# Identifying Crosscutting Concerns Using Partitional Clustering

GABRIELA CZIBULA, GRIGORETA SOFIA COJOCAR, ISTVAN GERGELY CZIBULA
Babeş - Bolyai University
Department of Computer Science
1, M. Kogalniceanu Street, Cluj-Napoca
ROMANIA
gabis@cs.ubbcluj.ro, grigo@cs.ubbcluj.ro, istvanc@cs.ubbcluj.ro

*Abstract: - Aspect mining* is a research direction that tries to identify crosscutting concerns in already developed software systems, without using aspect oriented programming. 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. In this paper we aim at presenting a partitional clustering algorithm for identifying crosscutting concerns in existing software systems. We experimentally evaluate our algorithm using the open source JHotDraw case study, for three distance functions, providing a comparison of the proposed approach with similar existing approaches. The experimental evaluation conducts us to the best semi-metric distance function to be used in the clustering process.

*Key–Words:* Aspect mining, crosscutting concerns, clustering

## 1 Introduction

Nowadays, software systems have become more and more complex and large. Usually, a software system is composed of many core concerns and (some) crosscutting concerns (like logging, exception handling). If core concerns can be cleanly separated and implemented using existing programming paradigms, this is not true for crosscutting concerns, as a crosscutting concern has a more system-wide behavior that cuts across many of the core concerns implementation modules. *Separation of concerns* [22] is a very important principle of software engineering that, in its most general form, refers to the ability to identify, encapsulate and manipulate those parts of software that are relevant to a particular concept, goal, or purpose. The aspect oriented paradigm (AOP) is one of the approaches proposed so far, for designing and implementing crosscutting concerns [13]. Aspect oriented techniques allow crosscutting concerns to be implemented in a new kind of module called *aspect*, by introducing new language constructs like pointcuts and advices.

Kiczales et al. introduce for the first time aspect oriented programming (AOP) in [13]. Since 1997 the aspect oriented paradigm has been slowly adopted by the industry, too, leading to the appearance of new research problems like software reverse engineering, reengineering, and refactoring to use the aspect-oriented paradigm in order to benefit from the advantages it brings.

*Aspect mining* is a 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.

There are many reasons for migrating a legacy system to an aspect oriented based system. An inadequate solution for crosscutting concerns implementation has a negative impact on the final system with consequences like duplicated code, *scattering* of concerns throughout the entire system and *tangling* of concern-specific code with that of other concerns. These consequences lead to software systems that are hard to maintain and to evolve. When aspect oriented techniques are used, the crosscutting concerns are cleanly separated from the core concerns.

The main contribution of this paper is to introduce a partitional clustering algorithm for identifying crosscutting concerns in existing software systems.

The rest of the paper is structured as follows. Section 2 presents some existing work in the field of *aspect clustering*. The main aspects related to the problem of clustering, particularly to partitional clustering are presented in Section 3. A new partitional clustering algorithm (*PACO*) for identifying crosscutting concerns is proposed in Section 4. An experimental evaluation of the proposed algorithm is presented in Section 5, together with a comparison of our approach with similar existing approaches. Some conclusions of the paper and further research directions are given in Section 6.

## 2 Related Work

Aspect mining is a relatively new research domain. However, many aspect mining techniques have been proposed. Some use metrics [18], some use formal concept analysis [4, 30, 31], or execution relations [3]. There are also a few approaches that use clone detection techniques [5, 28] or natural language processing [24]. A few techniques use clustering in order to identify crosscutting concerns [9, 21, 26, 29].

In the following we will briefly present some of the existing approaches in aspect mining.

Marin et al [18] have proposed an aspect mining technique that uses the *fanin* metric [10]. Their idea is to search for crosscutting concerns among the methods that have the value of the fanin metric greater than a given threshold. The result obtained by this technique can be viewed as a *partition* of the software system to be mined. The partition contains two clusters: the first one contains the methods that have the fanin greater than the given threshold and the second contains the remaining methods.

A graph based approach in aspect mining is introduced in [25]. The basic idea of this technique is to determine methods that are similar. The approach is to construct a graph between the methods of the software system, to determine the connex components of this graph, called *clusters*, and then to identify crosscutting concerns in the obtained clusters.

