KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT2007
Cluj-Napoca (Romania), June 6–8, 2007, pp. 252–260

A NEW GRAPH-BASED APPROACH IN ASPECT MINING

GABRIELA ŞERBAN[1] AND GRIGORETA SOFIA COJOCAR[2]

ABSTRACT. *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 ([3]), in order to make them easier to maintain and to evolve. This paper aims at presenting a new graph-based approach in aspect mining. We define the problem of identifying the crosscutting concerns as a search problem in a *graph* and we introduce *GRAM* algorithm for solving this problem. We evaluate based on some quality measures the results obtained by applying *GRAM* algorithm from the aspect mining point of view. The proposed approach is compared with a clustering approach in aspect mining ([5]) and a case study is also reported.

1. Introduction

1.1. **Aspect Mining.** *Separation of concerns* ([1]) 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. Some of the benefits of a good separation of concerns are reduced software complexity, improved comprehensability, limited impact of change, easy evolution and reuse.

Aspect Oriented Programming (AOP) ([3]) provides means to encapsulate concerns which cannot be modularized using traditional programming techniques. These concerns are called *crosscutting concerns*. Logging and exception handling are well known examples of crosscutting concerns. Aspect oriented paradigm offers a powerful technology for supporting the separation of crosscutting concerns. Such a concern is explicitly specified as an *aspect*. Aspects encapsulate the implementation of a crosscutting concern. A special tool, called *weaver*, integrates a number of aspects to obtain the final software system. *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.

2000 *Mathematics Subject Classification.* 68N99, 68R10.
Key words and phrases. Software Engineering, Aspect Mining, Graph.

©2007 Babeş-Bolyai University, Cluj-Napoca

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.

1.2. **Related Work.** Several approaches have been considered for aspect mining until now. Some approaches use clone detection techniques to identify duplicate code that might indicate the presence of crosscutting concerns ([12], [13]). Another approach uses metrics to identify crosscutting concerns that have the scattering symptom ([8]). There are also two approaches that use dynamic analysis to discover crosscutting concerns: one that analyzes the program traces to discover recurring execution relations ([11]), and one that uses formal concept analysis to analyze the execution traces ([14]).

A few aspect mining techniques proposed in the literature use *clustering* in order to identify crosscutting concerns ([5], [6], [7]). In [6] a vector space model based clustering approach in aspect mining is proposed. This approach is improved in [5], by defining a new *k-means* based clustering algorithm in aspect mining (kAM). In [7] the methods are clustered based on their names, and then the user can navigate among the clusters, visualize the source code of the methods and identify the crosscutting concerns.

In this paper we propose a new graph-based approach, as an alternative to the clustering approach in aspect mining.

The paper is structured as follows. A theoretical model on which we base our approach is introduced in Section 2. Section 3 presents a new graph based approach in aspect mining. An experimental evaluation of our approach, based on some quality measures, is presented in Section 4. The obtained results are compared with the ones obtained by applying kAM algorithm ([5]). Some conclusions and further work are outlined in Section 5.

2. Theoretical Model

In this section we present the problem of identifying *crosscutting concerns* as a problem of identifying a partition of a software system.

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

We consider a crosscutting concern as a set of methods $C \subset M$, $C = \{c_1, c_2, \ldots, c_{cn}\}$, methods that implement this concern. The number of methods in the crosscutting concern C is $cn = |C|$. Let $CCC = \{C_1, C_2, ..., C_q\}$ be the set of all crosscutting concerns that exist in the system M. The number of crosscutting concerns in the system M is $q = |CCC|$. Let $NCCC = M \setminus (\bigcup_{i=1}^{q} C_i)$ be the set of methods from the system M, methods that do not implement any crosscutting concerns.

Definition 1. ***Partition of a software system*** M.
The set $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ *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 the i-th *cluster* of $\mathcal{K}$.

In fact, the problem of aspect mining can be viewed as the problem of finding a partition $\mathcal{K}$ of the system M. If the result of an AM technique is a *partition* of the software system, we will call it **partitioning aspect mining technique**.

