IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-8 E-ISSN: 2347-2693

Computation of External view based Software Metrics: Java Based Tool

P.L. Powar[1*], M.P. Singh[2], Bharat Solanki[3], Jawwad Wasat Shareef[4]

[1*]Dept. of Mathematics and Computer Science, R.D University, Jabalpur, India
[2]Dept. of Computer Science, Dr. B. R. Ambedkar University Agra, Agra, India
[3] Dept. of Mathematics and Computer Science, R.D University, Jabalpur, India
[4]Dept. of Mathematics and Computer Science, R.D. University, Jabalpur, India

Corresponding Author: bharat.jbp@gmail.com

Available online at: www.ijcseonline.org

Received: 19/Jul/2017, Revised: 27/Jul/2017, Accepted: 17/Aug/2017, Published: 30/Aug/2017

Abstract: Component based software development (CBSD) strategies have been found to be boon for software development companies. The contribution of past researches in the area of software cost estimation of component based software development was excellent. It has been noticed that the use of the components in software development is hierarchical and this multilevel implementation increases the complexity of the software development. If the depth of the components in hierarchy and their association and aggregation with each other are available in advance then this information helps in estimating the cost and complexity of the software in early stage. The present paper compute the different metrics values during the design phase of the entire life cycle of software development. A component diagram consisting of the various components and their associations has been prepared using ArgoUML software tool. Considering a case of E-learning, our technique has been implemented on this special case, to compute various metrics for analyzing certain results of the software at designing stage which turns out to be the important information to control the development cost and the complexity of the software in advance. By computing external view based metrics we can assess effort estimation and complexity of CBSE at early stage.

Keywords: Software Metrics, Static metrics, dynamic metrics, External view, Component based software.

I. INTRODUCTION

In Component based software engineering (CBSE), component is an independent and replaceable part of a system that performs a clear function in the context of a well defined architecture. This option of CBSE results in better productivity, improved quality, reduction in time spent and cost to develop. Metrics used in component based software engineering are helpful in achieving the quality and managing risk in component based system by checking the factors that affect risk and quality. Metrics help the developer in identifying the probable risks so that proper corrective action can be taken beforehand. Various metrics [1] have been proposed to measure the different attributes of a component like functionality, interactivity, complexity, reusability etc.

A formal definition of software component has been given in [2] which states that "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party". In view of [3] component based software development is a process of connecting together the separate component parts in order to meet the requirement of the problem. Most of the research is inclined to improve the methods of software development process. However, a less attention has been given to evaluate components by using metrics.

Considering the metrics for both individual components and their assembly between the components, the brief survey of the traditional software metrics has been described [4] with the formal definition of direct and indirect component coupling metric. The complexity metrics presented in [5] provides formal specifications of software metrics and novel quantitative software measures by combining coupling, cohesion and interface metrics on the levels of the component and component based software system (CBSS). By using the effective software metrics[5] the investigation has been done regarding improved measurement tools and techniques.

Component based software development (CBSD) involves composing software system from existing software rather than building from the scratch. This principle embodies an element of "buy, don't build" that shifts the emphasis from programming software to composing software systems [6]. CBSD provides many advantages like reduced development time, effort, and increased quality. The quality of the

application by using CBSD is tremendous when it returns the expected results, is stable and adaptable and leads to reduce maintenance costs.

The paper is organized as follows:

Section 1 describes the available technology and methods in processing CBSE and their utilization. Section 2 covers the details of the various metrics according to their different views used in component based software. In order to use metrics for processing of CBSE, we have referred the techniques described in [7]. Existing work by various researchers have been discussed in Section 3. Research problem of the present paper has been described in Section 4. Section 5 provides the proposed methodology along with proposed algorithm. Section 6 provides application of tool with the help of case study. The discussions on the results are given in section 7. Conclusion of the work has been discussed in Section 8.

II. COMPONENT VIEWS AND METRICS

The two fundamental views of components which are as follows:

- a black-box view
- a white-box view

The specification is the black box view which describes all externally visible properties while the realization is the white box view which describes the internal realization of the component. It has been mentioned [7] that while studying the fundamental views of components many of researchers described the externally visible properties of the component (i.e. its interface in a general sense) initially. Later on some of the researchers extended their studies over the internal realization of the component. However, the classification of "a black box" and "a white box" is not enough to provide information regarding "realization" that belongs to the component itself and "realization" that belongs to its nested and external components. In order to resolve this problem the four different view [7] of component are:

- External view
- Shallow view
- Deep view
- Complete view

To explore the detailed study of above stated four views of component diagram, the Table 1 shows the corresponding different metrics for external view [7]. Similarly, the authors[7] have defined the same metric for the remaining three views. In the present paper, authors have considered the external view and computed its metrics. As its name implies, the external view shows all the externally visible properties of a component including the interfaces of its acquired components.

III. EXISTING WORK

A set of metrics that offers useful and simple results for the component selection process has been defined in [8] in which the authors have presented a collection of software component metrics focused on a main quality characteristic—the usability—of great importance to any software product. Moreover the usability of metrics for software components based on ISO 9126 Quality Model has been defined in a consistent way in [8].

