Clustering Object-Oriented Software Systems using Spectral Graph Partitioning

Spiros Xanthos
University of Illinois at Urbana-Champaign
201 North Goodwin
Urbana, IL 61801
xanthos2@cs.uiuc.edu

Abstract

In this paper we propose a novel method for identifying dense communities of classes (clusters) within an object-oriented software system. These clusters suggest the decomposition of the software system into smaller subsystems. Also, such clusters can imply relevance of functionality and thus be used as reusable components. To accomplish the identification we employ a technique from algebraic graph theory known as spectral graph partitioning. We create a graph representation of the object-oriented software system in which nodes stand for the classes and the edges stand for the discrete messages exchanged between the classes. Our approach is based on an iterative method for partitioning the graph in order to identify dense communities of classes.

1 Introduction

As it is well known, it is difficult to understand the functionality of large software systems because of their inherent complexity. It is even more difficult to extend or modify them. So, people working with large and complex software systems tend to break them down into smaller pieces [1] that are easier to comprehend. Software Clustering is a very important facility when trying to decompose an existing system into smaller parts. It helps identify the subsystems that have related functionality and are somehow independent from the other parts of the system. Therefore, the starting point in the process of reverse engineering a software system, must be to identify the clusters that constitute the system.

1.1 Object Oriented Software

In the object-oriented domain an essential principle, which is also a golden rule in designing reusable software, is that of modularity. A module, in a software system, is the single unit of the application design. Each module is comprised of a small number of classes [2, 3] that are strongly coupled. Modules must have a simple interface through which they can interact with other modules. Also coupling between modules should be loose [4, 5] in order to preserve their autonomy.

This paper presents a method for analyzing object oriented software systems trying to identify highly coupled communities of classes. Utilizing this we can obtain the modules that form the system. Also this method can identify clusters that are autonomous and might possibly imply reusable components. Finally this method can estimate the degree of modularity in a software system by recognizing the individual modules that constitute the system. These are accomplished by applying Algebraic Graph Theory techniques in the object oriented software

domain.

1.2 Background and Related Work

Software Clustering has been used in the past, mainly as an aid in the reverse engineering process of software [6, 7, 8]. These approaches are trying to identify the clusters of a system by utilizing information about the dependencies between the different parts of the system. Other research approaches of the software clustering problem have been used pattern matching techniques [9] and concept analysis [10].

Spectral Graph Partitioning techniques first appeared in the early seventies in the research work of Donath - Hoffman [11] and Fiedler [12, 13]; they explored the properties of the algebraic representations of the graphs (Adjacency Matrix, Laplacian Matrix) and introduced the idea of using eigenvectors to partition graphs. Renewed interest has been generated more recent and these techniques have been applied to many research disciplines, related to Computer Science and Electrical Engineering. Hendrickson and Leland used Spectral Graph Partitioning in parallel systems [14]. Simon et. al. used similar techniques in Scientific Computing [16, 15].

The innovation of this paper is the use of Spectral Graph Partitioning Techniques in the object oriented domain and the applications of these techniques for decomposing an object oriented system into smaller modules, some of which might be used as reusable components. Our method is based on spectral bisection techniques of Barnard and Simon [15], but these are generalized in several ways. Firstly, our algorithm doesn't try to partition the graph in two equal sub-graphs in each iteration, but it tries to achieve the minimum-cut set. Secondly, our algorithm doesn't stop when achieves a certain, predefined number of clusters but when it finds the set of the most cohesive modules, no matter which is their number.

2 Spectral Graph Theory

Define a graph, $G = (V, E)$ with vertex set V of cardinality n and edge set E. Let v_i denote the vertex with index i and e_{ij} denote an edge between v_i and v_j. The adjacency matrix of G is the $n \times n$ matrix $A = [a_{ij}]$ where a_{ij} is the weight of the edge e_{ij}. The degree matrix of G is the $n \times n$ matrix $D = [d_{ij}]$ defined by:

$$d_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

The Laplacian matrix of G is the $n \times n$ symmetric matrix

$$L = D - A$$

. The smallest eigenvalue of the Laplacian matrix is 0, with an associated eigenvector with all ones; in fact the multiplicity of 0 as an eigenvalue of L is equal to the number of connected components of G. If G is connected all other eigenvalues of L are greater than 0 [17].