Breu and Krinke [3] 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. This approach is adapted to static analysis in [14]. In this approach the recurring execution relations are obtained from the control flow graph of the program.

Qu and Liu [23] proposed a modified approach of that proposed by Breu and Krinke. They also mine for crosscutting concerns using the same execution relations defined by Breu and Krinke, but they obtain these relations using a method call tree. This approach uses a method call tree to generate method call traces. These traces are then investigated for the same recurring method patterns and the same constrains as those defined by Breu and Krinke.

Tonella and Ceccato [30] 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 [7] is applied on these execution traces and the concepts that satisfy some constraints are considered as aspect candidates.

Tourwé and Mens [31] 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 [5] 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 [28] 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 [4] 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 [24] 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 a few aspect mining techniques proposed in the literature that use *clustering* in order to identify crosscutting concerns [9, 26, 27, 29].

He and Bai [9] have proposed an aspect mining technique based on dynamic analysis. They obtain execution traces for each use case, but they apply clustering and association rules to discover aspect candidates.

Shepherd and Pollock [29] 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.

Şerban and Moldovan have proposed a clustering approach for identifying crosscutting concerns and a partitional clustering algorithm named *kAM* is introduced in [26].

An *evolutionary* approach in aspect mining is introduced in [27] by Şerban and Moldovan and two *genetic clustering* algorithms used to identify crosscutting concerns are proposed.

# 3 Partitional Clustering. The *k-medoids* clustering algorithm

*Unsupervised classification*, or *clustering*, as it is more often referred as, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects [8], being considered the most important unsupervised learning problem. 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 $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ be the set of objects to be clustered. The measure used for discriminating objects can be any *metric* or *semi-metric* function $d : \mathcal{O} \times \mathcal{O} \longrightarrow \Re$. The distance expresses the dissimilarity between objects.

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 $k, k \leq n$, 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 (i.e., a squared error function); the squared error criterion tends to work well with isolated and compact clusters [11].

In *k-medoids* or *PAM* (Partitioning around medoids) algorithm [12], each cluster is represented by one of the objects in the cluster. It finds representative objects, called *medoids*, in clusters. The algorithm starts with $k$ initial representative objects for the clusters (medoids), then iteratively recalculates the clusters (each object is assigned to the closest cluster - medoid), and their medoids until convergence is achieved. At a given step, a medoid of a cluster is replaced with a non-medoid if it improves the total distance of the resulting clustering [12].

The main disadvantages of *PAM* algorithm are:

- The performance of the algorithm depends on the initial centroids. So, the algorithm gives no guarantee for an optimal solution.
- The user needs to specify the number of clusters in advance.

# 4 A New Partitional Clustering Algorithm for Crosscutting Concerns Identification (PACO)

In this section we introduce a new partitional clustering algorithm (*PACO*) (Partitional Clustering Algorithm for Crosscutting Concerns Identification) for identifying crosscutting concerns in existing software systems. In order to discover the crosscutting concerns from the system, we analyze the source code of the software system to be mined. All classes, methods and relations between them are computed. Afterwards, *PACO* algorithm is used to identify a **partition** of a software system $S$ in which the methods belonging to a crosscutting concern should be grouped together. The final step is to manually analyzed the obtained results.

Let us consider that the software system to be mined consists of a set of classes $\mathcal{C} = \{c_1, c_2, \ldots, c_s\}$, each class containing one ore more methods. In our clustering approach, the objects to be clustered are the methods from the software system $S$, i.e., $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$. Our focus is to group the methods such that the methods belonging to the same crosscutting concern are placed in the same cluster.

## 4.1 Distance functions

In order to apply a clustering algorithm, the dissimilarity degree between any two methods from the software system $S$ have to be considered.

Crosscutting concerns in non AO systems have two symptoms: *code scattering* and *code tangling*. In the following we will define three distance functions in order to express the dissimilarity degree between the methods from the software system considering the *code scattering* and the *code tangling* symptoms.

### "Scattering" distance

The *code scattering* symptom means that the code that implements a crosscutting concern is spread across the system.