Recently authors describes the evaluation of number of component cycles in DSME tool that helps in deciding the super components has been described in [9]. Similarly utilization of components in a module helps in deciding most usable components in the system and hence importance of the components during the implementation of software system. Finally, it may be concluded that the Number of cycles and degree of utilization is one of the key component for the cost estimation of CBSE. Hence, by computing NC and degree of utilization, basically we would be in a position to predict the approximate cost of CBSE.

The concept of two suites of metrics, which cover static and dynamic aspects of component assembly has been initiated in [10, 11]. The static metrics measure complexity and criticality of component assembly, wherein complexity is measured using Component Packing Density and Component Interaction Density metrics. Further, four criticality conditions namely, Link, Bridge, Inheritance and Size criticalities have been identified and quantified. The complexity and criticality metrics are combined to form a Triangular Metric, which can be used to classify the type and nature of applications. Dynamic metrics are collected during the runtime of a complete application. Dynamic metrics are useful to identify super-component and to evaluate the degree of utilization of various components [12, 13].

IV. RESEARCH PROBLEM

Component Based Software Engineering is the widely used concept in the software industry. Metrics play an important role in determining the various characteristics of a component for example to find out which components are reusable and what particular function they will perform. Software requires to be evaluated before the development to avoid the wastage of resources and also requires evaluation during their life cycle to manage the software maintenance cost. Different software evaluation strategies with software metric measurement and effort estimation models have been introduced [6, 10]. The four views along with metrics [7] theoretically which have been described in section 2. In this paper, we have considered external view along with metrics proposed in[7] and implemented practically with the help of Java based parser tool.

In the present work, a study has been made on how quantification metrics are used in component based development that concentrates on the factors like Number of provided interface (NPI), Number of Acquired Interface (NAI), Number of Acquired Components (NAC), Deepest Provided Interface Nesting (DPIN) and Deepest Acquired Interface Nesting (DAIN). With the help of NAI, NAC, DPIN and DAIN software developer can easily find and predict the different complexities and architecture of CBSE at early stage i.e. at design stage.

V. PROPOSED METHODOLOGY

Evaluation of the software has been done by using quantification metrics and it can be visualized before software development using component diagram in ArgoUML. Such mapping and testing of the different metric values is a major challenge which has been taken into consideration in this paper. Java based parser tool developed in this paper has been implemented on the E-learning system. The proposed methodology is given as follows:

- **Step1:**Design the component diagram of any proposed software using ArgoUML tool as per need of customers.
- **Step 2:** Create XMI file of given component diagram with the help of Export option XMI given in ArgoUML. This XMI file contains all the information of component diagram like unique xmi.id, dependency.supplier, dependency.client etc.
- **Step 3:** Using Java based software and Netbeans tool the XMI file is then parsed using Java parser for extracting information related to various quantification metrics like NPI, NAI, NAC, DPIN in CBSE.

For Calculating NPI, NAI, NAC, and DPIN of components in CBSE, the following algorithms have been used:

A. Algorithm for calculating provided interface:

```
Algorithm getClientComponentId (xmlId: String)
Begin
    id:=0;
    For i=0 to sizeof(ComponentList) loop
        If ComponentList.get(i). xmlId = xmlId Then
            id: = i;
            Exit Loop;
        End If
    End Loop
    Return id;
End;
```

```
Algorithm getSupplierComponentId (xmlId: String)
Begin
    id:=0;
    For i=0 to sizeof(ComponentList) loop
        If ComponentList.get(i). xmlId = xmlId  Then
            id: = i;
            Exit Loop;
        End If
    End Loop
    Return id;
End;
```

```
Algorithm ProvidedInterface()
Begin
    ForEach D in DependancyList Loop
        IC:=0;
        For Each C in ComponentList loop
            If D.RefId = C.Id Then
                    IC := IC + 1;
            End If
        End Loop
    End Loop
    Return IC;
End;
```

B Algorithm for calculating Acquired interface:

```
Algorithm AcquiredInterface()
Begin
    count:=0;
    For Each D in DependancyList Loop
        cXmlRefId := D.getClientRefId();
        sXmlRefId := D.getSupplierRefId();
        ccId := getClientComponentId(cXmlRefId);
        scId := getSupplierComponentId (sXmlRefId);
        int cDepth:=ComponentList.getDepth(ccId);
        int sDepth:=ComponentList.getDepth(ccId);
        if(sDepth==0 && cDepth>0) {
            count:=count+1;
        }
    End Loop
    Return count;
End;
```

C Algorithm for calculating Acquired components:

```
Algorithm showAcquiredComponents ()
Begin
    count:=0;
    For Each C in ComponentList loop
        If C.IsRoot = "true" Then
            count:=count+1;
        End Loop
    End Loop
    Return count;
End;
```

D. Algorithm for calculating Deepest Provided Interface Nesting:

```
Algorithm showDeepestProvidedInterfaceNesting ()
Begin
    depth:=0;
    For Each C in ComponentList loop
        If depth < C.getDepth Then
```



```
            Depth := C.getDepth;
        End Loop
    End Loop
    Return depth;
End;
```

VI. IMLEMENTATION OF JAVA BASED TOOL FOR EXTRACTION OF EXTERNAL METRICS

In this section, we consider the model of E-learning system which has been designed with the help of Argo UML 0.34 (UML Modelling tool). Our aim is to implement the Java based tool on the component diagram of the E-learning system (cf. Fig. 1). A parser based tool developed in Java by using Netbeans 7.1.2 has been explored to implement Java based tool on component diagram to compute quantification metrics of external views. Using ArgoUML tool, a UML component diagram has been drawn. This tool works only with XMI files. For parsing the XMI file, SAX [14] – a Java API for XML to parse the XMI file is used. The version implemented in the Java based tool is SAX 2.0.1 as the SAX parser is an easy-to-use forward parser. The flow of process of how the Java based tool works is depicted in Figure 2.

