Hierarchical Clustering of Distributed Object-Oriented Software Systems: A Generic Solution for Software-Hardware Mismatch Problem

AMAL ABD EL-RAOUF
Computer Science Department
Southern Connecticut State University
501 Crescent St., New Haven, CT 06515
USA
abdelraoufa1@southernct.edu

Abstract: During the software lifecycle, the software structure is subject to many changes in order to fulfill the customer's requirements. In Distributed Object Oriented systems, software engineers face many challenges to solve the software-hardware mismatch problem in which the software structure does not match the customer's underlying hardware. A major design problem of Object Oriented software systems is the efficient distribution of software classes among the different nodes in the system while maintaining two features: low-coupling and high software quality. In this paper, we present a new methodology for efficiently restructuring Distributed Object Oriented software systems to improve the overall system performance and to solve the software-hardware mismatch problem. Our method has two main phases. In the first phase, we use the hierarchical clustering method to restructure the target software application. As a result, all the possible clustering solutions that could be applied to the target software application are generated. In the second phase, we decide on the best-fit clustering solution according to the customer hardware organization.

Key-Words: - software restructuring, hierarchical clustering, distributed systems, object oriented software, performance analysis, low coupling.

1 Introduction

One of the important advantages of software restructuring techniques is providing solutions for the software-hardware mismatch problem in which the software structure does not match the available hardware organization. In such class of problems, the solution is possible through two approaches; either to configure the hardware to match the software components (hardware reconfiguration), and/or to reconfigure the software structure to match the available hardware by reorganizing its components (software restructuring). The first approach is impractical especially in complex programs containing many interacting modules (or subtasks). The second approach is more practical especially in computing environments that contain a large number of users. It provides an optimal way to use the available system capabilities, reduce the overall computational cost, and improve the overall system performance.

The basic idea of distributed software restructuring techniques as introduced in [1] is to select the best alternative structure(s) for a constrained computing environment while reducing the overall resources need. These structures can be created through; granularity definition (merging of modules), alternative modules ordering, loop decomposition, or multiple servers support.

In Distributed Object Oriented systems, classes are the main units. Classes represent abstraction that should make adapting software easier and thus lower the cost of reuse, maintenance and enhancement [2]. These classes interact to form a functioning system. This kind of interaction results in a communication cost; a function call is a source of data communication.

On the other hand, the object oriented paradigm as described in [3] is based on several concepts such as inheritance, aggregation and association that produce complex dependencies between classes. That is why it is a challenge in the Distributed Object Oriented systems to create subsystems with low coupling and high cohesion as quality matrices for good design [4].

Clustering techniques aim to group the fine-grained active objects into clusters with the goal minimizing the interconnectivity between the different clusters while maximizing the interconnectivity inside the same cluster in order to improve the software quality.

In this paper, we introduce a new methodology to restructure distributed object oriented software systems. Our methodology is more appropriate for software companies that develop distributed object oriented software applications while the customer hardware platform is unknown. When the software is purchased, the development engineer would customize the system to fit the customer requirements. S/he would be able to use our technique to pick the level in the hierarchy that has the number of clusters matches the number of nodes in the customer hardware platform.

The rest of this paper is organized as follows: the second section presents related work and existing approaches in the field of restructuring software systems. Section three states the problem definition, including assumptions and our goals. Section four describes the analytical distributed object oriented performance model that we utilized to generate the communication matrix. Section five describes the hierarchal clustering technique used with our approach. In section six, we support our methodology with a case study. Finally, the last section draws our conclusions and potential future work.

2 Related Work

Fergany in [1] defines software restructuring as follows:

> "Software restructuring is the process of selecting an allowable order of subtasks (modules) which meet user and system performance requirements while achieving minimum total resource cost".

Fergany introduces restructuring techniques for distributed software, in which the best alternative structure(s) for a constrained computing environment is selected while reducing the overall resources need. Fergany created her different structures through what she called granularity definition. Granularity definition aims to merge relatively small modules into larger ones. Other structures are created using alternative modules ordering, loop decomposition, or multiple servers' support as shown in the work introduced in [5]. In this work, it has been shown that performing software restructuring ahead of allocation and scheduling phases improved the results obtained from these phases and reduced the overall resources cost. However these techniques are not targeting distributed object oriented systems.

In the literature, the work in the software restructuring field targeting distributed object oriented systems can be categorized into three main categories. The first category aims to re-define the granularity of the system to provide better overall performance. Different researchers used different granularity levels (modules vs. classes, procedures, fields), that is why it is difficult to compare the different restructuring methods. The second category computes a measure of similarity/dissimilarity between software grains. In this category, each measure used would lead to a different structure of the target software application. Finally the third category looks at the software restructuring as an optimization problem. In order to solve the optimization problem, some researchers define an objective function and some use well-known optimization techniques.

