Computing and Informatics, Vol. 29, 2010, 881–900

A HIERARCHICAL CLUSTERING BASED APPROACH IN ASPECT MINING

Gabriela Czibula, Grigoreta Sofia Cojocar

Department of Computer Science
Babeş-Bolyai University
1, M. Kogalniceanu Street
400084, Cluj-Napoca, Romania
e-mail: {gabis, grigo}@cs.ubbcluj.ro

Manuscript received 14 September 2006; revised 10 April 2008

Communicated by Jacek Kitowski

Abstract. *Clustering* is a division of data into groups of similar objects. *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, in order to make them easier to maintain and to evolve. The aim of this paper is to present a new hierarchical clustering based approach in aspect mining. For this purpose we propose *HAC* algorithm (*Hierarchical Agglomerative Clustering in aspect mining*). Clustering is used in order to identify crosscutting concerns. We evaluate the obtained results from the aspect mining point of view, based on two quality measures that we have previously introduced and a newly defined one. The proposed approach is compared with other similar existing approaches in aspect mining and two case studies are also reported.

Keywords: Hierarchical clustering, aspect mining, crosscutting concern, quality measure, evaluation

Mathematics Subject Classification 2000: 68N99, 62H30

1 INTRODUCTION

1.1 Clustering

Unsupervised classification, or clustering, as it is more often referred to, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects [25], being considered one of the most important *unsupervised learning* problems. The inferring process is carried out with respect to a set of relevant characteristics or attributes of the analyzed objects.

The resulting subsets or groups, distinct and non-empty, are to be built so that the objects within each cluster are more closely related to one another than objects assigned to different clusters. Central to the clustering process is the notion of degree of similarity (or dissimilarity) between 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 l initial attributes $A_1, A_2, \ldots, A_l$ and is therefore described by an l-dimensional vector $O_i = (O_{i1}, \ldots, O_{il}), O_{ik} \in \Re, 1 \leq i \leq n, 1 \leq k \leq l$. Usually, the attributes associated with objects are standardized in order to ensure an equal weight to all of them [25].

The measure used for discriminating objects can be any *metric* or *semi-metric* function $d : X \times X \longrightarrow \Re^+$ (Euclidian distance, Manhattan distance, Hamming distance, etc).

1.2 Aspect Mining

Aspect Oriented Programming (AOP) is a relatively new paradigm that is used to design and implement *crosscutting concerns* [31]. 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 brings to software engineering are: better modularization, higher productivity, software systems that are easier to maintain and to 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.

1.3 Related Work

Although aspect mining is a relatively new research domain, many aspect mining techniques have already been proposed. In the following we present a short overview of the existing aspect mining techniques.

Marin et al. [1] have proposed an aspect mining technique that uses the *fanin* metric [33]. Their idea is to search for crosscutting concerns among the methods that have the value of the fanin metric greater than a given threshold.

Breu and Krinke [17] have proposed an aspect mining technique based on dynamic analysis. The mined software system is run and program traces are generated. From program traces, recurring execution relations that satisfy some constraints are selected. Among these recurring execution relations they search for aspect candidates. In [8] this approach is adapted to static analysis. In this approach the recurring execution relations are obtained from the control flow graph of the program.

Tonella and Ceccato [21] have also proposed an aspect mining technique based on dynamic analysis. An instrumented version of the mined software system is run and execution traces for each use case are obtained. Formal concept analysis [29] is applied on these execution traces and the concepts that satisfy some constraints are considered as aspect candidates.

Tourwé and Mens [22] have proposed an aspect mining technique based on identifier analysis. The identifiers associated with a method or class are computed by splitting up its name based on where capitals appear in it. They apply formal concept analysis on the identifiers to group entities with the same identifiers. The groups that satisfy some constraints and that contain a number of elements larger than a given threshold are considered as aspect candidates.

Bruntink et al. [18, 15] have studied the effectiveness of clone detection techniques in aspect mining. They did not propose a new aspect mining technique, but they tried to evaluate how useful clone detection techniques are in aspect mining.

Shepherd et al. [20] have proposed an aspect mining technique based on clone detection. They search for code duplication in the source code using the program dependency graph. The obtained results are further analyzed to discover crosscutting concerns.

Breu and Zimmermann [5] have proposed an history based aspect mining technique. They mine CVS repositories for add-call transactions on which they apply formal concept analysis. Concepts that satisfy some constraints are considered aspect candidates.

Sampaio et al. [13] have proposed an aspect mining technique to discover aspect candidates early in the development lifecycle. They use natural language processing techniques on different documents (requirements, interviews, etc.) to discover words that are used in many sentences. The words that have a high frequency and have the same meaning in all the sentences are considered aspect candidates.

There are just a few aspect mining techniques proposed in the literature that use *clustering* in order to identify crosscutting concerns [7, 10, 14, 16].

He and Bai [7] have proposed another aspect mining technique based on dynamic analysis. They also obtain execution traces for each use case, but they apply clustering and association rules to discover aspect candidates.

Shepherd and Pollock [16] have proposed an aspect mining tool based on clustering. They use hierarchical clustering to find methods that have common substrings in their names. The obtained clusters are then manually analyzed to discover crosscutting concerns.

In [10] a vector space model based clustering approach in aspect mining is proposed. This approach is improved in [14], by defining a *k-means* based clustering algorithm in aspect mining (kAM).

