Dependency Analysis for Component based Systems using Minimum Spanning Tree

Jyoti Rani
Ajay Kumar Garg Engineering College, Ghaziabad

Kirti Seth
Ajay Kumar Garg Engineering College, Ghaziabad

ABSTRACT

Dependency analysis is advantageous technique that has many applications in software engineering activities. In component-based system (CBS), Dependencies can solve implicit problems such as integration testing, regression testing, change processing, component reusing and version control. In order to promote testing of the CBS, it is necessary to analyze the mutual impact between components and form a description of dependencies. During the present time dependency analysis is one of the important research fields in CBS. This paper presents a minimum spanning tree approach to analyze dependency in Component Based Systems (CBS). First we calculate the dependency of each component using Minimum Spanning Tree in component based system and then calculate the dependency of each component using Analytical Hierarchal Process. Finally we calculate the Correlation Coefficient of the two techniques.

General Terms

analytical hierarichal process; minimum spanning tree; correlation coefficient.

Keywords

component; interaction; interfaces; dependency; component based systems; Component dependency graph; analytical hierarichal process; minimum spanning tree; correlation coefficient.

1. INTRODUCTION

Interaction in component-based systems (CBS) takes place when a component delivers an interface and other components use it, and also when a component submits an event and other component receives it. Dependencies are promoted by interactions. Higher dependency leads to a complex system, which results in poor understanding and a higher maintenance cost.[1]

The dependency among components can be illustrated as the assurance of a component on other components to support a specific functionality or configuration[1].

In a Component-Based System (CBS) dependencies rise from the individual dependencies of each component that compose the system as well as from the possible casual composition of the dependencies among those components. A component's required interface expresses its dependencies on services provided by other components. This interface, however, contains only a fraction of the information necessary to analyze dependencies embedded in a CBS, partly because interfaces of a component are typically limited to listing names and type signatures of the component's attributes and services. That information is not sufficient to have a good understanding of the component's assumptions about the services on which it depends, that is, when, under what conditions, and for what purpose the services are necessary[4].

Practically, as soon as a new component is installed in a system, it has an impact on a part of the system. The new component may refer to certain components, and also be used by other components. This is a kind of explicit direct dependency. In addition, there are also indirect dependencies, derived from the components which are used by the new component, and also implicit dependencies, that are related to the system environment. In Component Based Systems, there are four types of dependencies: explicit dependency, explicit indirect dependency, implicit direct dependency and implicit indirect dependency.[5]

Dependence analysis involves the identification of interdependent elements of a system. It is referred to as a "reduction" technique, since the interdependent elements induced by a given inter-element relationship forms a subset of the system. It has been widely studied for purposes such as code restructuring during optimization, automatic program parallelization, test-case generation, and debugging. Dependence analysis as applied to program code is based on the relationships among statements and variables in a program. Techniques for identifying and exploiting dependence relations at the architectural level have also been developed. Dependence relationships at the architectural level arise from the connections among components and the constraints on their interactions. These relationships may involve some form of control or data flow, but more generally involve source structure and behavior. Source structure (or structure, for short) has to do with system dependencies such as "imports", while behavior has to do with dynamic interaction dependencies such as "causes". Structural dependencies allow one to locate source specifications that contribute to the description of some state or interaction. Behavioral dependencies allow one to relate states or interactions to other states or interactions. Both structural and behavioral dependencies are important to capture and understand when analyzing an architecture[6].

Dependencies between different components within a complete system exist in the way that the whole application becomes unstable if those dependencies are broken or violated. Normally, there is no reason why this should happen during work with an application or a tool. However, this can necessarily occur in the case that (distributed) applications change during run time. Such changes can appear in many ways. In the following we will concentrate on two fields of applications where those changes are not only normal but intended in a special way[8].

2. REVIEW

Similar work has been done by many researchers. Binbin Qu,Qian Liu,Yansheng Lu has been mentioned a new dynamic dependency analysis framework for COM[3].