The first distance function that we propose illustrates the *code scattering* symptom. Consequently, we consider the distance $d_S(m_i, m_j)$ between two methods $m_i$ and $m_j$ as expressed in Equation (1).

$$d_S(m_i, m_j) = \begin{cases} 1 - \frac{|in(m_i) \cap in(m_j)|}{|in(m_i) \cup in(m_j)|} & if\ in(m_i) \cap in(m_j) \neq \emptyset \\ \infty & otherwise \end{cases} \quad (1)$$

where, for a given method $m \in \mathcal{M}$, $in(m)$ defines the set of methods and classes that invoke $m$, as expressed in Equation (2).

$$in(m) = \{m\} \cup \{m' \in \mathcal{M} \cup \mathcal{C} |\ m'\ \ invoke\ \ m\}. \quad (2)$$

In our view, the distance between two methods as defined in (1) expresses the following idea: if two methods are invoked by common methods or classes, they should belong to the same cluster. This means that scattered methods would be placed in the same cluster, and the methods that do not represent aspects would not be placed together with methods from aspects as the latest are invoked from multiple places. Consequently, methods that belong to the same crosscutting concern are close (considering distance $d_S$) to each other.

Many programming languages allow the definition of inner classes, classes that are defined inside of another class. The definition of classes inside of other classes are used in order to group classes that are related. This semantic information can be relevant for identifying crosscutting concerns. Grouping of classes frequently appears in real life software projects, and our distance consider this situation, also. The set $in(m)$ contains not only the class *c* that invoke the method, but also the class that contains class $c$.

**"Tangling" distance**

The *code tangling* symptom means that the code that implements some concern is mixed with code from other (crosscutting) concerns.

The second distance function that we propose illustrates the *code tangling* symptom. For a method $m$ from the software system to be mined we denote by $r(m)$ the set of relevant properties for each invocation context $inv \in in(m)$. Each invokation of the metod $m$ is analyzed and the set $r(m)$ is constructed by collecting all the local variables defined and used, all used attributes, all classes accesed and parameter types used by the invoker method. The ideea is to collect all the information that can be considered as the context of the invocation for method $m$.

Consequently, we consider the distance $d_T(m_i, m_j)$ between two methods $m_i$ and $m_j$ as expressed in Equation (3).

$$d_T(m_i, m_j) = \begin{cases} 1 - \frac{|r(m_i) \cap r(m_j)|}{|r(m_i) \cup r(m_j)|} & if\ r(m_i) \cap r(m_j) \neq \emptyset \\ \infty & otherwise \end{cases} \quad (3)$$

where, for a given method $m \in \mathcal{M}$, $in(m)$ is defined as in Equation (2).

In our view, the distance between two methods as defined in (3) expresses the following idea: if two methods are invoked in similar contexts, they should belong to the same cluster. Consequently, methods that belong to the same crosscutting concern are close (considering distance $d_T$) to each other.

**"Scattering-Tangling" distance**

The third distance function that we propose considers both the *code scattering* and *code tangling* symptoms. Consequently, we consider the distance $d_{ST}(m_i, m_j)$ between two methods $m_i$ and $m_j$ as expressed in Equation (4).

$$d_{ST}(m_i, m_j) = min\{d_S(m_i, m_j), d_T(m_i, m_j).\} \quad (4)$$

## 4.2 *PACO* algorithm

In order to avoid the two main disadvantages of the traditional *k-medoids* algorithm, *PACO* algorithm uses a heuristic for choosing the number of medoids (clusters) and the initial medoids. This heuristic is particular to aspect mining and it will provide a good enough choice of the initial medoids.

After selecting the initial medoids, *PACO* behaves like the classical *k-medoids* algorithm.

In the following we will consider that the distance $d$ between two methods from the software system is given by one of the distances $d_S$ (Equation (1)), $d_T$ (Equation (3)) or $d_{ST}$ (Equation (4)).