In [6], Jain et al review different techniques for data clustering. This paper presents an overview of pattern clustering methods from a statistical pattern recognition perspective, with a goal of providing useful advice and references to fundamental concepts accessible to the broad community of clustering practitioners.

In [7], Bellur et al present a method to group the fine-grained active objects into clusters while minimizing communications among them. Some of the clustering approaches consider the cost of object update as a part of the communication as in [8]. In [9], the author introduce a parallel programming environment, called distributed object-oriented virtual computing environment (DOVE), for clustered computers based on distributed object model.

Other work in clustering considers automatic clustering approach. Early work is done by Schwanke. In [10], Schwanke uses the shared neighbors' technique to solve the problem of automatic clustering. In his method he uses heuristic algorithm in order to capture patterns that commonly appear in software systems. In [11], Schwanke introduce a tool that applies reverse engineering to the software system in order to obtain better software modularity. The granularity is at the level of "the procedure". The outputs of his method are modules that have procedures referencing same names, or according to Schwanke "design coupled Procedures".

Mancoridis et al. present in [12] a methodology to facilitate the automatic recovery of the modular structure of a software system from its source code. In [13], Mancoridis et al introduce their clustering tool "Bruch" that is implemented with the goal of software recovery and allowing incremental software structures maintenance. In [14], Mancoridis et al use "genetic algorithms" as an optimization technique to solve the automatic clustering problem and to avoid the local optima of the hill-climbing algorithms. The objective was to maximize the interconnectivity within each of the generated clusters.

Another restructuring approach can be found in the literature is to re-define the granularity by splitting modules in order to get finer size. This approach is called *Partitioning* as the work introduced in [15, 16, 17].

In addition, the work presented in [16, 17, 18] consider real-time application. The granularity is at the level of "tasks". Tasks may violate the deadlines; in this case a factor is added to the objective function. This factor is represented by a penalty of missing the deadline and should be minimized.

Other restructuring techniques consider clustering granularity at the level of the "method". As a result methods could be moved from one cluster to another according to specific criteria, examples are shown in [19, 20]

In [21], Welch et al introduce a very promising technique using automatic partitioning within concurrent programs. The partitioning process is performed by constructing a call-rendezvous graph (CRG) for an application program. The CRG is augmented with edge weights depicting inter-program-unit communication and concurrency relationships. The partitioning algorithm has the goal of producing a set of partitions among which there is a small amount of communication and there is a large degree of potential for concurrent execution.

Work from the literature that falls into the second category is introduced in [22]. Tzerpos et al address the problem of software clustering, by defining metric that can be used in evaluating the similarity of two different decompositions of a software system. His metric calculates a distance between two partitions of the same set of software resources. To calculate the distance, he used a heuristic algorithm that computes the minimum number of operations needed in order to transform one partition from one cluster to another.

Tzerpos and Holt present in [23] a software clustering algorithm in order to discover clusters based on subsystem patterns that are commonly observed in decompositions of large software systems. Moreover, the work introduced in [24] also uses objective function in the restructuring method. The objective function aims to minimize the amount of communication among classes, the processing power fragmentation in the processors on different nodes and the penalty factor of missing the deadline.

Other work consider restructuring distributed object oriented software targeting specific hardware architecture, an example is the pipeline architecture used in [25].

In our previous work [26, 27], we present a two-phase methodology for efficiently restructuring the distributed object oriented software systems. The first phase introduces a recursive graph clustering technique to partition the OO system into subsystems with low coupling. Then we faced the problem of software-hardware mismatch as the resultant number of clusters is not matching the available number of the distributed nodes. Hence we needed a second phase of clustering to map the generated clusters to the set of available machines in the target distributed architecture.

In this paper, we propose a new restructuring methodology that solves the software-hardware mismatch problem. We use the hierarchal clustering technique in which the data are not partitioned into a particular number of clusters. Instead, the result is a series of partitions, which may run from a single cluster containing all objects to a number of clusters each containing a single object. In the second phase, we just need to find the level of the hierarchy that has the number of clusters equal to the number of available machines in the target distributed architecture.

3 Problem Definitions

In this paper, we consider restructuring distributed object oriented applications for mapping on a distributed system. The restructuring process is the process of mapping the distributed object oriented application classes to the different network nodes in order to attain better performance.

To achieve this goal, we investigate the possibilities of merging heavily related classes to identify clusters of a dense community within the distributed object oriented system.