Zimmermann and et al.(2011) have also proposed two large software systems: Microsoft VISTA and ECLIPSE. Their results showed that components that have outgoing dependencies to components with higher object-oriented complexity tend to have fewer field failures for VISTA, but the opposite relation holds for ECLIPSE[10].

Usha Kumari and Shuchita Upadhyaya (December 2011) have designed an interface complexity metric for black-box components to quantify an important aspect of complexity of a component-based system[2].

Marlon Vieira and Debra Richardson has given a technique to analyze dependencies in large component-based systems. This technique proposed an explicit representation of component dependencies by using a deployable Extensible Markup Language (XML) description[4].

Robert Leitch and Eleni Stroulia have been proposed a model for quantifying the quality of a design from a maintainability perspective. Based on this model, they propose a novel strategy for predicting the "Return on Investment" (ROI) for possible design restructurings using procedure level dependency alalysis[11].

Saleh Alhazbi and Aman Jantan has been discussed dependencies analysis significance when updating component-based system dynamically and presented a service-based matrix model and nested graph as approaches to capture components' dependencies they discussed using dependencies analysis for safe dynamic updating in component-based software sysrems[12].

Arun Sharma, P. S. Grover and Rajesh Kumar have been proposed a link-list based dependency representation and implements it by using Hash Map in Java[1].

3. DEPENDENCY IN COMPONENT BASED SOFTWARE DEVELOPMENT

In Component-based development (CBD) paradigm, Component-based software system (CBSS) are established using a set of mutually dependent components which work together. Some of these components may be developed in-house, while others may be third-party components, without source code.

The main objective of this approach is to minimize the development effort, time and cost by means of software reuse. CBSD advances quality, productivity, reliability and maintainability of the software system[2].

Dependency between components can be defined as the reliance of a component on other components to support a specific functionality; therefore, we consider dependency as a binary relationship between two components: dependent and antecedent. Dependent component is one that related to its antecedents where changes in them might lead dependent to malfunction or fail. Antecedent is the component that has an effect on the dependent one if it is removed or modified, on the other hand, Alhazbi and Jantan. Some times it may occur that a component has to take help of other components to perform its functionality. A component A is dependent on component B means that A must be checked if B changes. Maximization of such components builds a CBS complex[20].

CBS requirements analysis and component selection is widely recognized as a commutal process which plays an interior role in overall CBS development Individual components usually provide fix capabilities that might not satisfy all system requirements and some of them may be unnecessary in a given system. This reduces the chance of a match between a component and stakeholder requirements. Therefore, it is difficult to find a supplier who can meet all stakeholder requirements[14]

3.1 Benefits of CBSD

Software developers create software components mainly with an intention of being reused in various software systems. Components are designed to interact with its environment through its well-defined interfaces but to encapsulate their implementation. Component-based software development brings the potential for

1. significant reduction in the development cost and time-to-market of enterprise softwar systems because developers can assemble such systems from a set of reusable components rather than building them from scratch,

2.increasing the reliability of enterprise software systems - each reusable component undergoes several review and inspection stages in the course of its original development and previous uses, and CBSD relies on explicitly defined architectures and interfaces,

3. improving the maintainability of enterprise software systems by allowing new, higher quality components to replace old ones, and

4. enhancing the quality of enterprise software systems - application-domain experts develop components, then software engineers who specialize in Component software engineering assemble those components into enterprise software systems

3.2 Some common definition proposed by researchers:

Incoming dependency: A component has an incoming dependency if syntactically another component appropriates its data or functionality.

Outgoing dependency: A component has an outgoing dependency if syntactically it appropriates data or functionality of another component.

Dependant: A component is a dependent with respect to another component if it has an outgoing dependency on that component.

Dependee: A component is a dependee with respect to another component if it has an incoming dependency from that component[10].

In general, there are eight types of dependency as follows[5]:

Data dependency: Data dependence is generated by data integration between different COTS components. In general, data dependency represents that the data defined in one component, but used in another one.