The main idea of *PACO*'s heuristic for choosing the initial medoids and the number *p* of clusters (medoids) is the following:

(i) The initial number *p* of clusters is *n* (the number of methods from the software 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) In order to choose the next medoid we reason as follows. For each remaining method (that was not chosen as medoid), 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 positive given threshold (*distMin*). If such a method does not exist, it means that $m$ is very close to all the medoids and should not be chosen as a new medoid. From the aspect mining point of view this means that $m$ should belong to the same (crosscutting) concern with an existing medoid. In this case, the number *p* of medoids will be decreased.

(iv) The step (iii) will be repeatedly performed, until *p* 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 (cluster), 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 *PACO* algorithm, if such a non-deterministic case exists, the first selection is made. Future improvements of *PACO* algorithm will deal with these kind of situations.

The main idea of the *PACO* algorithm that we apply in order to group methods from a software system is the following:

(i) The initial number $p$ of clusters and the initial medoids are determined by the heuristic described above.

(ii) The clusters are recalculated, i.e., each object is assigned to the closest medoid.

(iii) Recalculate the medoid $i$ of each cluster $k$ based on the following idea: if $h$ is an object from $k$ such that $\sum_{j \in k}(d(j,h) - d(j,i))$ is negative, then $h$ becomes the new medoid of cluster $k$.

(iv) Steps (ii)-(iii) are repeatedly performed until there is no change in the partition $\mathcal{K}$.

We give, next, *PACO* algorithm.

Algorithm PACO is
**Input:**
- the software system $\mathcal{S} = \{m_1, \ldots, m_n\}$
- the semi-metric $d$ between methods
- the threshold $distMin$,
- $noMaxIter$ the maximum number of iterations allowed.
**Output:**
- the partition $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ of the system $S$.
**Begin**
$k \leftarrow n$ //the initial number of medoids
//the index $i_1$ of the first medoid is chosen
$i_1 \leftarrow argmax_{i=1,n} \left\{ \sum_{j=1, j \neq i}^{n} d(m_i, m_j) \right\}$
//the number of the already chosen medoids
$nr \leftarrow 1$
While $nr < k$ do
$D \leftarrow \{j \mid 1 \leq j \leq n,\ j \notin \{i_1, ..., i_{nr}\},\ d = min_{l=1,nr}\left\{d(m_j, m_{i_l})\right\},\ d > distMin\}$
If $D = \emptyset$ then
//the number of medoids is decreased
$k \leftarrow k - 1$
else
$nr \leftarrow nr + 1$//another medoid is chosen
$i_{nr} \leftarrow argmax_{j \in D}\left\{min_{l=1,nr-1}\{d(m_j, i_l)\}\right\}$
endif
endwhile
$p \leftarrow k$ //the number of medoids
//the initial medoids are determined
For $j \leftarrow 1, k$ do
$medoid_j \leftarrow m_{i_j}$
endfor
While ($\mathcal{K}$ changes) and
(no. of iterations $\leq noMaxIter$) do
For $j \leftarrow 1, p$ do
$K_j \leftarrow \{s_i \mid 1 \leq i \leq n,\ \forall l,\ l \neq j,$
$d(m_i, medoid_j) \leq d(m_i, medoid_l)\}$
endfor
For $i \leftarrow 1, p$ do
If $K_i = \emptyset$ then
//the number of clusters is decreased
@ Remove element $K_i$ from $\mathcal{K}$
else
@ Calculate the new medoid of $K_i$
For each $h \in K_i$ do
@ Compute $dist = \sum_{j \in k}(d(j,h) - d(j, medoid_i))$
If $dist < 0$ then
$medoid_i \leftarrow h$
endif
endfor
endif
endfor
endwhile
**End.**

We mention that *PACO* algorithm provides a partition of a software system $S$, partition that ideally would contain separate clusters for each crosscutting concern.

Regarding to *PACO* algorithm, we have to notice the following:

- If, at a given moment, a cluster becomes empty, this means that the number of clusters will be decreased.
- Because the initial medoids are selected based on the heuristic described above, the dependence of the algorithm on the initial medoids is eliminated.
- We have chosen the value 1 for the threshold $distMin$, because distances greater than 1 are obtained only for unrelated entities (Equations (1), (3), (4)). Our intuition for choosing

the value for the threshold $distMin$ was experimentally confirmed. The most appropiate value for $distMin$ is dependent on the nature of the analyzed system. In the future we plan to find the most appropriate value for the threshold $distMin$ using supervised learning techniques [19] and to give a rigorous proof for our selection.