A part of a formal model for clustering in aspect mining is introduced in [12] and some quality measures for evaluating the results of clustering based aspect mining techniques are presented.

An *evolutionary* approach in aspect mining is introduced in [2] and two *genetic clustering* algorithms used to identify crosscutting concerns are proposed.

To our knowledge, there are no other techniques proposed in aspect mining, besides the ones described above.

The main contributions of this paper are:

- To introduce a new hierarchical agglomerative clustering algorithm in aspect mining.
- To evaluate the obtained results from the aspect mining point of view, based on two quality measures previously introduced in [12] and a newly defined one.
- To provide a comparison of our algorithm with other existing similar approaches.

The paper is structured as follows. Section 2 defines the problem of aspect mining as a clustering problem. A new *hierarchical agglomerative* clustering algorithm in aspect mining (HAC) is proposed in Section 3. An experimental evaluation of our approach, based on some quality measures, is presented in Section 4. We also provide a comparison of the proposed algorithm with other similar approaches. Some conclusions and further work are outlined in Section 5.

2 CLUSTERING APPROACH IN ASPECT MINING

2.1 Theoretical Model

In this section we present the problem of identifying *crosscutting concerns* as a clustering problem.

Let $M = \{m_1, m_2, \ldots, m_n\}$ be an object oriented software system to be mined, where $m_i, 1 \leq i \leq n$ is a method from a class of the system. We denote by n ($|M|$) the number of methods in the system.

We consider a crosscutting concern C as a set of methods that implement this concern, i.e., $C \subset M$, $C = \{c_1, c_2, \ldots, c_{cn}\}$. The number of methods in the crosscutting concern C is $cn = |C|$. Let $\mathcal{SC} = \{C_1, C_2, \ldots, C_q\}$ be the set of all crosscutting

concerns that exist in the system M. The number of crosscutting concerns in the system M is $q = |\mathcal{SC}|$. Let $\mathcal{NC} = M \setminus (\bigcup_{i=1}^{q} C_i)$ be the set of methods from the system M that do not implement any crosscutting concerns.

Definition 1 (Partition of a software system M)**.** The set $\mathcal{K} = \{K_1, K_2, \ldots, K_p\}$ is called a *partition* of the system $M = \{m_1, m_2, \ldots, m_n\}$ iff $1 \leq p \leq n$, $K_i \subseteq M, K_i \neq \emptyset, \forall 1 \leq i \leq p$, $M = \bigcup_{i=1}^{p} K_i$ and $K_i \cap K_j = \emptyset$, $\forall i, j, 1 \leq i, j \leq p, i \neq j$.

In the following we will refer to K_i as to the i^{th} *cluster* of $\mathcal{K}$ and to $\mathcal{K}$ as to a *set of clusters*.

In fact, the problem of aspect mining can be viewed as the problem of finding a partition $\mathcal{K}$ of the system M.

2.2 Identification of Crosscutting Concerns

We propose the following steps for identifying the crosscutting concerns that have the scattered code symptom:

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

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

Grouping – The remaining set of methods is grouped into clusters using a clustering algorithm $\mathcal{CA}$. The clusters are then sorted by the average distance from the point 0_l in descending order, where 0_l is the l dimensional vector with each component 0 (l is the number of attributes characterizing a method).

Analysis – The obtained clusters are analyzed in order to discover which clusters contain methods belonging to crosscutting concerns. We analyze the clusters whose distance from 0_l point is greater than a given threshold.

3 A NEW HIERARCHICAL AGGLOMERATIVE CLUSTERING APPROACH IN ASPECT MINING

In this section we propose a new hierarchical agglomerative clustering algorithm in aspect mining (*HAC*). This algorithm is used in the *Grouping* step of the crosscutting concerns identification process presented in Subsection 2.2.

In our approach, the objects to be clustered are the methods from the software system, $X = \{m_1, m_2, \ldots, m_n\}$. The methods belong to the application classes or are called from the application classes.

Based on the vector space model, we will consider each method as an l-dimensional vector: $m_i = (m_{i1}, \ldots, m_{il})$. We have considered two vector-space models:

- A method m is characterized by a 2-dimensional vector $\{FIV, CC\}$, where FIV is the value of the *fan-in* metric [33] and CC is the number of calling classes. We denote this model by $\mathcal{M}_1$.
- A method m is characterized by an l-dimensional vector $\{FIV, B_1, B_2, \ldots B_{l-1}\}$, where $l-1$ is the number of classes from the software system S (called application classes), FIV is the value of the *fan-in* metric and B_i is the value of the attribute corresponding to the application class AC_i $(1 \leq i \leq l-1)$, as follows:

$$B_i = \begin{cases} 1 & \text{if } m \text{ is called from at least one method belonging to} \\ & \text{application class } AC_i \\ 0 & \text{otherwise.} \end{cases}$$

We denote this model by $\mathcal{M}_2$.

In our approach we will consider that the distance between two methods m_i and m_j is expressed using the *Euclidian distance*, as:

$$d_E(m_i, m_j) = \sqrt{\sum_{k=1}^{l}(m_{ik} - m_{jk})^2}.$$

We have chosen *Euclidian distance* as distance metrics for expressing the dissimilarity between two methods based on the study from [11]. In this paper we have analyzed the influence of several distance metrics (*Euclidian*, *Manhattan* and *Hamming*) for different clustering algorithms in aspect mining. It was experimentally concluded that *Euclidian distance* is the most appropriate for clustering based aspect mining.

The clustering approach that we propose in this section is based on hierarchical agglomerative clustering [26], that is why in the following we briefly present *hierarchical clustering*.

3.1 Hierarchical Clustering

Hierarchical clustering methods represent a major class of clustering techniques [26]. There are two types of hierarchical clustering algorithms: divisive and agglomerative.

Given a set of n objects, the divisive (top-down) methods start from one cluster containing all n objects and split it until n clusters are obtained.

The agglomerative (bottom-up) methods begin with n clusters (each cluster containing a single object), merging them until a single cluster is obtained. At each step, the two most similar clusters are chosen for merging.

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) [28].

The reasons for choosing a hierarchical clustering approach in this paper are:

- Some researchers consider that *hierarchical* clustering algorithms are typically more effective in detecting the true clustering structure of a data set than *partitional* clustering algorithms [23].
- To avoid the main disadvantages of the *partitional* clustering algorithms. In *partitional* clustering the user usually needs to specify some input parameters in advance (the number of clusters – in *k-means* or *k-medoids* clustering [26], a threshold for point density in clusters – for *density* based clustering [25]) and the algorithms give no guarantee for an optimal solution because of their dependance on some initial settings (centroids, medoids, etc.).
- We have introduced in [14] a *partitional* clustering algorithm in aspect mining and, now, we aim to determine if a *hierarchical* clustering approach is more appropriate in aspect mining than a *partitional* one.

3.2 *HAC* Algorithm

In this subsection we present a new Hierarchical Agglomerative Clustering algorithm in aspect mining (*HAC*). We will use this algorithm for obtaining a partition of the software system M.

HAC is based on the idea of hierarchical agglomerative clustering, but stops when a given number of clusters is reached. In order to obtain the number of clusters to be determined, *HAC* uses an heuristic. This heuristic is particular to aspect mining and provides a good enough choice for the number of clusters.

The main idea of *HAC*'s heuristic for choosing the number k of clusters is to determine k representative methods (called *medoids*) from M, performing the following steps:

(i) The initial number k of clusters is n (the number of methods from the system).

(ii) The method chosen as the first medoid is the most "distant" method from the set of all methods (the method that maximizes the sum of distances from all other methods).

(iii) For each remaining methods (that were not chosen as medoids), we compute the minimum distance (*dmin*) from the method and the already chosen medoids. The next medoid is chosen as the method m that maximizes *dmin* and this distance is greater than a given positive threshold (*distMin*). If such a method does not exist it means that m is very close to its nearest medoid nc and should not be chosen as a new medoid (from the aspect mining point of view m and nc should belong to the same (crosscutting) concern). In this case, the number k of clusters will be decreased.

(iv) The step (iii) will be repeatedly performed, until k medoids will be reached.

We have to notice that step (iii) described above assures, from the aspect mining point of view, that near methods (with respect to the given threshold *distMin*) will

be merged in a single (crosscutting) concern, instead of being distributed in different (crosscutting) concerns.

We mention that at steps (ii) and (iii) the choice could be a non-deterministic one. In the current version of *HAC* algorithm, if such a non-deterministic case exists, the first selection is chosen. Improvements of *HAC* algorithm can tackle these kinds of situations.

We have chosen the value 1 for the threshold *distMin*. The reason for choosing this value is based on the following intuition: if the distance between two methods m_i and m_j is less than or equal to 1, we consider that they are similar enough to be placed in the same (crosscutting) concern. In our opinion, from the aspect mining point of view, using *Euclidian distance* as metrics and the vector space models proposed above, the value 1 for $distMin$ makes the difference between a crosscutting and a non-crosscutting concern. Our intuition for choosing the value for the threshold $distMin$ was also experimentally confirmed. In the future we plan to find the most appropriate value for the threshold $distMin$ using supervised learning techniques [4, 32] and to give a rigorous proof for our selection.

The linkage metric between clusters used in *HAC* is complete-link [26].

Below we give the *HAC* algorithm.

Algorithm HAC is
Input:
- the set $M = \{m_1, \ldots, m_n\}$ of methods from the software system to be mined;
- the metric d_E between methods in a multidimensional space;
- $distMin > 0$ the threshold for merging the clusters.

Output:
- $\mathcal{K} = \{K_1, \ldots, K_p\}$ the partition of methods in M.

Begin
$k \leftarrow n$ //the initial number of clusters
$i_1 \leftarrow argmax_{i=1,n} \left\{\sum_{j=1, j\neq i}^{n} d_E(m_i, m_j)\right\}$
//the index i_1 of the first medoid is chosen
$nr \leftarrow 1$ // the number of already chosen medoids
While $nr < k$ do
$D \leftarrow \{j \mid 1 \leq j \leq n,\ j \notin \{i_1, \ldots, i_{nr}\},\ d = min_{l=1,nr} \{d_E(m_j, m_{i_l})\},\ d > distMin\}$
If $D = \emptyset$ then
$k \leftarrow k - 1$ //the number of clusters is decreased
Else
$nr \leftarrow nr + 1$ //another medoid is chosen
$i_{nr} \leftarrow argmax_{j \in D} \{min_{l=1,nr-1}\{d_E(m_j, m_{i_l})\}\}$
EndIf
EndWhile
For i ← 1 to n do
$K_i \leftarrow \{m_i\}$ //each method is put in its own cluster

```
   EndFor
   K ← {K_1, ..., K_n} //the initial partition
   clusNo ← n //the number of clusters
   While clusNo > k do //the desired number of clusters is not reached
      //the most similar clusters are chosen for merging
      (K_i, K_j) ← argmin_(K_i*, K_j*) max_(m' ∈ K_i*, m'' ∈ K_j*) {d_E(m', m'')}
      K_new ← K_i ∪ K_j
      //the most similar clusters are merged
      K ← (K \ {K_i, K_j}) ∪ {K_new}
   EndWhile
   //K is the output partition of the software system M
End.
```

4 EXPERIMENTAL EVALUATION

In order to evaluate the results of *HAC* algorithm from the aspect mining point of view, we use two quality measures defined in [12] and a newly defined one (Subsection 4.1).