Control dependency: Control dependency is generated by control integration in CBSs, it is not explicit dependency. Control dependency is realized by broadcasting, remote procedure calls or by general passing.

Time dependency: Time dependency means that the behavior of one component precedes or follows the behavior of another component in CBSs.

State dependency: State dependence means that the behavior of a basic component will not happen unless the system, or some part of the system, is in a specified state.

Cause and effect dependency: Cause and effect dependency means that the behavior of one component implies the behavior of another component.

Input/Output dependency: Input/Output dependency means that a component requires/provides information from/to another component.

Context dependency: Context dependency means that a

component runs must be special context environment.

Interface dependency: Interface dependency is generated by user interface integration. Usually, the interface-event dependency is the main dependency form in CBSs.

4. COMPONENT DEPENDENCY GRAPH (CDG)

Component Dependency Graph of a CBS is defined as G=(S,D,s,t), is a directed graph, where S is a non empty set of vertices each represents a component in the system, D is a set of dependency edges between two vertices each represents a direct dependency between components, s is a starting node, t is a terminating node. Fig 1 describes the direct dependency, where D={(A,B),(B,D),(C,D),(C,B),(C,A),(E,B),(E,D)}

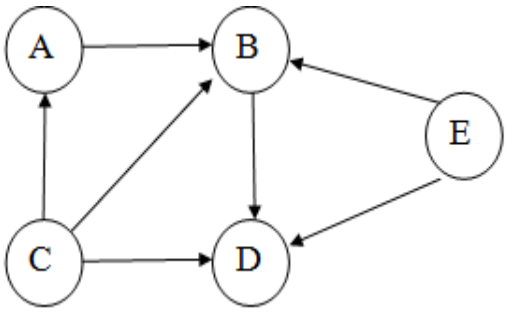

Fig 1 : Component Dependency Graph

5. PROPOSED APPROACH

We propose a new approach to analyze dependency in Component Based System (CBS). This approach contains the following steps:

1. Construct a Component Dependency Graph (CDG) of a Component Based System(CBS).

2. Assign weights to every edge of Component Dependency Graph.

3. Calculate the minimum spanning tree for CDG by any one of the existing algorithms (Prim,s algorithm or Kruskal,s Algorithm).

4.The dependency of the individual component is the minimum weight of that component.

Fig. 2 describes the flowchart which helps to calculate dependency of each component using Minimum Spanning Tree. First construct Component Dependency Graph, which have different components and interaction with other components. Dependency of each component is the minimum weight of that component. Here weight is the probability of dependency in Component Dependency Graph.

Weight is directly proportional to the probability of the dependency it means that if weight increases then probability of dependency of a component increases, similarly if weight decreases then probability of dependency of a component decreases.

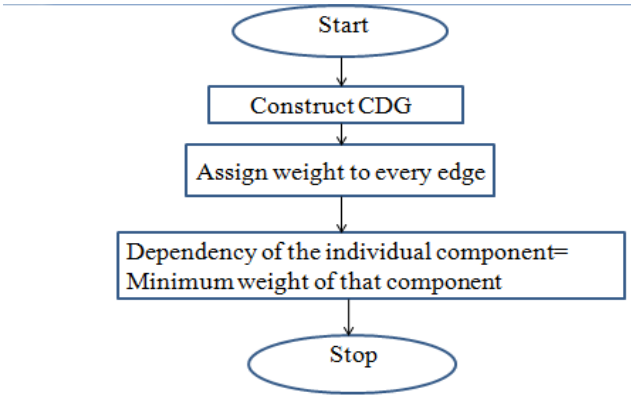

Fig 2: Flow chart of Proposed Approach

6. SPANNING TREE

A spanning tree of a graph is a subgraph that contains all the vertices and is a tree. A graph may have many spanning trees. A spanning tree of G is a selection of edges of G that form a tree spanning every vertex that is, every vertex lies in the tree, but no cycles are formed. "A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices." The weight of spanning tree of a graph is the sum of the weights of all the edges in the spanning tree. Different spanning trees of a Graph will have different weights.

6.1 Minimum Spanning Tree

A single graph can have many different spanning trees. We can also assign a weight to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. "A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree".

7. DEMONSTRATION OF THE PROPOSED APPROACH

For the analysis of proposed work we consider a hypothetical model as shown in figure. This model is a CDG for any CBS. There are eight components in the graph. The dependency of each component is assigned as the weight of each edge.

1.Construct a Component Dependency Graph(CDG)

We construct a CDG of 8 components (A,B,C,D,E,F,G,H) and 15 edges which are connected to the components in the graph. Edges connects the component with each other as shown in Fig 3 CDG.

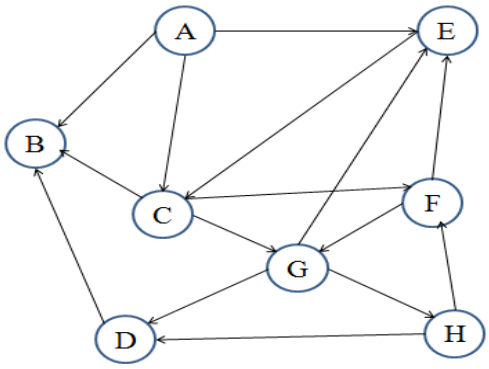

Fig 3: Component Dependency Graph

2 .Assign weights to every edge in the component dependency graph in Fig 4.

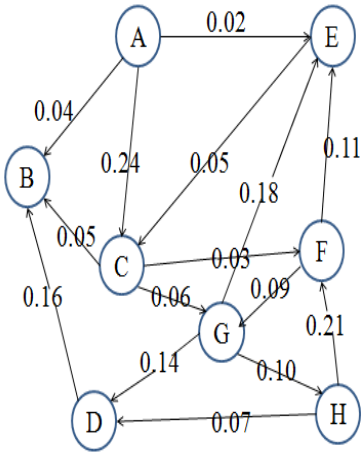

Fig 4: Weights in Component Dependency Graph

3. Calculate minimum spanning Tree, which shown in Fig 5.

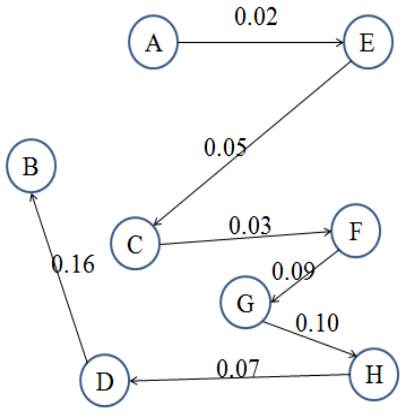

Fig 5: MST of Component Dependency Graph

4 .Figure 5 gives the minimum spanning tree of the given Component Dependency Graph using Prim's Algorithm.

8. ANALYTICAL HIERARICHAL PROCESS

Analytical Hierarchy Process is a multi-criteria decision making process. AHP is a comprehensive, analytical and structured framework

The AHP, as a compensatory method, accepts complete aggregation among criteria and develops a linear additive model. The weights and counts are achieved generally by pairwise comparisons between all options with each other.

To make a decision in an adapted way to calculate priorities we need to break down the decision into the following steps.

1. Define the problem and calculate the kind of knowledge sought.

2. Structure the decision hierarchy from the top with the goal of the decision, then the objectives from a broad perspective, through the intermediate levels to the lowest level.

3. Construct a set of pairwise comparison matrices. Each element in an upper level is used to compare the elements in the level immediately below with respect to it.

4. Use the priorities obtained from the comparisons to weigh the priorities in the level immediately below. Do this for every element. Then for each element in the level below add its weighed values and obtain its overall or global priority. Continue this process of weighing and adding until the final priorities of the alternatives in the bottom most level are obtained[15, 16,17].

9. CORRELATION COEFFICIENT

A correlation coefficient ammounts the strength and direction of a linear joining between two variables. The range of Correlation Coefficient is from -1 to +1. The closer the absolute value is to 1, the stronger the relationship. A zero correlation indicates that there is no linear relationship between the variables. Correlation quantifies the extent to which two quantitative variables, X and Y, "go together." When high values of X are associated with high values of Y, a positive correlation exists. When high values of X are associated with low values of Y, a negative correlation exists[18,19].

Correlation is frequently used as a descriptive tool in non-experimental research. Two measures are correlated if there is something in common. The intensity of the correlation is described by a number called the correlation coefficient which is almost always denoted by the letter r.

The correlation coefficient is a tool used to appraise the similarity of two sets of measurements obtained on the same observations. "Correlation" relates to a process for finding whether relationships exist between two variables the or not. The concept of 'correlation' is a statistical tool which studies the relationship between two variables and Correlation Analysis involves various methods and techniques used for studying and measuring the extent of the relationship between the two variables. The coefficient can be either negative or positive.

"Two variables are said to be in correlation if the change in one of the variables results in a change in the other variable".

"A single summary number that gives us a good idea about how closely one variable is related to another variable.

The main idea behind correlation coefficient is to calculate an index which reflects how much two measurements sequences are related to each other. This coefficient will take values from -1 to +1.

A value 0 implies that the two sequences of measurements have nothing in common. A coefficient close to zero indicates that no methodical co-varying exists between the variables.

A value +1 infers that two sequences of measurements are measuring the same thing. A positive correlation coefficient infers that two variables systematically vary in the same direction: as one variable increases, the other variable tends to increase. The closer the coefficient is to +1, the stronger the positive association. In other words, as one variable goes up so does the other.

A value -1 infers that the two measurements are measuring the same thing but one measurement varies inversely to the other. A negative correlation coefficient infers that two variables systematically vary in opposite directions: as one variable increases, the other variable tends to decrease. The closer the coefficient is to -1, the stronger the negative association. The coefficient of correlation indicates how much information is shared by two variables, or in other words, how much these two variables have in common.

10. EMPRICAL VALIDATION

We calculate the component selection efforts by Analytical Hierarchical Process(AHP). Finally we calculate the correlation coefficient between these two outputs.

$$\text{Correlation Coefficient } (r) = \frac{\sum(X-\bar{X})(Y-\bar{Y})}{\sqrt{(X-\bar{X})^2.(Y-\bar{Y})^2}}$$

Where,

N equals the number of score-pairs,

$\overline{X} = \sum x/n$ $\overline{Y} = \sum y/n$

The symbol ∑ is "sigma", which is a mathematical shorthand meaning "sum up".

The value of the correlation coefficient between the outputs calculated by AHP method comes to be 0.8114.Hence we conclude that the proposed technology is valid.

Table 1. Dependency of Each Component using MST and AHP

Components	Dependency using MST(X)	Dependency using AHP(Y)
A	0.00	0.0700
B	0.52	0.2043
C	0.07	0.0906
D	0.36	0.1624
E	0.02	0.1404
F	0.10	0.1147
G	0.19	0.0967
H	0.29	0.12056

Correlation Coefficient of the two techniques MST and AHP of Table 1 calculated. Correlation Coefficient between X and Y is 0.8114, which says that the dependency using MST and AHP are strongly related.

11. CONCLUSION

Understanding and tracking dependence among components in CBSD is increasingly difficult in large and complex systems. Dependency analysis helps to answer the following questions in a component based systems: If a component is updated, which other components in the system are affected, what is the effect on a system if a new component is installed, which components are more important than others and which components are isolated.

Dependencies in a Component-Based System (CBS) take place from the individual dependencies of each component. Dependency in component based system rise when a component provide an interface and another component use it or when a component sends an event and another component receive it. This paper proposed a minimum spanning tree based approach and Analytical Hierarchal Process for analyzing dependency in component based system and also calculates the correlation coefficient between the two techniques which shows that the technique is valid because the value of correlation coefficient is 0.81, which is near about to 1.

