STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 2, 2006

A GRAPH ALGORITHM FOR IDENTIFICATION OF CROSSCUTTING CONCERNS

GABRIELA ŞERBAN AND GRIGORETA SOFIA MOLDOVAN

Abstract. The purpose of this paper is to present 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 *GAAM* algorithm (*Graph Algorithm in Aspect Mining*) for solving this problem. We evaluate the results obtained by applying *GAAM* algorithm from the aspect mining point of view, based on a set of quality measure that we have previously defined in [3]. The proposed approach is compared with a clustering approach in aspect mining ([4]) and a case study is also reported.

Keywords: graph, algorithm, aspect mining.

1. Introduction

1.1. **Aspect Mining.** The Aspect Oriented Programming (AOP) is a new paradigm that is used to design and implement *crosscutting concerns* [2]. A *crosscutting concern* is a feature of a software system that is spread all over it, and whose implementation is tangled with other features' implementation. A well known example of crosscutting concern is logging. In order to design and implement a crosscutting concern, AOP introduces a new modularization unit called *aspect*. Better modularization, and higher productivity are some of the advantages that AOP brings to software engineering.

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.

1.2. **Related Work.** Several approaches have been considered for aspect mining until now. One approach was to develop tools that would help the user to navigate and to analyze the source code in order to find crosscutting concerns. Some of them

Received by the editors: September, 30, 2006.

2000 *Mathematics Subject Classification.* 68N99, 68R10.

1998 *CR Categories and Descriptors.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement –*Restructuring, reverse engineering, and reengineering*; G.2.2 [**Mathematics of Computing**]: Discrete Mathematics – *Graph Theory*.

rely on lexical analysis, and some also include a type-based search ([1], [15], [16], [17]). Other approach uses clone detection techniques to identify duplicate code, that might indicate the presence of crosscutting concerns ([13], [18], [19]). These are all static approaches that analyze the source code for crosscutting concerns. There are also two dynamic approaches: one that analyzes the event traces ([12]), and one that uses formal concept analysis to analyze the execution traces ([21]). In [20] formal concept analysis is used again, but in a static manner. A comparison of three different approaches can be found in [14].

There is also a clustering approach that constructs the clusters based on the methods' names ([7]). The user can then navigate among the clusters, visualize the source code of the methods and identify the crosscutting concerns.

There are just a few aspect mining techniques proposed in the literature that use *clustering* in order to identify crosscutting concerns ([4], [5], [6], [7]).

In [5] a vector space model based clustering approach in aspect mining is proposed. This approach is improved in [4], by defining a new *k-means* based clustering algorithm in aspect mining (kAM).

In [3], a part of a formal model for clustering in aspect mining is introduced and a set of quality measures for evaluating the results of clustering based aspect mining techniques is presented.

In this paper we propose a new graph-based approach, as an alternative to the clustering approach in aspect mining. Such an approach has not been reported, in the literature, so far.

The paper is structured as follows. A theoretical model on which we base our approach is introduced in Section 2. Section 3 presents our approach. 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 ([4]). 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 consider a crosscutting concern as a set $C = \{c_1, c_2, ..., c_{cn}\}$ with $C \subset M$, of 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.

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 such that $CCC \subset \mathcal{K}$. So, in Definition 2 we introduce the notion of **partitioning aspect mining technique**, that will be used in our approach.

Definition 2. ***Partitioning aspect mining technique.***
Let $\mathcal{T}$ be an aspect mining technique and M a software system to be mined. We say that $\mathcal{T}$ is a ***partitioning aspect mining technique*** *if the result obtained by $\mathcal{T}$ is a* ***partition*** (Definition 1) *$\mathcal{K}$ of M.*

For a software system, 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, in a Java program etc.*
- **Grouping** - The remaining set of methods is grouped in order to obtain a partition of the software system M (in our approach $GAAM$).
- **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 ([4]) in aspect mining, or using $GAAM$ algorithm, that will be introduced in the next section.

3. Our Approach

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 ([4]).

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 each method as a l-dimensional vector: $m_i = (m_{i1}, \ldots, m_{il})$.

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.