Figure 1 shows the component diagram of E-learning system. The external view of Teacher essentially includes:

- All the information in the offered interfaces.
- All the information in the offered interfaces of the acquired components.

The diagram consists of components, interfaces and a Dependency indicator. The eight components of the system (cf. Figure 1) are as follows :

(i) User Management
(ii) Teacher
(iii) Course Management
(iv) Study Material Management
(v) Messaging
(vi) Report Generation
(vii) Authorization System
(viii) Key Authentication

Teacher (cf. (ii)) is the largest component. Teacher uses two external components, (vii) and (viii), and four internal components, (iii), (iv), (v) and (vi). The internal components (iii), (iv), (v) and (vi) are referred to as sub-components of (ii), while (vii) and (viii) are referred to as acquired-components of (ii). The component (iii) uses component (vii) and (viii). Thus, (vii) and (viii) are acquired components of (iii) as well as (ii), which is the super-component of the Course management. Similarly, we may assess for other sub-components (iv), (v) and (vi). An important constraint of our component model is that all components acquired by a component must also be acquired by its super-component.

The quantification metrics [7] implemented in the Java based tool are: NPI, NAI, NAC, and DPIN. Table 1 shows some of the quantification metrics currently obtained by using Java based tool, derived through XMI file. The Java coding of XMI parser for evaluating NPI, NAI, NAC, and DPIN of components is shown in Table 3 to 6. The XMI representation of UML component diagram is illustrated in Table 7.

VII. RESULT AND DISCUSSION

Figure 1 shows a system with eight components, viz User, Teacher, Course, Study Material, Messaging, Report Generation, Authorization, and Authentication. Results obtained for NPI, NAI, NAC and DPIN describe in Table 1, the authors have computed NPI, NAI, NAC and DPIN (cf. screen shots given in Figure 3 to 6. The significance of these metrics (viz. NPI, NAI, NAC and DPIN) in evaluating the software at the early stage is as follows:

A. Provided Interfaces (NPI):
Provided Interfaces are the interfaces of the components to which any other component can use. These are exposed interfaces of the components. In the case study Teacher provides one interface to the User, whereas Authorization and Authentication both provide four interfaces each to sub-components of Teacher. **Hence the total numbers of provided Interfaces are nine.**

B. Acquired Interfaces (NAI):
Interfaces of the acquired components, acquired by a component are referred to as the acquired interfaces. In the case study User uses one interface to acquire component Teacher, Teacher uses one interface to acquire component Authorization and also Teacher use s one interface to acquire component Authentication and **hence the number of acquired interfaces are also three.**

C. Acquired Components (NAC):
All the components which are used by a component named are called acquired components or external components. In the case study, User uses one external component Teacher; Teacher uses two external components Authorization & Authentication. Authorization and Authentication are acquired components of Teacher and Teacher is an acquired component of User. **Hence the total number of acquired components are three.**

D. Deepest Provided Interface Nesting (DPIN):
Components can be nested in system and the nesting process can be extended up any internal level. The most internal level of component nesting which is using the provided interface of some other component is known as Deepest Provided Interface Nesting. It is a measure of the complexity of the component system. This also indicates the importance

of the component. In the case study, deepest component using the provided interface of the acquired component are sub-components of Teacher namely Course, Study Material, Messaging, and Report Generation. **Hence the deepest provided interface nesting is two.**

The Table 2 shows the different quantification metrics extracted for E-learning system.

VIII. CONCLUSION AND FUTURE WORK

The evaluation of number of acquired components (NAI) in Java based parser tool helps in deciding to identify the interconnection of different components in the system, it also shows the dependency of these on each others. Through NAI and NPI user can identify the total number of interfaces in the system. It is used to measure the integration efforts for that individual component. If the value will be higher, then integration efforts will be complex and then maintenance efforts will also increase. Through NPI software developer can check the dependency and usability of component in the system. NAC shows the dependency of component to other components. DPIN shows that which component exists at deep level i.e. components within component (nested component). Higher value of DPIN results in complex integration efforts that can increase the maintenance efforts of the system. It also shows depth of the components in hierarchy and their association and aggregation with each other at designing stage that will help in effort estimation to develop software in early stage.

Finally, it may be concluded that the NAC, NAI, NPI and DPIN play an important role to assess effort estimation and complexity of CBSE at early stage. Hence, by computing all these metrics, basically we would be in a position to predict the approximate cost of CBSE. The proposed tool can also be modified to extract the metrics of other views namely shallow view, deep view etc. for component based systems, which will be considered in a future version.

