Software Architecture Reconstruction through Clustering: Finding the Right Similarity Factors

Ioana Şora

Department of Computer and Software Engineering, Politehnica University of Timişoara, Timişoara, Romania

Abstract. Clustering is very often used for the purpose of automatic software architecture reconstruction. This work investigates the importance of taking into account different factors for the similarity metric, besides the traditional factor based on direct coupling/cohesion: indirect coupling as computed from the topology of the dependency graph, and global architectural layering resulting from the orientation of dependencies. We experiment with using these factors, individually or combined, for defining the similarity metrics within a set of clustering algorithms.

1 Introduction

Software architecture is a model of the software system expressed at a high level of abstraction, concentrating on the interaction of "black box" elements. Knowing and having an explicit representation of the system architecture is essential for understanding, evaluating and maintaining a large software application. Often, the documentation is incomplete, outdated or is completely missing, only the code being available. Reconstructing the architectural model from the available code remains the saving alternative in these cases.

The reverse engineering community developed many techniques to help reconstruct the architecture of software systems, as they are surveyed and classified in [8]. Automatic reconstruction techniques aim at finding the logical cluster structure of software systems, with as few user intervention as possible and with minimal prior knowledge. Software clustering refers to the decomposition of a software system into meaningful subsystems. To be meaningful, the automatic approach must produce clusterings that can help developers to understand the system, grouping together parts that relate to each other from a logical design point of view.

Our goal is to improve automatic reconstruction techniques, in order to obtain a reconstructed architectural model of a better quality - one which is evaluated to be better by a human expert.

This article is organized as follows: Section 2 resumes the state of the art and builds the motivation for our approach. Section 3 states the goals of this work and introduces our reconstruction approach. Experimental results regarding the influence of different similarity factors on the quality of the reconstructed model are described in Section 4.

Sora I..
Software Architecture Reconstruction through Clustering: Finding the Right Similarity Factors.
DOI: 10.5220/0004599600450054
In *Proceedings of the 1st International Workshop in Software Evolution and Modernization* (SEM-2013), pages 45-54
ISBN: 978-989-8565-66-2

2 Background

Reconstructing the architecture of a software system can take one of the following approaches:

- The top-down approach, when certain assumptions of the overall system organization are known and they are validated by examining the existing artifacts with help of interactive tools in a human-controlled reconstruction process.
- The bottom-up approach, when (quasi)-automatic unsupervised tools build hypotheses starting from the examination of the existing artifacts.

In the category of top-down, human-controlled or interactive approaches there are notable tools such as: Rigi by H. Muller et al [14]; the Reflexion Model technique of Murphy, Notkin, and Sullivan [15]; the Reflexion model combined with clustering [7]; ACDC - the pattern driven approach of Tzerpos and Holt [20].

In the category of bottom-up, automatic or quasi-automatic approaches for architectural reconstruction, techniques have been imported from the domain of data mining. Clustering algorithms have been largely used in data mining to identify groups of objects whose members are similar in some way. Clustering algorithms group together entities into groups, by maximizing the sum of relationships between entities grouped together and minimizing the sum of relationships between different groups. In reverse software engineering, clustering is used for architectural reconstruction, by grouping together in subsystems modules (classes, functions, etc) that relate to each other.

There are several research approaches in this domain, which differ by:

- the graph clustering algorithm which is used [22]
- the software-engineering defined criteria used for grouping modules together (the similarity metric)

The basic assumption driving this software clustering approach is that software systems are organized into subsystems characterized by internal cohesion and loosely coupling with each other. A reference tool of this category is Bunch, developed by Mitchell and Mancoridis [12], using a search based algorithm (hill climbing) and a modularization quality metric MQ defined as a formula on coupling and cohesion.