We have considered two vector-space models that illustrate only the *scattered code* symptom. Future development will also consider the *code tangling* symptom.

- 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. We denote this model by $\mathcal{M}_1$.
- The vector associated with the method m is $\{FIV, B_1,\ B_2, ...B_{l-1}\}$, where FIV is the fan-in value of m and B_i $(1 \leq i \leq l-1)$ is 1, if the method m is called from a method belonging to the application class C_i, and 0, otherwise. We denote this model by $\mathcal{M}_2$.

As in a vector space model based clustering ([22]), we consider the *distance* between methods as a measure of dissimilarity between them.

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

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

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_l in descending order, where 0_l is the l dimensional vector with each component 0 (l is the dimension of the vector space model). Then, we analyze the clusters whose distance from 0_l point is greater than a given threshold.

3.1. **The Methods Graph.** In this section we introduce the concept of *methods graph* and auxiliary definitions needed to define our search problem.

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

Definition 3. *Let $M = \{m_1, m_2, \ldots, m_n\}$ be a software system and d_E* (Equation 1) *the metric between methods in a multidimensional space. The* ***methods graph*** *corresponding to the software system M, $\mathcal{MG}_M$, is an undirected graph defined as follows: $\mathcal{MG}_M = (\mathcal{V}, \mathcal{E})$, where:*

- *The set $\mathcal{V}$ of vertices is the set of methods from the software system, i.e., $\mathcal{V}\{m_1, m_2, \ldots, m_n\}$.*

- *The set* $\mathcal{E}$ *of edges is* $\mathcal{E} = \bigcup_{i=1}^{n} \{(m_i, m_j) \,|\, 1 \leq j \leq n,\, j \neq i,\, d_E(m_i, m_j) = min\{d_E(m_i, m_k),\ 1 \leq k \leq n,\ k \neq i,\ (m_i, m_k) \notin \mathcal{E}\} \wedge d_E(m_i, m_k) \leq distMin\}$*, 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 models proposed above, the value 1 for $distMin$ makes the difference between a crosscutting concern and a non-crosscutting one.

In Definition 4 below we will define the problem of computing a partition of the software system M.

Definition 4. *Let* $M = \{m_1, m_2, \ldots, m_n\}$ *be a software system,* d_E (Equation 1) *the metric between methods in a multidimensional space and* $\mathcal{MG}_M$ *the corresponding* ***methods graph*** (Definition 3). *We define the problem of computing a partition* $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ *of* M *as the problem of computing the connex components of* $\mathcal{MG}_M$.

3.2. ***GAAM* Algorithm.** In this subsection we briefly describe *GAAM* 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 metric d_E (Equation 1) between methods in a multidimensional space.

The main steps of *GAAM* algorithm are:

(i) Create the *methods graph*, $\mathcal{MG}_M$, as shown in Definition 3. 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 connex components of $\mathcal{MG}_M$. These components give a partition $\mathcal{K}$ of the software system M.

4. Experimental Evaluation

In order to evaluate the results of *GAAM* algorithm from the aspect mining point of view, we use three quality measures defined in [3]: *DIV*, *PAM* and *PREC*.

These measures will be applied on a case study (Subsection 4.1). The obtained results will be reported in Subsection 4.1. Based on the obtained results, *GAAM* algorithm will be compared with *kAM* algorithm proposed in [4].

In order to compare two partitions of a software system M from the aspect mining point of view, we introduce the Definition 5. The definition is based on the properties of the quality measures defined above ([3]).

Definition 5. *If $\mathcal{K}_1$ and $\mathcal{K}_2$ are two partitions of the software system M, CCC is the set of crosscutting concerns in M and $\mathcal{T}$ is a partitioning 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:*

$$DIV(CCC, \mathcal{K}_1) \geq DIV(CCC, \mathcal{K}_2), \;\; PREC(CCC, \mathcal{K}_1, \mathcal{T}) \geq PREC(CCC, \mathcal{K}_2, \mathcal{T}),$$
$$PAM(CCC, \mathcal{K}_1) \leq PAM(CCC, \mathcal{K}_2).$$

Remark 1. *If at least one of the inequalities from Definition 5 are not satisfied, we cannot decide which of the partitions $\mathcal{K}_1$ or $\mathcal{K}_2$ is better.*

4.1. **Results.** In order to evaluate the results of $GAAM$ 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 in **190** classes and **1963** methods.