Fiedler [12, 13] explored the properties of the eigenvector x_2 associated with the second smallest eigenvalue λ_2 (they are known as Fiedler vector and Fiedler value respectively). The elements of x_2 stand as weights on the corresponding vertices of G such that differences of elements provide information about the distances between the vertices. According to this we can split graph G in two other graphs by putting in the first graph the vertices whose corresponding elements in Fiedler vector have values smaller than 0 and in the second graph the vertices whose corresponding elements in Fiedler vector have values greater than 0. This partition of the graph minimizes the total weight of the edge cut. λ_2 is also called the algebraic connectivity of G [12]. Based on this we can state an algorithm for the spectral bisection of a graph. G is partitioned in two subgraphs N^+ and N^- using its Fiedler vector:

```
step 1:
x_2 ← fiedler eigenvector of the graph G
step 2:
for each vertex v_i of G
if x_2(v_i) < 0
    N^- = N^- ∪ v_i
  else
    N^+ = N^+ ∪ v_i
  end if
end for
```

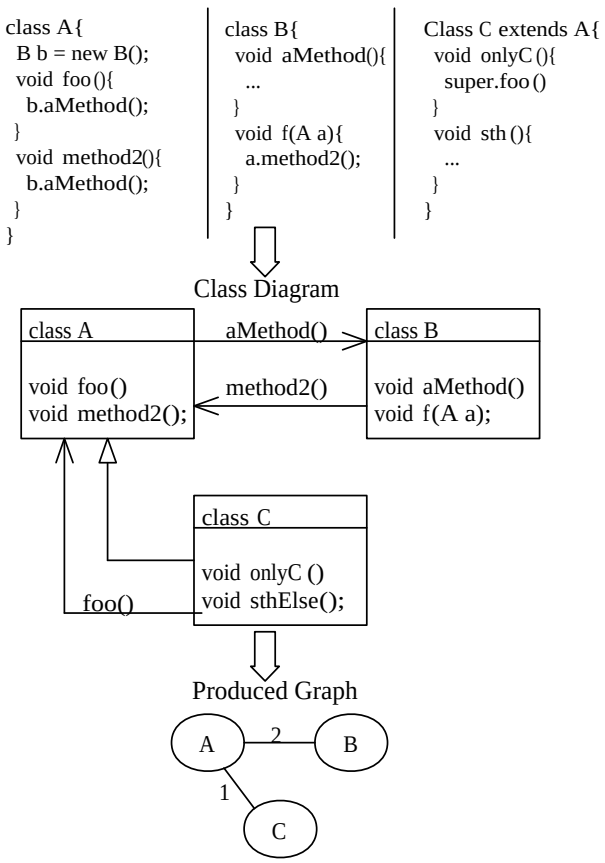

Figure 1: Creating the Graph Representation

3 Algorithm

From the class diagram of the software system we create a graph representation in which the vertices stand for the classes and the the edges stand for **discrete messages** exchanged between the classes. This can be seen in ***Figure 1***.

Then, the produced graph is bipartitioned iteratively: in each iteration, we partition the graph into two sub-graphs. The iteration stops when at least one of the produced sub-graphs is less cohesive than their parent graph. This is determined by examining if the number of internal edges of each sub-graph (those between the vertices of the sub-graph) exceeds the number of external edges (those between the vertices of the sub-graph and all the other vertices) (Alg. 2).If the external edges are more than the internal the algorithm stops. This strategy for breaking the iteration has been found through experiments to be the optimum. At the end the modules of the system are the sub-graphs that have been found by the algorithm.