For any clustering methodology, challenges are to decide about two important aspects: the clustering granularity level and the measure of similarity/dissimilarity between software grains.

In our method we have the granularity at the level of "classes". In addition, we opt the communication time to be the measure of similarity among the different classes in the software application.

The main objective is to propose a group of sub-systems, each has maximum communication cost among the inner-classes and the communication cost among the sub-systems is minimized. This helps in composing the system into clusters that have low coupling and better overall system performance. In order to achieve this objective, we used the MinMax algorithm described as follows:

- Given n data objects and the pairwise similarity matrix D = (dij)
  where dij is the similarity between class i; and class j, in our case it represents the communication matrix.
- We start to merge the two clusters C1, C2 with the maximum similarity using the min-max clustering principle. The similarity between C1,C2 is defined to be:
  $$S(C1,C2) = \sum_{i\in c1}\sum_{j\in c2} dij$$
  S(C1,C2) is also known as the overlap between C1 and C2.
- The similarity within a cluster C1 is the sum of pairwise similarities within C1: S(C1,C1). Therefore, S(C1,C1) represents the self-similarity of cluster C1.
- The clustering principle requires minimizing S(C1,C2) while maximizing S(C1,C1) and S(C2,C2) simultaneously.
- These requirements lead to the minimization of the MinMax objective function:

$$F_{MM}= S(C1,C2)/S(C1,C1)+S(C1,C2)/S(C2,C2)$$

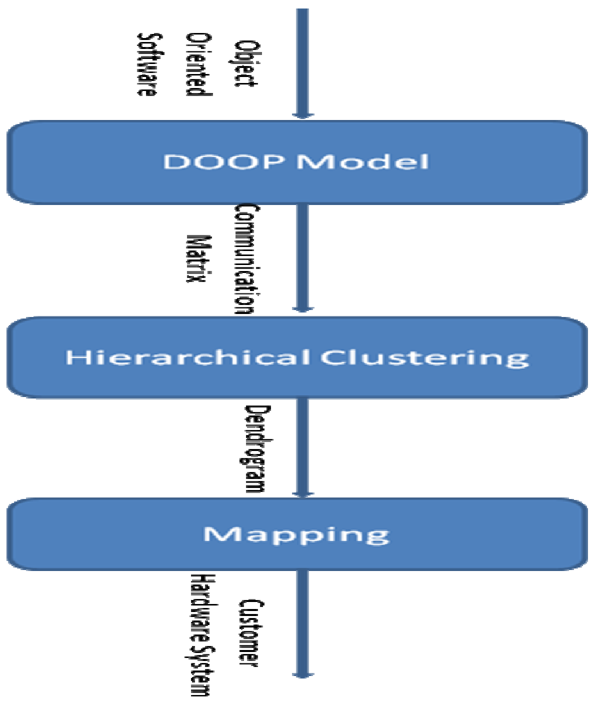

Figure 1: Restructuring Methodology Stages

The next challenge is to accurately compute our clustering metric. In our method, we used the distributed object oriented performance model (DOOP) to calculate the communication time among all classes in the application represented in the communication matrix as described in the following section.

Then, the Hierarchical clustering technique utilizes the generated communication matrix to build a dendrogram or a tree. The dendrogram is a hierarchal graph that consists of a number of levels. Each level has a number of clusters. Moving from one level to the next in the dendrogram, results in a coarser clustering, with a smaller number of larger clusters.

When it is time to map to the target distributed architecture, the solution is to find the appropriate level in the dendrogram that has the same number of clusters as the number of the available nodes in the target architecture.

The different stages of the methodology together with the input and output of every stage are shown in figure1.

4 Generating the Communication Matrix Using the DOOP Model

In order to decide which classes should be combined, a measure of similarity between classes is required. In our Restructuring methodology, the merging decision of two classes is based on the communication time cost between these two classes. Therefore, we need to generate a communication matrix as shown in figure 2.

Figure 2 shows the communication matrix of a software application that includes five different classes (5×5 matrix). Each element of the matrix represents the communication cost between two classes in the object oriented software application. For example (d23) is the communication cost between class 2 and class 3. The communication cost between a class and itself (dii) is always equal to zero.

0	d12	d13	d14	d15
d21	0	d23	d24	d25
d31	d32	0	d34	d35
d41	d42	d43	0	d45
d51	d52	d53	d54	0

Figure 2: the Communication Matrix

In distributed object oriented systems, accurate calculation of communication time cost is a challenge due to the dependency among classes, the frequent remote requests and the decentralization of the functionality. Most approaches to evaluate communication time in distributed object oriented systems are based on either the system measurements after its development which is very expensive approach or mapping to a conventional performance model that would add an extra layer to the analysis phase.