12. FUTURE WORK

This data can be used to measure the interaction complexity of the Component Based System and also can analyze several interaction and dependency related issues with the proposed approach. For future work this approach will be validated on some other applications and the result can also be calculated in the similar manner for other approaches and then Correlation Coefficient can be calculated between the other approaches.

REFERENCES

[1] Sharma, A., Grover, P.S., Kumar, R., 2009. "Dependency Analysis for Component-Based Software Systems" Volume 34 Number 4.

[2] Kumari, U., Upadhyaya, S., 2011. "An Interface Complexity Measure for Component-based Software Systems" International Journal of Computer Applications (0975 – 8887) Volume 36– No.1.

[3] Qu, B., Liu, Q., Lu, Y., 2010. "A Framework for Dynamic Analysis Dependency in Component-Based System".

[4] Vieira, M. and Richardson, D.,2002. "Analyzing Dependencies in Large Component-Based Systems". Proceedings of the 17 th IEEE International Conference on Automated Software Engineering (ASE'02), 2002, pp 241 – 244.

[5] Liangli, M. and Houxiang,, 2006. "The Design of Dependency Relationships Matrix to improve the testability of Component-based Software".

[6] Stafford, A., Richardson, D. and Wolf, A.L., 1998. "Architecture-level Dependence Analysis in support of Software maintenance". ISAW '98 Proceedings of the third international workshop on Software architecture. New York, NY, USA, pp 129-132.

[7] Abate, P. and Boender,J.,2009, "Strong Dependencies between Software Components", Third International Symposium on Empirical Software Engineering and Measurement, 24 May 2009, pp 89-99.

[8] Won, M., "Managing Dependencies in Component-Based Distributed Applications".

[9] Li, B., "Managing Dependencies in Component-Based Systems Based on Matrix Model".

[10] Zimmermann, T., Nagappan, N., Herzig, K., Premraj, R. and Williams, L., 2011. "An Empirical Study on the Relation between Dependency Neighborhoods and Failures". Fourth IEEE International Conference on Software Testing, Verification and Validation, Berlin, 21-25 March 2011, pp 347-35

[11] Leitch, R. and Stroulia, E.,2003. "Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis". Ninth International Software Metrics Symposium, Canada , 3-5 Sept. 2003, pp 309-322.

[12] Alhazbi, S. and Jantan, A.,2007. "Dependencies Management in Dynamically updateable Component-Based Systems", *Journal of Computer Science*, Vol.3, Issue 7, pp. 499-505.

[13] Vieira, M. and Richardson, D.,2002. "The Role of Dependencies in Component Based Systems Evolution", Proceeding of the International Workshop on Principles of Software Evolution,New York, USA, pp 62-65.

[14] Mahmood, S. and Lai, R. ,2006. "Analyzing Component Based System Specification". AWRE. Adelaide, Australia, 2 Feb 2006,pp 1055-1076.

[15] Coyle, G.: Practical Strategy. Open Access Material. AHP, "THE ANALYTIC HIERARCHY PROCESS (AHP)"

[16] Forman, E.H., 2001. "The Analytic Hierarchy Process – An Exposition".Gill, N.S. 2006. Importance of Software Component Characterization For Better Software Reusability. *ACM SIGSOFT Software Engineering Notes*, Vol.31 Issue1,pp 1-3.

[17] T.L. Saaty, Int. J. Services Sciences, Vol. 1, No. 1, 2008, "Decision making with the analytic hierarchy process".

[18] Pearson's r, Spearman rho Other Coefficients of Note Coefficient of Determination r2 "Correlation Coefficients The Meaning of Correlation".

[19] Callaghan, K., Ph.D, "The Correlation Coefficient".

[20] Ratneshwer and Tripathi , A. 2011. "Dependence Analysis of Component Based Software through Assumptions". *IJCSI International Journal of Computer Science Issues*, Vol. 8, Issue 4