As observed by many researchers, clustering software based on a metric for similarity and dissimilarity derived only from coupling and cohesion does not provide satisfactory results [9]. Various researches have tried to do software clustering by taking into account other categories of informations as similarity metrics: A form of indirect coupling is taken into account by Chiricota [6]. The LIMBO approach of Andritsos and Tzerpos [1] considers even non-software informations, such as historical data (time of last modification, author) held by version control system repositories, the physical organization of applications in terms of files and folders. Anquetil and Letherbridge [3] use for clustering the symbolic textual information available in the identifier used as names. Recent researches ([2], [5], [13], [10]) agree that unsupervised clustering approaches based only on a coupling/cohesion criteria tend to produce results that are not acceptable for the domain experts and propose different measures for improvements.

3 Our Work

3.1 Goal and Approach

Our goal is to investigate ways of improving the quality of bottom-up, automatic, unsupervised reconstruction. We have built the Architecture Reconstruction Tool Suite (ARTs) as an extensible tool chain for experimenting with different methods for clustering. In the architectural reconstruction community there have been developed a lot of different approaches and methods but they are used and studied in isolation. Our goal is to integrate into an architectural reconstruction toolsuite the different partial solutions, in order to compare their relative efficiency and also study the ways how they can be combined. Also, we propose a new approach of including extracted architectural information in the grouping criteria.

The architecture of ARTs is depicted in Figure 1.

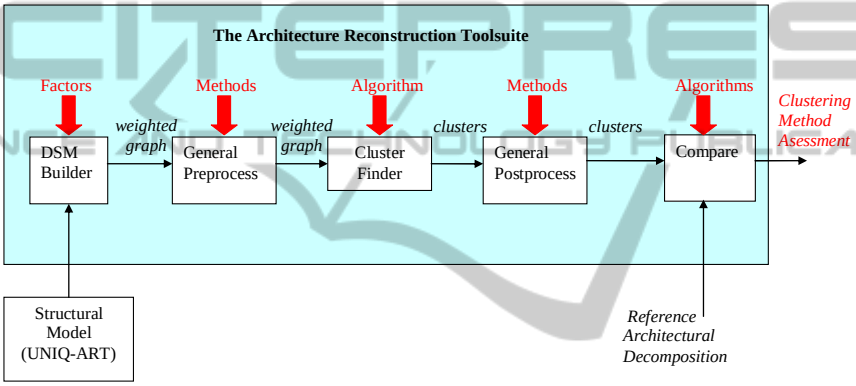

Fig. 1. The Architectural Reconstruction Toolsuite (ARTs).

The input for ARTs is a primary structural and dependency model extracted from code by static analysis and represented according to the UNIQ-ART meta-model [17]. The primary models represent relationships between the *Units*(classes or modules) and their parts, and may describe systems implemented in object oriented (Java, C#) or procedural (C) languages.

Each tool of this chain accepts different plug-ins in order to customize it.

The central tool is the *ClusterFinder*, which operates on a weighted graph representing an abstraction of the system in order to produce its decomposition into clusters. It may be implemented by different clustering algorithms. Currently we have implemented both flat decomposition algorithms as well as hierarchical decomposition algorithms. The algorithms are:

- Minimum Spanning Tree based algorithms (MST [23] and MMST [4])
- Metric Based ([6]);
- Search based (hill-climbing)
- Hierarchical clustering: Single Linkage, Complete Linkage, Weighted Average, Unweighted Average

The *DSMBuilder* selects information from the primary model and creates the abstract weighted graph. Different grouping criteria can be used (in isolation or composed) as factors leading to the weights values, as detailed in subsection 3.2.

In order to improve the clustering process, some preprocessings and postprocessings can be performed, optionally, independent of the chosen clustering algorithm. We implemented elimination of omnipresent modules [14] and orphan adoption [19].

A clustering method, defined by the combination of grouping factors, clustering algorithm, pre- and postprocessings, is evaluated by comparing its result with a given authoritative decomposition. The evaluation methods (the *Comparer*) are detailed in subsection 3.3.

