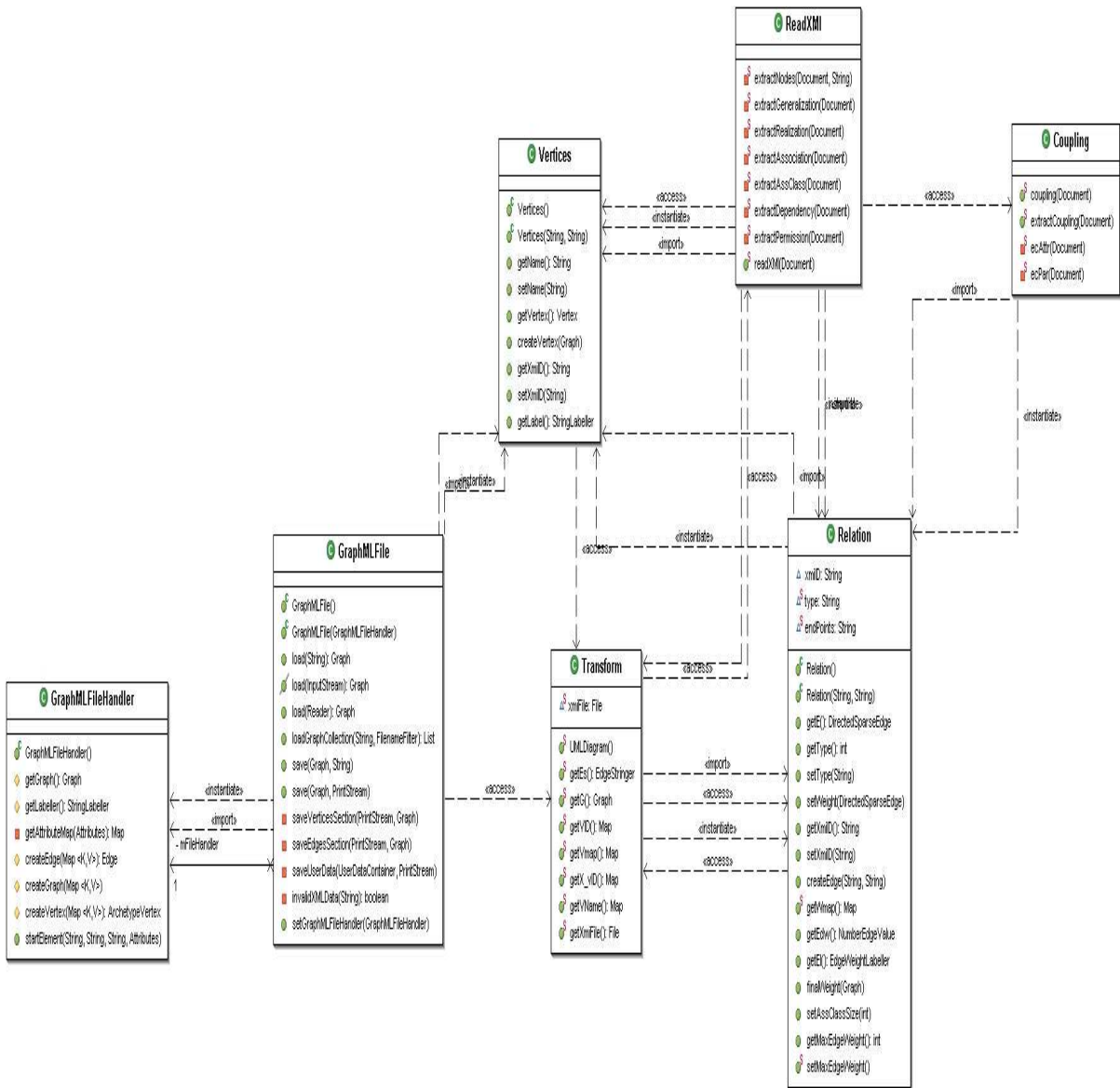


Figure 8: XMI 2 GraphML Package

### 4.3.2 XMI2GraphML Dependencies

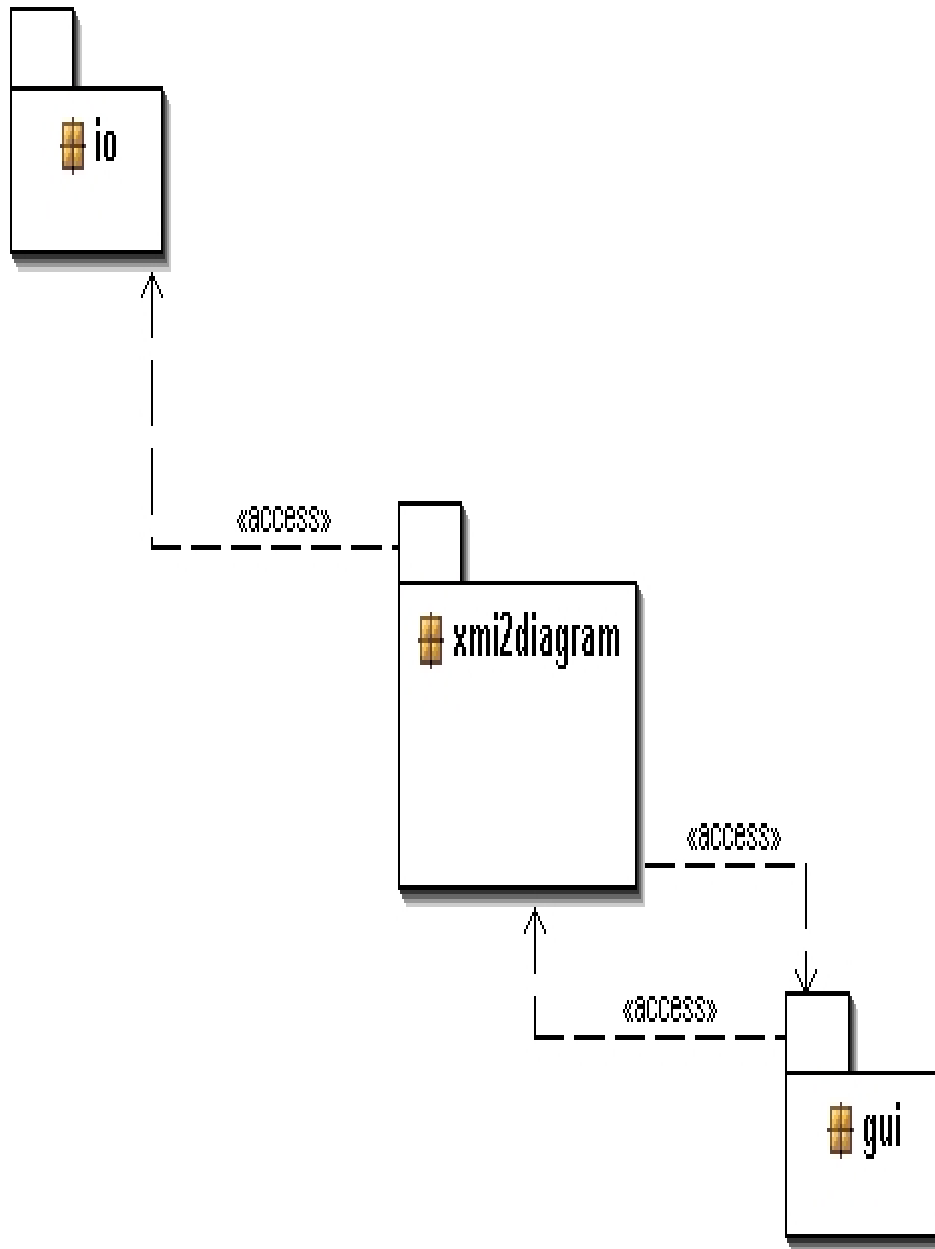


Figure 9: XMI 2 GraphML Package Dependencies Level # 0



### 4.3.3 XMI2GraphML

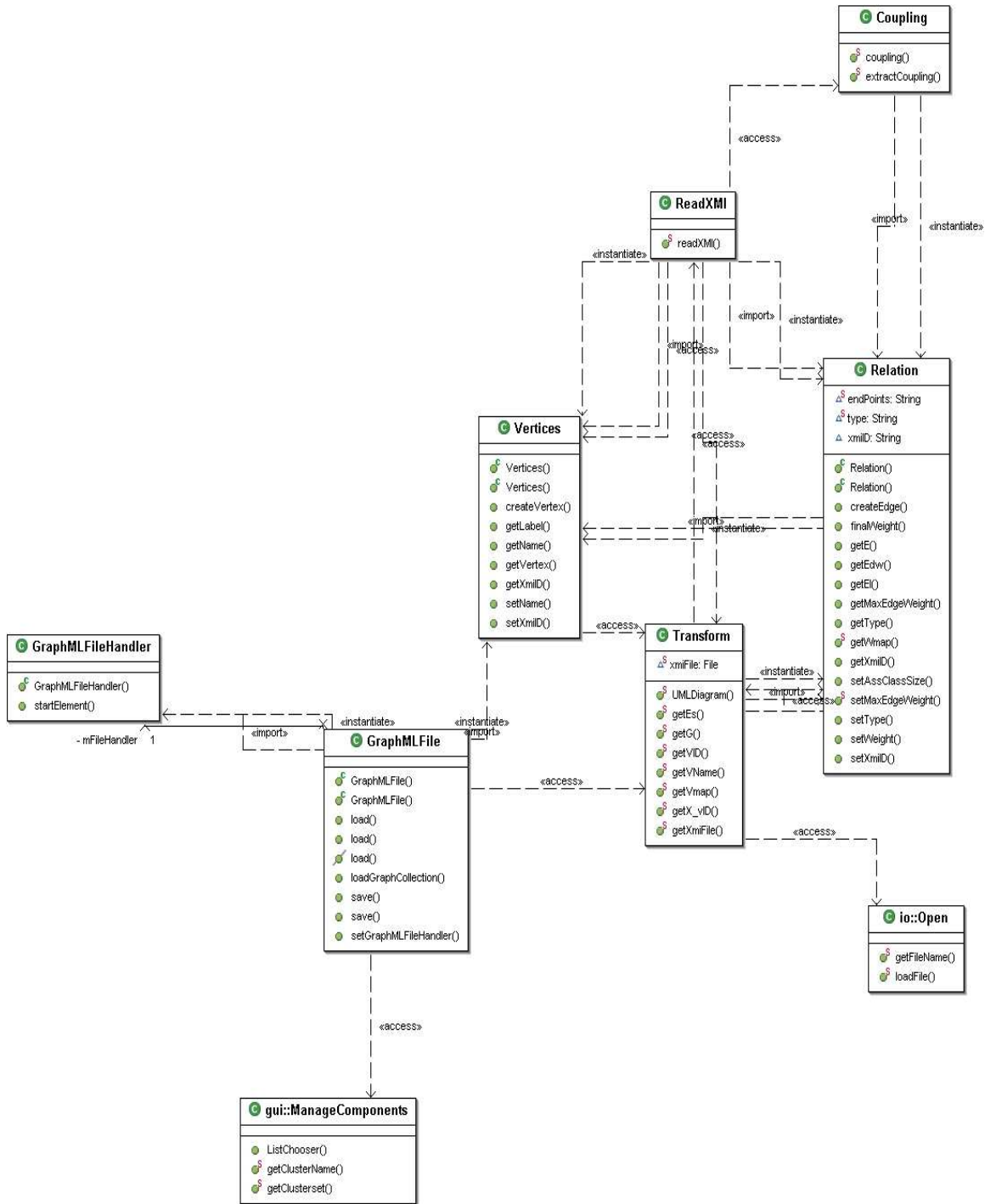


Figure 10: XMI 2 GraphML Package Dependencies Level # 1

#### **4.3.3.1 Transform Class**

Check the correctness of the input file and generate a normalized document that can be parsed.

#### **4.3.3.2 Read XMI Class**

Extract design elements and relations from XMI file and call vertices class, relation class, and coupling class respectively to build a directed weighted graph

#### **4.3.3.3 Vertices Class**

Each design element will be represented as a vertex labeled by the corresponding design element name.

#### **4.3.3.4 Relation Class**

Each kind of relation will be represented as a directed edge, refer to Table 1, the weight of the edge will be calculated in two stages. First, set the importance weight as the edge weight, if multiple relations with the same direction exist, set the summation of their weights. In second stage, multiply this weight with the complexity weight as illustrated in Table 1.

#### **4.3.3.5 Coupling Class**

Extract coupling between design elements, and using relation class to create an edge if not exist and the weight equal to total number of coupling \* importance weight of coupling.

#### **4.3.3.6 GraphML File Class**

A JUNG library class, we edit it to obtain graphML schema we need.

#### **4.3.3.7 GraphML File Handler Class**

A JUNG library class that is used by GraphML File Class.