In [28], the Distributed Object Oriented Performance (DOOP) model was introduced. The DOOP model analyzes and evaluates the overall time cost considering the communication overheads while preserving the features, properties and relationships between objects. According to the model, each node in the distributed object oriented system will be represented as shown in figure 3. The performance model consists of two main parts: the execution server and the communication server. The major components of the model are described in the following subsections.

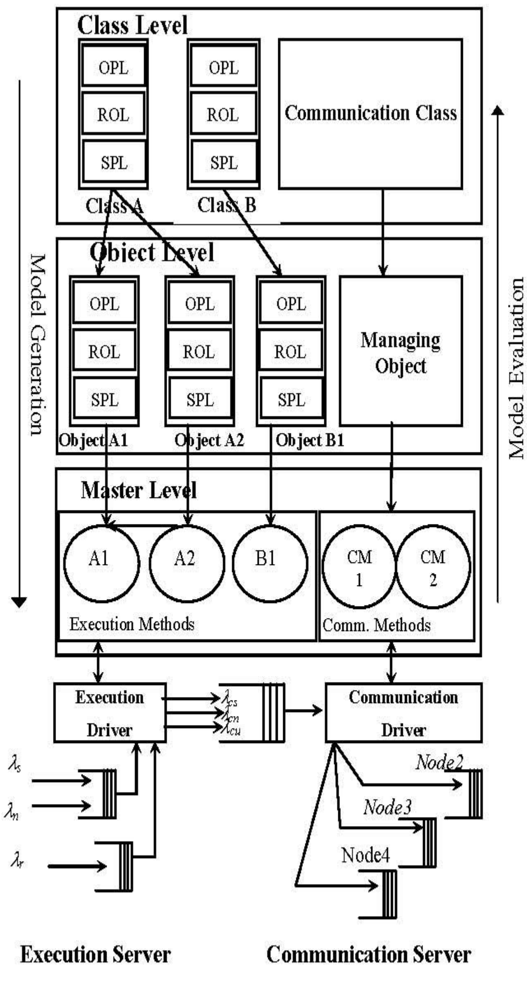

Figure 3: The DOOP Model Node Structure

4.1 Input Queues

Different arrivals enter to the input queues of the execution server at different rates according to their type. There are three categories of arrivals: first, the external user request (EUR), which submits a request to execute the entire software driver. Second type of arrivals is Remote Request (RR), which is a request from another node in the distributed system to perform a computation activity within this target node. The third type is response, which carries information representing the results of a remote procedure call (RPC) sent earlier by the target node to others.

4.2 Execution Server

Requests arrived to the input queues activates methods through the execution driver. It calculates the execution time cost using all the information in the upper layers of the model, and then forwards the required communication activities to the communication server.

The model has three levels of abstraction to emulate the natural way of handling the design of an object oriented software application: first, the class level to identify the classes in the system, each with its own attributes and methods. The second is the object level to create objects as instances of the existing classes and hold their information. The third level is the master level to execute all the software modules.

The model maintains the structure of the application itself by using the Performance Image (PI) structure. The PI will be automatically generated for all classes and objects defined within the system. The PI itself is a multi-layers representation as shown in figure 3. Each layer holds the performance cost that is partially added in the performance evaluation process. The PI consists of three layers. The first layer is the Object creation Performance Layer (OPL), which is responsible for the performance calculations of object creation. The OPL layer will include the cost equations of all the possible constructors.

The second layer is the Related Classes/Objects Performance Layer (RPL), which includes all the information about related classes/objects by composition, aggregation, inheritance or access relationships. The third layer is the Service Performance Layer (SPL), in which CSM (Computation Structure Model) defined in [29] is used to generate a cost equation for each method in the system.

All cost equations contribute in the overall estimated execution time calculations. The detailed evaluation process is illustrated in [28].

4.3 Execution-to-Communication Buffer

The arrivals to the communication buffer or queue are the data and/or information that need to be sent to other nodes within the distributed systems. There are two arrivals categories to the communication queue. First is the portion of the external user request (EUR) that needs to be processed on or propagated to other nodes (i.e. remote procedure call RPC). The second category represents the communication activities due to remote request (RR), which may create a message for another RPC (i.e. nested RPCs), for updating, or sending back the results of the RR.

4.4 Communication Server

The communication server handles all communication activities among nodes in distributed systems. A built-in communication class is generated in the class level; the managing object is an instance of the communication class, which is constructed in the object level as shown in figure 3.