3.2 Grouping Criteria

Following three criteria can be used (in isolation or composed) to build the similarity metrics:

- the strength of the direct dependencies coupling (DC)
- indirect coupling (IC)
- global architectural information regarding the architectural layer (LA)

The direct coupling factor, which is the baseline grouping criteria, can be adjusted by applying factors derived from the indirect coupling or global architectural information. The similarity metric value between two units A and B is given by aggregating the individual factors:

$$Similarity(A, B) = DC(A, B) \cdot IC(A, B) \cdot LA(A, B)$$

Also, in future work new grouping criteria could be added, for example introducing another factor derived from the symbolic textual information extracted from the identifier names.

The Direct Coupling Factor (DC). The main factor is the direct coupling factor which quantifies the statical dependencies between units. An unit A depends on a unit B if there are explicit references in A to elements of B.

In previous work [18] we empirically defined 11 different dependency types, characterized by a dependency type weight $w_{DepType}$. The values of the weights have been empirically finetuned in order to reflect the relative importance of different dependencies types for the strength of the coupling.

The value of the direct coupling factor between A and B is given by the sum of all dependency types that exist between them:

$$DC(A, B) = \sum_{DepType} w_{DepType} \cdot count_{DepType}(A, B)$$

For certain dependency types such as function calls or variable accesses, the specific weight $w_{DepType}$ is adjusted with a counter $count_{DepType}(A, B)$ representing the relative number of the accesses from A to B, reported to the total number of possible accesses.

The Indirect Coupling Factor (IC). We start from the observation that if two units A and B have neighbors (units they interact with) which also interact with each other, this corresponds to a form of indirect coupling. In this case, the two units A and B have a higher probability to be part of the same subsystem (cluster).

First we calculate the ESM (Edge Strength Metric, defined in [6]) value for each edge of the given dependency graph.

To determine the importance of ESM value, a confidence level $cl \in [0; 1]$ is introduced when computing the indirect coupling factor IC:

$$IC(A, B) = ESM(A, B) \cdot (1 - cl)$$

Thus the higher the pre-given confidence level, the higher the impact of the IC factor and the higher the importance given to cycles, with 0 meaning it will have no impact in the algorithm used and 1 meaning it will have maximum impact (and as some of the edges will have an ESM value of 0, it will practically cut some of the edges before the algorithm).

The Architectural Layer Distance Factor (LA). One of the advantages of top-down reconstruction approaches is that they start with some general assumptions about the global architecture. In a bottom-up unsupervised approach we may not have such a-priori global architectural knowledge. We propose a new approach of including *extracted* architectural information in the grouping criteria.

One kind of architectural information which may be extracted in a bottom-up approach is layering information. Units belonging to a layer may depend only on units belonging to lower layers. Layers are determined by applying a partitioning algorithm like [16] on the directed graph of dependencies. In future implementations, an algorithm such as [11] may improve the determination of layers also in the presence of cyclic dependencies.

We make the observation that two units which are situated in layers of very different levels are highly unlikely to be part of the same architectural subsystem, even if there is a strong dependency between them. On the other hand, two units that are situated on the same or on close layers have a higher chance to be part of the same architectural subsystem. This observation is reflected in the architectural layer distance factor.

We define δ as the absolute value of the difference between the layers of A and B, divided to the total number of layers in order to normalize the value:

$$\delta(A, B) = \frac{|Layer(A) - Layer(B)|}{TotalLayers}$$

The similarity metric is proportional with the architectural layer distance factor LA, defined as:

$$LA(A, B) = Ladjustement(\delta(A, B))$$

The layer distance adjustement is a decreasing function

$$Ladjustement : [0, 1] \rightarrow [0, 1]$$

We experimented with layer distance adjustement functions decreasing at different rates, such as linear or exponential.

When applying any of the adjustement functions, units that are mutually dependent and are situated on the same layer have $\delta = 0$, and the value of the linear or exponential adjustement function is 1, thus the similarity is given only by the dependency strength. For any other case, the bigger the distance is, the smaller will be the value of the adjustement function, reducing accordingly the dependency strength.