These measures will be applied on two case studies, the obtained results being reported in Subsection 4.3. Based on the proposed measures, *HAC* algorithm will be compared with other existing similar approaches.

4.1 Quality Measures

In this subsection we present three quality measures. These measures (*DISP*, *ACC* and *PAM*) evaluate a partition from the aspect mining point of view.

DISP and *PAM* are measures already defined in [12], but *ACC* is newly defined.

In the following, let us consider a partition $\mathcal{K} = \{K_1, \ldots, K_p\}$ of a software system $M = \{m_1, m_2, \ldots, m_n\}$ and $\mathcal{SC} = \{C_1, C_2, \ldots, C_q\}$ the set of all crosscutting concerns from M (Subsection 2.1).

Such a partition can be obtained using a clustering algorithm, as *kAM* or *HAC*.

Definitions 2, 3 and 4 introduce evaluation measures for a partition of a software system from the aspect mining point of view.

Definition 2 (Dispersion of crosscutting concerns – DISP [12])**.** The dispersion of the set $\mathcal{SC}$ in the partition $\mathcal{K}$, denoted by $DISP(\mathcal{SC}, \mathcal{K})$, is defined as

$$DISP(\mathcal{SC}, \mathcal{K}) = \frac{1}{q} \sum_{i=1}^{q} disp(C_i, \mathcal{K}). \tag{1}$$

$disp(C, \mathcal{K})$ is the dispersion of a crosscutting concern C and is defined as:

$$disp(C, \mathcal{K}) = \frac{1}{|D_C|}, \tag{2}$$

where

$$D_C = \{k | k \in \mathcal{K} \ \ and \ \ k \cap C \neq \emptyset\}. \tag{3}$$

D_C is the set of clusters that contain elements which are also in C.

In our view, $DISP(\mathcal{SC}, \mathcal{K})$ defines the dispersion degree of crosscutting concerns in clusters. For a crosscutting concern C, $disp(C, \mathcal{K})$ indicates the number of clusters that contain elements belonging to C.

Lemma 1. If $\mathcal{K}$ is a partition of the software system M and $\mathcal{SC}$ is the set of crosscutting concerns in M, then inequality (4) holds:

$$0 < DISP(\mathcal{SC}, \mathcal{K}) \leq 1. \tag{4}$$

Proof. Because $\forall C_i \in \mathcal{SC}$, C_i is a subset of M ($C_i \subset M$), and $\mathcal{K}$ is a partition of M, there must be at least one cluster $K_{C_i} \in \mathcal{K}$ such that $C_i \cap K_{C_i} \neq \emptyset$. It follows that:

$$D_{C_i} \neq \emptyset \tag{5}$$

From (5) and the definition of D_{C_i} (3), we have:

$$\emptyset \subset D_{C_i} \subseteq \mathcal{K}. \tag{6}$$

From (6) it follows that:

$$1 \leq |D_{C_i}| \leq |\mathcal{K}|, \ \forall C_i \in \mathcal{SC}. \tag{7}$$

From (7) and (2) we have:

$$\sum_{i=1}^{q} \frac{1}{|\mathcal{K}|} \leq \sum_{i=1}^{q} disp(C_i, \mathcal{K}) \leq \sum_{i=1}^{q} 1$$

$\Rightarrow$

$$\frac{1}{|\mathcal{K}|} \leq DISP(\mathcal{SC}, \mathcal{K}) \leq 1. \tag{8}$$

Inequality (8) implies (4), so Lemma 1 is proved. □

Remark 1. Larger values for *DISP* indicate better partitions with respect to $\mathcal{SC}$, meaning that *DISP* has to be maximized.

Definition 3 (ACCuracy of a clustering based aspect mining technique – ACC)**.** Let $\mathcal{T}$ be a clustering based aspect mining technique.

The accuracy of $\mathcal{T}$ with respect to a partition $\mathcal{K}$ and the set $\mathcal{SC}$, denoted by $ACC(\mathcal{SC}, \mathcal{K}, \mathcal{T})$, is defined as:

$$ACC(\mathcal{SC}, \mathcal{K}, \mathcal{T}) = \frac{\sum_{i=1}^{q} acc(C_i, \mathcal{K}, \mathcal{T})}{q}. \tag{9}$$

$acc(C, \mathcal{K}, \mathcal{T})$ is the accuracy of $\mathcal{T}$ with respect to the crosscutting concern C and is defined as:

$$acc(C, \mathcal{K}, \mathcal{T}) = \begin{cases} \frac{|C \cap K_C|}{|C|}, & \text{if } K_C \text{ is the first cluster in which } C \text{ was} \\ & \text{discovered by } \mathcal{T} \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

We consider that a crosscutting concern C was discovered in a cluster K_C if $C \cap K_C \neq \emptyset$, i.e., C has methods in cluster K_C.