REFERENCES

[1] S. S Ali, A. Ghafoor and R.A. Paul, "*Metrics-guided quality management for component-based software systems*", *P*roceedings of the 25th Annual International Computer Software and Applications Conference, 2001. COMPSAC 2001, Institute of Electrical and Electronics Engineers (IEEE), Jan 2001, pg. 303-308.

[2] C. Szyperski, "*Component Software"*, Addison-Wesley Professional; 2002.

[3] N.S. Gill, Balkishan, Dependency and interaction oriented Complexity metrics of component based systems. ACM Sigsoft Software Engineering notes, March 2008, vol 33, issue 2, pg. 1-5.

[4] J. Chen, W. Yeap, S. Bruda, "*A Review of Component Coupling metrics for component based development, World Congress on Software Engineering (WCSE'09)*", May 2009, Xiamen, China, DOI: 10.1109/WCSE.2009.391.

[5] J. Chen, H. Wang, Y. Zhon, S. Bruda, "*Complexity metrics for Component based Software System*", International Journal of Digital Content Technology and its Applications, March. 2011, volume 5, number 3, pg. 235-244,.

[6] R. S. Pressman, *Software engineering: A practitioner's approach*. McGraw Hill.

[7] G. Falcone, C. Atkinson, "*A Basis for a Metric Suite for Software Components*", 12th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 8th July 2008 - Paphos, Cyprus, pg. 53-63.

[8] M. F. Bertoa and A. Vallecillo, "*Usability metrics for software components*", 8th international workshop on quantitative approaches in object-oriented software engineering (QAOOSE'2004), Oslo, Norway, June 2004.

[9] P.L. Powar, M.P Singh, S Upadhyay, and B. Solanki, "*Dynamic software metric estimation (DSME): Tool using ArgoUML*", International Journal of Advanced Research in Computer Science, May-June 2017, volume 8 number 5, pg. 591-602.

[10] L.V. Narasimhan, P.T. Parthasarathy, M. Das, "*Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE)*", issues in Informing Science and Information Technology, January 2009, volume 6, pg. 731-740.

[11]L.V. Narasimhan, B. Hendradjaya, "*Some theoretical consideration for a suite of metrics for the integration of software components*", issues of information Science and Information Technology, 2007, volume 177, issue -3, pg 844-864.

[12] R. K. Pandey, J.W. Shareef, "*CAME: Component Assembly Metrics Extraction using UML*", ACM SIGSOFT Software Engineering Notes, July 2013, volume 38 number 4, pg. 1-12.

[13] R. K. Pandey, J.W. Shareef, "*Design of a Component Interface Complexity Measurement Tool for Component-Based Systems*", ACM SIGSOFT Software Engineering Notes, July 2015, Volume 40 Number 1, pg. 1-12.

[14] *SAX*, Retrieved on March 5, 2011 http://sax.sourceforge.net.

Author Profiles

P. L. Powar received her M.Sc. and Ph.D. in Mathematics form Rani Durgavati University, Jabalpur, India. Currently, she is a Professor in Deparment of Mathematics and Computer Science, Rani Durgavati University, Jabalpur, India. Her research areas are Cutting Stock Problem, Spline Approximation Theory, Finite Element Methods, Software Engineering and Topology.

Manu Pratap Singh is a Associate Professor, in Institute of Engineering & Technology, Agra, Dr. B. R. Ambedkar University, Agra. He receives Young Scientist award in 2005 by IAPS, Allahabad Paper Presented in IFORS, Sandton, South Africa in 2008. His current research interest is Artificial Neural Networks, Artificial Intelligence, Soft Computing, Software Engineering.

Bharat Solanki received his M.Sc. (Mathematics) and M.CA from Rani Durgavati University, Jabalpur, India. Presently persuing Ph.D. in Computer Science. His research area is Software Engineering.

Jawwad Wasat Shareef received the MCA degree from Indira Gandhi National Open University, India and Ph.d in Computer Science from Rani Durgavati Universitry, Jabalpur, India. Currently he is working in Department of Mathematics and Computer Science, Rani Durgavati University, Jabalpur. His research areas are Software Engineering, Data mining, Soft computing.

Metrics for external view	Types	Full form	Interpretation
Quantification Metrics	NPI	Number of Provided Interface	The total number of interface provided by the component.
	NAI	Number of Acquired Interface	The total number of interfaces acquired by the component.
	NAC	Number of Acquired Components	The total number of acquired components.
	DPIN	Deepest Provided Interface Nesting	The depth of nesting of the provided interface of the component with the deepest nesting.