3.3 Evaluation Approach

In our case, a clustering method is defined by the combination of: grouping factors, clustering algorithm, pre- and postprocessings. The existing approaches of evaluating clustering methods can be divided into two categories: approaches which rely on a authoritative decomposition and approaches which do not rely on such. Evaluation criteria which do not rely on reference decompositions, such as the MQ metric [12], are not suitable for our purpose because they already quantify coupling and cohesion as main grouping criteria. Since our work investigates the importance of different grouping criteria, the only way to evaluate the results of a clustering method is to measure how close they are to the decomposition indicated by a human expert.

A clustering method is evaluated by comparing the results it produces for a set of test systems with the corresponding authoritative decompositions of these systems. It may be arguable that different experts may indicate different decompositions, at different granularity levels, but this can be handled if the reference decompositions are specified hierarchical.

Different strategies for comparing the similarity degree of two decompositions of the same system have been proposed [21]. In this work we have so far used the MoJo metric, but other metrics(such as Precision/Recall, EdgeSim, etc) could be also used in the *Comparator*. The MoJo metrics counts the minimum number of operations (moves and joins) one needs to perform in order to transform one decomposition C_1 into another decomposition C_2. The direct MoJo metric is actually a dissimilarity measure, since a big value of the metric indicates that the decompositions are not similar. In order to have a similarity measure, we use another quality measurement based on MoJo, the MoJo similarity measurement which is defined as:

$$similarityMoJo(C_1, C_2) = [1 - \frac{MoJo(C_1, C_2)}{N}] \times 100\%$$

This metric describes the normalized similarity degree of two clusterings, C_1 and C_2, of a system with N units. Since the MoJo metric is not symmetric, for a pair C_1,C_2 the metric is applied in both directions and the maximum value is taken.

4 Results

4.1 Tuning of Algorithms

First, all implemented algorithms required a tuning process in order to establish the ranges of optimal values for their specific parameters.

Each algorithm has its very own set of specific parameters: The MST algorithm has as parameter a *Threshold* value that is used by the algorithm as a decision factor when edge removal is considered; The MMST algorithm has as parameter a *Closeness factor* value that represents the threshold used by the algorithm as a decision factor when uniting two clusters is considered. The Metric Based algorithm has as parameter a *Threshold* value that is used as a decision factor when considering removing an edge together with the ESM metric value. The Hill Climbing algorithm has as parameters the *climbDegree* which specifies how many of the possible variations should be considered at each step and the *generationMethod*. The Hierarchical algorithms have as parameters a *granularity factor* which determines the point of cutting off the final clusters.

In order to determine the optimal parameter values, we proceeded as follows: We choose a set of test systems to be clustered and we determined their reference decompositions, either by detailed code inspection or by requesting the opinion of their developers. For each algorithm, several runs have been made with different values for the specific parameters, for all test system. We noticed that the parameter values for which the obtained decomposition is closest to the reference (the maximum of the MoJo similarity) may vary from one system to another, thus some average values have been determined as the recommended values for the parameters of each algorithm. Discussing the exact parameter values obtained by tuning for each algorithm is not relevant for the main goal this paper; for example, an analysis of parameter values for the MST and MMST algorithms has been included in our previous work [18].

Also, tuning has shown that general pre- and postprocessings such as elimination of omnipresent modules (library classes) and orphan adoption have a clear positive impact and have been included by default in all further experiments.

4.2 Evaluation of the Impact of different Grouping Criteria

After the step of tuning each algorithm, we carried out experiments in order to compare the results when composing the grouping criteria from different factors : Direct coupling only (DC) which represents the baseline of other comparisons, Direct coupling and Layer architecture (DC + LA), Direct coupling and Indirect coupling (DC + IC), Direct coupling, Indirect coupling and Layer architecture (DC + IC + LA).