## 4.4 CLUSTER Package



### 4.4.1 Isolated CLUSTER

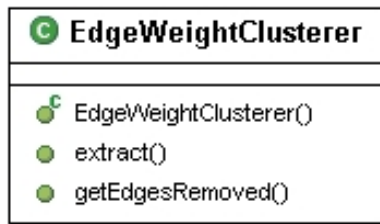


Figure 11: Cluster Package

### 4.4.2 CLUSTER Dependencies

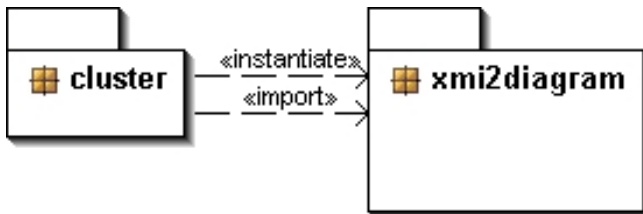


Figure 12: Cluster Package Dependencies Level # 0

### 4.4.3 CLUSTER

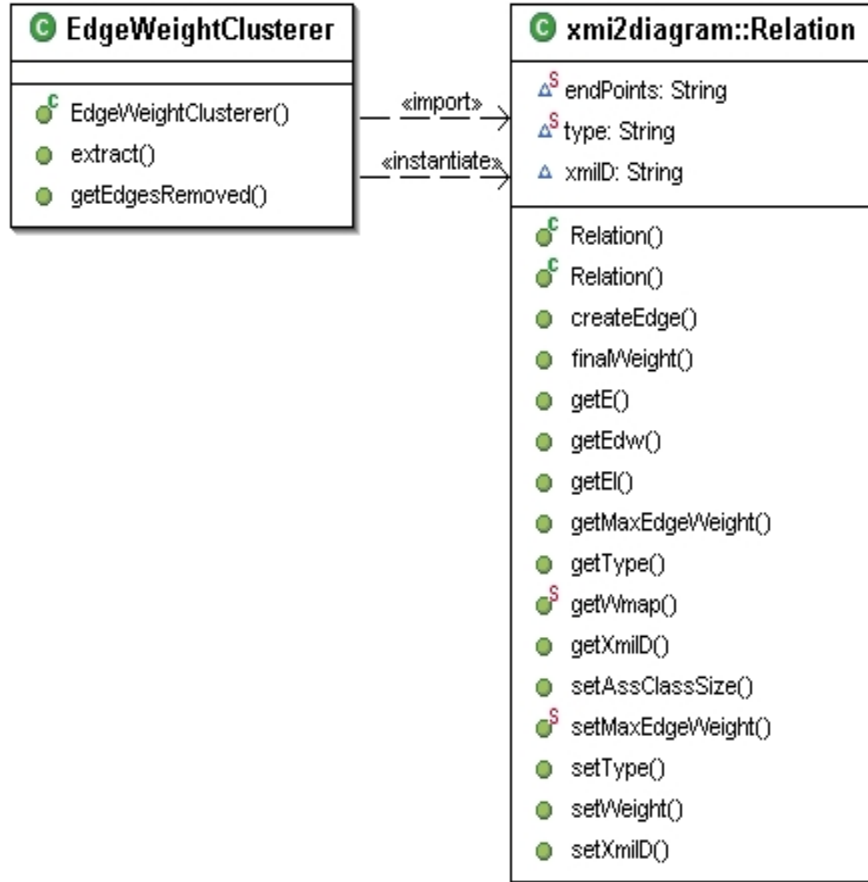


Figure 13: Cluster Package Dependencies Level # 1

#### 4.4.3.1 Edge Weight Cluster Class

Begin with the whole graph as a cluster after specifying a threshold, edges below this threshold will be progressively removed and clusters will be generated.

