A Formal Model for Partitioning based Aspect Mining

GRIGORETA SOFIA (MOLDOVAN) COJOCAR[1]
GABRIELA SERBAN[1]

Babeş Bolyai University
Department of Computer Science
1, M. Kogalniceanu Street, Cluj - Napoca
RO-400085, Romania
[1] (grigo, gabis)@cs.ubbcluj.ro

Abstract. *Separation of concerns* 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 a software system that are relevant to a particular concept, goal, or purpose. *Aspect Oriented Programming* provides means to encapsulate concerns which cannot be modularized using traditional programming techniques. These concerns are called *crosscutting concerns*. *Aspect Mining* is a research direction that tries to identify crosscutting concerns in legacy systems. The aim of this paper is to introduce a new formal model for partitioning based aspect mining. Such a model was not defined in the literature, yet. The applicability of the proposed formal model is studied on three different aspect mining techniques.

Keywords: formal model, aspect mining, partitioning.

(Received November 17, 2006 / Accepted March 27, 2007)

1 Introduction

Separation of concerns ([10]) 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 a software system 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) ([5]) 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. The 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.

In order to apply AOP principles to legacy software systems, it is necessary to analyze the existing implementation to discover the crosscutting concerns and to refactor them into aspects. The research on *aspect mining* refers to the identification and analysis of non-localized crosscutting concerns throughout an existing legacy software system ([3]). The goal of aspect mining is to support aspect-oriented refactoring to improve software comprehensibility, reusability and maintainability.

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.

A software system cannot contain only crosscutting concerns. It is composed of core concerns and concerns that crosscut them.

There is no general formal model for aspect mining defined in the literature, yet. In [7, 8] we have defined a formal model for clustering based aspect mining techniques. This is the only formal model defined for aspect mining, so far.

The paper is structured as follows. In Section 2 we briefly present the formal model for clustering based aspect mining ([7, 8]), that is extended by our model. Section 3 presents the new formal model for partitioning based aspect mining. The applicability of this formal model on three different aspect mining techniques is presented in Section 4. Some conclusions and further work are given in Section 5.

2 Formal Model For Clustering Based Aspect Mining

In this section we briefly present the formal model for clustering based aspect mining that we have previously introduced in [7, 8]. We aim at extending this model for partitioning based aspect mining.

2.1 Definitions

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a software system, represented by a multiset of elements. An *element* s_i can be a statement, a method, a class, a module, etc.

In the following, we will consider a crosscutting concern as a multiset of elements that implement this concern, $C \subset S$, $C = \{c_1, c_2, \ldots, c_{cn}\}$, $C \neq \emptyset$. The number of elements in the crosscutting concern C is $cn = |C|$. Let $CCC = \{C_1, C_2, \ldots, C_q\}$ be the set of all crosscutting concerns that exist in the system S, $C_i \cap C_j = \emptyset$, $\forall i, j,\ 1 \leq i, j \leq q,\ i \neq j$. Let $NCCC = S \setminus (\bigcup_{i=1}^{q} C_i)$ be the multiset of elements from the system S that do not implement any crosscutting concerns. As a software system cannot be composed only of crosscutting concerns, $NCCC \neq \emptyset$.

Definition 1 ([7, 8]) ***Partition of a system** S.*
*The set $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ is called a **partition** of the system S iff:*
(1) $1 \leq p \leq n$
(2) $K_i \subseteq S, K_i \neq \emptyset, \forall\, i, 1 \leq i \leq p$
(3) $S = \bigcup_{i=1}^{p} K_i$
(4) $K_i \cap K_j = \emptyset,\ \forall\, i, j, 1 \leq i, j \leq p, i \neq j.$

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

Formally, the problem of aspect mining can be viewed as the problem of identifying a partition $\mathcal{K}$ of the software system S.

A partition of a software system S can be obtained by an aspect mining technique, as a clustering based aspect mining one ([12]) or a graph based one ([11]).