$G \leftarrow$ Graph of the application
partitionAlgorithm(graph G)
  $x_2 \leftarrow G$ Fiedler Vector
  $G_1, G_2 \leftarrow$ partition(x_2)
  if(highlyCohesive(G_1, G_2))
    partitionAlgorithm(G_1)
    partitionAlgorithm(G_2)
$*$partition(x_2) *returns the two subgraphs of* G *according to* x_2
$*$highlyCohesive(G_1, G_2) *determines whether the two subgraphs are more cohesive than their parent graph.*

Alg. 1: The Partitioning Algorithm

$\sum_{k=1}^{n} v_{ij} : i, j \in E_{in} < \sum_{k=1}^{n} v_{im} : i \in E_{in}, m \in E_{out}$
where
V_{ij} : *Edge from vertex* i *to vertex* j
E_{in} : *The vertices that belong to the graph.*
E_{out} : *The vertices that do not belong to the graph.*

Alg. 2: Determine if a graph is highly cohesive

3.1 Example

In ***Figure 2*** a full application of the algorithm is presented:

1. The class diagram of the application is created .
2. The class diagram is converted to a graph.
3. The graph is partitioned in two subgraphs.
4. The one of the two subgraphs is then partitioned again.
5. The algorithm stops because the three clusters that have been found are more cohesive than their possible subgraphs.

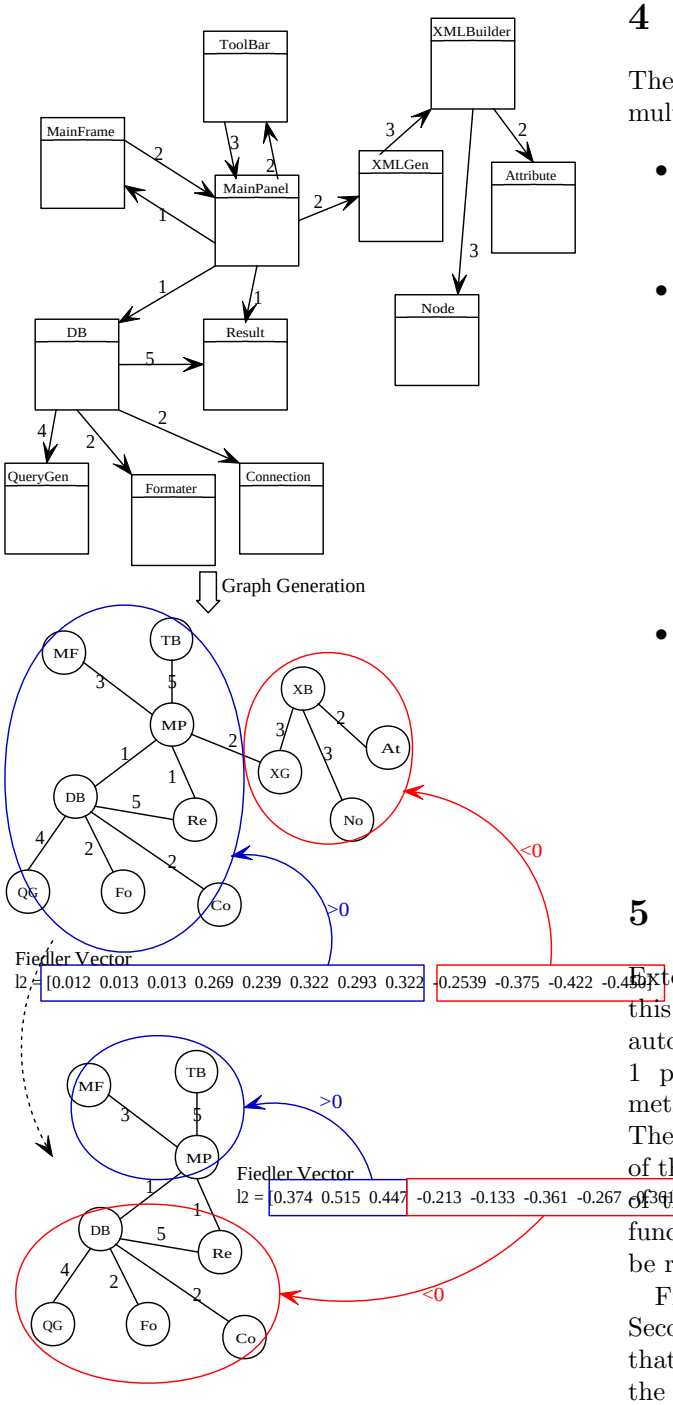

Figure 2: Application of The Algorithm

4 Results and Contribution

The results produced by the algorithm can have multiple uses:

- The modules that are found can be used as the starting point for the reverse engineering process of a software system.
- During the process of identifying each module, the edges that connect the vertices of the module with the rest of the graph are removed. These edges are in fact the interface of the module with the other parts of the application. So, beyond the identification of the modules themselves our methodology identifies the Facade [18] of each of them. Having the Facade of a module is easier to transfer and use it in another system (reusability).
- Our methodology minimizes the communication between the modules of the system. This is achieved because it finds the minimum cut set of edges in each iteration. Therefore, it can also be used to identify the modules that should be assigned in different machines, in a distributed environment.