Table 1. External view based metrics

Module	Quantification metrics	Value of metrics
E-learning system	NPI	9
	NAI	3
	NAC	3
	DPIN	2

Table 2. Quantification metrics for component diagram described

```
    public void showProvidedInterfaces() {
        int count=0;
        for (int i = 0; i < GlobalLists.lstComponentDependencies.size(); i++) {
            for(int j=0;j < GlobalLists.lstComponents.size(); j++) {
                if (GlobalLists.lstComponents.get(j)!=null
&& GlobalLists.lstComponentDependencies.get(i)!=null
&& GlobalLists.lstComponentDependencies.get(i).getSupplierComponentId()!=null
&& !GlobalLists.lstComponentDependencies.get(i).equals("")) {

System.out.println(GlobalLists.lstComponentDependencies.get(i).getSupplierComponentId()+"          :::
"+(GlobalLists.lstComponents.get(j).getXmlId()));

if(GlobalLists.lstComponentDependencies.get(i).getSupplierComponentId().equals(GlobalLists.lstCom
ponents.get(j).getXmlId())) {
                        count++;
                    }
                }
            }
        }
        lblNPI.setText("Number of Provided Interfaces :"+count);
```

Table 3. Java implementation function for calculating NPI


```
public void showAcquiredInterfaces() {
    int count=0;
    String usedComponent="";
    for (int i = 0; i < GlobalLists.lstComponentDependencies.size(); i++) {
      if (!GlobalLists.lstComponentDependencies.get(i).equals("") &&
GlobalLists.lstComponentDependencies.get(i).getSupplierComponentId()!=null &&
GlobalLists.lstComponentDependencies.get(i).getClientComponentId()!=null) {
        int ccId = this.getClientComponentId(GlobalLists.lstComponentDependencies.get(i).getClientComponentId());
        int scId = this.getClientComponentId(GlobalLists.lstComponentDependencies.get(i).getSupplierComponentId());
        System.out.println(GlobalLists.lstComponents.get(scId).getName()+" ::
"+GlobalLists.lstComponents.get(scId).getDepth()+" :: "+GlobalLists.lstComponents.get(scId).getIsRoot().equals("true"));
        if(GlobalLists.lstComponents.get(scId).getDepth()==0
            && GlobalLists.lstComponents.get(scId).getIsRoot().equals("true") &&
usedComponent.indexOf(GlobalLists.lstComponents.get(scId).getName())==-1)
            //&& GlobalLists.lstComponents.get(ccId).getDepth()>0
            //&& GlobalLists.lstComponents.get(ccId).getIsRoot().equals("false"))
        {
          usedComponent+=GlobalLists.lstComponents.get(scId).getName()+",";
          count++;
        }
      }
    }
    lblNPI.setText("Number of Acquired Interfaces :"+count);
```

Table4: Java implementation function for calculating NAI

```
public void showAcquiredComponents() {
    int count=0;
    for(int j=0;j < GlobalLists.lstComponents.size(); j++) {
      if(GlobalLists.lstComponents.get(j).getIsRoot().equals("true")) {
        count++;
      }
    }
    //acquired components are all roots components - 1
    lblNPI.setText("Number of Acquired Components :"+(count-1));
```

Table 5 : Java implementation function for calculating NAC

```
public void showDeepestProvidedInterfaceNesting() {
    int count=0;
    for(int j=0;j < GlobalLists.lstComponents.size(); j++) {
      if(count<GlobalLists.lstComponents.get(j).getDepth()) {
        count=GlobalLists.lstComponents.get(j).getDepth();
      }
    }
    lblNPI.setText("Deepest Provided Interface Nesting :"+(count+1));
}
```

Table 6: Java implementation function for calculating DPIN

```
<UML:Model xmi.id = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000A8E'
  name = 'untitledModel' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
  isAbstract = 'false'>
```



```
<UML:Namespace.ownedElement>
 <UML:Component xmi.id = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000A90'
  name = 'User Management' isSpecification = 'false' isRoot = 'true' isLeaf = 'false'
  isAbstract = 'false'>
  <UML:ModelElement.clientDependency>
   <UML:Dependency xmi.idref = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000AA7'/>
  </UML:ModelElement.clientDependency>
 </UML:Component>
 <UML:Component xmi.id = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000A91'
  name = 'Teachers' isSpecification = 'false' isRoot = 'true' isLeaf = 'false'
  isAbstract = 'false'>
  <UML:Namespace.ownedElement>
   <UML:Component xmi.id = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000A94'
    name = 'Course Management-Teachers' isSpecification = 'false' isRoot = 'false'
    isLeaf = 'true' isAbstract = 'false'>
    <UML:ModelElement.clientDependency>
     <UML:Dependency xmi.idref = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000A9F'/>
     <UML:Dependency xmi.idref = '-64--88--23-1--586162ab:15c6d8e6389:-8000:0000000000000AA0'/>
    </UML:ModelElement.clientDependency>
   </UML:Component>
```