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 in order to obtain a partition of the software system M (in our approach using *GRAM* algorithm).
- **Analysis** - A part of the obtained clusters are analyzed in order to discover which clusters contain methods belonging to crosscutting concerns.

We mention that at the **Grouping** step, a partition of the software system can be obtained using a clustering algorithm ([5]) in aspect mining, or using *GRAM* algorithm, that will be introduced in the next section.

3. A New Graph-Based Approach In Aspect Mining

In this section we present the problem of obtaining a *partition* (Definition 1) of a software system as a search problem in a *graph*. This graph based approach is, in fact, a method to identify the clusters in the system and can be viewed as an alternative to a *clustering* algorithm in aspect mining ([5]).

In our approach, the objects to be grouped (clustered) are the methods from the software system: $m_1, m_2, \ldots, m_n$. The methods belong to the application classes or are called from the application classes.

Based on the vector space model, we will consider that the vector associated with the method m is $\{FIV, CC\}$, where FIV is the fan-in value ([8]) of m (the number of methods that call m) and CC is the number of calling classes for m.

In our approach, we will consider the *Euclidian distance* between methods as a measure of dissimilarity between them.

After a partition of the software system is determined using a **partitioning aspect mining technique**, the clusters are sorted by the average distance from the point 0_2 in descending order, where 0_2 is the two-dimensional vector with each component 0 (in our case two is the dimension of the vector space model). Then, we analyze the clusters whose distance from 0_2 point is greater than a given threshold.

3.1. **The Graph Of Concerns.** In this section we introduce the concept of *graph of concerns* and auxiliary definitions needed to define our search problem. The concept of *graph of concerns* introduced below is different from the concept of *concerns graph* defined in [2] by Robillard and Murphy and it is used in a different context.

We mention that the idea of constructing the *graph of concerns* is specific to aspect mining and will be explained later.

Definition 2. *Let $M = \{m_1, m_2, \dots, m_n\}$ be a software system and d_E the* Euclidian distance *metric between methods in a multidimensional space. The* ***graph of concerns*** *corresponding to the software system M, denoted by $\mathcal{GC}_M$, is an undirected graph defined as follows: $\mathcal{GC}_M = (\mathcal{V}, \mathcal{E})$, where:*

- *The set $\mathcal{V}$ of vertices is the set of methods from the software system, i.e., $\mathcal{V} = \{m_1, m_2, \dots, m_n\}$.*
- *The set $\mathcal{E}$ of edges is $\mathcal{E} = \{(v_1, v_2) \mid v_1, v_2 \in \mathcal{V},\ v_1 \neq v_2,\ d_E(v_1, v_2) \leq$ **distMin**$\}$, where* ***distMin*** *is a given threshold.*

We have chosen the value 1 for the threshold *distMin*. The reason for choosing this value is the following: if the distance between two methods m_i and m_j is less or equal to 1, we consider that they are similar enough to be placed in the same (crosscutting) concern. We mention that, from the aspect mining point of view, using *Euclidian distance* as metric and the vector space model proposed above, the value 1 for $distMin$ makes the difference between a crosscutting concern and a non-crosscutting one.

In Definition 3 below we will define the problem of computing a partition of the software system M.

Definition 3. *Let $M = \{m_1, m_2, \dots, m_n\}$ be a software system, d_E (*Euclidian distance*) the metric between methods in a multidimensional space and $\mathcal{GC}_M$ the corresponding graph of concerns* (Definition 2)*. We define the problem of computing a partition $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ of M as the problem of computing the connected components of $\mathcal{GC}_M$.*

3.2. ***GRAM*** **Algorithm.** In this subsection we briefly describe *GRAM* algorithm for determining a *partition* $\mathcal{K}$ of a software system M. This algorithm will be used in the **Grouping** step (Section 2) for identification of crosscutting concerns.

Let us consider a software system $M = \{m_1, m_2, \ldots, m_n\}$ and the *Euclidian distance* d_E between methods in a multidimensional space, and the problem introduced in Definition 3.

