Restructuring Distributed Object-Oriented Software Using Hierarchical Clustering

Amal Abd El-Raouf
Computer Science Department
Southern Connecticut State University
501 Crescent St., New Haven, CT 06515
USA
abdelraoufa1@southernct.edu

Abstract: The software restructuring techniques present solutions for the software-hardware mismatch problem in which the software structure does not match the available hardware platform. In Distributed Object Oriented (DOO) systems, software engineers face many challenges to solve the software-hardware mismatch problem. One important aspect of DOO software systems is the efficient distribution of software classes among the different nodes while maintaining low-coupling and high software quality. In this paper, we present a new methodology for efficiently restructuring the DOO software systems to improve the performance and to solve the software-hardware mismatch problem. In our method, we use the hierarchical clustering technique to opt the classes to be grouped together and according to the customer hardware organization, we pick the level of the hierarchy that have the appropriate number of clusters to be allocated to the set of available nodes in the customer distributed system.

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 [1, 2, 3, 4]. 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 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 described in [2] 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. 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. Moreover, the Object-oriented paradigm [5, 6, 7] 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 DOO systems to create subsystems with low coupling and high cohesion as quality matrices for good design [8].

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 our previous work [9, 10], we introduced 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 best fits our problem. We use the hierarchical 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. Hence, we do not need a second phase to map the clusters to the set of available machines in the target distributed architecture.

This new methodology is more appropriate for software companies that develop DOO applications and 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 simply use our methodology results to pick the level in the hierarchy that has the number of clusters matches the number of nodes in the customer hardware platform.

This paper is organized as follows: the second section states the problem definition, including assumptions and our goals. Section three describes how we generate the communication matrix using the analytical DOO performance model. Section four describes the hierarchical clustering technique used with our approach. In Section five, we support our methodology with a case study. Finally, section six draws our conclusions.

2 Problem Definitions

In this paper, we consider restructuring DOO applications for mapping on a distributed system. The restructuring process is simply the process of mapping the DOO 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 DOO system. 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 decomposing the system into subsystems that have low coupling and are suitable for distribution. The Hierarchical clustering technique is used to build a dendrogram or a tree. 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 process is as simple as 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 are shown in figure 1.

3 Generating the Communication Matrix

In order to decide which classes should be combined, a measure of dissimilarity 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. Each element of the matrix represents the communication cost between two classes in the OO application. For example C23 will be the communication cost between class 2 and class 3. Communication cost between a class and itself is always equal to zero.

<table>
<thead>
<tr>
<th></th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
<th>C15</th>
</tr>
</thead>
<tbody>
<tr>
<td>C21</td>
<td>0</td>
<td>C23</td>
<td>C24</td>
<td>C25</td>
</tr>
<tr>
<td>C31</td>
<td>C32</td>
<td>0</td>
<td>C34</td>
<td>C35</td>
</tr>
<tr>
<td>C41</td>
<td>C42</td>
<td>C43</td>
<td>0</td>
<td>C45</td>
</tr>
<tr>
<td>C51</td>
<td>C52</td>
<td>C53</td>
<td>C54</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2: the Communication Matrix

In DOO system, accurate calculation of communication time is a challenge due to the dependency between classes and frequent remote requests. Most approaches to evaluate communication time in DOO systems are based on either the system measurements after its development or mapping it to a conventional performance model and hence no way to preserve the OO features during the analysis phase.

In [11], 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 a DOO 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 execution server evaluates the execution cost of the software modules that reside on the target node. The communication server provides the analysis of the communication activities as well as the evaluation of the communication cost.

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 $\lambda_{ck}$ is given by:

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

where $\lambda_{cs}$, $\lambda_{cn}$ and $\lambda_{cu}$ represent the communication arrival due to External User Request (EUR), Remote Request (RR), and updating objects’ data on other nodes, respectively,

$$\lambda_{cs} = \beta_s \lambda_s, \quad \lambda_{cn} = \beta_n \lambda_n, \quad \lambda_{cu} = \sum_{i=1}^{N} \lambda_{CUi}$$

Where, $\beta_s$ and $\beta_n$ are the message multipliers for EUR and RR. Let $\lambda_{cu}$ 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, $P_{i1}$ defined to be the probability that object i is updated due to EUR, $P_{i2}$ is the probability that object i is modified due to RR. $\lambda_{cu}$ can be expressed as:

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

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

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

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 is the communication channel capacity. Furthermore, the average communication service time for node (k) will be:

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

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

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.

If each individual class is allocated to a separate node in the DOOP model, we can use the above equations to compute the average communication cost between a specific class and all other classes in the system. The computed values are then used as the elements of our Communication Matrix.

### 4 Hierarchal Clustering Technique

In this section we describe the agglomerative hierarchal clustering technique that we used in our restructuring methodology.

#### 4.1 The Dendrogram

The input to the algorithm is the communication matrix generated in section 3 and the output is a tree-like structure or a dendrogram created through recursive merging of existing OO classes.

Generating the dendrogram is a bottom-up approach; each class starts in its own cluster in the first level. Then, in succeeding steps, the two clusters that have the maximum communication time cost are aggregated into a new combined cluster. In this way, the number of clusters in the DOO system is reduced by one in each
step. At the top level, all classes are combined into a single huge cluster. Since, for n classes there are \((n - 1)\) merges, there are \(2^{(n-1)}\) 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 dissimilarity update formula [12, 13].

The last update step could be done by a number of different ways according to the clustering method. Hierarchical clustering common methods are: single-link, complete-link and average-link clustering.

In this work, we utilize the average-link clustering method, that consider the communication cost between one cluster and another cluster to be equal to the average communication cost from any member of one cluster to any member of the other cluster. The linkage criteria between two classes A and B is:

\[
1 - \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b).
\]

Where \(d(a,b)\) is the communication cost between the two classes \(a\) in \(A\) and \(b\) in \(B\).

4.2 Mapping to the Hardware Architecture

The basic advantage of using the hierarchical 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 \(m\), you go down \(m-1\) levels. The level you reach is the key level. This key level would have \(m\) clusters that are directly mapped to the \(m\) nodes in the distributed system.

5 Case Study

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

In the first step we used the Distributed Object Oriented Performance (DOOP) model to generate the 27 \(\times 27\) communication matrix. This matrix is the input of the hierarchical clustering function.

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 last one “agnes”.

As described in [14], the agnes clustering method has the following features:

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

The command used in R is:

\[
\text{plot( agnes( data, diss = FALSE, method = "average" ) )}
\]

The arguments of the command are set as follows:

- Data is the Communication Matrix generated in step1.
- 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.

The result of the agnes clustering is typically visualized as a dendrogram as shown in Figure 4. The horizontal coordinate shows the different classes in terms of \(V_i\), where \(i\) is the class number. Each merge is represented by a horizontal line.

The resultant dendrogram is used to map the OO 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.

![Figure 4: A dendrogram for clustering 28-classes OO software application](image-url)
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

6 Conclusion
In this paper, we introduced a new methodology for restructuring distributed object oriented software systems. This new methodology has two main goals, first: grouping OO software classes into clusters that have low coupling and high cohesion to improve the system quality, second: solving the software-hardware mismatch problem by producing a hierarchical clustering dendrogram (or tree). The dendrogram includes different levels of hierarchy with different number of clusters. Hence, the software engineer would be able to locate the appropriate level (key level) that matches the customer system requirements.

References: