ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 8, August 2013

Multi-Parameter Summarization for Software Architecture Recovery

Sarvar Begum[1], Manjula.K.S[2], D. Venkata Swetha Ramana [3]

Student, CSE, RYMEC, Bellary, India [1]

Student, CSE, RYMEC, Bellary, India [2]

Senior Lecturer, CSE, RYMEC, Bellary, India [3]

Abstract: Software architecture is identified as an important element in the successful development and evolution of software systems. In spite of the significant role of architecture representation and modeling, many existing software systems like legacy or eroded ones do not have a consistent architecture representation. There have been several algorithms on Architecture recovery utilizing various aspects of similarity measures, clustering, lexical rules and distance measures. It is understood from the literature that no single technique can give best interpretation or desired result in the summarization process. Therefore in this work we propose a multi parameter summarization for extracting high level software architecture with the help of Bipartite graph matching and semantic similarity.

Keywords: Software architecture recovery, bipartite matching, semantic similarity

I. INTRODUCTION

Software is a set of modules. Architecture is a structure of the system which is a set of software elements and externally visible properties of those elements and the relationships among them. The term SA frequently indicates the documentation of a system's "Software Architecture". Documenting software architecture helps in the interaction between stake holders, describes early decisions about the high level design, and allows reuse of design components between projects.

The term SA is used to denote three concepts

- High level structure of the software system
- Order for creating such a high level structure
- Documentation of this high level structure

It is known that maintenance of software system requires a large portion of programmers effort and also a large amount of time will be spent in understanding the program's logic. So it could greatly help the maintainers if we help them in understanding the program logic (by providing documentation).One way to help the maintainers is to give a complete overview of the existing system. This overview may contain the main components of the system, relationships among these components and conditions on these relationships. This kind of overview is called software architecture.

The explanation of architecture of a system can help a maintainer's attention to the more critical parts of the system which need to be understood in more detail.

Most of the SAR profession are involved in finding the methods which help us to identify the architectural description of the system. There is a large amount of existing code which needs to be maintained and would also benefit from an architectural description. Thus, there is a need to *recover architectural descriptions for existing systems.*

This topic has gained the concentration of researchers lately, and developers have started to document software architectures. The current architecture of a software system may differ from the documented architecture if architecture changes are made during software implementation or maintenance and no related effort is made to maintain the related architecture documents. Although in theory, architectural integrity can be enforced by a continuous review process, in practice this is rarely done. To evaluate how well the architecture of a software system corresponds to its documentation, we use architecture conformance analysis; it can also help in keeping the document of software architecture up to date.

SAR is a reverse engineering process and is defined as process of extracting architectural information from its lower level abstraction such as source code. As a source code changes, the maintenance of document related to source code becomes difficult. Most of the times when the programmer modifies the source code, he or she will not do the modification to the related document. As a result, the documentation becomes outdated. So, the software architecture recovery plays an important role in maintenance and evaluation of software systems. SAR is a flavour of reverse engineering that concerns all activities for making existing of software architecture explicit. In SAR, the analysis must show all the historical design decision by looking at the existing implementation and documentation of the system

The use of software architecture can have a positive impact on four aspects of software development.

1. Understanding: Software architecture helps us to understand the system at different levels of abstraction. For making specific architectural choices, architectural description exposes the high-level conditions on system design.
2. Reuse: Architectural descriptions allows reuse at multiple levels. Architectural design supports, in addition, both reuse of large components and also frameworks into which components can be integrated.
3. Evolution: Software architecture can expose the dimensions along which a system is expected to evolve.By making explicit the "load-bearing walls" of a system, system maintainers can better understand the ramifications of changes, and thereby more accurately estimate costs of modifications. Moreover, architectural descriptions can separate concerns of the functionality of a component from the ways in which that component is connected to (interacts with) other components. This allows one to change the connection mechanism to handle evolving concerns about performance, interoperability, prototyping, and reuse.
4. Analysis: Architectural descriptions provide new opportunities for analysis, including high-level forms of system consistency checking, conformance to an architectural style, conformance to quality attributes, and domain-specific analyses for architectures that conform to specific styles.

Challenges:

- To find a process that can support reveiling software architecture within a sub system
- The level of automization that can be achieved in this process
- The restrictions on the process of architecture recovery(E.g., Recovering all design decisions)

II. EXISTING METHODS

A. *Appriori algorithm*

In graph based architecture recovery methods, finding the frequent sub graphs from the main graph is the major step. We can use the apriori based algorithm to find the frequent sub graphs. The main disadvantage of apriori based algorithm is that it requires a candidate generation step and it creates the significant overhead while joining two size-k and size-k+1 subgraphs.

