Aspect Mining using a Vector-Space Model Based Clustering Approach

Grigoreta Sofia Moldovan
Department of Computer Science
Babeş-Bolyai University
Str. Mihail Kogălniceanu, Nr. 1
Cluj-Napoca, Romania
grigo@cs.ubbcluj.ro

Gabriela Şerban
Department of Computer Science
Babeş-Bolyai University
Str. Mihail Kogălniceanu, Nr. 1
Cluj-Napoca, Romania
gabis@cs.ubbcluj.ro

ABSTRACT

Aspect Oriented Programming is a programming paradigm that addresses the issue of crosscutting concerns. Aspect mining is a process that tries to identify crosscutting concerns in existing software systems. The goal is to refactor the existing systems to use aspect oriented programming to make them easier to maintain and to evolve.

This paper presents a new approach in aspect mining that uses clustering and proposes two techniques: a *k-means* based clustering technique and a *hierarchical agglomerative* based clustering technique. We are trying to identify the methods that have the code scattering symptom. For a method, we consider as indication of code scattering a big number of calling methods and, also, a big number of calling classes. In order to group the best methods (candidates), we use in our approach the vector-space model for defining the similarity between methods. For testing the efficiency of the proposed techniques, a number of Java applications are being used.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*; I.5.3 [**Computing Methodologies**]: Pattern Recognition—*Clustering*

Keywords

Aspect Mining, Clustering.

1. INTRODUCTION

The Aspect Oriented Programming(AOP) is a new paradigm that is used to design and implement *crosscutting concerns* [11]. A *crosscutting concern* is a feature of a software system that is spread all over the system, and whose implementation is tangled with other features' implementation. Logging, persistence and connection pooling are well-known examples of crosscutting concerns. In order to design and implement a crosscutting concern, AOP introduces a new modularization unit called *aspect*. At compile time, the aspect is woven to generate the final system, using a special tool called *weaver*. Some of the benefits that the use of AOP to software engineering brings are: better modularization, higher productivity, software systems that are easier to maintain and evolve.

Aspect mining is a relatively new research direction that tries to identify crosscutting concerns in already developed software systems without using AOP. The goal is to identify them and then to refactor them to aspects, to achieve a system that can be easily understood, maintained and modified.

Crosscutting concerns in non AO systems have two symptoms: *code scattering* and *code tangling*. *Code scattering* means that the code that implements a crosscutting concern is spread across the system, and *code tangling* means that the code that implements some concern is mixed with code from other (crosscutting) concerns.

The paper is structured as follows: section 2 presents the main issues related to the clustering problem, section 3 explains our approach, and section 4 presents the applications we have used to test our approach and the results we have obtained. Section 5 presents our conclusions and some future research directions.

1.1 Related Work

Several approaches have been considered for aspect mining until now. One approach was to develop tools that would help the user to navigate and to analyze the source code in order to find crosscutting concerns. Some of them rely on lexical analysis, and some also include a type-based search([6], [8], [20]). Other approach uses clone detection techniques to identify duplicate code, that might indicate the presence of crosscutting concerns([16] [15], [2]). These are all static approaches that analyze the source code for crosscutting concerns. There are also two dynamic approaches: one that analyzes the event traces [1], and one that uses formal concept analysis to analyze the execution traces [18]. In [19] formal concept analysis is used again, but in a static manner. A comparison of three different approaches can be found in [3].

There is also a clustering approach that constructs the clusters based on the methods names [17]. The user can then navigate among the clusters, visualize the source code of the methods and identify the crosscutting concerns.

2. CLUSTERING

Clustering, or unsupervised classification, is a data mining activity that aims to partition a given set of objects into groups (classes or clusters) such that objects within a cluster would have high similarity to each other and low similarity to objects in other clusters. The inferring process is carried out with respect to a set of relevant characteristics or attributes of the analyzed objects. Similarity and dissimilarity between objects are calculated using metric or semi-metric functions applied to the attribute values characterizing the objects.