The Communication Class is mainly concerned with the internal and the external communication activities that take place among objects in a distributed object oriented system. It holds information about all the cooperating objects, their locations (node holding each object), number of object copies, the size of data exchange between objects methods, links between nodes, the message size, message multiplier, communication arrival rate and others. This information is used to calculate the cost of communication processes and also the cost of updating the objects. The communication driver will cooperate with higher layers to retrieve the information necessary to calculate the overall communication cost as illustrated in [28].

4.5 Output Queues

The output queues represent the input queues to other nodes in the distributed object oriented system communicating with this node. Communication is performed through a point-to-point connection. The output of the communication server will go through the physical communication link connecting nodes, and then to the corresponding input queue.

4.6 Communication time evaluation

In our restructuring scheme, we utilize the DOOP model in the evaluation process of the communication activities among classes as shown below.

Assume that the overall arrival rate to the communication queue λ_{ck} is given by:

$$\lambda_{ck} = \lambda_{cs} + \lambda_{cn} + \lambda_{cu}$$

where λcs, λcn and λcu represent the communication arrival due to External User Request (EUR), Remote Request (RR), and updating objects' data on other nodes, respectively.

$$\lambda_{cs} = \beta_s \lambda_s , \qquad \lambda_{cn} = \beta_n \lambda_n ,$$

$$\lambda_{cu} = \sum_{i=1}^{N} \lambda_{CUi}$$

Where, β_s and β_n are the message multipliers for EUR and RR. Let λ_{cui} be the arrival rate corresponding to object i data updating.

Since the updating process to an object i occurs due to processing EUR or RR, Pi1 is defined to be the probability that object i is updated due to EUR, Pi2 is the probability that object i is modified due to RR. λ_{cui} can be expressed as:

$$\lambda_{cui} = P_{i1} \lambda_s + P_{i2} \lambda_n$$

Hence, the expected communication service time for each class will be:

$$t_{cs} = \frac{m_s}{R} \qquad t_{cn} = \frac{m_n}{R} \qquad t_{ui} = \frac{m_{ui}}{R}$$

where t_{cs}, t_{cn} and t_{ui} are the expected communication service time for EUR, RR and for update requests from object i. While m_s, m_n and m_{ui} are the expected message sizes of EUR, RR and of sending object i updating data. R represents the communication channel capacity.

Furthermore, the average communication service time for node (k) will be:

$$t_{ck} = P_{cs} t_{cs} + P_{cn} t_{cn} + \sum_{i=1}^{N} P_{ui} t_{ui}$$

$$p_{cs} = \frac{\lambda_{cs}}{\lambda_{ck}} \qquad p_{cn} = \frac{\lambda_{cn}}{\lambda_{ck}} \qquad p_{ui} = \frac{\lambda_{ui}}{\lambda_{ck}}$$

Where P_{cs}, and P_{cn} are the probabilities of activating communication service by the external user requests and by remote request respectively. P_{ui} is the probability of sending object i's data update to other nodes.

In our restructuring method, each individual class is allocated to a separate node and represented by the DOOP model. The above equations are used to compute the average communication cost dij between a specific class i and other classes in the system. The computed values represent the elements of our Communication Matrix.

5 Hierarchal Clustering Technique

In this section we present the agglomerative hierarchal clustering technique that we use in our restructuring methodology.

Hierarchal clustering common methods are: single-link, complete-link and average-link clustering.

In single-link (or nearest neighbor) hierarchical clustering, we merge in each step the two clusters whose two closest members have the smallest distance (or the two clusters with the smallest minimum pairwise distance). The distance between clusters is given by:

$$D(r,s) = \mathrm{Min}\ \{\ d(i,j)\ \}$$

Where object i is in cluster r and object j is cluster s

In complete-link (or farthest neighbor) hierarchical clustering, we merge in each step the two clusters with the maximum pairwise distance. The distance between clusters is given by:

$$D(r,s) = \mathrm{Max}\ \{\ d(i,j)\ \}$$

Where object i is in cluster r and object j is cluster s

In the average-link (or group average) hierarchical clustering method, the distance between two clusters is defined as the average of distances between all pairs of objects.

$$D(r,s) = Trs\ /\ (\ Nr * Ns)$$

Where Trs is the sum of all pairwise distances between cluster r and cluster s. Nr and Ns are the sizes of the clusters r and s respectively. At each stage of hierarchical clustering, the clusters r and s, for which D(r,s) is the minimum, are merged.

Agglomerative clustering is uniquely determined for a given linkage, independent of any objective function.