The main steps of *GRAM* algorithm are:

(i) Create the *graph of concerns*, $\mathcal{GC}_M$, as shown in Definition 2. We mention that the threshold *distMin* used for creating the edges in the graph is chosen to be 1. The reason for this choice was explained above.
(ii) Determine the connected components of $\mathcal{GC}_M$. These components give a partition $\mathcal{K}$ of the software system M.

3.3. **Example.** In the following, we present a small example that shows how methods are grouped in clusters by *GRAM* algorithm. If we have the classes shown in Table 1, the values of the attribute set are presented in Table 2 and the corresponding graph of concerns is shown in Figure 1. The obtained clusters are given in Table 3.

```
public class A {
   private L l;
   public A(){l=new L(); methB();}
   public void methA(){ l.meth(); methB();}
   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	2	1
B.B	0	0
B.methC	0	0
B.methD	0	0
L.L	1	1
L.meth	3	2

TABLE 2. Attribute values.

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

TABLE 3. The obtained clusters.

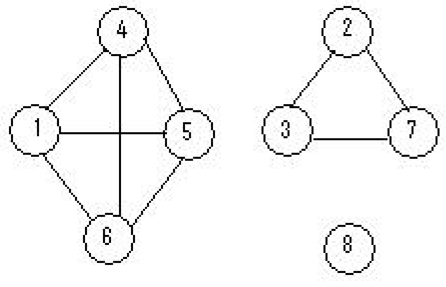

FIGURE 1. Graph of concerns.

4. Experimental Evaluation

In order to evaluate the results of *GRAM* algorithm from the aspect mining point of view, we use a set of quality measures defined in [4].

These measures will be applied on a case study (Subsection 4.2). The obtained results will be reported in Subsection 4.2. Based on the obtained results, *GRAM* algorithm will be compared with *kAM* algorithm proposed in [5].

4.1. **Quality Measures.**
In this subsection we present three quality measures. These measures (*DIV*, *ACT* and *PAN*) evaluate a partition from the aspect mining point of view.

DIV is a measure already defined in [4], but *ACT* and *PAN* are newly defined. All these measures evaluate a partition of a software system from the aspect mining point of view.

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 $CCC = \{C_1, C_2, ..., C_q\}$ the set of all crosscutting concerns from M (Section 2). The partition $\mathcal{K}$ can be obtained using *GRAM* algorithm or using a clustering algorithm, like *kAM* ([5]).

$DIV(CCC, \mathcal{K})$ is introduced in [4] and defines the degree to which each cluster contains methods from different crosscutting concerns or methods from other concerns. $DIV(CCC, \mathcal{K})$ takes values in $[0, 1]$ and larger values for *DIV* indicate better partitions with respect to *CCC*, meaning that *DIV* has to be maximized.

Definition 4. ***ACcuracy of a partitioning based aspect mining Technique - ACT.***

Let $\mathcal{T}$ be a partitioning aspect mining technique.

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

$$ACT(CCC, \mathcal{K}, \mathcal{T}) = \frac{1}{q}\sum_{i=1}^{q} act(C_i, \mathcal{K}, \mathcal{T}).$$

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

$$act(C, \mathcal{K}, \mathcal{T}) = \sum_{j=1}^{r} \frac{|C \cap K_j|}{|C|}$$

where r $(1 \leq r \leq p)$ is the last cluster analyzed by $\mathcal{T}$.

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

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

Based on Definition 4, $ACT(CCC, \mathcal{K}, \mathcal{T}) \in (0, 1]$. Larger values for ACT indicate better partitions with respect to CCC, meaning that ACT has to be maximized.

Definition 5. ***Percentage of ANalyzed methods for a partition - PAN.***

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 $\mathcal{K}$ with respect to the set CCC, denoted by $PAN(CCC, \mathcal{K})$, is defined as:

$$PAN(CCC, \mathcal{K}) = \frac{1}{q} \sum_{i=1}^{q} pan(C_i, \mathcal{K}).$$

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

$$pan(C, \mathcal{K}) = \frac{1}{n} \sum_{j=1}^{s} |K_j|$$

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

$PAN(CCC, \mathcal{K})$ defines the percentage of the number of methods that need to 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 when all the methods that implement it were analyzed.

Based on Definition 5, it can be proved that $PAN(CCC, \mathcal{K}) \in (0, 1]$. Smaller values for PAN indicate shorter time for analysis, meaning that PAN has to be minimized.