Abstractly, a clustering based aspect mining technique $\mathcal{T}$ can be viewed as a tuple of functions:

$$\mathcal{T} = (divide, select, order),$$

where:

- $divide$ is a function that maps a software system S to a partition $\mathcal{K}$ of the system S, i.e., $divide(S) = \mathcal{K}$. Consequently, the domain of $divide$ is the set of all software systems, and its codomain is the set of partitions of software systems.
- $select$ is a function that indicates the clusters from $\mathcal{K}$ that will be analyzed by the user of the aspect mining technique, i.e., $select(S, \mathcal{K}) = \mathcal{SK}, \mathcal{SK} \subseteq \mathcal{K}$.
- $order$ is a function that indicates the order in which the selected clusters (given by the function $select$) will be analyzed by the user of the technique.

In order for a technique $\mathcal{T}$, to be efficient, Equation (1) should hold:

$$CCC = \mathcal{SK}. \tag{1}$$

In practice, the above equality is hard to be satisfied, that is why, it is acceptable that $CCC \subseteq \mathcal{SK}$. However, as smaller the set $\mathcal{SK} \setminus CCC$ is, as efficient $\mathcal{T}$ is.

The following definitions classify the partitions obtained by a clustering based aspect mining technique considering the efficiency condition given by Equation (1).

Definition 2 ([7, 8]) ***Good partition of a system** S.*
*Being given a partition $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ of the system S, $\mathcal{K}$ is called a **good partition** of the system S with respect to the set $CCC = \{C_1, C_2, ..., C_q\}$ of its crosscutting concerns, iff:*
(1) $p \geq q$
(2) $\forall\, C, C \in CCC,\ \exists K_C \in \mathcal{K}$ *such that* $C \subseteq K_C$ *and* $\forall\, A, A \in CCC \setminus \{C\},\ A \cap K_C = \emptyset$.

Intuitively, $\mathcal{K}$ is a good partition of the system S if all the elements implementing a crosscuting concern C_i $(1 \leq i \leq q)$ are in the same cluster K_{j_i} $(1 \leq j_i \leq p)$ and this cluster does not contain elements from other crosscutting concerns.

Definition 3 ([7, 8]) ***Optimal partition of a system** S.*
*Being given a partition $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ of the system S, $\mathcal{K}$ is called an **optimal partition** of the system*

S with respect to the set $CCC = \{C_1, C_2, ..., C_q\}$ of all crosscutting concerns, iff:
(1) $p \geq q$
(2) $\forall\, C, C \in CCC,\ \exists K_C \in \mathcal{K}$ *such that* $C = K_C$.

Intuitively, $\mathcal{K}$ is an optimal partition of the system S if all the elements implementing a crosscuting concern C_i $(1 \leq i \leq q)$ are in the same cluster K_{j_i} $(1 \leq j_i \leq p)$ and they are the only elements in K_{j_i}.

Remark 1 *An optimal partition of a software system S is a good partition in which the elements implementing a crosscutting concern are the only elements in their corresponding cluster.*

2.2 Quality Measures

In this subsection we briefly present two quality measures that can be used to determine if a partition $\mathcal{K}$ of a software system S is optimal with respect to its set of crosscutting concerns CCC. The proofs of Lemma 1, Lemma 2, and Theorem 4 can be found in [7, 8].

Definition 4 ([7, 8]) ***DISPersion of crosscutting concerns - DISP.***
The dispersion of the set CCC in the partition $\mathcal{K}$, denoted by $DISP(CCC, \mathcal{K})$, is defined as

$$DISP(CCC, \mathcal{K}) = \frac{1}{|CCC|} \sum_{i=1}^{|CCC|} disp(C_i, \mathcal{K}). \quad (2)$$

In (2) $disp(C_i, \mathcal{K})$ is the dispersion of a crosscutting concern C_i and is defined as:

$$disp(C_i, \mathcal{K}) = \frac{1}{|D_{C_i}(\mathcal{K})|}, \quad (3)$$

where

$$D_{C_i}(\mathcal{K}) = \{k \mid k \in \mathcal{K} \text{ and } k \cap C_i \neq \emptyset\}. \quad (4)$$

D_{C_i} is the set of clusters that contain elements from C_i.