## 4.5 Graphical User Interface Package



### 4.5.1 Isolated GUI

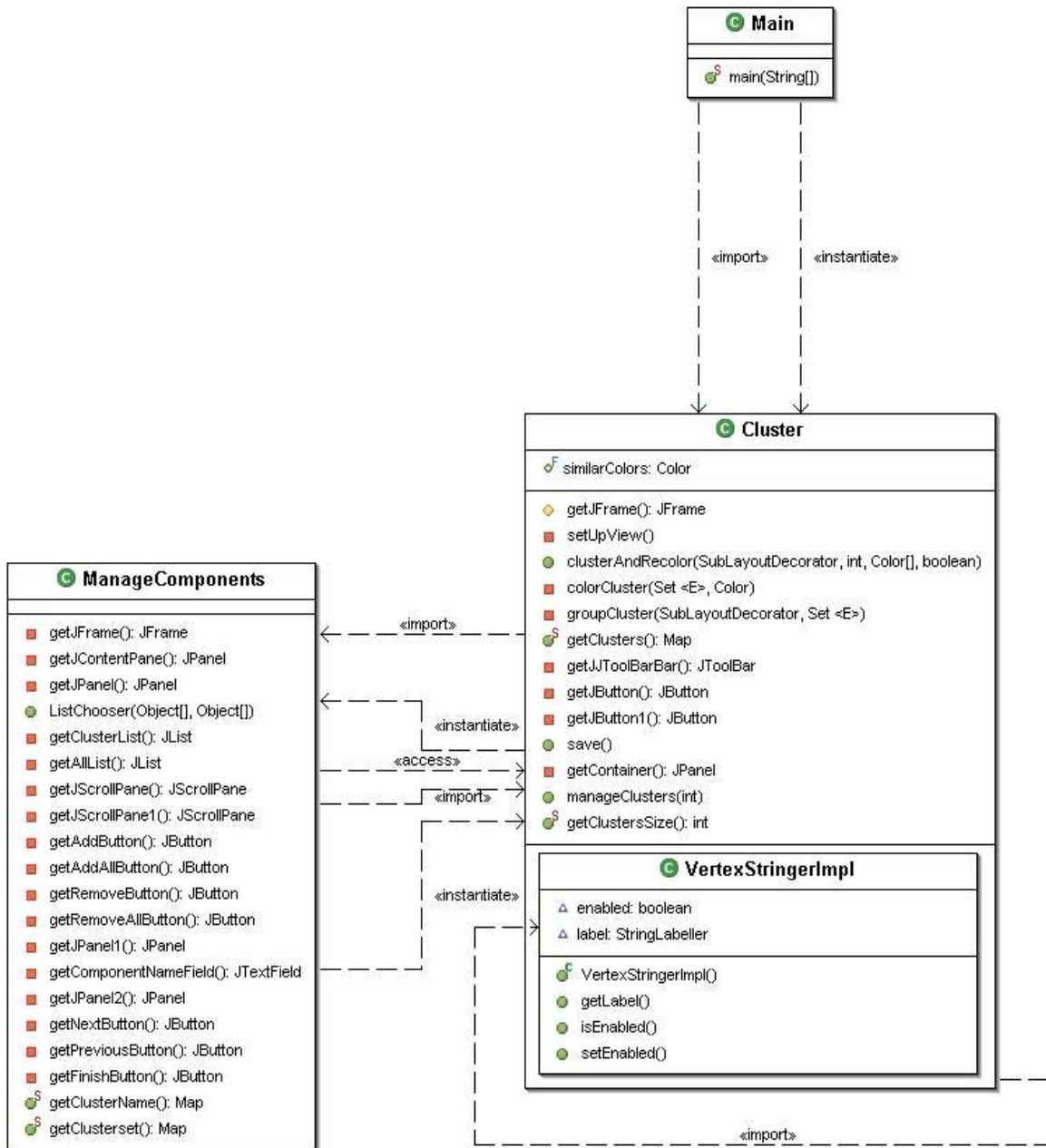


Figure 14: Graphical User Interface Package

## 4.5.2 GUI Package Dependency

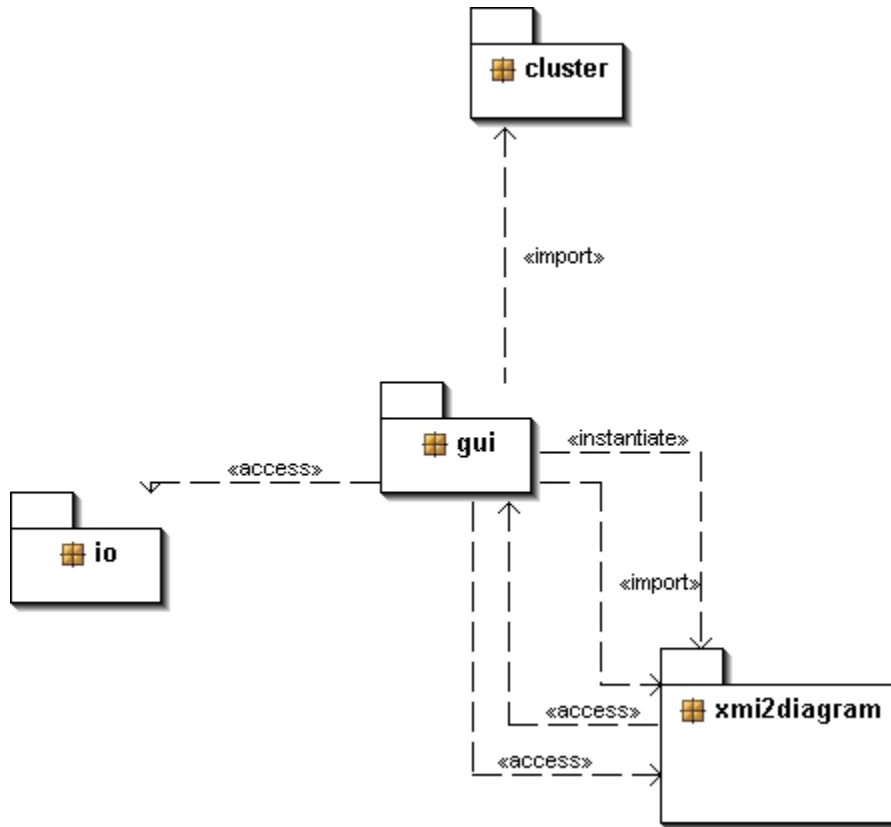


Figure 15: GUI Package Dependencies Level # 0

### 4.5.3 GUI Connections

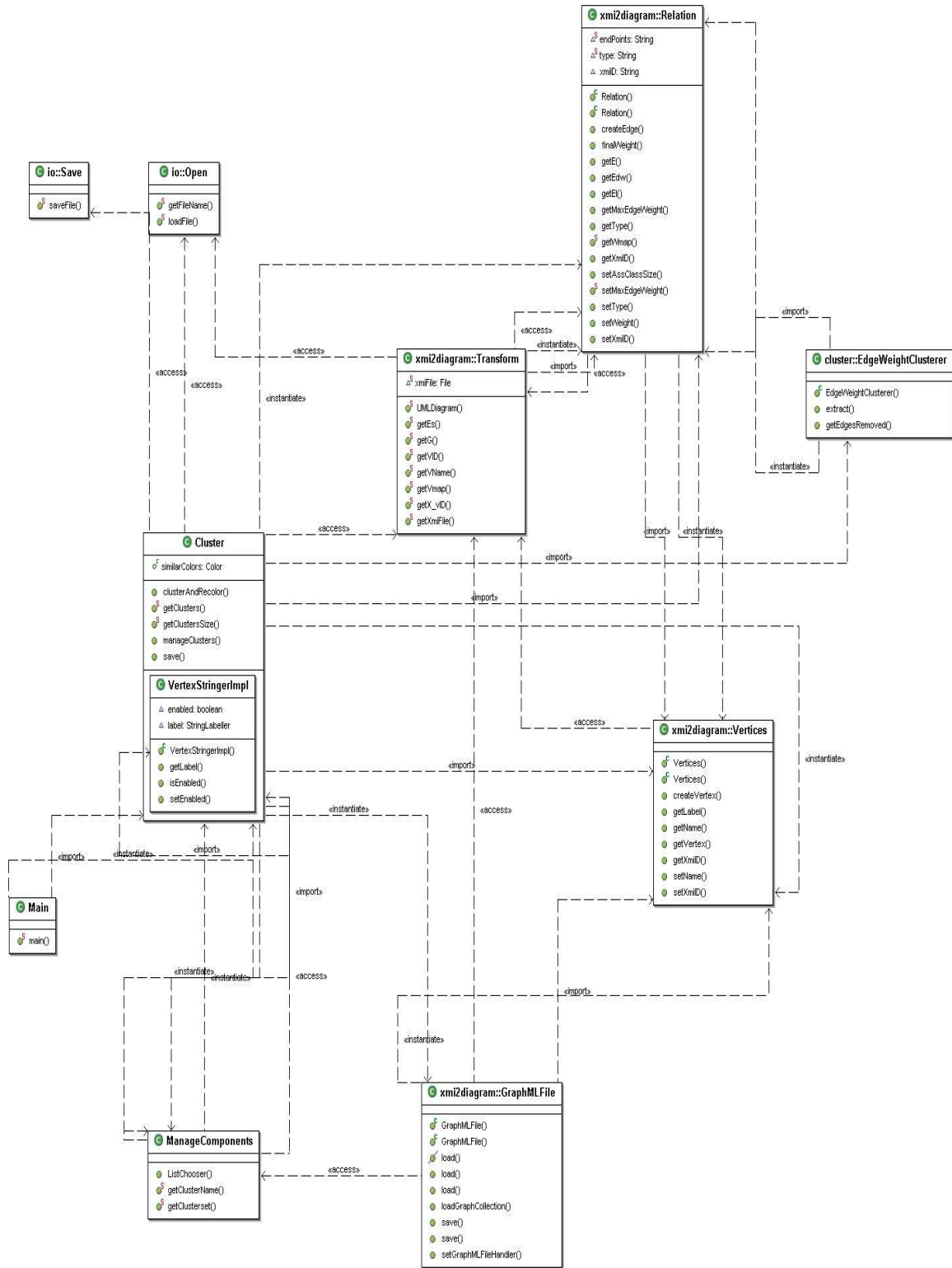


Figure 16: GUI Package Dependencies Level # 1

### **4.5.3.1 Cluster Class**

Main GUI represents the graph, set threshold using slider, represent the cluster graph, and call Manage Components Class preparing to export the new XMI file.

### **4.5.3.2 Manage Components Class**

Dual list manager GUI to name each component and manage them contents.

### **4.5.3.3 Main Class**

Main class is used to launch this tool.