Table 7: XMI representation of component diagram

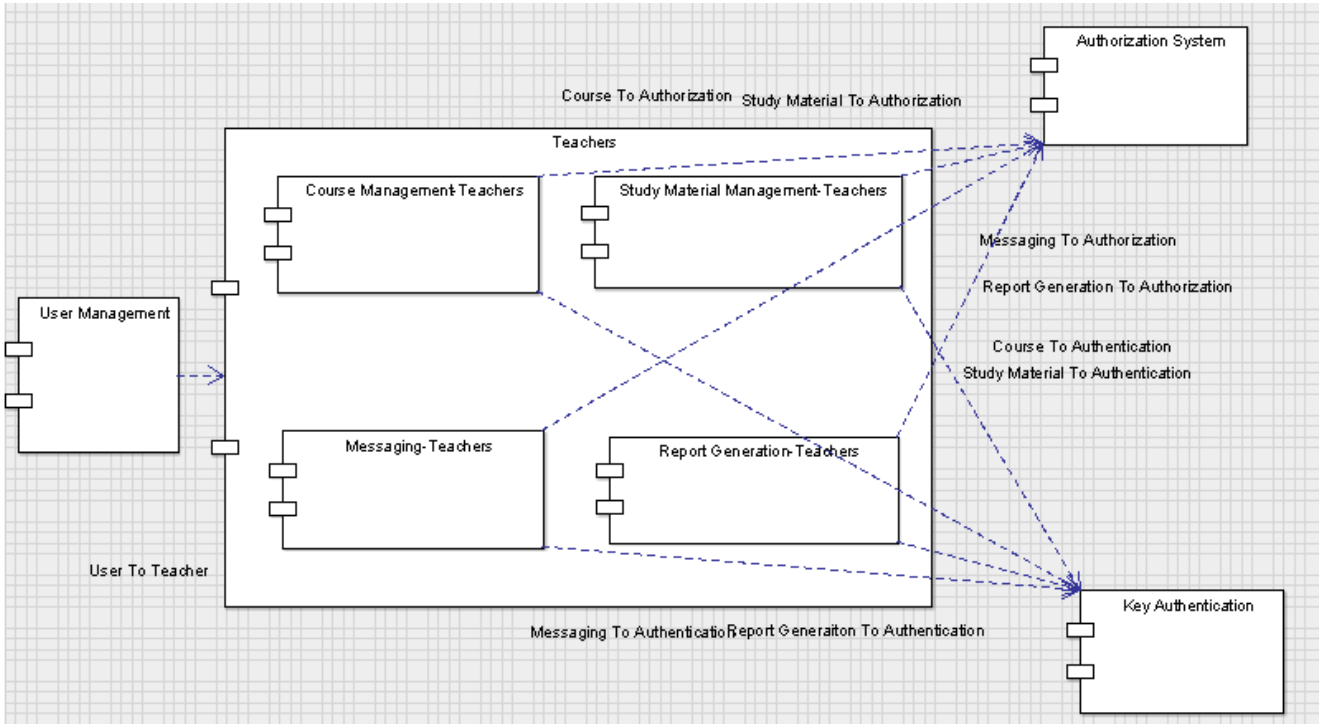

Figure 1. Component diagram for E-learning System

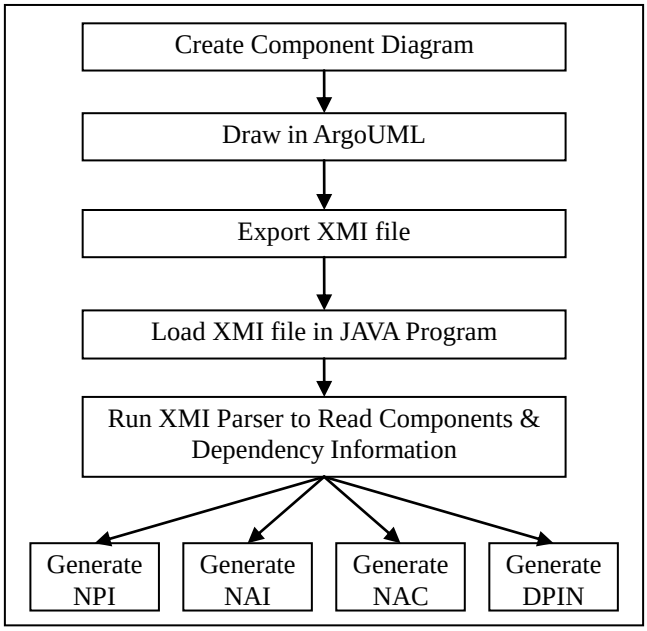

Figure 2. Working of Java based tool for E-Learning System