B. *Approximate graph matching algorithm*

A graph pattern matching can be described as software architecture recovery because both uses recursive graph equations that match to an iterative graph matching process and in graph matching process ,an architecture is represented as a graph where nodes represent modules and edges represent the relationships among these modules..In this process,we can consider two graphs G_1 and G_2 and a function f that maps the nodes and edges of G_1 to the nodes and edges of G_2. As a result of graph matching process which is exploratory in nature, we get a pattern graph which is not the final graph in the proposed software architecture recovery. Here we generate all the possible sub graphs of main graph and we find the exact matrix between pattern graph and a subgraph.The pattern-graph represents a macroscopic view and structural constraints for a part or the whole of the system architecture to be recovered. The goal is to find a sub graph that matches the pattern graph. The source graph provides the search space for matching process. This search space is divided into sub spaces using data mining techniques and each sub space is a sub set of main search space.

This Search Algorithm generates a search tree that corresponds to the recovery of each module M_i in AQL(Architectural Query Language).It

It Consists of a

i) root node for matching the main seed of the Source region with the first place holder $n_{i,1}$ in the pattern region G_i^{pr}.

ii) A number of non-leaf tree-nodes at different levels of the search-tree that correspond to different alternative matching of the place holders in the Pattern region with nodes in the source region

iii) Leaf tree-nodes that correspond to solution paths where the placeholders have been matched and constraints have been met.

A place holder is set by each phase for the process of matching by the search tree which is divided into number of phases. This helps us to manage the complexity matching process of a large graph,which is divided into K incremental phases so that the recovery process performs a Multiphase matching. Each Partial Matching at phase i where i takes values from 1,2,3,……k generating a search tree which is a part of Multiphase Search space .

In this algorithm, the result of previous phase which is stored in queue is discarded by storing the result of current phase. The algorithm should back track by doing one of the following three steps if the Current phase i of the matching process fails to identify a matched graph G_i^m ---

i) Discarding the result that was stored queue in its previous phase G^m_{i-1}

ii) Restoring the search tree for previous phase i-1

iii) Expanding the search tree to find another solution G^{m1}_{i-1}

iv) Advancing to the current phase i and generating a new search tree from G^{m1}_{i-1}

In the Nth phase of this algorithm, we are backtracking to the root n times. Hence the complexity increases by exponential order.

C. Drawbacks of existing methods

The drawbacks of existing system are as follows:
Tools taking document and source code as input require huge calculations and hence require huge processing power of CPU
Tools using appriori algorithm have overhead of candidate generation step
Tools using approximate graph matching algorithm increases the complexity to exponential order.
The user require a prior knowledge of architecture to specify the query for required architectural components
Sometimes we may have only a dll or exe as source of input for recovering the architecture.

III. Proposed System

In proposed system, we extract all the unique methods, variable or classes or terms from a dll file or an executable file using the concept of reflection. **Reflection** uses the type system. Every compiled C# program is encoded into a relational database—this is called metadata. With reflection we act upon the data in this special database. Astonishing features rely on System. Reflection. Structurally, metadata is a normalized relational database. This means that metadata is organized as a set of cross-referencing rectangular tables. The main value of Reflection is that it can be used to inspect assemblies, types, and members. It's a very powerful tool for determining the contents of an unknown assembly or object and can be used in a wide variety of cases.

Then we use the levenshtein algorithm to compute the similarity of each token with respect to all other tokens and construct a cost matrix which is used as input to bipartite matching algorithm to find the score. Thus we use the bipartite matching and similarity measure to find the final architecture of the assembly.

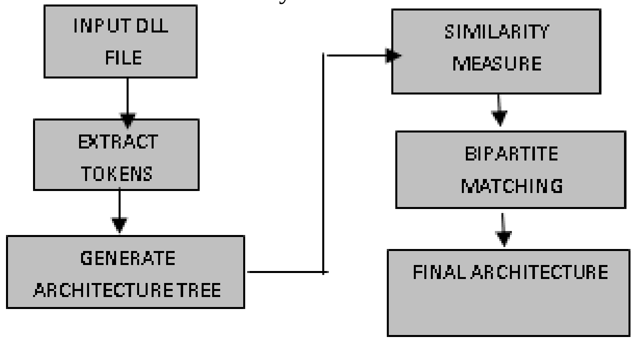

Fig 1: Proposed system

In the proposed system, the important step is to find the similarity measure of each token with respect to all other tokensAn interesting method where strings are represented as a graph which is defined as G = (V, E), where V (V1→ partitions of first string V2→partitions of second string)are nodes representing the secondary structural elements(partitioned tokens) of the strings and E are edges representing the connections between the various partitions of tokens in the string. We use bipartite graph matching technique, where secondary structure elements of two strings (tokens) A and B are represented as nodes in weighted bipartite graph. An edge is defined as degree of similarity between two nodes each from different string. The weight of edges is calculated using levenshtein distance (The distance between two strings reflects the number of prescribed edit operations that are required in order to transform one string into the other) between strings and is represented as cost matrix in the proposed system. Then the similarity is found by choosing such a set of edges, which has maximum weight. In other words, the bipartite matching algorithm calculates the score by traversing the edge which has maximum cost (edge between two strings which are most similar i.e., higher cost in cost matrix).The score is 1 if strings are same, otherwise score is positive value less than 1.Since there is no back tracking as in approximate graph matching algorithm, the complexity is decreased to linear order. Finally, the average score is calculated by adding score of each token with respect to all other tokens divided by number of tokens and displayed along with token. This score represents the similarity measure of each token with respect to all other tokens. Thus, tokens having similar name will have nearly similar score.
The proposed system uses the two algorithms.