## 4.6 Overall Class Diagram

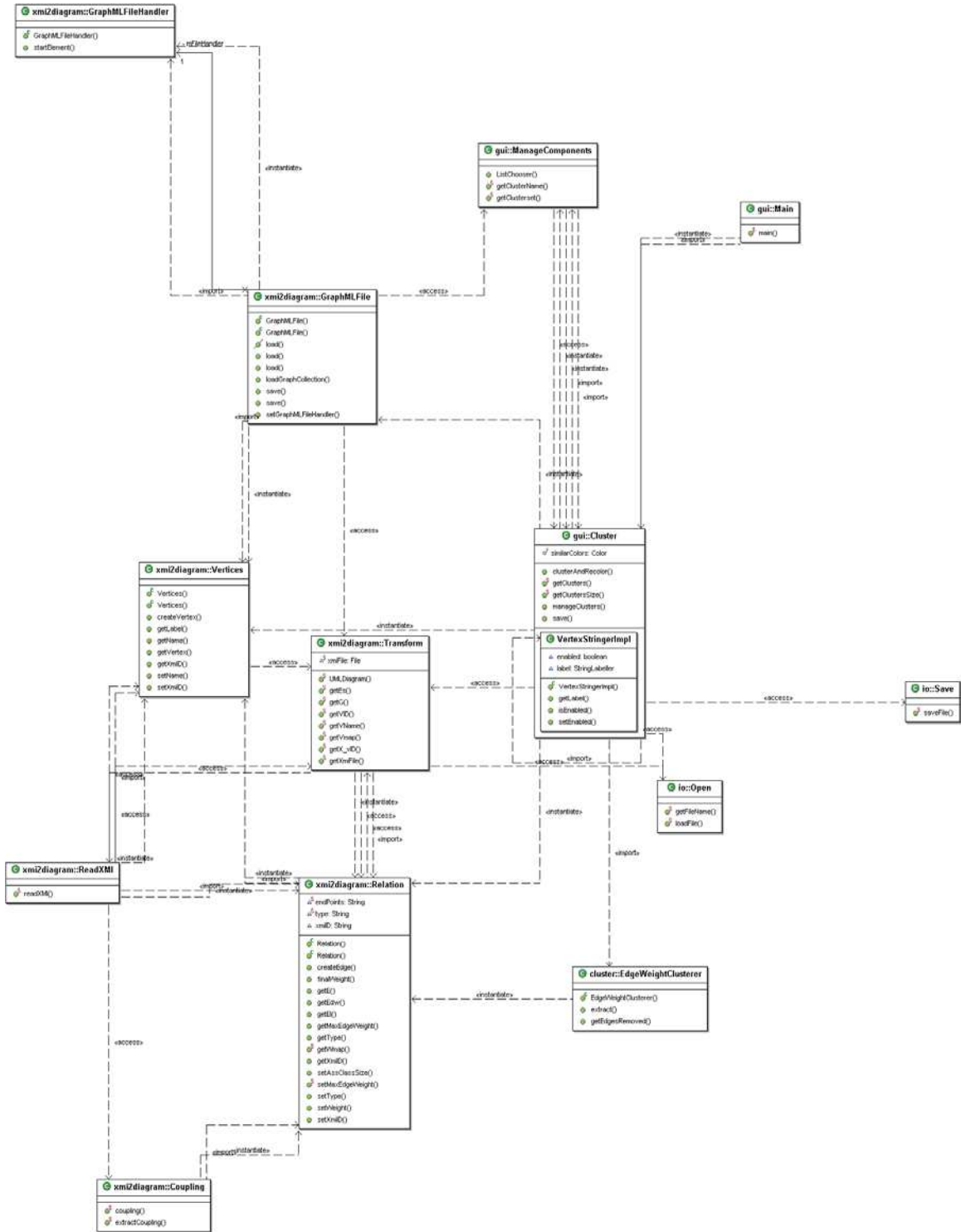
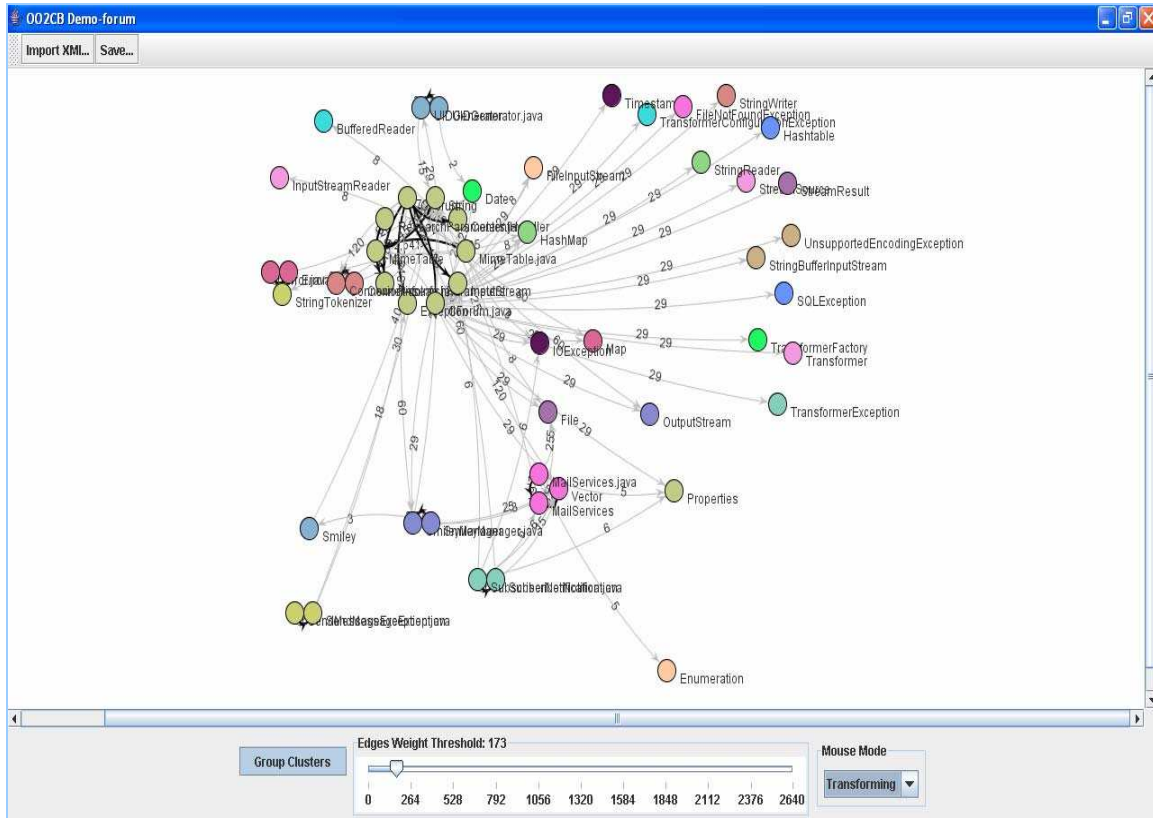


Figure 17: Overall Class Diagram

## 5 Case Study

I arbitrarily choose an open source Forum Software to analyze its transformation in the case study.



**Figure 18: Window#1 – Graph Representation**

Total Number of Nodes = 52

Total Number of Edges = 85

DESIGN ELEMENT TYPE	CLASS	INTERFACE	COMPONENTS	TOTAL
Quantity	39	3	10	52
Internal Design	Attribute	Parameters		
Quantity	95	226		

**Table 2: Analytic of Design Elements**

RELATION TYPE	QUANTITY
Generalization	1
Abstraction	0
Association	0
Association class	0
Dependency	11
Permission	47
Import / Export Coupling Attribute	41
Import / Export Coupling Parameter	99
Total Relations	199
Parallel Edges	114
Total Edges	85

**Table 3: Analytic of Relations**

### Generated GraphML:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphML xmlns="http://graphml.graphdrawing.org/xmlns/graphml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml
file:///C:/Documents%20and%20Settings/Hind/Desktop/GraphMLSchema.
xsd">
    <graph edgedefault="directed"
Edge_Weight_Key="edu.uci.ics.jung.graph.decorators.EdgeWeightLabe
ller@10bbf9e"
Clustering_Weight_Key="edu.uci.ics.jung.graph.decorators.EdgeWeig
htLabeller@513d61"
IndexDefaultKey="edu.uci.ics.jung.graph.decorators.Indexer@19bfb3
0"
StringLabeller.LabelDefaultKey="edu.uci.ics.jung.graph.decorators
.StringLabeller@194d372">
```

```

    <clusters>
      <cluster id="1">
        <node id="21" xmi.id=".:0000000000000890"
name="Date"/>
      </cluster>
      <cluster id="2">
        <node id="41" xmi.id=".:0000000000000898"
name="Hashtable"/>
      </cluster>
      <cluster id="3">
        <node id="9" xmi.id=".:000000000000080E"
name="SubscriberNotification"/>
        <node id="48" xmi.id=".:000000000000080D"
name="SubscriberNotification.java"/>
      </cluster>
      <cluster id="4">
        <node id="30" xmi.id=".:0000000000000862"
name="FileInputStream"/>
      </cluster>
      <cluster id="5">
        <node id="22" xmi.id=".:000000000000088B"
name="Timestamp"/>
      </cluster>
      <cluster id="6">
        <node id="42" xmi.id=".:0000000000000FAD"
name="BufferedReader"/>
      </cluster>
      <cluster id="7">
        <node id="51" xmi.id=".:00000000000008AB"
name="Transformer"/>
      </cluster>
      <cluster id="8">
        <node id="39" xmi.id=".:000000000000083C"
name="SmileyManager.java"/>

```