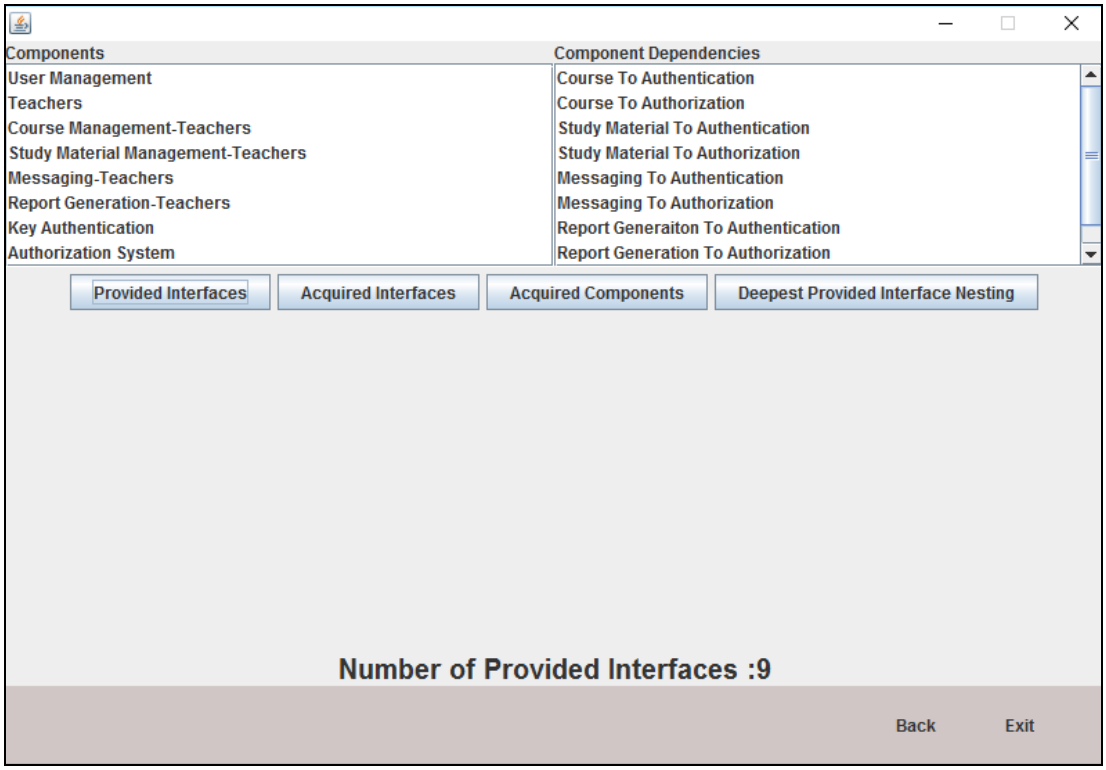

Figure 3: GUI for displaying number of provided interface

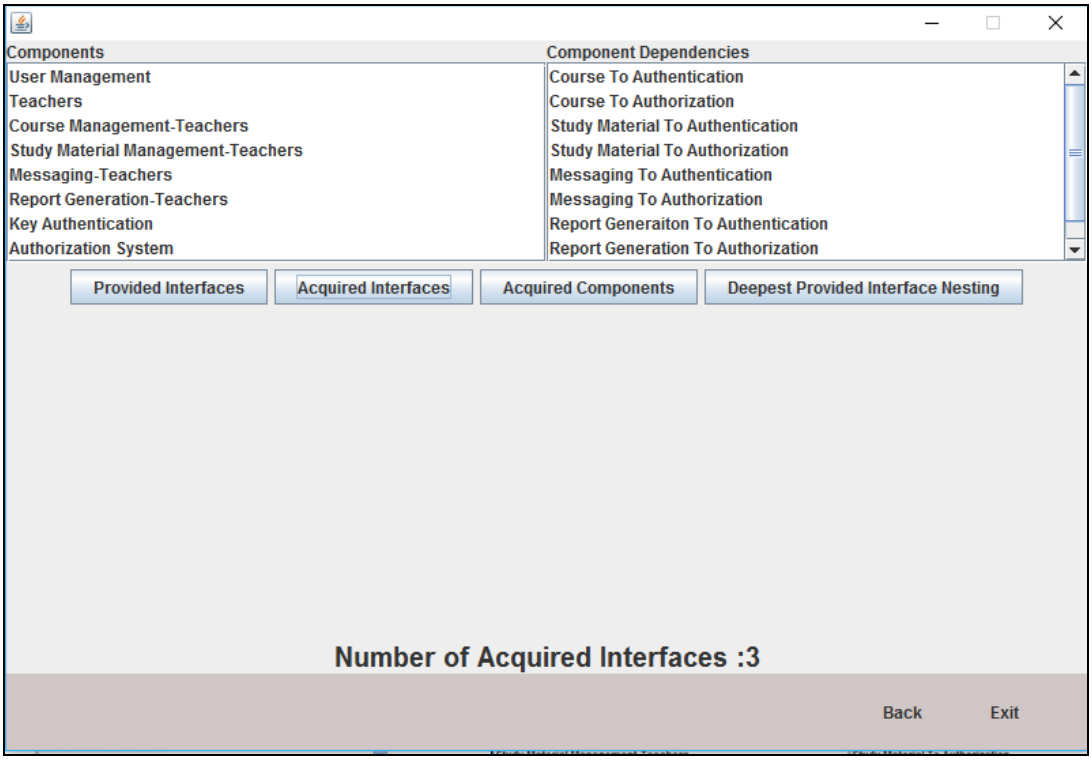

Figure 4: GUI for displaying number of acquired interfaces