In the next sub-section we illustrate how to generate the Dendrogram. Then, the following sub-section shows how to use the generated Dendrogram in order to identify the recommended clusters to be mapped to the different nodes of the distributed system.

5.1 Generating the Dendrogram

The input to the hierarchal clustering algorithm is the communication matrix described in section 4 and the output is a tree-like structure or a *Dendrogram*. The Dendrogram is created through recursive merging of the existing object oriented classes.

Generating the Dendrogram is a bottom-up approach; each class starts in its own cluster in the first level. Then, in succeeding steps, during each recursive procedure the two clusters that have the largest pairwise linkage; communication time cost in our case; are aggregated into a combined cluster. In this way, the number of clusters in the distributed object oriented system is reduced by one in each step.

At the top level, all classes in the software application are combined into a single remaining cluster. Since, for n classes there are (n – 1) merges, there are $2^{\wedge(n-1)}$ possible orderings for the leaves in a cluster tree, or a Dendrogram. At each step, the communication costs between clusters are recomputed by the Lance–Williams similarity/dissimilarity update formula given in [30]. This update step could be done by a number of different ways according to the clustering method. Figure 4 shows a detailed step by step description of the hierarchal clustering Algorithm.

```
ALGORITHM   HierarchalCluster(W, CIndx)
INPUT:      W = Communication Cost matrix
            (weights matrix).
            CIndx = the cluster index representing
            the resultant merged cluster after each
            merge process.

OUTPUT:     Matrix G = a matrix in which each
            row indicates the new cluster numbers
            as a result of each merge process, to
            which each node in the distributed
            system belongs.

STEP 1      Let CurrW be the extracted
            Communication matrix of the
            distributed system indicated by
            CIndex.

STEP 2      Merge the CurrW into larger cluster
            CIndx = ClusterMerge(CurrW)

STEP 3      Create Matrix G so each row holds
            the indices of the resultant merged
            clusters.

STEP 4      Update CIndx with new merged
            cluster created in the matrix G.

STEP 5      Update the Communication cost

STEP 6      Recursively Merge the recently
            merged cluster(s) and add the
            resultant merged clusters to a new
            row of the Matrix G.
```

Figure 4: The Hierarchal Clustering Algorithm

5.2 Mapping to the Hardware Architecture

The basic advantage of using the hierarchal clustering that you don't need to determine the number of clusters in advance and this is the case of any software company that develops the OO software product while the customer system is unknown.

The mapping process starts when a customer purchases the software and it should be customized to fit his/her needs. The software engineer would then use the generated dendrogram in a top-down approach. The start is at the top level and if the available number of nodes of the customer distributed system is *(m)*, s/he goes down *(m-1)* levels. The level reached in this step is the *key level.* This key level would have *(m)* clusters that are directly mapped to the *m* nodes in the distributed system.

6 Case Study

We consider applying our restructuring methodology to an object oriented application that consists of 28 classes.

In the first step we used the Distributed Object Oriented Performance (DOOP) model to generate the 27 × 27 communication matrix. This matrix is the input of the hierarchal clustering algorithm.

To conduct our experiment, we used the free open source R that has several functions for hierarchical clustering. Among hclust, Diana, and agnes, we chose the hierarchal clustering method "agnes".

As described in [31], the agnes clustering methods has the following features:

- It constructs a hierarchy of clusters.
- It yields the agglomerative coefficient which measures the amount of clustering structure found.
- In addition to the usual tree, it also provides the banner, which is a novel graphical display.

The command used in R is:

```
plot( agnes( data, diss = FALSE, method = "average" ) )
```

The arguments of the command are set as follows:

- data is the communication matrix generated in the first step.
- Dissimilarity is set to false as we are trying to merge classes with maximum communication time in-between.
- Method used is the average-link method explained in the previous section.

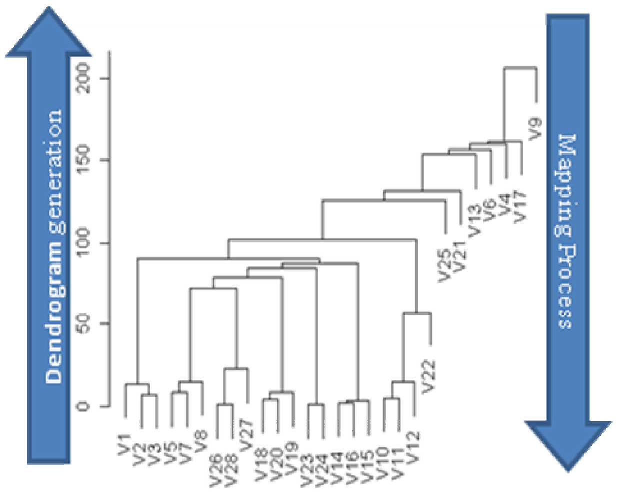

Figure 5: A Dendrogram for clustering 28-classes OO software application

The result of the agnes clustering method is typically visualized as a Dendrogram as shown in Figure 5.