```

        <node id="8" xmi.id=".:000000000000083D"
name="SmileyManager"/>
    </cluster>
    <cluster id="9">
        <node id="29" xmi.id=".:0000000000000FBD"
name="StringTokenizer"/>
    </cluster>
    <cluster id="10">
        <node id="44" xmi.id=".:00000000000008B7"
name="TransformerFactory"/>
    </cluster>
    <cluster id="11">
        <node id="50" xmi.id=".:000000000000087E"
name="StringWriter"/>
    </cluster>
    <cluster id="12">
        <node id="10" xmi.id=".:0000000000000876"
name="StringBufferInputStream"/>
    </cluster>
    <cluster id="13">
        <node id="28" xmi.id=".:000000000000089C"
name="Map"/>
    </cluster>
    <cluster id="14">
        <node id="47" xmi.id=".:000000000000086A"
name="IOException"/>
    </cluster>
    <cluster id="15">
        <node id="7" xmi.id=".:0000000000000F0D"
name="Enumeration"/>
    </cluster>
    <cluster id="16">
        <node id="1" xmi.id=".:00000000000008C0"
name="StreamSource"/>

```

```

        </cluster>
        <cluster id="17">
            <node id="40" xmi.id=".:000000000000084D"
name="UIDGenerator.java"/>
            <node id="43" xmi.id=".:000000000000084E"
name="UIDGenerator"/>
        </cluster>
        <cluster id="18">
            <node id="19" xmi.id=".:00000000000008AF"
name="TransformerConfigurationException"/>
        </cluster>
        <cluster id="19">
            <node id="33" xmi.id=".:0000000000000872"
name="OutputStream"/>
        </cluster>
        <cluster id="20">
            <node id="32" xmi.id=".:00000000000007F8"
name="Error"/>
            <node id="5" xmi.id=".:00000000000007F7"
name="Error.java"/>
        </cluster>
        <cluster id="21">
            <node id="26" xmi.id=".:0000000000000866"
name="FileNotFoundException"/>
        </cluster>
        <cluster id="22">
            <node id="50" xmi.id=".:000000000000087E"
name="StringWriter"/>
        </cluster>
        <cluster id="23">
            <node id="35" xmi.id=".:0000000000000882"
name="UnsupportedEncodingException"/>
        </cluster>
        <cluster id="24">

```

```

        <node id="12" xmi.id=".:000000000000087A"
name="StringReader"/>
    </cluster>
    <cluster id="25">
        <node id="25" xmi.id=".:0000000000000FB7"
name="InputStreamReader"/>
    </cluster>
    <cluster id="26">
        <node id="4" xmi.id=".:000000000000085B"
name="File"/>
    </cluster>
    <cluster id="27">
        <node id="15" xmi.id=".:0000000000000843"
name="Smiley"/>
    </cluster>
    <cluster id="28">
        <node id="37" xmi.id=".:00000000000008A0"
name="Properties"/>
    </cluster>
    <cluster id="29">
        <node id="24" xmi.id=".:00000000000007EB"
name="ConnexionInfo.java"/>
        <node id="45" xmi.id=".:00000000000007EC"
name="ConnexionInfo"/>
    </cluster>
    <cluster id="30">
        <node id="2" xmi.id=".:00000000000008A4"
name="Vector"/>
        <node id="14" xmi.id=".:000000000000081A"
name="MailServices"/>
        <node id="3" xmi.id=".:0000000000000818"
name="MailServices.java"/>
    </cluster>
    <cluster id="31">

```

```

        <node id="16" xmi.id=".:0000000000000887"
name="SQLException"/>
    </cluster>
    <cluster id="32">
        <node id="38" xmi.id=".:0000000000000894"
name="HashMap"/>
    </cluster>
    <cluster id="33">
        <node id="27" xmi.id=".:00000000000008BC"
name="StreamResult"/>
    </cluster>
    <cluster id="34">
        <node id="34" xmi.id=".:00000000000008B3"
name="TransformerException"/>
    </cluster>
    <cluster id="35">
        <node id="51" xmi.id=".:00000000000008AB"
name="Transformer"/>
    </cluster>
    <cluster id="36">
        <node id="13" xmi.id=".:000000000000082F"
name="MimeTable.java"/>
        <node id="46" xmi.id=".:000000000000086E"
name="InputStream"/>
        <node id="31" xmi.id=".:00000000000007CD"
name="CForum.java"/>
        <node id="23" xmi.id=".:0000000000000C0F"
name="Exception"/>
        <node id="20" xmi.id=".:0000000000000803"
name="ResearchParameters"/>
        <node id="36" xmi.id=".:0000000000000831"
name="MimeTable"/>
        <node id="17" xmi.id=".:0000000000000802"
name="ResearchParameters.java"/>

```



```

        <node id="18" xmi.id=".:00000000000007D7"
name="CForum"/>
        <node id="52" xmi.id=".:00000000000008F1"
name="String"/>
        <node id="11" xmi.id=".:00000000000008CE"
name="ContentHandler"/>
    </cluster>
    <cluster id="37">
        <node id="49" xmi.id=".:0000000000000825"
name="SendMessageException"/>
        <node id="6" xmi.id=".:0000000000000824"
name="SendMessageException.java"/>
    </cluster>
</clusters>
<edges>
    <edge source="24" target="21" directed="true"
RelType="Dependency" edge_weight="2.0" isRemoved="True"/>
    <edge source="31" target="50" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="14" target="2" directed="true"
RelType="Coupling" edge_weight="200.0" isRemoved="False"/>
    <edge source="18" target="52" directed="true"
RelType="Coupling" edge_weight="780.0" isRemoved="False"/>
    <edge source="18" target="23" directed="true"
RelType="Coupling" edge_weight="2640.0" isRemoved="False"/>
    <edge source="40" target="21" directed="true"
RelType="Dependency" edge_weight="2.0" isRemoved="True"/>
    <edge source="18" target="28" directed="true"
RelType="Coupling" edge_weight="60.0" isRemoved="True"/>
    <edge source="24" target="45" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
    <edge source="13" target="38" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>

```

```
<edge source="48" target="14" directed="true"
RelType="Dependency" edge_weight="6.0" isRemoved="True"/>
<edge source="18" target="2" directed="true"
RelType="Coupling" edge_weight="120.0" isRemoved="True"/>
<edge source="32" target="52" directed="true"
RelType="Coupling" edge_weight="20.0" isRemoved="True"/>
<edge source="48" target="2" directed="true"
RelType="Dependency" edge_weight="6.0" isRemoved="True"/>
<edge source="31" target="16" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
<edge source="13" target="4" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>
<edge source="31" target="11" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
<edge source="18" target="36" directed="true"
RelType="Coupling" edge_weight="60.0" isRemoved="True"/>
<edge source="13" target="36" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
<edge source="3" target="7" directed="true"
RelType="Dependency" edge_weight="5.0" isRemoved="True"/>
<edge source="20" target="52" directed="true"
RelType="Coupling" edge_weight="15.0" isRemoved="True"/>
<edge source="14" target="52" directed="true"
RelType="Coupling" edge_weight="125.0" isRemoved="True"/>
<edge source="39" target="8" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
<edge source="36" target="38" directed="true"
RelType="Coupling" edge_weight="25.0" isRemoved="True"/>
<edge source="31" target="36" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
<edge source="15" target="52" directed="true"
RelType="Coupling" edge_weight="40.0" isRemoved="True"/>
<edge source="9" target="2" directed="true"
RelType="Coupling" edge_weight="15.0" isRemoved="True"/>
```