Let $X = \{O_1, O_2, \ldots, O_n\}$ be the set of objects to be clustered. Using the vector-space model, each object is measured with respect to a set of m initial attributes $A_1, A_2, \ldots$ $\ldots, A_m$ (a set of relevant characteristics of the analyzed objects) and is therefore described by an m-dimensional vector $O_i = (O_{i1}, \ldots, O_{im}), O_{ik} \in \Re, 1 \le i \le n, 1 \le k \le m$. Usually, the attributes associated to objects are standardized in order to ensure an equal weight to all of them ([7]).

The measure used for discriminating objects can be any *metric* function d. We used the *Euclidian distance*:

$$d(O_i, O_j) = d_E(O_i, O_j) = \sqrt{\sum_{l=1}^{m} (O_{il} - O_{jl})^2}$$

The *similarity* between two objects O_a and O_b is defined as

$$sim(\vec{O_a}, \vec{O_b}) = \frac{1}{d(\vec{O_a}, \vec{O_b})}$$

A large collection of clustering algorithms is available in the literature. [7], [9] and [10] contain comprehensive overviews of the existing techniques. Most clustering algorithms are based on two popular techniques known as *partitional* and *hierarchical* clustering.

In the following, an overview of both techniques is presented.

2.1 Partitioning Methods. The *k-means* Clustering Algorithm

A well-known class of clustering methods is the one of the partitioning methods, with representatives such as the *k-means* algorithm or the *k-medoids* algorithm. Essentially, given a set of n objects and a number $k, k \le n$, such a method divides the object set into k distinct and non-empty clusters. The partitioning process is iterative and heuristic; it stops when a "good" partitioning is achieved.

Finding a "good" partitioning coincides with optimizing a criterion function defined either locally (on a subset of the objects) or globally (defined over all of the objects, as in *k-means*). These algorithms try to minimize certain criteria (a squared error function); the squared error criterion tends to work well with isolated and compact clusters ([10]).

Partitional clustering algorithms are generally iterative algorithms that converge to local optima.

The most widely used partitional algorithm is the iterative *k-means* approach. The objective function that the *k-means* optimizes is the squared sum error (SSE). The SSE of a partition $K = \{K_1, K_2, \ldots K_k\}$ is defined as:

$$SSE(K) = \sum_{K_j \in K} \sum_{O_i \in K_j} d^2(O_i, f_j) \qquad (1)$$

where the cluster K_j is a set of objects $\{O_1^j, O_2^j, \ldots, O_{n_j}^j\}$ and f_j is the centroid (mean) of K_j:

$$f_j = \left(\frac{\sum_{k=1}^{n_j} O_{k1}^j}{n_j}, \ldots, \frac{\sum_{k=1}^{n_j} O_{km}^j}{n_j} \right)$$

Hence, the *k-means* algorithm minimizes the intra-cluster distance. The algorithm starts with k initial centroids, then iteratively recalculates the clusters (each object is assigned to the closest cluster - centroid) and their centroids until convergence is achieved.

2.2 Hierarchical Methods. The Hierarchical Agglomerative Clustering Algorithm ($HACA$)

Hierarchical clustering methods represent a major class of clustering techniques. There are two styles of hierarchical clustering algorithms. Given a set of n objects, the agglomerative (bottom-up) methods begin with n singletons (sets with one element), merging them until a single cluster is obtained. At each step, the most similar two clusters are chosen for merging. The divisive (top-down) methods start from one cluster containing all n objects and split it until n clusters are obtained.

The agglomerative clustering algorithms that were proposed in the literature differ in the way the two most similar clusters are calculated and the linkage-metric used (single, complete or average).

3. CLUSTERING APPROACH IN ASPECT MINING

Our approach is to try to discover crosscutting concerns by finding measures of the two symptoms: *code scattering* and *code tangling*. The version presented here is just for *scattering*. Our goal is to group the methods by the number of calling methods(the *fan-in* metric) and also by the number of calling modules(in this case we have considered classes as modules). In [14] an approach to aspect mining is presented that also uses the *fan-in* metric, but in our opinion the number of calling classes is also important. A method might have a high fan-in value, but all the calling methods belong to the same class. This might show a high-coupling between the two classes and it might, even, indicate that some refactoring is needed [4].

In order to group methods, we use two clustering algorithms: the *k-means* algorithm and the Hierarchical Agglomerative Clustering Algorithm $HACA$ (section 2).