For a given crosscutting concern $C \in \mathcal{SC}$, $acc(C, \mathcal{K}, \mathcal{T})$ defines the proportion of methods from C that were discovered by $\mathcal{T}$.

In all clustering based aspect mining techniques, only a part of the clusters are analyzed, meaning that some crosscutting concerns or parts of them may be missed.

Lemma 2. If $\mathcal{K}$ is a partition of the software system M and $\mathcal{SC}$ is the set of crosscutting concerns in M, then inequality (11) holds:

$$0 \leq ACC(\mathcal{SC}, \mathcal{K}, \mathcal{T}) \leq 1. \tag{11}$$

Proof. From (10) we have that:

$$\forall C \in \mathcal{SC},\ 0 \leq |C \cap K_C| \leq |C|. \tag{12}$$

From (12) it follows that:

$$0 \leq acc(C, \mathcal{K}, \mathcal{T}) \leq 1. \tag{13}$$

From (13) and (9) we have:

$$0 \leq ACC(\mathcal{SC}, \mathcal{K}, \mathcal{T}) \leq 1 \tag{14}$$

and Lemma 2 is proved. □

Remark 2. Larger values for ACC indicate better partitions with respect to $\mathcal{SC}$, meaning that ACC has to be maximized.

Definition 4 (Percentage of analyzed methods for a partition – PAM)**.** [12] Let us consider that the partition $\mathcal{K}$ is analyzed in the following order: $K_1, K_2, \ldots, K_p$.

The percentage of analyzed methods for a partition K with respect to the set $\mathcal{SC}$, denoted by $PAM(\mathcal{SC}, \mathcal{K})$, is defined as:

$$PAM(\mathcal{SC}, \mathcal{K}) = \frac{1}{q} \sum_{i=1}^{q} pam(C_i, \mathcal{K}). \tag{15}$$

$pam(C, \mathcal{K})$ is the minimum percentage of the methods that need be analyzed in the partition $\mathcal{K}$ in order to discover the crosscutting concern C, and is defined as:

$$pam(C, \mathcal{K}) = \frac{1}{n} \sum_{j=1}^{s_C} |K_j| \tag{16}$$

where $s_C = min\{t \,|\, 1 \leq t \leq p \text{ and } K_t \cap C \neq \emptyset\}$ is the index of the first cluster in the partition $\mathcal{K}$ that contains methods from C.

$PAM(\mathcal{SC}, \mathcal{K})$ defines the percentage of the minimum number of methods that need be analyzed in the partition in order to discover all crosscutting concerns that are in the system M. We consider that a crosscutting concern was discovered the first time a method that implements it was analyzed.

Lemma 3. If $\mathcal{K}$ is a partition of the software system M and $\mathcal{SC}$ is the set of crosscutting concerns in M, then inequality (17) holds:

$$\frac{|K_1|}{n} \leq PAM(\mathcal{SC}, \mathcal{K}) \leq 1. \tag{17}$$

Proof. $\forall C_i \in \mathcal{SC}$ we have that $1 \leq s_{C_i} \leq p$. It follows that:

$$|\mathcal{K}_1| \leq \sum_{j=1}^{s_{C_i}} |K_j| \leq n, \quad \forall i,\ 1 \leq i \leq q. \tag{18}$$

From (18) and (16) we have:

$$\frac{|K_1|}{n} \leq pam(C_i, \mathcal{K}) \leq 1 \quad \forall i,\ 1 \leq i \leq q. \tag{19}$$

From (19) it follows that:

$$q \cdot \frac{|K_1|}{n} \leq \sum_{i=1}^{q} pam(C_i, \mathcal{K}) \leq q. \tag{20}$$

From (20) and (15) we have:

$$\frac{|K_1|}{n} \leq PAM(\mathcal{SC}, \mathcal{K}) \leq 1. \tag{21}$$

So, Lemma 3 is proved. □

Remark 3. Smaller values for PAM indicate shorter time for analysis, meaning that PAM has to be minimized.

Based on the quality measures defined above, the comparison of the results obtained by different aspect mining techniques can be made from three different criteria:

Partitioning: the degree to which each crosscutting concern is well placed in the partition (using measure *DISP*).

Selection: how well the clusters to be analyzed are chosen (using measure *ACC*).

Ordering: how relevant is the order in which the clusters are analyzed (using measure *PAM*).

In order to compare two partitions obtained by clustering algorithms in aspect mining from the aspect mining point of view, we introduce Definition 5. The definition is based on the properties of the quality measures defined above and considers all the three criteria presented above.

Definition 5. If $\mathcal{K}_1$ and $\mathcal{K}_2$ are two partitions of the software system M, $\mathcal{SC}$ is the set of crosscutting concerns in M and $\mathcal{T}$ is a clustering based aspect mining technique, then $\mathcal{K}_1$ is *better* than $\mathcal{K}_2$ *from the aspect mining point of view* iff the following inequalities hold:

$$DISP(\mathcal{SC}, \mathcal{K}_1) \geq DISP(\mathcal{SC}, \mathcal{K}_2), \quad ACC(\mathcal{SC}, \mathcal{K}_1, \mathcal{T}) \geq ACC(\mathcal{SC}, \mathcal{K}_2, \mathcal{T}),$$

$$PAM(\mathcal{SC}, \mathcal{K}_1) \leq PAM(\mathcal{SC}, \mathcal{K}_2).$$

For the above definition we can remark the following:

Remark 4. If at least one of the inequalities from Definition 5 is not satisfied, we cannot decide which of the partitions $\mathcal{K}_1$ or $\mathcal{K}_2$ is better from the aspect mining point of view (considering all the three criteria simultaneously).