We carried out these experiments looking for the impact of using different grouping criteria on the quality of the automatic decomposition, measured by its closeness to the authoritative decomposition.

Table 1 contains the results obtained when applying the different clustering algorithms, with different grouping criteria, for the clustering of a test system. The test system analyzed in Table 1 is the ARTs toolsuite implementation, a medium-sized system of 360 classes, and its architecture is well known to the experimenters. The table presents the maximum values of the MoJo similarity metric, obtained for any specific

parameter settings for each algorithm. Columns $\Delta 1$, $\Delta 2$ and $\Delta 3$ compute the differences in MoJo similarity, obtained when using different additional factors vs. the baseline factor.

Table 1. Experimental results - influence of different grouping factors on the clustering results.

Factors Algorithms	DC [0]	DC + LA [1]	$\Delta 1$ [1]-[0]	DC + IC [2]	$\Delta 2$ [2]-[0]	DC+IC+LA [3]	$\Delta 3$ [3]-[0]
MST	64.2	75.8	11.6	55.6	-8.6	71.1	6.9
MMST	57.5	65.6	8.1	50.7	-6.8	60.3	2.8
Metric	70.8	74.6	3.8	76.2	5.4	72.2	1.4
HillClimb	47.8	61.2	13.4	49.1	1.3	59.1	11.3
SL	71.3	82.7	11.4	71.5	0.2	81.2	9.9
WA	66.9	76.5	9.6	64.3	-2.6	73.8	6.9
average improvements			9.65		-1.85		6.53

As the table shows, including an Architectural Layer factor in all clustering algorithms always produces decompositions that are closer to the reference solution. Including an Indirect Coupling factor, however, does not have a clear positive impact on the quality of the resulting decomposition. Including both Architectural Layer and Indirect Coupling factors is not better than using only the Architectural Layer factor.

We have used several other test systems, some open source software such as junit, xercesImpl, jEdit, Ant and some developed as our university projects. We determined their reference decompositions either by performing detailed analysis of their code or by asking their developers. The sizes of the test systems go from 110 classes up to 1400 classes. By experimenting also with these systems, we obtained average improvement values for $\Delta 1$, $\Delta 2$, $\Delta 3$ in ranges similar to these presented in Table 1.

We conclude that the architectural layer factor always improves the quality of the clustering result, and the exponential adjustement function works better than the linear one. From a quantitative point of view, the improvements are biggest for systems with many classes that that have many dependencies spanning big layer distances.

From our experiments we concluded that the Indirect Coupling factor does not bring real improvements. It also has a negative effect on many cases. Although it may seem surprising, we can explain this finding by the following facts: the Indirect Coupling as defined by the Edge Strength Metric hampers the grouping of inheritance hierarchies; also, in the case of smaller systems, the Indirect Coupling metric tends to agglomerate everything in a few very big clusters. The granularity of the selected reference model also affects the results, positive results were obtained on large and/or complex systems or when using a more coarse grained reference model.

Also, the experiments pointed out another aspect which is worth to be investigated in future work - how the different factors of the similarity metric may have an influence

on the stability of the clustering algorithms, by increasing the range of parameter values that lead to optimal results and thus simplifying the tuning of the algorithms.

5 Conclusions

Taking into account global architectural information is essential for improving the results of coupling/cohesion guided software architecture reconstruction. In the case of unsupervised automatic software clustering, we propose to make such global architectural information available in form of the Architectural Layer distance factor, which can be computed at the reconstruction time in a bottom-up manner and used as part of the grouping criteria. Our experiments show that this way of taking into account the global topology of the whole dependency graph in form of the Architectural Layer distance factor is more effective than taking into account only local topologies of the dependency graph in form of the Indirect Coupling factor. This conclusion applies to all the investigated clustering algorithms, thus it demonstrates that the improvement is due to the grouping criteria.