In our approach, the objects to be clustered are methods $X = \{M_1, M_2, ...M_n\}$. The methods belong to the application classes or are called from the application classes. We consider two vector-space models:

- The vector associated with the method M is $\{FIV, CC\}$, where FIV is the fan-in value and CC is the number of calling classes. We denote this model by $\mathcal{M}_1$.
- The vector associated with the method M is $\{FIV, B_1, B_2, ...B_m\}$, where FIV is the fan-in value and B_i is the value of the attribute corresponding to the application class C_i. The value of B_i is 1, if the method M is called from a method belonging to C_i, and 0, otherwise. We denote this model by $\mathcal{M}_2$.

In the following, we will briefly describe the application of *k-means* and *HACA* in the context of aspect mining.

k-means

We applied a modified version of the *k-means* algorithm in order to optimally divide the set of methods into clusters. We define the "optimal" partition $K = \{K_1, K_2, ...K_p\}$ as the partition that minimizes $SSE(K)$ and we will refer to p as the "optimal" number of clusters ($p \leq k$).

We also mention that in order to assure a "good" choice of the initial centroids, we choose as initial centroids the most dissimilar initial methods (objects).

We mention that, for simplicity, we will continue to refer this method as *k-means*.

HACA

With the optimal number of clusters p determined after applying *k-means*, we have applied a modified version of the traditional *HACA* algorithm in order to determine p clusters in data (the agglomerative algorithm stops when p clusters are reached).

We also mention that we have used complete-link as a linkage metric, because, in general, complete-link generates compact clusters [10] and is a better choice for our approach (single-link produces elongated clusters).

3.1 Identification steps

The approach consists of the following steps:

Step 1. Computation

Computation of the set of methods in the selected source code, and computation of the attributes set values, for each method in the set.

Step 2. Filtering

Methods belonging to some data structures classes like *ArrayList, Vector* are eliminated. Also, we eliminate the methods belonging to some built-in classes like *String, StringBuffer, StringBuilder, etc.*

Step 3. Grouping

The remaining set of methods is grouped into clusters using *k-means* or *HACA*. The clusters are sorted by the average distance from the point 0_m in descending order, where 0_m is the m dimensional vector with each component 0.

Step 4. Analysis

The clusters obtained are analyzed to discover which clusters contain methods belonging to crosscutting concerns. We analyze the clusters whose distance from 0_m point is greater than a threshold (eg. two).

The first three steps are done automatically, but the last one must be done manually.

3.2 Example

In the following, we present a small example of the application of our techniques. If we have the classes shown in Table 1, the values of the attributes set when $\mathcal{M}_1$ is used are presented in Table 2 and the clusters obtained are shown in Table 3:

```
public class A {
   private L l;
   public A(){l=new L();}
   public void methA(){ l.meth();}
   public void methB(){ l.meth();}
}
public class L {
   public L(){}
   public void meth(){}
}
public class B {
   public B(){}
   public void methC(L l){ l.meth();}
   public void methD(A a){a.methA();}
}
```

Table 1: Code example.

Method	FIV	CC
A.A	0	0
A.methA	1	1
A.methB	0	0
B.B	0	0
B.methC	0	0
B.methD	0	0
L.L	1	1
L.meth	3	2

Table 2: Attribute values when $\mathcal{M}_1$ is used.

Cluster	Methods
C1	{ L.meth }
C2	{A.methA, L.L }
C3	{ A.A, A.methB, B.B, B.methC, B.methD }

Table 3: The obtained clusters.

3.3 Characteristics of our clustering approaches

Our clustering approaches have some advantages, reducing the well known main disadvantages of *k-means* and *HACA*:

- we have adapted the traditional *k-means* approach in order to determine the "optimal" number of clusters;
- the *k-means* dependence on the initial centroids is reduced by a good selection of the initial centroids (the most dissimilar initial objects);

- the *HACA* based approach that we have used, instead of merging all the methods in a single cluster, determines a good enough partition into clusters;
- the *HACA* based approach uses the complete-link as linkage metric, choice that is better for our approach (we are looking for compact clusters in data).

4. CASE STUDIES

For each case study, we have generated the vector-space models $\mathcal{M}_1$ and $\mathcal{M}_2$ as inputs for clustering, and we have applied two clustering algorithms: *k-means* and *HACA*.