# 5 Experimental Evaluation

In this section we want to evaluate how well did *PACO* algorithm succeed in grouping the elements from crosscutting concerns in clusters. For this purpose we consider a simple Java code example and the open source JHotDraw case study. The evaluation will be made considering a measure that is described in Subsection 5.1.

## 5.1 Evaluation measure

In order to evaluate the results we use a quality measure, called *DISP*, that we have previously introduced in [20]. This measure defines the dispersion degree of crosscutting concerns in clusters, considering, for each crosscutting concern, the number of clusters that contain elements belonging to the concern.

We give below the formal definition of *DISP* measure. In the following $CCC$ denotes the set of crosscutting concerns existing in a software system, $\mathcal{K}$ denotes a partition of the set $\mathcal{M}$ of methods from the software system to be mined. The partition $\mathcal{K}$ can be obtained using a clustering algorithm (*PACO* in this paper).

**Definition 1** [20] ***DISPersion of crosscutting concerns - DISP.***
*The dispersion of the set $CCC$ of crosscutting concerns in the partition $\mathcal{K}$, denoted by $DISP(CCC, \mathcal{K})$, is defined as*

$$DISP(CCC, \mathcal{K}) = \frac{1}{|CCC|} \sum_{i=1}^{|CCC|} disp(C_i, \mathcal{K}). \tag{5}$$

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

$$disp(C, \mathcal{K}) = \frac{1}{|D_C|}, \tag{6}$$

*where*

$$D_C = \{k | k \in \mathcal{K} \text{ and } k \cap C \neq \emptyset\}. \tag{7}$$

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

The values of the *DISP* measure are in the interval $[0, 1]$. The proof can be found in [20]. In order to obtain more cohesive partitions, $DISP$ measure has to be maximized.

## 5.2 Example

In this subsection we present a small example that shows how methods are grouped in clusters by *PACO* algorithm using distances $d_S$ (Equation (1)), $d_T$ (Equation (3)) and $d_{ST}$ (Equation (4)). We have chosen the example below in order to provide the reader with an easy to follow example of crosscutting concerns identification.

Let us consider the Java code example given in Table 1.

```
public class A {
 public A(L l){
  mC();
  mB(); }
 public void mA(L l){
  l.m1();
  mB();
  l.m2(); }
 public void mB(){ }
 public void mC(){ }
}
public class L {
 public L(){}
 public void m1(){}
 public void m2(){}
}
public class B {
 private L l=new L();
 private A a;
 public B(){
  a=new A(l);
  a.mA(l);
  a.mB(); }
 public void mC(){
  l.m1(); }
 public void mD(){
  a.mB();
  a.mC(); }
}
```

Table 1: Java Code example.

For the code illustrated in Table 1, the set of crosscutting concerns is $CCC = \{C_1\}$, where $C_1 = \{L.m1(), L.m2()\}$.

#### 5.2.1 Distance $d_S$

The distances between the methods from the system described above are given in Table 2. We present only the non-zero distances that are not $\infty$.

| Method | Method | Distance |
|---|---|---|
| A.A | A.mA | 0.5 |
| A.A | A.mB | 0.57 |
| A.A | A.mC | 0.66 |
| A.A | B.B | 0.66 |
| A.A | L.L | 0.5 |
| A.A | L.m1 | 0.85 |
| A.mA | A.mB | 0.57 |
| A.mA | A.mC | 0.85 |
| A.mA | B.B | 0.66 |
| A.mA | L.L | 0.5 |
| A.mA | L.m1 | 0.66 |
| A.mA | L.m2 | 0.8 |
| A.mB | A.mC | 0.5 |
| A.mB | B.B | 0.85 |
| A.mB | A.mD | 0.85 |
| A.mB | L.L | 0.75 |
| A.mB | L.m1 | 0.66 |
| A.mB | L.m2 | 0.75 |
| A.mC | A.mD | 0.8 |
| A.mC | L.L | 0.85 |
| A.mC | L.m1 | 0.75 |
| A.mC | L.m2 | 0.85 |
| B.B | L.L | 0.66 |
| B.mC | L.m1 | 0.8 |
| L.L | L.m1 | 0.85 |
| L.m1 | L.m2 | 0.66 |

Table 2: Distances $d_S$ between the methods.