The horizontal coordinate shows the different classes in terms of V_i where i is the class number. Each merge process is represented by a horizontal line.

The resultant Dendrogram is used to map the object oriented application to any customer architecture according to the available number of nodes.

In our case, the customer has 7 nodes available in his distributed system. Starting from the top level, we go down six levels to reach the key level. A horizontal line in this key level would intersect with seven clusters. Table1 shows the OO classes' assignment to the different clusters in the system

Cluster/Node	Classes
1	V9
2	V17
3	V4
4	V6
5	V13
6	V21
7	All other classes

Table 1: Classes assigned to 7-nodes distributed system

7 Conclusion

In this paper, we introduced a new methodology for restructuring distributed object oriented software systems. This new methodology has two main objectives, first: grouping object oriented software classes into clusters that have low coupling to improve the software quality. Second: solving the software-hardware mismatch problem by producing a hierarchical clustering Dendrogram (or tree).

The Dendrogram consists of different levels of hierarchy each with different number of clusters. Hence, the Dendrogram would be used to locate the appropriate level *(key level)* that matches the customer system requirements.

Further work could be done in two directions. First: to use different clustering techniques as in [32, 33] or different data modeling such as the model described in [34]. Second: rather than using the communication time cost as a metric to decide about the merging clusters, we could use other metrics as the metrics presented in [35]. Finally, a comparison between future work results and our approach's results is needed.

References:

[1] T. Fergany, "*Software Restructuring in Performance-Critical Distributed Real-time Systems,*" Ph.D. Thesis, University of Connecticut., USA, 1991.

[2] B. Meyer, *"Object-Oriented Software Construction,"* Prentice-Hall International (UK), Ltd, 1988.

[3] B. Ostereich, "*Developing Software with UML: OO Analysis and Design in Practice*," Addison Wesley, June 2002.

[4] Ian Sommerville, "*Software Engineering*," 8th Edition, Addison-Wesley Publishers Ltd, New York, 2007.

[5] H. Sholl and T. Fergany,"Performance-Requirements - Based Loop Restructuring for Real-Time Distributed Systems," *Proceedings of the International Conference on Mini and Microcomputers, from Micro to Supercomputers*, Florida, Dec. 1988.

[6] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, 1999, pp. 264–323.

[7] U. Bellur, G. Craig, and D. Lea, "Clustering: Composition for Active Object Systems," *Proceedings of the 27th Hawaii International Conference on System Science*, Jan. 1994.

[8] J. Cheng, and A. Hurson,"Effective Clustering of Complex Objects in Object-Oriented Databases," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Denver, USA, May 1991.

[9] W. Chang, and C. Tseng,"Clustering Approach to Grouping Objects in Message-Passing Systems," *The Journal of Object-Oriented Programming*, Vol. 8, No. 6, Oct. 1995.

[10] R. W. Schwanke and M. A. Platoff, "Cross references are features," *Proceedings of the 2nd International Workshop on Software configuration management*, New York, NY, USA: ACM Press, 1989, pp. 86–95.

[11] R. W. Schwanke, "An intelligent tool for reengineering software modularity," *Proceedings of the 13th international conference on Software engineering*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 83–92.

[12] C. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Ganser, "Using Automatic Clustering to produce High-level System Organizations of Source Code," *Proceedings of the International Workshop on Program Understanding*, 1998.

[13] S. Mancoridis, B. S. Mitchell, Y. F. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," *In the IEEE Proceedings of the International Conference on Software Maintenance (ICSM'99)*, Oxford, UK, August, 1999, pp 50-59..

[14] D. Doval, S. Mancoridis and B.S. Mitchell, "Automatic Clustering of Software Systems using a Genetic Algorithm," *IEEE Proceedings of the International Conference on Software Tools and Engineering Practice (STEP'99)*, Pittsburgh, PA, Aug. 1999.

[15] K. Karlapalem, Q. Li, and S. Vieweg, "Method Induced Partitioning Schemes in Object-Oriented Databases," *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96)*, 1996.

[16] T. Fergany, H. Sholl, and R. Ammar, "SRS: A Tool for Software Restructuring in Real- Time Distributed Environment," *In the Proceedings of the 4th International Conference on Parallel and Distributed Computing and Systems*, Oct. 1991.