4.1 Theatre

The second case study is a web application, called *Theatre*, developed by an undergraduate student as her graduation project. It allows searching for a show, reserving tickets for a show, canceling reserved tickets; it displays the configuration of a showr0om and the occupied places. The application was developed using applets, servlets and databases. It has 27 classes (4 applets, 9 servlets, and 14 additional classes), and 336 methods.

The "optimal" number of clusters obtained by *k-means* for the model $\mathcal{M}_1$ is 9 and for the model $\mathcal{M}_2$ is 15. The distribution of methods inside each cluster is shown in Table 4. The first two clusters are identical for all the algorithms independent of the vector-space model used.

Cluster	C1	C2	C3	C4	C5	C6	C7	C8	C9
Methods	1	1	2	2	11	14	37	206	62

Table 4: No. of methods inside each cluster when $\mathcal{M}_1$ and *k-means* are used.

The first cluster contains one method *PrintStream.println(String)* which was used inside the applet classes to print logging information. The second cluster also contains a method used for logging *LogWriter.log(...)*, from the servlets classes. The application contains two distinct logging methods, because the applets are not implicitly allowed to write to files, so applets logging information are written to the Java console of the browser.

The next two clusters contain methods used for the construction of the user interface, and constructors for writing to files.

The application uses a connection pool implemented using the *Singleton* design patterns. In all the cases, all the methods belonging to the database connection were grouped into the same cluster; the difference is the number of the cluster they appear in. In one case, they are the only methods contained in the cluster (when $\mathcal{M}_2$ is used as vector-space model and *k-means* as the clustering algorithm).

4.2 Carla Laffra - Dijkstra's Algorithm

We have considered as case study, Carla Laffra's implementation of the Dijkstra algorithm [12], also. The "optimal" number of clusters obtained by *k-means* for the model $\mathcal{M}_1$ is 7 and for the model $\mathcal{M}_2$ is 5. The distribution of methods in clusters is shown in Table 5.

Cluster	C1	C2	C3	C4	C5
Methods	1	2	20	40	90

Table 5: No. of methods inside each cluster when $\mathcal{M}_2$ and *k-means* are used.

In each case, the methods *Component.repaint()* and *DocText.showline(String)* appear in the first or the second cluster. The method *Component.repaint()* is used each time a new step is executed, or when the algorithm is finished or when a new execution is started with new input data. This method might be considered as part of the crosscutting concern that refreshes the user interface after the execution of different operations. The method *DocText.showline(String)* is used to display guiding information in the TextArea or to display error messages when some preconditions are not met. The last usage may be considered as a crosscutting concern.

In [18] two crosscutting concerns were discovered: locking and unlocking the user graphical interface each time a functionality was executed. The methods used to implement them were not found in the clusters we have analyzed (a better choice for the threshold will, certainly, influence the final results). Another explanation can be the fact that the approach used in [18] is trying to discover *tangled code*, and the current version of our approach is only trying to discover *scattered code.*

4.3 JHotDraw

Our last case study is JHotDraw, version 5.2 which contains 190 classes. The "optimal" number of clusters obtained by *k-means* for the model $\mathcal{M}_1$ is 20 and for the model $\mathcal{M}_2$ is 34. After analyzing the results we have observed that better results were reported by using the model $\mathcal{M}_1$ and *HACA* algorithm. That is why we briefly present only these results.

Most of the crosscutting concerns discovered in [14] were also discovered by our approach and they were not eliminated during step 4. The first six clusters contain methods like *Point.Point(...)*, *FigureEnumeration.hasMoreElements()* or *Figure.displayBox()* which cannot be considered as crosscutting concern seeds.

The first occurences of methods belonging to crosscutting concerns in the obtained clusters are as follows: *Observer* in cluster 7, *Policy Enforcement* in cluster 7, *Persistence* in cluster 10, *Composite* in cluster 11 and *Contract Enforcement* in cluster 17.

5. CONCLUSIONS AND FURTHER WORK

We have presented a new clustering approach in aspect mining based on vector-space models. It tries to identify the methods used to implement crosscutting concerns that have the *scattered* symptom. For that we compute the *fan-in* metric of each method that is called inside the application classes, and the number of classes that call this method. The obtained results are divided in clusters using two clustering algorithms: *HACA* and *k-means.* Some of the obtained clusters are then manually analyzed to determine if they contain methods used to implement crosscutting concerns.