5 Validation

Extended measures that confirm the correctness of this methodology have been made. Also a tool that automates the process has been developed. Table 1 presents the results of the application of the methodology in some open source Java programs. The modules produced by our tool are in the most of the cases highly cohesive. Also the classes in some of these modules have been found to have relevant functionality, an indication for a module that could be reused.

First column holds the names of the programs. Second column has the number of modules-cluster that have been identified in each of the programs and the third column has the number of those modules that has been found to be highly cohesive. A module is considered highly cohesive when the messages

Program	Modules	Highly Cohesive
Pygmy	2	2
JFlex	3	3
classycle	12	8
JSpider	21	8
Apache Ant	44	26

Table 1: Evaluation Results

extended between its classes exceed the number of messages exchanged between its classes and the rest of the system by a factor of three or more.

References

[1] D. Parnas. *On the Criteria To Be Used in Decomposing Systems into Modules.* CACM, 15 (12), 1972.

[2] Bertrand Meyer. *Object Success A Managers Guide to Object Orientation, its Impact on the Corporation, and its Use for Reengineering the Software Process.* Prentice Hall, 1995.

[3] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard. *Object-Oriented Software Engineering A Use Case Driven Approach.* Reading, MA, USA / Wonkingam, England, Addison Wesley / ACM Press, 1992.

[4] C. Ghezzi, M. Jazayeri, D. Mandrioli. *Fundamentals of Software Engineering.* Englewood Cliffs, New Jersey, USA, Prentice Hall, 1991.

[5] Clemens Szyperski. *Component Software.* Addison Wesley, 2002.

[6] H. Muller, M. Orgun, S. Tilley, J. Uhl. *A reverse engineering approach to subsystem structure identification.* Journal of Software Maintenance: Research and Practice, 1993

[7] S. Choi, W Scacchi. *Extracting and restructuring the design of large systems.* IEEE Software, 1990

[8] R. Schwanke. *An intelligent tool for re-engineering software modularity.* International Conference on Software Engineering, 1991

[9] K Sartipi, K Kontogiannis. *A graph pattern matching approach to software architecture recovery* International Conference on Software Maintenance, 2001

[10] M. Siff, T. Reps. *Identifying modules via concept analysis.* Proceedings of the 19th International Conference on Software Maintenance, 1997

[11] 7] W.E. Donath and A.J. Hoffman. *Lower bounds for the partitioning of graphs.* IBM Journal of Research and Development, 17, pp. 420-425, September 1973.

[12] M. Fiedler. *Algebraic connectivity of graphs.* Czechoslovak Mathematical Journal, 23(98), pp.298-305, 1973.

[13] M. Fiedler. *A property of eigenvectors of non-negative symmetric matrices and its applications to graph theory.* Czechoslovak Mathematical Journal, 25(100), pp. 619-633, 1975.

[14] B. Hendrickson R. Leland. *An improved spectral graph partitioning algorithm for mapping parallel computations.* SIAM Journal on Scientific Computing. Volume 16 , Issue 2, March 1995

[15] S. Barnard and H. Simon *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems.* Proceedings of the 6th SIAM conference on Parallel Processing for Scientific Computing, 711-718, 1993.

[16] A. Pothen H. Simon K. Liou. *Partitioning sparse matrices with eigenvectors of graphs Source.* SIAM Journal on Matrix Analysis and Applications, Volume 11, Issue 3, July 1990.

[17] B. Mohar. *Some applications of Laplace eigenvalues of graphs.* Algebraic Methods and Applications, volume 497 of NATO ASI Series C, pages 227-275. Kluwer, Dordrecht, 1997.

[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* Massachusetts, Addison-Wesley, 1995