[17] S.Wathne, H. Sholl, and R. Ammar, "Task Partitioning of Multichannel, Distributed, Real-Time Systems," *ISCA 8th International Conference on Computer Applications in Industry*, November 1995.

[18] S. Shah and H. Sholl, "Task Partitioning of Incompletely Specified Real-Time Distributed Systems," *ICSA 9th International Conference on Parallel and Distributed Systems*, Dijon France, September 1996.

[19] Istvan Gergely Czibula, Gabriela (Serban) Czibula, "Hierarchical Clustering Based Automatic Refactorings Detection", *In the WSEAS TRANSACTIONS on ELECTRONICS*, Issue 7, Volume 5, July 2008,pp 291-302.

[20] X. Xu, C. H. Lung, M. Zaman, and A. Srinivasan, "Program restructuring through clustering techniques," *In the Proceedings of the Source Code Analysis and Manipulation, Fourth IEEE International Workshop on (SCAM'04),* Washington, DC, USA: IEEE Computer Society, 2004, pp. 75–84.

[21] L.R. Welch, B. Ravindran, J. Henriques, and D.K. Hammer, "Metrics and Techniques for Automatic Partitioning and Assignment of Object-based Concurrent Programs," *In the Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, Oct. 1995.

[22] V. Tzerpos and R. C. Holt, "Mojo: A distance metric for software clusterings," *Proceedings of the sixth Working Conference on Reverse Engineering*, 1999, pp. 187–193.

[23] V. Tzerpos and R. C. Holt, "ACDC: An algorithm for comprehension-driven clustering," *In the Proceedings of the Working Conference on Reverse Engineering*, 2000, pp. 258–267.

[24] A Abd El-Raouf, T. Fergany and R. Ammar, "Restructuring Of Distributed Object Oriented Software," *In the WSEAS Transactions on Computers*, Issue 5, Volume *3*, ISSN: 1109-2750 pp: 1179-1184, November 2004.

[25] A. Abd El-Raouf, R. Ammar, and T. Fergany "Object Oriented Performance Modeling and Restructuring on a Pipeline Architecture, " *The Journal of Computational Methods in Science and Engineering , JCMSE*, Volume 6, pp 59-71, IOS Press, 2006.

[26] S. Hamad, R. Ammar, A. Abd El-Raouf, and Mohammed Khalifa "A Performance-driven Clustering Approach To Minimize Coupling In A DOO System," *The 20th International Conference on Parallel and Distributed Computing Systems*, Las Vegas, Nevada, 24-26 Sept. 2007.

[27] S. Hamad, R. Ammar, T. Fergany and A Raouf, "A Double K-Clustering Approach for Restructuring Distributed Object-Oriented Software," *the International Symposium on Computers and Communication, ISCC*, July 6-9, Marrakech, Morocco, 2008.

[28] A. Abd El-Raouf, "*Performance Modeling and Analysis of Object Oriented Software Systems*," PhD Dissertation, University of Connecticut Department of Computer Science & Engineering, 2005.

[29] H. Sholl and T. Booth "Software Performance Modeling Using Computation Structures", *IEEE transaction on Software Engineering,* VOL. SE-1, No.4, Dec 1975.

[30] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome. "*Hierarchical clustering: The Elements of Statistical Learning*," New York: Springer, 2001 pp. 272–280

[31] Kaufman, L., & Rousseeuw, P. J., "*Finding Groups in Data: An Introduction to Cluster Analysis*," New York: John Wiley & Sons, Inc, 1990.

[32] I. G. Czibula and G. Serban, "Improving Systems Design Using a Clustering Approach", *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 6, no. 12, 2006, pp. 40–49.

[33] N. P. Lin, C.-I. Chang, H.E. Chueh, H. J. Chen, and W. H. Hao, "A Deflected Grid-based Algorithm for Clustering Analysis", *In the WSEAS Transactions on Computers*, Issue 3, vol. 7, 2008, pp. 125–132.

[34] D. I. Hunyadi, M. A. Musan, "Data Modeling at Conceptual Level Object-Role Modeling (ORM)," *In the Proceedings of the 9th WSEAS International Conference on Evolutionary Computing (EC'08)*, Sofia, Bulgaria, May 2-4, 2008.

[35] D. S. Kushwaha and A. K. Misra, "A Cognitive Complexity Metric Suite for Object-Oriented Software", *In the WSEAS TRANSACTIONS on COMPUTERS*, Issue 3, Volume 5, March 2006, pp 604-611