```
        <edge source="13" target="30" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>
        <edge source="31" target="22" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="31" target="44" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="31" target="26" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="13" target="29" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>
        <edge source="36" target="52" directed="true"
RelType="Coupling" edge_weight="525.0" isRemoved="False"/>
        <edge source="48" target="47" directed="true"
RelType="Dependency" edge_weight="6.0" isRemoved="True"/>
        <edge source="31" target="41" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="31" target="51" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="13" target="42" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>
        <edge source="18" target="8" directed="true"
RelType="Coupling" edge_weight="60.0" isRemoved="True"/>
        <edge source="31" target="46" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="31" target="47" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="8" target="2" directed="true"
RelType="Coupling" edge_weight="25.0" isRemoved="True"/>
        <edge source="31" target="18" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
        <edge source="3" target="2" directed="true"
RelType="Dependency" edge_weight="5.0" isRemoved="True"/>
        <edge source="31" target="30" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
```

```
    <edge source="17" target="20" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
    <edge source="31" target="34" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="48" target="37" directed="true"
RelType="Dependency" edge_weight="6.0" isRemoved="True"/>
    <edge source="31" target="2" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="35" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="3" target="37" directed="true"
RelType="Dependency" edge_weight="5.0" isRemoved="True"/>
    <edge source="5" target="32" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
    <edge source="3" target="4" directed="true"
RelType="Dependency" edge_weight="5.0" isRemoved="True"/>
    <edge source="18" target="20" directed="true"
RelType="Coupling" edge_weight="180.0" isRemoved="False"/>
    <edge source="6" target="49" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
    <edge source="31" target="1" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="28" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="18" target="33" directed="true"
RelType="Coupling" edge_weight="60.0" isRemoved="True"/>
    <edge source="40" target="43" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
    <edge source="13" target="25" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>
    <edge source="31" target="33" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="12" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
```

```
<edge source="31" target="8" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="27" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="43" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="18" target="11" directed="true"
RelType="Coupling" edge_weight="780.0" isRemoved="False"/>
    <edge source="9" target="52" directed="true"
RelType="Coupling" edge_weight="60.0" isRemoved="True"/>
    <edge source="31" target="10" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="37" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="21" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="31" target="4" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="49" target="52" directed="true"
RelType="Coupling" edge_weight="30.0" isRemoved="True"/>
    <edge source="48" target="46" directed="true"
RelType="Dependency" edge_weight="6.0" isRemoved="True"/>
    <edge source="45" target="52" directed="true"
RelType="Coupling" edge_weight="45.0" isRemoved="True"/>
    <edge source="43" target="52" directed="true"
RelType="Coupling" edge_weight="15.0" isRemoved="True"/>
    <edge source="31" target="19" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
    <edge source="8" target="52" directed="true"
RelType="Coupling" edge_weight="25.0" isRemoved="True"/>
    <edge source="3" target="14" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
    <edge source="48" target="9" directed="true"
RelType="Dependency" edge_weight="2641.0" isRemoved="False"/>
```