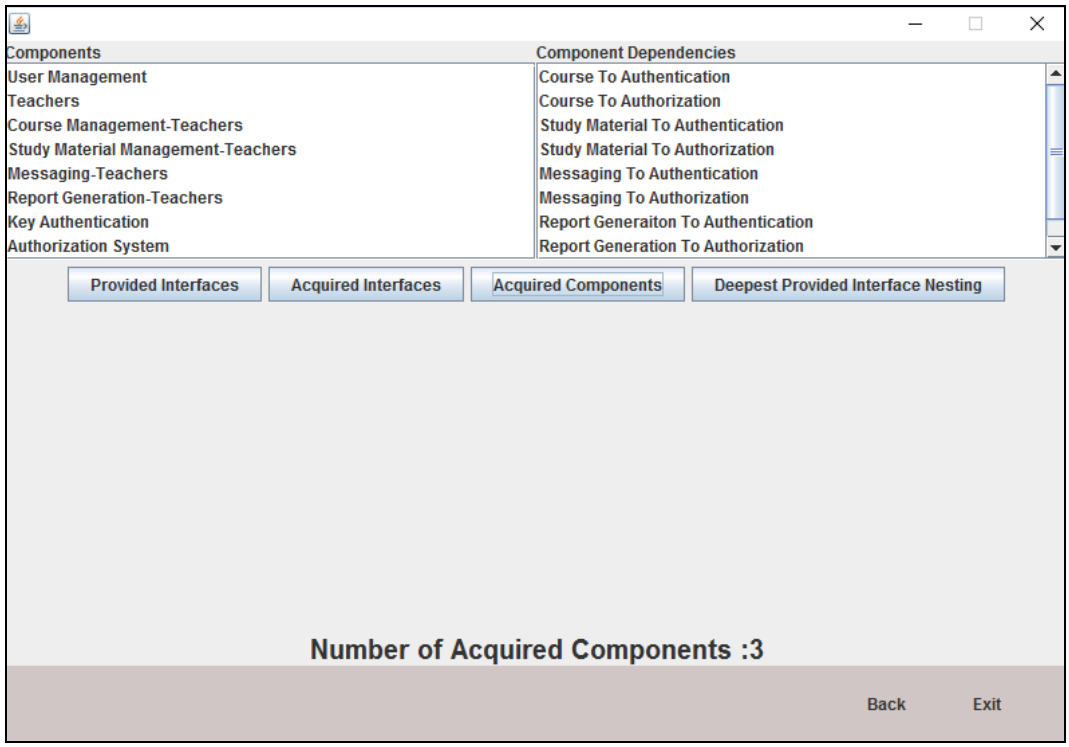

Figure 5: GUI for displaying number of acquired components

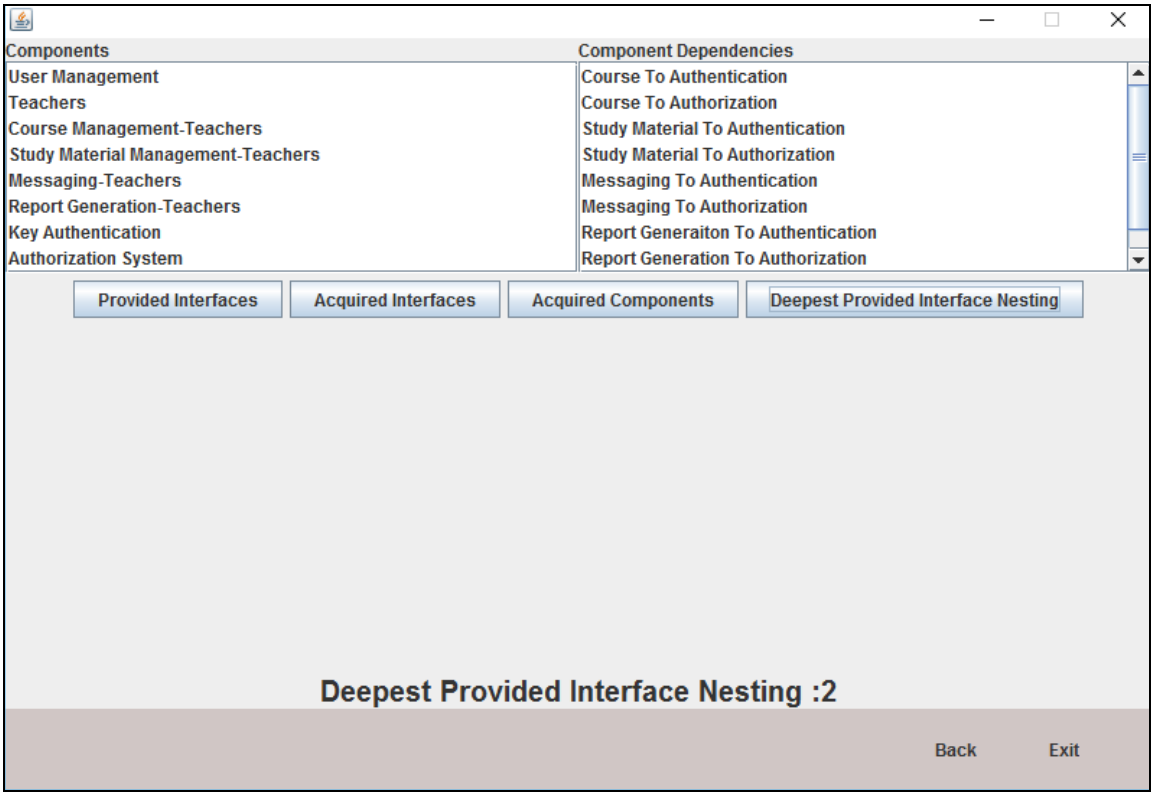

Figure 6: GUI for displaying deepest provided interface nesting