Acknowledgements

The author thanks all the students who, in recent years, have participated in the implementation of parts of ARTs: Gabriel Glodean, Mihai Gligor, Adrian Oros, Bogdan Zavada, Diana Brata.

References

1. Periklis Andritsos and Vassilios Tzerpos. Information-theoretic software clustering. IEEE Trans. Software Eng., 31(2):150–165, 2005.
2. N. Anquetil and J. Laval. Legacy software restructuring: Analyzing a concrete case. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pages 279–286, 2011.
3. Nicolas Anquetil and Timothy C. Lethbridge. Recovering software architecture from the names of source files. Journal of Software Maintenance, 11(3):201–221, May 1999.
4. Markus Bauer and Mircea Trifu. Architecture-aware adaptive clustering of OO systems. Software Maintenance and Reengineering, European Conference on, 0:3, 2004.
5. Fabian Beck and Stephan Diehl. On the congruence of modularity and code coupling. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ESEC/FSE '11, pages 354–364, New York, NY, USA, 2011. ACM.
6. Yves Chiricota, Fabien Jourdan, and Guy Melancon. Software components capture using graph clustering. In Proceedings IWPC, pages 217–226, 2003.
7. Andreas Christl, Rainer Koschke, and Margaret-Anne D. Storey. Automated clustering to support the reflexion method. Information & Software Technology, 49(3):255–274, 2007.
8. S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. Software Engineering, IEEE Transactions on, 35(4):573–591, 2009.
9. Fernando Brito e Abreu and Miguel Goulão. Coupling and cohesion as modularization drivers: Are we being over-persuaded? In Proceedings of the Fifth Conference on Software Maintenance and Reengineering, CSMR, pages 47–57, 2001.

10. M. Hall, N. Walkinshaw, and P. McMinn. Supervised software modularisation. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on, pages 472–481, 2012.
11. Jannik Laval, Nicolas Anquetil, Usman Bhatti, and Stphane Ducasse. oZone: Layer identification in the presence of cyclic dependencies. Science of Computer Programming, (0):–, 2012.
12. Brian S. Mitchell and Spiros Mancoridis. On the automatic modularization of software systems using the bunch tool. IEEE Trans. Software Eng., 32(3):193–208, 2006.
13. S. Muhammad. Evaluating relationship categories for clustering object-oriented software systems. IET Software, 6:260–274(14), June 2012.
14. Hausi A. Müller, Scott R. Tilley, and Kenny Wong. Understanding software systems using reverse engineering technology perspectives from the rigi project. In Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1, CASCON '93, pages 217–226. IBM Press, 1993.
15. G.C. Murphy, D. Notkin, and K.J. Sullivan. Software reflexion models: bridging the gap between design and implementation. Software Engineering, IEEE Transactions on, 27(4):364–380, 2001.
16. Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pages 167–176, New York, NY, USA, 2005. ACM.
17. Ioana Sora. A meta-model for representing language-independent primary dependency structures. In Joaquim Filipe and Leszek A. Maciaszek, editors, ENASE 2012 - Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering, pages 65–74. SciTePress, 2012.
18. Ioana Sora, Gabriel Glodean, and Mihai Gligor. Software architecture reconstruction: An approach based on combining graph clustering and partitioning. In Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on, pages 259–264, 2010.
19. V. Tzerpos and R.C. Holt. The orphan adoption problem in architecture maintenance. In Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on, pages 76–82, 1997.
20. V. Tzerpos and R.C. Holt. Accd: an algorithm for comprehension-driven clustering. In Reverse Engineering, 2000. Proceedings. Seventh Working Conference on, pages 258–267, 2000.
21. Zhihua Wen and V. Tzerpos. Evaluating similarity measures for software decompositions. In Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on, pages 368–377, 2004.
22. T.A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In Proceedings of the Fourth Working Conference on Reverse Eengineering, pages 33–43, 1997.
23. C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. IEEE Transactions on Computers, C(20):68–86, 1971.