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

The results obtained by $GAAM$ are compared with the results obtained by kAM algorithm proposed in ([4]).

In Table 1 we present the comparative results.

Algorithm	Model	DIV	PREC	PAM
GAAM	$\mathcal{M}_1$	0.844	0.875	0.073
kAM	$\mathcal{M}_1$	0.842	0.875	0.073
GAAM	$\mathcal{M}_2$	0.993	0.875	0.073
kAM	$\mathcal{M}_2$	0.993	0.875	0.081

TABLE 1. The values of the quality measures for JHotDraw case study.

From Table 1 we observe, based on Definition 5, that $GAAM$ algorithm provides **better** results from the aspect mining point of view, than kAM algorithm, for both vector space models $\mathcal{M}_1$ and $\mathcal{M}_2$.

Moreover, $GAAM$ with vector space model $\mathcal{M}_2$ provides the best results.

We can conclude that vector space model $\mathcal{M}_2$ is more appropriate, from the aspect mining point of view.

5. Conclusions and Future Work

We have presented in this paper a new *graph-based approach* in aspect mining. For this purpose we have proposed $GAAM$ algorithm, that identifies a partition of a software system. This partition will be 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.

We have given a definition in order to compare two partitions from the aspect mining point of view. Based on this definition, we showed that $GAAM$ algorithm provides **better** partitions than kAM algorithm (previously introduced in [4]).

Further work can be done in the following directions:

- To apply this approach for other case studies like JEdit ([11]).
- To compare the results provided by $GAAM$ 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 $GAAM$, by improving the vector space model used.
- To determine the distance metric used that will provide better results from the aspect mining point of view (Minkowski, Manhattan, Hamming, etc).
- To identify the heuristics for constructing the *methods graph* that will lead to better results from the aspect mining point of view.

References

[1] 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, pp. 406–416.

[2] 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, pp. 220–242.

[3] 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.

[4] Serban, G., Moldovan, G.S., "A new k-means based clustering algorithm in aspect mining", In: 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06), 2006, pp. 60–64.

[5] 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, to be published.

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

[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, pp. 132–141.

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

[10] 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.
[11] jEdit Programmer's Text Editor: http://www.jedit.org, 2002.
[12] Breu, S., Krinke, J., "Aspect Mining using Event Traces", In: *Proceedings of International Conference on Automated Software Engineering*, 2004, pp. 310–315.
[13] 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.
[14] Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., Tourwé, T., "A Qualitative Comparison of Three Aspect Mining Techniques", In: *IWPC '05, Proceedings of the 13th International Workshop on Program Comprehension*, IEEE Computer Society, 2005, pp. 13–22.
[15] Griswold, W.G., Kato, Y., Yuan, J.J., "AspectBrowser: Tool Support for Managing Dispersed Aspects", Technical Report CS1999-0640, UCSD, 3, 2000.
[16] Hannemann, J., Kiczales, G., "Overcoming the Prevalent Decomposition of Legacy Code", In: *Advanced Separation of Concerns Workshop,at the International Conference on Software Engineering. (ICSE)*, 2001.
[17] Zhang, C., Gao, G., Jacobsen, H., "Multi Visualizer", http://www.eecg.utoronto.ca/ czhang/amtex/.
[18] Sheperd, 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.
[19] Morales, O.A.M., "Aspect Mining Using Clone Detection", Master's thesis, Delft University of Technology, The Netherlands, 2004.
[20] Tourwé, T., Mens, K., "Mining Aspectual Views using Formal Concept Analysis", In: *Proc. IEEE International Workshop on Source Code Analysis and Manipulation*, 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)*, 2004, pp. 112–121.
[22] Jain, A., Dubes, R., "Algorithms for Clustering Data", Prentice Hall, Englewood Cliffs, New Jersey, 1998.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: grigo@cs.ubbcluj.ro