The case studies used to test our techniques have shown that the first clusters obtained contain almost the same methods independently of the clustering algorithm used. Most of the methods belonging to these clusters are used to implement crosscutting concerns. We also mention that the approach proposed in this paper can be used for large applications(with large number of classes).

Further works can be done in the following directions:

- to discover a set of attributes that can indicate the *tangling* symptom for methods. This set of attributes can be easily integrated into our approach just by modifying the used vector-space model;
- to apply other steps of filtering, for example to eliminate the *get/set* type methods, as in [14];
- to use other vector-space models in the clustering approach, and to identify the models that will lead to better results;
- to apply other clustering techniques in the context of aspect mining;
- to use other approaches for clustering that were proposed in the literature (such as variable selection for hierarchical clustering [5], search based clustering [13]);
- to isolate conditions in order to decide the clustering methods and the metric that will lead to better results;
- to apply this approach for other case studies like PetStore and TomCat as in [14];
- to identify and to explain the reasons for success and failure in our approach.

6. REFERENCES

[1] S. Breu and J. Krinke. Aspect Mining using Event Traces. In *Proceedings of International Conference on Automated Software Engineering*, pages 310–315, 2004.

[2] M. Bruntik, A. van Deursen, R. van Engelen, and T. Tourwe. An Evaluation of Clone Detection Techniques for Identifying Crosscutting Concerns. In *Proceedings International Conference on Software Maintenance(ICSM 2004)*. IEEE Computer Society, 2004.

[3] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwe. A Qualitative Comparison of Three Aspect Mining Techniques. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, pages 13–22. IEEE Computer Society, 2005.

[4] M. Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[5] E. B. Fowlkes, G. Gnanadesikan, and J. R. Kettering. *Design, Data, and Analysis: By Some Friends of Cuthbert Daniel.* Wiley, New York, NY, 1987.

[6] W. G. Griswold, Y. Kato, and J. J. Yuan. AspectBrowser: Tool Support for Managing Dispersed Aspects. Technical Report CS1999-0640, UCSD, 3 2000.

[7] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2001.

[8] J. Hannemann and G. Kiczales. Overcoming the Prevalent Decomposition of Legacy Code. In *Advanced Separation of Concerns Workshop,at the International Conference on Software Engineering (ICSE)*, May 2001.

[9] A. Jain and R. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[10] A. Jain, M. N. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[11] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, 1997.

[12] C. Laffra. Dijkstra's Shortest Path Algorithm. http://carbon.cudenver.edu/ hgreenbe/courses/dijkstra/ DijkstraApplet.html.

[13] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, pages 50–59. IEEE Computer Society, 1999.

[14] M. Marin, A. van, Deursen, and L. Moonen. Identifying Aspects Using Fan-in Analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE2004)*, pages 132–141. IEEE Computer Society, 2004.

[15] O. A. M. Morales. Aspect Mining Using Clone Detection. Master's thesis, Delft University of Technology, The Netherlands, August 2004.

[16] D. Sheperd, , E. Gibson, and L. Pollock. Design and Evaluation of an Automated Apect Mining Tool. In *Proceedings of Mid-Atlantic Student Workshop on Programming Languages and Systems*, 2004.

[17] D. Shepherd and L. Pollock. Interfaces, Aspects, and Views. In *Proceedings of Linking Aspect Technology and Evolution Workshop(LATE 2005)*, March 2005.

[18] P. Tonella and M. Ceccato. Aspect Mining through the Formal Concept Analysis of Execution Traces. In *Proceedings of the IEEE Eleventh Working Conference on Reverse Engineering (WCRE 2004)*, pages 112–121, November 2004.

[19] T. Tourwe and K. Mens. Mining Aspectual Views using Formal Concept Analysis. In *Proc. IEEE International Workshop on Source Code Analysis and Manipulation*, 2004.

[20] C. Zhang, G. Gao, and H. Jacobsen. Multi Visualizer. http://www.eecg.utoronto.ca/ czhang/amtex/.