After applying *PACO* algorithm using distance $d_S$, the obtained clusters are shown in Table 3.

#### 5.2.2 Distance $d_T$

The distances between the methods from the system described above are given in Table 4. We present only the non-zero distances that are not $\infty$.

After applying *PACO* algorithm using distance $d_T$, the obtained clusters are shown in Table 5.

#### 5.2.3 Distance $d_{ST}$

The distances between the methods from the system described above are given in Table 6. We present only the non-zero distances that are not $\infty$.

| Cluster | Methods |
|---|---|
| C1 | { B.mC } |
| C2 | { B.mD } |
| C3 | { L.m1, L.m2 } |
| C4 | { A.A, A.mA, A.mB, A.mC, B.B, L.L } |

Table 3: The clusters obtained by *PACO* using distance $d_S$.

| Method | Method | Distance |
|---|---|---|
| A.A | A.mB | 0.28 |
| A.A | A.mC | 0.61 |
| A.A | L.m1 | 0.64 |
| A.A | L.m2 | 0.83 |
| A.mA | A.mB | 0.28 |
| A.mA | A.mC | 0.61 |
| A.mA | L.m1 | 0.64 |
| A.mA | L.m2 | 0.83 |
| A.mB | A.mC | 0.28 |
| A.mB | L.L | 0.28 |
| A.mB | L.m1 | 0.37 |
| A.mB | L.m2 | 0.37 |
| A.mC | L.L | 0.61 |
| A.mC | L.m1 | 0.64 |
| A.mC | L.m2 | 0.73 |
| L.L | L.m1 | 0.64 |
| L.L | L.m2 | 0.83 |
| L.m1 | L.m2 | 0.28 |

Table 4: Distances $d_T$ between the methods.

After applying *PACO* algorithm using distance $d_{ST}$, the obtained clusters are shown in Table 7.

Analyzing the results obtained for the Java code example described in Table 1, for distances $d_S$, $d_T$, $d_{ST}$ we obtain the folowing:

- For each distance used, *PACO* algorithm identifies the existing crosscutting concern $C_1$.
- Both elements of the crosscutting concern $C_1$ (i.e., $L.m1$ and $L.m2$) are grouped in the same cluster, for each distance.
- The value of $DISP$ measure is 1 for every distance.
- The best results are obtained using distance $d_S$, as the cluster containing the crosscutting concern $C_1$ does not contain methods from other concerns.

| Cluster | Methods |
|---|---|
| C1 | { B.B } |
| C2 | { B.mC } |
| C3 | { B.mD } |
| C4 | { A.A, A.mA, A.mB, A.mC, L.L, L.m1, L.m2 } |

Table 5: The clusters obtained by *PACO* using distance $d_T$.

### 5.3 JHotDraw case study

In this subsection we consider the open source JHotDraw version 5.4b1 case study [6] for evaluating *PACO* algorithm. It 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. It consists of **396** classes and **3359** methods.

The set of crosscutting concerns used for the evaluation is : *Adapter*, *Command*, *Composite*, *Consistent behavior*, *Contract enforcement*, *Decorator*, *Exception handling*, *Observer*, *Persistence*, and *Undo*. The set of crosscutting concerns and their implementing methods was constructed using the results reported by Marin et al. and publicly available at [17].

We have applied *PACO* algorithm for JHotDraw case study for the distance functions $d_S$, $d_T$ and $d_{ST}$. The obtained results are shown in Table 8.

Analyzing the obtained results, we can conclude the folowing:

- The best results are obtained using distance $d_S$, as the value of $DISP$ measure is greater for this distance. Considering both the simple example from Subsection 5.2 and JHotDraw case study, we can experimentally conclude that distance semi-metric $d_S$ is the best distance to be used by $PACO$ algorithm in the clustering process.