Remark 5. However, the importance of the above mentioned comparison criteria may depend on the user of the aspect mining technique. In our view, the most important criterion is *Selection* (how many crosscutting concerns were discovered), followed by *Partitioning* (how well the crosscutting concerns are grouped) and the last one is *Ordering* (how quickly the crosscutting concerns are discovered).

4.2 Case Studies

Many software applications have been used as case studies in aspect mining. Most of them are publicly available so that everybody can use them (PetStore, Tomcat, JHotDraw v5.4b1 [30], Eclipse v3.2, Carla Laffra's implementation of Dijkstra algorithm [34]), and a few are not publicly available. The complete list of aspect candidates obtained by an aspect mining technique for a particular case study is publicly available only for a few case studies: PetStore, Tomcat and JHotDraw obtained by Fan-in, and JHotDraw and Laffra's Dijkstra obtained by Dynamo-Execution traces. Laffra's Dijkstra is a small application and we have considered that the results mentioned in the papers [10, 21] are enough. That is why we have chosen JHotDraw and Laffra's Dijkstra as case studies for our evaluation.

In the following we will briefly describe JHotDraw and Laffra's Dijkstra case studies considered for evaluating the results of *HAC* algorithm.

Laffra's Dijkstra. The first case study is a Java applet that implements Dijkstra algorithm in order to determine the shortest path in a graph [34]. It was developed by Carla Laffra and it consists of *6* classes and *90* methods.

The set of crosscutting concerns used for the evaluation of this case study is the union of those obtained by Tonella and Ceccato reported in [21], and those obtained by us and reported in [10]. The crosscutting concerns discovered by Tonella and Ceccato and reported in [21] are *(un)locking of GUI* and those discovered by us in [10] are *exception handling* and *consistent behaviour*.

JHotDraw v5.2. The second case study is a Java GUI framework for technical and structured graphics, developed by Erich Gamma and Thomas Eggenschwiler, as a design exercise for using design patterns [30]. It consists of 190 classes and 1 963 methods. We did not include the classes from the `test` package in our experiments.

The set of crosscutting concerns used for the evaluation of this case study is: *Consistent behaviour*, *Composite*, *Observer*, *Command*, *Contract Enforcement*, *Decorator*, and *Change Monitoring*. The set of crosscutting concerns and their implementing methods was constructed using the results reported by Marin et al. in [19].

4.3 Results

In this subsection we give a comparative analysis of the results obtained by *HAC* algorithm with the results obtained by existing similar approaches.

After an in-depth analysis of the existing aspect mining techniques (Subsection 1.3) we have concluded the following:

- Some techniques are dynamic and they depend on the data used during executions [7, 17, 21].
- For most of the techniques [17, 7, 16] only parts of the results are publicly available.
- Most of the existing aspect mining techniques have an associated tool. However, few tools are publicly available so that other people can use them.
- There is no case study used by all these techniques and there is no complete case study available, i.e., for which all existing crosscutting concerns are reported.
- Although measures are essential when evaluating the results obtained by different aspect mining techniques, they were not used very often. Some measures have been used to evaluate the results obtained by aspect mining techniques [6, 9, 15]. In most cases, the measures used are applicable only to those particular aspect mining techniques.

Considering the above and the fact that the quality measures that we have presented in Subsection 4.1 are particular to clustering based aspect mining techniques,

we have focused our evaluation only on aspect mining techniques which use clustering [2, 7, 14, 16]. Shepherd and Pollock have proposed in [16] an aspect mining tool based on clustering that does not automatically identify the crosscutting concerns. The user of the tool has to manually analyze the obtained clusters in order to discover crosscutting concerns. This is another reason for which we did not include this approach in our evaluation.

He and Bai ([7]) have proposed an aspect mining technique based on dynamic analysis and clustering that also uses association rules. They first use clustering to obtain crosscutting concern candidates and then use association rules to determine the position of the source code belonging to a crosscutting concern in order to ease refactoring. The technique proposed by the authors cannot be reproduced, as they do not report neither the clustering algorithm used, nor the distance metric between the objects to be clustered. Also, the results obtained for the case study used by the authors for evaluation are not available. For these reasons, we cannot provide a comparison with this technique.

An evolutionary clustering approach for identifying crosscutting concerns was proposed in [2]. The worst-case time complexity of the genetic clustering algorithm introduced in [2], named *GAM*, is $O(NoOfRuns \cdot NoOfGenerations \cdot (NoOf-Individuals^2 + NoOfIndividuals(n + n \cdot p + p^2 \cdot l))$, where *NoOfRuns* is the number of executions of the algorithm in order to validate its results, *NoOfGenerations* is the number of generations to be created, *NoOfIndividuals* is the number of individuals in a generation (population), p is the number of clusters to be obtained, n is the number of methods from the software system to be mined, and l is the dimension of the vector space model used for characterizing a method. This time complexity is too large and the algorithm does not scale for medium and large software systems. As can be seen in Tables 1 and 2 the values of the quality measures for *GAM* algorithm are not better than those obtained by *HAC* algorithm. That is why this algorithm is not suitable for aspect mining.