```

        <edge source="39" target="2" directed="true"
RelType="Dependency" edge_weight="3.0" isRemoved="True"/>
        <edge source="13" target="47" directed="true"
RelType="Dependency" edge_weight="8.0" isRemoved="True"/>
        <edge source="14" target="4" directed="true"
RelType="Coupling" edge_weight="25.0" isRemoved="True"/>
        <edge source="18" target="46" directed="true"
RelType="Coupling" edge_weight="180.0" isRemoved="False"/>
        <edge source="18" target="45" directed="true"
RelType="Coupling" edge_weight="120.0" isRemoved="True"/>
        <edge source="39" target="15" directed="true"
RelType="Dependency" edge_weight="3.0" isRemoved="True"/>
        <edge source="31" target="38" directed="true"
RelType="Dependency" edge_weight="29.0" isRemoved="True"/>
        <edge source="49" target="23" directed="true"
RelType="Genaralization" edge_weight="18.0" isRemoved="True"/>
    </edges>
</graph>
</graphml>

```

**Figure 19: Forum Case Study GraphML File**

WEIGHTS AVERAGE	404.2771
Weights Standard Deviation	889.2613

**Table 4: Analytic of Relations**

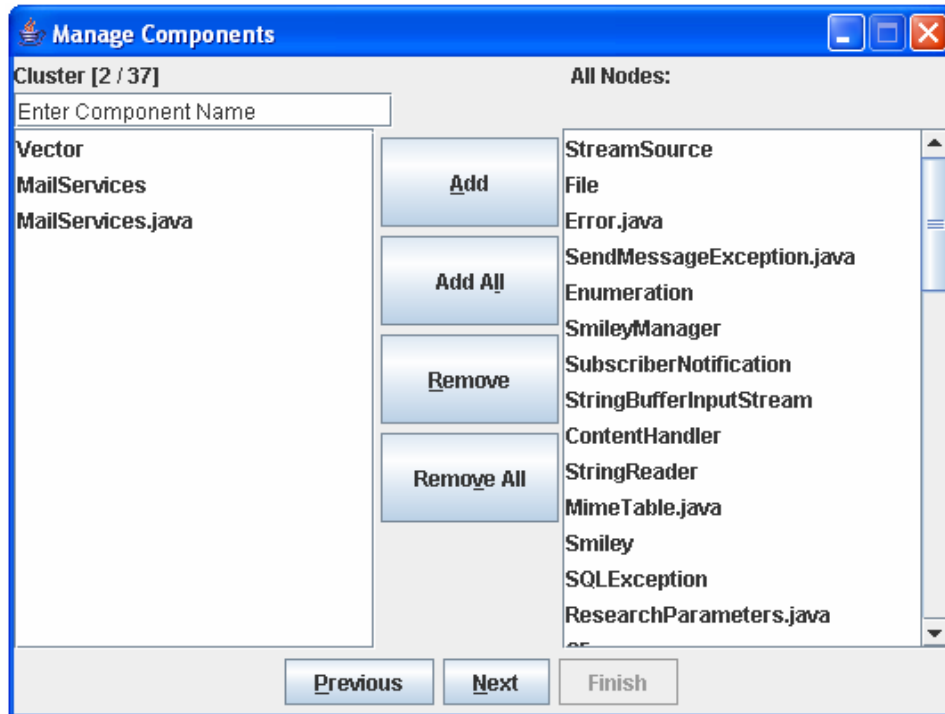
Highest Couple design models = 2640 between CForum & Exception, the second highly coupled is when edge weight = 780 there were 2 edges has this weight and the most frequent weight was 29 it replicated for 26 times.

6 Components remain as they were.

29 Components contains one design models

2 Components were added

To refine the results user can manually manage the components; naming them and also add or remove some of its constituents.



**Figure 20: Window#2 – Components Manager**

## **6 Conclusion and Future Work**

In today's fast paced world, there is a huge demand for renovating object-oriented systems into component-based systems. Renovation of object-oriented systems into component-based systems decreases the complexity of the systems through lowering the number of system constituent. This is achieved by grouping them into individual components that provide a well defined system feature. Decreasing the complexity, in turn, leads to improving the understandability of the system, reducing maintenance cost, enhancing the ability to modify or facilitating the evolution of the system.

This paper demonstrated the generation of component-based system from an object-oriented design using UML class diagrams. It was achieved by developing a system recovery tool that converts an object-oriented structural design into component-based software. This tool was tested on java programming and it was also ascertained that it can be used for any programming language.

This system tool could be enhanced in the future by improving the weights for relations. One way to do this could be by using neural networks to specify these weights rather than deriving them empirically.



## References

- [1] Baudry, B., L. Traon, Y., G. Sunye, “Testability Analysis of a UML Class Diagram”, IEEE, 2002, Ottawa, Canada, pp. 54-65.
- [2] Brandes U., C. Pich, “GraphML Transformation”, Springer, 2004, pp. 89-99.
- [3] Burd E.L., M. Munro, “Enriching Program Comprehension for Software Reuse”, Proceeding of Fifth Int’l Workshop Program Comprehension, pp.130-137, 1997.
- [4] Cheng D., R. Kannan, S. Vempala, G. Wang, “A Divide-and-Merge Methodology for Clustering”, PODS, 2005.
- [5] Chiricota, Y.; F. Jourdan, and G. Melancon, “Software components capture using graph clustering”, Proceedings of 11th IEEE International Workshop on Program Comprehension, 10-11 May 2003, pp. 217- 226.
- [6] Chitnis, M., P. Tiwari, L. Ananthamrthy, “The UML Class Diagram: Part1”, 2003.
- [7] Dagon S., “A Cluster Algorithm for Graphs”, CWI, 2000.
- [8] Deursen, A., B. Elsinga, P. Klint, and R. Tolido. “From Legacy to Component: Software Renovation in Three Steps”, CAP Gemini. Institute, CWI, 2000.
- [9] Ding C., X. He, “Cluster merging and splitting in hierarchical clustering algorithms”, IEEE ICDM, 2002, pp. 139- 146.
- [10] Ding C., X. He, H. Zha, M. Gu, H. Simon, “A Min-max Cut Algorithm for Graph Partitioning and Data Clustering.” IEEE 1<sup>st</sup> Conference on Data Mining, 2001, pp. 107 –114.
- [11] Errickson-Connor B., “Truth or consequences: legacy application modernization”, Business Integration Journal, 2003.
- [12] G. Marcela, M. Piattini, C. Calero, “Empirical Validation of Class Diagram Metrics”, IEEE, 2002. pp. 195-203.

- [13] Genero, M., M. Piattini, C. Calero, "A Survey of Metrics for UML Class Diagrams", JOT, 2005.
- [14] Hitz M., B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems", Proc. Int'l Symp. Applied Corporate Computing, Monterrey, Mexico, 1995.
- [15] Jain H., N. Chalimeda, N. Ivaturi, B. Reddy, "Business Component Identification - A Formal Approach", Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing, p.183, September 04-07, 2001.
- [16] Kang, D., B. Xu, J. Lu, W. Chu, "A Complexity Measure for Ontology Based on UML", IEEE, 2004, pp. 222-228.
- [17] Koschke R., "Atomic Architectural Component Recovery for Program Understanding and Evolution," Proceedings of the International Conference on Software Maintenance, October 2002.
- [18] Lee E., B. Lee, W. Shin, C. Wu, "A Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System", IEEE, 2003.
- [19] Lee J., S. Jung, S. Kim, W. Hyun, D. Ham, "Component Identification Method with Coupling and Cohesion", IEEE, 2001.
- [20] Li B., "Managing Dependencies in Component-Based Systems Based On Matrix Model", Proc. Of Net.Object.Days 2003, 22-25, Sept. 2003, Erfurt, Germany.
- [21] Luo J., R. Jiang, L. Zhang, H. Mei, J. Sun, "An Experimental Study of Two Graph Analysis Based Component Capture Methods for Object-Oriented Systems". IEEE, 2004, pp. 390-398.
- [22] Manso M., M. Genero, M. Piattini, "No-redundant Metrics for UML Class Diagram Structural Complexity", CAiSE, 2003, pp. 127-142.
- [23] Mehta A, and GT. Heineman, "Evolving legacy system features into fine-grained components", ACM, 2002, pp. 417-427.
- [24] MOF 2.0/XMI Mapping Specification, v2.1, OMG, 2005.

- [25] Ncube, C., and N. Maiden. "PORE: Procurement – Oriented Requirement Engineering Method for the Component-Based Systems Engineering Development Paradigm", International Workshop on Component-Based Software Engineering, IEEE, 1999.
- [26] Neville J., M. Alder, D. Jensen, "Clustering Relational Data Using Attribute and Link Information", in Proceedings of the Text Mining and Link Analysis Workshop, 18th International Joint Conference on Artificial Intelligence, 2003.
- [27] Ovlinger J., K. Lieberherr, "Class Graph Views", Northeastern University, 1998.
- [28] Sneed H., "Extracting Business Logic from existing COBOL programs as a basis for Redevelopment", IEEE, 2001.
- [29] Tansalarak N., K.T. Claypool, "CGC: An Architecture to support Better and Faster Component Evolution", Second International Workshop on Unanticipated Software Evolution, Warsaw, Poland, 2003.
- [30] Tzerpos V., R. C. Holt, "Software Botryology Automatic Clustering of Software Systems"
- [31] Washizaki H., H. Yamamoto, Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components", Software Metrics Symposium, 2003. Proceedings. Ninth International (2003), pp. 211-223.
- [32] Yi T., F. Wu , C. Gan, "A comparison of metrics for UML class diagrams", ACM SIGSOFT Software Engineering Notes, v.29 n.5, 2004.

## **Books**

- [33] Cheesman J., and J. Daniels, "UML Components: A Simple Process for Specifying Component-Based Software", Addison-Wesley, 2001.
- [34] Heineman G., W. Council, "Component-Based Software Engineering: Putting the Pieces Together", Addison Wesley, 2001.
- [35] Kay M., "XSLT 2.0 Programmer's Reference", Wrox, 3rd edition, 2004.

- [36] Larman C., “Applying UML and Patterns: an Introduction to Object-Oriented Analysis and Design and Iterative Development”, 3rd Ed., Prentice Hall, 2005.
- [37] Sommerville J., “Software Engineering”, 6th Ed., Addison Wesley, 2001.
- [38] Szyperski C., D. Gruntz, and S. Murer, “Component Software: Beyond Object-Oriented Programming”, Addison Wesley Professional, 2nd edition, November 2002.
- [39] Tennison J., “Beginning XSLT”, Wrox Press, Chicago, Illinois, May 2002.

## **Web Sites**

- [40] <http://graphml.graphdrawing.org/>
- [41] <http://jung.sourceforge.net/>
- [42] <http://www.omg.org/technology/documents/formal/xmi.htm>
- [43] <http://www.sdmetrics.com>
- [44] <http://www.uml.org>
- [45] <http://www.w3.org/TR/xslt>