- Levenshtein algorithm
- Bipartite matching algorithm

Levenshtein algorithm

Algorithm Name: Levenshtein algorithm
Input : A pair of strings (extracted tokens)
Output : Similarity measure of strings in terms of number edit operations to transform one string to another.
Description : Select a pair of tokens from architecture tree and find the similarity measure. This is done on every possible pair of tokens from architecture tree.

Bipartite matching:

Algorithm Name: Bipartite matching
Input : A pair of strings (strings) and their corresponding cost matrix
Output : Similarity score between given pair of strings based on cost matrix.
Description : Accepts the given input and represents it as graph where represent weight of edges.The algorithm traverses the edge that has maximum cost (maximum similarity) to find the final similarity.

In the next step, the proposed system selects the tokens with maximum score as most significant or most related components of the architecture. The similarity measure of one token with respect to all other tokens is displayed as score in the previous module. This module will select the tokens based on this score. It is possible to recover the required percentage of tokens by adjusting the value in the source code. The tokens having similar names will be displayed near to each other.

IV. CONCLUSION

Architecture recovery occurs much later in a program's lifecycle, after it has been deployed for long enough that many of the original developers are no longer around or have forgotten many details that drove the original development, including the program's underlying architecture and the rationale for it. If the program must now be changed in some way, the changes must respect the forgotten architecture. Therefore, it is necessary to recover the program's architecture and the rationale for the architecture from a detailed and thorough examination of the program's code and any other available related artifacts. This recovery is very much detective work, relying on intuition and experience about how code, in general, works and some lucky discoveries. The sources of information can be any of the following: the executable file of the previous project, the past versions of the code, comments in code, documentation about the code.

In this paper I have used a bipartite graph matching technique to recover the Software Architecture. It is more efficient than Apriori algorithm. To avoid the overheads incurred in Apriori algorithm, we used a non Apriori based algorithm, to recover the Software Architecture and experimental results have shown that the recovered architecture from developed system is more efficient than Apriori based. This algorithm runs in linear time which improves the performance in terms of iterations when compared to A* algorithm.

Thus in this work we present a method that obtains the architecture of the software from its executable file and uses multiple parameters such as similarity measure and bipartite matching algorithm to identify most significant components of the architecture.

V. REFERENCES

[1] Graph theory with applications j.a.bondy and U.S.R. Murthy.
[2] Planar graph drawing T.Nishizeki .
[3] Discrete mathematics and its applications Series Editor KENNETH H. ROSEN.
[4] "Improving the software architecture through fuzzy clustering technique",shaheda akthar sk.md.rafi ,ijcse .
[5] A.Inokuchi,T.Washio and H.Motoda, "an Apriori based algorithm for mining frequent substructures from graph data", In PKDD' 00,2000 .
[6] D.R. Harris, H. B. Reubenstein, and A. S. Yeh, " Reverse engineering to the architectural level",in Proceedings of the 17th ICSE, pages 186–195, 1995.
[7] Jun Haun, Weiwang,Jan Prins,Jiong Yang, "SPIN: Mining Maximal frequent Subgraphs from Graph Databases".
[8] Kamran Sartipi and Kostas Kontogiannis , "A user-assisted approach to component clustering", Accepted for the Journal of Software Maintenance: Research and Practice(JSM), 2002.
[9] Kamran Sartipi and Kostas Kontogiannis, " Interactive software architecture recovery: An incremental supervised clustering approach". Technical Report WE&CE#2002.
[10]Kamran Sartipi, Kostas Kontogiannis, and Farhad Mavaddat., "A pattern matching framework for software architecture recovery and restructuring". In Proceedings of the IEEE IWPC, pages 37–47, Limerick, Ireland, June2000.
[11]Sartipi.k, " Software architecture recovery based on pattern matching ".
[12]Sartipi, "On Modeling Software Architecture Recovery as Graph architecture".
[13]Venkatesh Karthik Srinivasan,Thomas Reps, "Software-Architecture Recovery from Machine Code".
[14]Pavankumar kolla, kolla HariPriyanka, R Manjula**,** "Software Architecture Recovery through Graph Mining Technique and Reduction of Complexity Involved in A* Algorithm".