Algorithm	Case study	Model	DISP	ACC	PAM
GAM	**JHotDraw**	$\mathcal{M}_1$	0.424	0.293	0.144
GAM	**JHotDraw**	$\mathcal{M}_2$	out of time		
GAM	**Laffra**	$\mathcal{M}_1$	0.666	0.583	0.199
GAM	**Laffra**	$\mathcal{M}_2$	out of time		

Table 1. The values of the quality measures for *GAM* algorithm

Consequently, considering the above, the results obtained by *HAC* are compared only with the results obtained by *kAM* algorithm presented in [14] using the measures introduced in Subsection 4.1.

In Table 2 we present the comparative results after applying *HAC* and *kAM* algorithms, for the vector space models $\mathcal{M}_1$ and $\mathcal{M}_2$ (Section 3), with respect to the quality measures described in Subsection 4.1, for the two case studies described in Subsection 4.2.

Algorithm	Case study	Model	DISP	ACC	PAM
HAC	**JHotDraw**	$\mathcal{M}_1$	0.452	0.319	0.070
HAC	**JHotDraw**	$\mathcal{M}_2$	0.422	0.278	0.0812
kAM	**JHotDraw**	$\mathcal{M}_1$	0.441	0.278	0.073
kAM	**JHotDraw**	$\mathcal{M}_2$	0.422	0.278	0.0819
HAC	**Laffra**	$\mathcal{M}_1$	0.75	0.666	0.124
HAC	**Laffra**	$\mathcal{M}_2$	0.75	0.666	0.160
kAM	**Laffra**	$\mathcal{M}_1$	0.75	0.666	0.168
kAM	**Laffra**	$\mathcal{M}_2$	0.75	0.666	0.148

Table 2. The values of the quality measures for *HAC* and *kAM* algorithms.

From Table 2 we observe, based on Definition 5, that:

- For *JHotDraw* case study, *HAC* algorithm provides *better* results than *kAM* algorithm from the aspect mining point of view, for both vector space models $\mathcal{M}_1$ and $\mathcal{M}_2$.
- For *Laffra* case study, *HAC* algorithm provides *better* results than *kAM* algorithm from the aspect mining point of view, for vector space model $\mathcal{M}_1$. For vector space model $\mathcal{M}_2$ the values of the quality measures *DISP* and *ACC* are the same for both algorithms, but the value of *PAM* is slightly greater for *HAC*. This means that the obtained partitions by both algorithms are the same, but the time needed for the manual analysis of the results is slightly greater for *HAC*. However, considering Remark 5 from Subsection 4.1, the *Ordering* criterion has the lowest importance compared to the *Partitioning* and *Selection* criteria.
- *HAC* algorithm with vector space model $\mathcal{M}_1$ is the *best* from the aspect mining point of view (Definition 5) for both case studies.

Based on the above analysis, we can conclude that vector space model $\mathcal{M}_1$ is more appropriate for clustering based aspect mining, but it can be improved in order to illustrate the *code tangling* symptom, too.

In our view, the vector space model used for clustering in aspect mining has a significant influence on the obtained results.

5 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new *hierarchical agglomerative* clustering approach in aspect mining that uses a newly defined algorithm, *HAC*.

We have evaluated the obtained results from the aspect mining point of view based on three quality measures.

We have given a definition in order to compare two partitions from the aspect mining points of view. Based on this definition, we have shown that *HAC* algorithm provides *better* partitions than *kAM* algorithm [14] and that vector space model $\mathcal{M}_1$ is more appropriate for clustering based aspect mining.

Further work can be done in the following directions:

- To use other approaches for clustering that were proposed in the literature (such as search based clustering [27], or fuzzy clustering [3]).
- To apply this approach for other case studies like JEdit [24].
- To identify a choice for the threshold $distMin$ that will lead to better results.
- To improve the results obtained by *HAC*, by improving the vector space model used for clustering in aspect mining.

Acknowledgement

This work was supported by CNCSIS-UEFISCU, project No. PNII-IDEI 2286/2008.

REFERENCES

[1] Marin, M.—van Deursen, A.—Moonen, L.: Identifying Crosscutting Concerns Using Fan-in Analysis. In: ACM Transactions on Software Engineering and Methodology, Vol. 17, 2007, Issue 1, pp. 1–37.

[2] Serban, G.—Moldovan, G. S.: Aspect Mining using an Evolutionary Approach. WSEAS Transactions on Computers, Vol. 6, 2007, No. 2, pp. 298–305.

[3] Yang, Y.—Huang, S.: Image Segmentation by Fuzzy C-Means Clustering Algorithm with a Novel Penalty Term. Computing and Informatics, Vol. 26, 2007, No. 1, pp. 17–31.

[4] Khorsi, A.: Towards Hybridization of Knowledge Representation and Machine Learning. Computing and Informatics, Vol. 26, 2007, No. 2, pp. 123–147.

[5] Breu, S.—Zimmermann, T.: Mining Aspects from History. In: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, September 2006, pp. 221–230.

[6] Ceccato, M.—Marin, M.—Mens, K.—Moonen, L.—Tonella, P.—Tourwé, T.: Applying and Combining Three Different Aspect Mining Techniques. Software Quality Control, Vol. 14, 2006, No. 3, pp. 209–231.

[7] He, L.—Bai, H.: Aspect Mining Using Clustering and Association Rule Method. International Journal of Computer Science and Network Security, Vol. 6, 2006, pp. 247–251.