- As distance $d_{ST}$ illustrates both *scattering* and *tangling* symptoms, we would expect to obtain better result with this distance. But, as shown in Table 8 with $d_{ST}$ distance, *PACO* provides the worst results. That is why we have to improve the distance $d_{ST}$ in order to better illustrate both *scattering* and *tangling* symptoms.

We have compared the results obtained by *PACO* algorithm with the results obtained by *kAM* algorithm proposed in [26]. *kAM* algorithm is based on the idea of k-means clustering and uses an heuristic for choosing the initial centroids and the initial number of clusters. The similarity between two methods is computed using a vector space model based approach. The value of the $DISP$ measure obtained by *kAM* for the same case study is $0.4005$.

| Method | Method | Distance |
|---|---|---|
| A.A | A.mB | 0.28 |
| A.A | A.mC | 0.61 |
| A.A | B.B | 0.66 |
| A.A | L.m1 | 0.64 |
| A.A | L.m2 | 0.83 |
| A.mA | A.mB | 0.28 |
| A.mA | A.mC | 0.61 |
| A.mA | B.B | 0.66 |
| A.mA | L.m1 | 0.64 |
| A.mA | L.m2 | 0.8 |
| A.mB | A.mC | 0.25 |
| A.mB | B.B | 0.85 |
| A.mB | B.mD | 0.85 |
| A.mB | L.L | 0.28 |
| A.mB | L.m1 | 0.37 |
| A.mB | L.m2 | 0.37 |
| A.mC | B.mD | 0.8 |
| A.mC | L.L | 0.61 |
| A.mC | L.m1 | 0.64 |
| A.mC | L.m2 | 0.73 |
| B.B | L.L | 0.66 |
| B.mC | L.m1 | 0.8 |
| L.L | L.m1 | 0.64 |
| L.L | L.m2 | 0.83 |
| L.m1 | L.m2 | 0.28 |

Table 6: Distances $d_{ST}$ between the methods.

Comparatively, considering the $DISP$ measure, *PACO* algorithm has obtained better results than *kAM* algorithm. This means that in the partition obtained by *PACO* algorithm the methods from the crosscutting concerns were better grouped than in the partition obtained by *kAM* algorithm. However, the elements of crosscutting concerns are spread in two or more clusters of a partition for both algorithms, as the values of the $DISP$ measure are less than 0.5 for both *PACO*

| Cluster | Methods |
|---|---|
| C1 | { B.mC } |
| C2 | { B.mD } |
| C3 | { A.A, A.mA, A.mB, A.mC, B.B, L.L, L.m1, L.m2 } |

Table 7: The clusters obtained by *PACO* using distance $d_{ST}$.

| Distance function | DISP |
|---|---|
| $d_S$ | 0.4444 |
| $d_T$ | 0.4433 |
| $d_{ST}$ | 0.4207 |

Table 8: Results for JHotDraw case study.

and *kAM* algorithms.

We did not provide a comparison of the considered approach with the two other existing clustering based aspect mining approaches for the following reasons:

- Shepherd and Pollock have proposed in [29] 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.
- The technique proposed by He and Bai [9] 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.

The non-clustering aspect mining techniques cannot be compared with our approach because of the following reasons:

- some techniques are dynamic and they depend on the data used during executions [3, 30];
- for the static techniques [18] only parts of the results are publicly available;
- there is no case study used by all these techniques.

# 6 Conclusions and Future Work

We have presented in this paper a new partitional clustering algorithm (*PACO*) that can be used for identifying crosscutting concerns in existing software systems. The proposed algorithm uses an heuristic for choosing the number of clusters and the initial medoids, reducing this way the disadvantages of the traditional *k-medoids* method.

In order to evaluate the obtained results, we have considered JHotDraw case study.

Further work can be done in the following directions:

- To improve *PACO* algorithm by improving the heuristic used for selecting the initial medoids.
- To use other clustering techniques that were proposed in the literature [1, 2, 15, 16] in order to identify crosscutting concerns in existing software systems.
- To improve the distances used for discriminating the methods in the clustering process in order to better illustrate the *code scattering* and *code tangling* symptoms.
- To apply *PACO* algorithm on real software systems.
- To approach the problem of *aspect mining* by using machine learning techniques [19].