4.2. **Results.** In order to evaluate the results of *GRAM* algorithm, we consider as case study JHotDraw, version 5.2 ([9]).

This 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. It consists of **190** classes and **1963** methods.

In this subsection we present the results obtained after applying *GRAM* algorithm described in Subsection 3.2, for the vector space model presented in Section 3, with respect to the quality measures described in Subsection 4.1, for the case study presented above.

The results obtained by *GRAM* are compared with the results obtained by *kAM* algorithm proposed in [5].

In Table 1 we present the comparative results.

Algorithm	DIV	ACT	PAN
GRAM	0.857	0.299	0.346
kAM	0.842	0.278	0.361

Table 4. The values of the quality measures for JHotDraw case study.

From Table 1 we observe, based on the properties of the quality measures defined in the above subsection, that *GRAM* algorithm provides **better** results from the aspect mining point of view, than **kAM** algorithm.

In our view, the vector space model has a significant influence on the obtained results. We are working on improving the vector space model in order to handle the tangling code symptom, too.

So far no comparison between existing aspect mining techniques was reported in the literature. No comparison between *GRAM* and other similar approaches is provided for the following reasons:

- some techniques are dynamic and they depend on the data used during executions ([11], [14]);
- for the static techniques ([8], [7]) only parts of the results are publicly available;
- there is no case study used by all these techniques.

5. Conclusions and Future Work

We have presented in this paper a new *graph-based approach* in aspect mining. For this purpose we have introduced *GRAM* algorithm, that identifies a partition of a software system. This partition is analyzed in order to identify the crosscutting concerns from the system. In order to evaluate the obtained results from the aspect mining point of view, we have used a set of quality measures. Based on these measures, we showed that *GRAM* algorithm provides **better** partitions than *kAM* algorithm (previously introduced in [5]).

Further work can be done in the following directions:

- To apply this approach for other case studies, like JEdit ([10]).
- To compare the results provided by *GRAM* with the results of other approaches in aspect mining.
- To identify a choice for the threshold *distMin* that will lead to better results.

- To improve the results obtained by *GRAM*, by improving the vector space model used.

References

[1] David. L. Parnas. On The Criteria To Be Used in Decomposing Systems Into Modules. *Communications of the ACM*, 15(12), 1972, pp. 1053–1058.

[2] Robillard, M.P., Murphy, G.C.: Concern graphs: finding and describing concerns using structural program dependencies. In: *Proceedings of the 24th International Conference on Software Engineering.* Orlando, Florida (2002) 406–416

[3] 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.* Volume 1241. Springer-Verlag (1997) 220–242

[4] 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) 13–16

[5] Serban, G., Moldovan, G.S.: 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).* (2006) 60–64

[6] 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) 36–40

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

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

[9] JHotDraw Project: http://sourceforge.net/projects/jhotdraw (1997)

[10] jEdit Programmer's Text Editor: http://www.jedit.org (2002)

[11] Breu, S., Krinke, J.: Aspect Mining using Event Traces. In: *Proceedings of International Conference on Automated Software Engineering.* (2004) 310–315

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

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

[14] 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).* (2004) 112–121

[1] Department of Computer Science, Babeş-Bolyai University, 1, M. Kogalniceanu Street, Cluj-Napoca, Romania,

E-mail address: `gabis@cs.ubbcluj.ro`

[2] Department of Computer Science, Babeş-Bolyai University, 1, M. Kogalniceanu Street, Cluj-Napoca, Romania,

E-mail address: `grigo@cs.ubbcluj.ro`