[8] Krinke, J.: Mining Control Flow Graphs for Crosscutting Concerns. In: 13th Working Conference on Reverse Engineering: IEEE International Astrenet Aspect Analysis (AAA) Workshop, 2006, pp. 334–342.

[9] Marin, M.—Moonen, L.—van Deursen, A.: A Common Framework for Aspect Mining Based on Crosscutting Concern Sorts. In: WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), IEEE Computer Society, Washington, DC, USA, 2006, pp. 29–38.

[10] Moldovan, G. S.— Serban, G.: Aspect Mining Using a Vector-Space Model Based Clustering Approach. In: Proceedings of Linking Aspect Technology and Evolution (LATE) Workshop, 2006, pp. 36–40.

[11] Moldovan, G. S.—Serban, G.: A Study on Distance Metrics for Partitioning Based Aspect Mining. Studia Universitatis Babes-Bolyai, Informatica, Vol. LI, 2006, No. 2, pp. 53–60.

[12] Moldovan, G. S.—Serban, G.: Quality Measures for Evaluating the Results of Clustering Based Aspect Mining Techniques. In: Proceedings of Towards Evaluation of Aspect Mining (TEAM), ECOOP, 2006, pp. 13–16.

[13] Sampaio, A.—Loughran, N.—Rashid, A.—Rayson, P.: Mining Aspects in Requirements. In: Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005), Chicago, Illinois, USA, 2005.

[14] Serban, G.—Moldovan, G. S.: A New k-Means Based Clustering Algorithm in Aspect Mining. In: 8$^{\text{th}}$ International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '06), IEEE Computer Society, 2006, pp. 26–29.

[15] Bruntink, M.—van Deursen, A.—van Engelen, R.—Tourwé, T.: On the Use of Clone Detection for Identifying Crosscutting Concern Code. In: IEEE Transactions on Software Engineering, Vol. 31, 2005, No. 10, pp. 804–818.

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

[17] Breu, S.—Krinke, J.: Aspect Mining Using Event Traces. In: Proceedings of International Conference on Automated Software Engineering (ASE), 2004, pp. 310–315.

[18] Bruntink, M.—van Deursen, A.—Tourwé, T.—van Engelen, R.: An Evaluation of Clone Detection Techniques for Identifying Crosscutting Concerns. In: ICSM '04: Proceedings of the 20$^{\text{th}}$ IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 2004, pp. 200–209.

[19] Marin, M.—van Deursen, A.—Moonen, L.: Identifying Aspects Using Fan-in Analysis. In: Technical Report SEN-R0413, Centrum voor Wiskunde en Informatica, September 2004.

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

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

[22] Tourwé, T.—Mens, K.: Mining aspectual views using formal concept analysis. In: SCAM '04: Proceedings of the Source Code Analysis and Manipulation, Fourth IEEE International Workshop on (SCAM '04), IEEE Computer Society, Washington, DC, USA, 2004, pp. 97–106.

[23] Sander, J.—Qin, X.—Lu, Z.—Niu, N.—Kovarsky, A.: Automatic Extraction of Clusters from Hierarchical Clustering Representations. In: Proceedings of the 7$^{\text{th}}$ Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2003, LNAI2637, pp. 75–87.

[24] jEdit Programmer's Text Editor: `http://www.jedit.org`, 2002.

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

[26] Jain, A.—Murty, M. N.—Flynn, P.: Data Clustering: A Review. ACM Computing Surveys 31, 1999, pp. 264–323.

[27] Mancoridis, S.—Mitchell, B. S.—Chen, Y.—Gansner, E. R.: 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, IEEE Computer Society, 1999, pp. 50–59.

[28] Jain, A.—Dubes, R.: Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[29] Ganter, B.—Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[30] JHotDraw Project: `http://sourceforge.net/projects/jhotdraw`, 1997.

[31] Kiczales, G.—Lamping, J.—Menhdhekar, A.—Maeda, C.—Lopes, C.—Loingtier, J. M.—Irwin, J.: Aspect-Oriented Programming. In: Proceedings European Conference on Object-Oriented Programming, Vol. 1241, Springer-Verlag, 1997, pp. 220–242.

[32] Mitchell, T. M.: Machine Learning. McGraw-Hill, New York, 1997.

[33] Henderson-Sellers, B.: Object-Oriented Metrics: Measures of Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[34] Laffra, C.: Dijkstra's Shortest Path Algorithm. `http://carbon.cudenver.edu/~hgreenbe/courses/dijkstra/DijkstraApplet.html`, 1996.

Gabriela Czibula has graduated from Babeş-Bolyai University of Cluj-Napoca, Faculty of Mathematics and Computer Science in 1992. She has received the Ph. D. degree in computer science in 2003, with the "cum laude" distinction. She is Professor at the Department of Computer Science, Faculty of Mathematics and Computer Science Babeş-Bolyai University of Cluj-Napoca, Romania. Her research interests include artificial intelligence, machine learning, multiagent systems, software engineering.

Grigoreta Sofia Cojocar has graduated from Babeş-Bolyai University of Cluj-Napoca, Faculty of Mathematics and Computer Science in 2002. She has received the Ph. D. degree in computer science in 2008, with the "cum laude" distinction. She is an Assistant Professor at the Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University of Cluj-Napoca, Romania. Her research interests include software engineering, aspect mining, formal modeling.