**Acknowledgements:** This work was supported by the research project ID_2268/2009, sponsored by the Romanian National University Research Council (CNCSIS).

*References:*

[1] J. Avila, A. Ramirez, C. Cruz, and I. Vasquez-Alvarez. The clustering algorithm for nonlinear system identification. *WSEAS Transactions on Computers*, 7(7), 2008, pp.1179–1188.

[2] W. Barbakh and C. Fyfe. A novel construction of connectivity graphs for clustering and visualization. *WSEAS Transactions on Computers*, 7(5), 2008, pp.424–434.

[3] S. Breu and J. Krinke. Aspect Mining Using Event Traces. In *Proc. Intern. Conference on Automated Software Engineering (ASE)*, 2004, pp. 310–315.

[4] S. Breu and T. Zimmermann. Mining Aspects from Version History. In S. Uchitel and S. Easterbrook, editors, *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*. ACM Press, September 2006.

[5] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwé. On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10), 2005, pp.804–818.

[6] E. Gamma. JHotDraw Project. http://sourceforge.net/projects/jhotdraw.

[7] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. Translator-C. Franzke.

[8] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[9] L. He and H. Bai. Aspect Mining using Clustering and Association Rule Method. *International Journal of Computer Science and Network Security*, 6(2A), February 2006, pp. 247–251.

[10] B. Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

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

[12] L. Kaufman and P. J. Rousseeuw. Clustering by means of Medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, 1987, pp. 405–416.

[13] 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, Springer-Verlag, 1997, pp. 220–242.

[14] J. Krinke. 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.

[15] N. P. Lin, C.-I. Chang, H.-E. Chueh, H.-J. Chen, and W.-H. Hao. A deflected grid-based algorithm for clustering analysis. *WSEAS Transactions on Computers*, 7(3), 2008, pp. 125–132.

[16] 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*, IEEE Computer Society, 1999, pp. 50–59.

[17] M. Marin. Fan-in jhotdraw v5.4b1 results. http://swerl.tudelft.nl/bin/view/AMR/FanInAnalysisResults#JHotDraw_v_54b1.

[18] 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).*, IEEE Computer Society, 2004, pp. 132–141.

[19] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[20] G. Moldovan and G. Serban. Clustering based aspect mining formalized. *WSEAS Transactions on Computers*, 6(2), 2007, pp. 199–206.

[21] G. S. Moldovan and G. Serban. Aspect Mining using a Vector-Space Model Based Clustering Approach. In *Proceedings of Linking Aspect Technology and Evolution (LATE) Workshop*, Bonn, Germany, March, 20 2006, pp. 36–40.

[22] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of ACM*, 15(12), 1972, pp. 1053–1058.

[23] L. Qu and D. Liu. Aspect Mining Using Method Call Tree. In *Proceedings of International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, 2007, pp. 407–412.

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

[25] G. Serban and G. S. Moldovan. A Graph Algorithm for Identification of Crosscutting Concerns. *Studia Universitatis Babes-Bolyai, Informatica*, LI(2), 2006, pp. 53–60.

[26] G. Serban and G. S. Moldovan. A New k-means Based Clustering Algorithm in Aspect Mining. In *Proceedings of 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*, Timisoara, Romania, September, 26-29 2006, IEEE Computer Society, pp. 69–74.

[27] G. Serban and G. S. Moldovan. Aspect Mining using an Evolutionary Approach. *WSEAS Transactions on Computers*, 6(2), 2007, pp. 298–305.

[28] D. Shepherd, E. Gibson, and L. Pollock. Design and Evaluation of an Automated Aspect Mining Tool. In *2004 International Conference on Software Engineering and Practice*, IEEE, June 2004, pp. 601–607.

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

[30] P. Tonella and M. Ceccato. Aspect Mining through the Formal Concept Analysis of Execution Traces. In *Proc. of the 11th Working Conference on Reverse Engineering (WCRE'04)*, Washington, DC, USA, IEEE Computer Society 2004, pp. 112–121.

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