$DISP(CCC, \mathcal{K})$ 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 ([7, 8]) *If $\mathcal{K}$ is a partition of the software system S and CCC is the set of crosscutting concerns in S, then inequality (5) holds:*

$$0 < DISP(CCC, \mathcal{K}) \leq 1. \quad (5)$$

Remark 2 *Larger values for $DISP$ indicate better partitions with respect to CCC, meaning that $DISP$ has to be maximized.*

Definition 5 ([7, 8]) ***DIVersity of a partition - DIV.***
The diversity of a partition $\mathcal{K}$ with respect to the set CCC, denoted by $DIV(CCC, \mathcal{K})$, is defined as

$$DIV(CCC, \mathcal{K}) = \frac{1}{|\mathcal{K}|} \sum_{i=1}^{|\mathcal{K}|} div(CCC, K_i). \quad (6)$$

$div(CCC, k)$ is the diversity of a cluster $k \in \mathcal{K}$ and is defined as:

$$div(CCC, k) = \frac{1}{|V_k| + \tau(k)} \quad (7)$$

where

$$V_k = \{C \mid C \in CCC \text{ and } k \cap C \neq \emptyset\} \quad (8)$$

is the set of crosscutting concerns that have elements in k, and

$$\tau(k) = \begin{cases} 1 & \text{if } k \cap NCCC \neq \emptyset \\ 0 & \text{if } k \cap NCCC = \emptyset. \end{cases} \quad (9)$$

$\tau(k)$ is 1 if the cluster k contains elements that do not implement any crosscutting concerns, and 0 otherwise.

$DIV(CCC, \mathcal{K})$ defines the degree to which each cluster contains elements from different crosscutting concerns or elements from other concerns.

Lemma 2 ([7, 8]) *If $\mathcal{K}$ is a partition of the software system S and CCC is the set of crosscutting concerns in S, then inequality (10) holds:*

$$0 < DIV(CCC, \mathcal{K}) \leq 1. \quad (10)$$

Remark 3 *Larger values for DIV indicate better partitions with respect to CCC, meaning that DIV has to be maximized.*

In the following we introduce Lemma 3 that gives a necessary (but not sufficient) condition that must exist between the number p of elements of a partition $\mathcal{K}$ and the number q of crosscutting concerns, in order to obtain the maximum value for DIV.

Lemma 3 *If $\mathcal{K} = \{K_1, K_2, \ldots, K_p\}$ is a partition of a software system S, $CCC = \{C_1, C_2, \ldots, C_q\}$ is the set of crosscutting concerns in S, and $p \leq q$ then $DIV(CCC, \mathcal{K}) < 1$.*

Proof: We prove this lemma by contradiction. We suppose that $DIV(CCC, \mathcal{K})$ is 1.

From Definition 5, Equations (7), (8), and (9) we have that:

$$0 < div(CCC, k) \leq 1,\ \forall\, k, k \in \mathcal{K}. \quad (11)$$

From Equation (6) and inequality (11) it follows that:

$$DIV(CCC, \mathcal{K}) = 1 \Leftrightarrow div(CCC, k) = 1,\ \forall k,\ k \in \mathcal{K}. \quad (12)$$

From (8) and (9) we have that $|V_k| \in \mathbf{N}$ and $\tau(k) \in \{0, 1\}$, so

$$|V_k| + \tau(k) \in \mathbf{N}. \quad (13)$$

Based on (7) and (13) we have that $\forall k,\ k \in \mathcal{K}$, $div(CCC, k) = 1$ iff $|V_k| + \tau(k) = 1$.

Using (8) and (9) it follows that:

$$\forall\ k, k \in \mathcal{K},\ |V_k| + \tau(k) = 1 \text{ iff}$$

$$|V_k| = 1 \text{ and } \tau(k) = 0 \quad (14)$$

or

$$|V_k| = 0 \text{ and } \tau(k) = 1. \quad (15)$$

Since $DIV(CCC, \mathcal{K}) = 1$, from (12), (14), and (15) we have that: there exists a $t,\ 0 \leq t \leq p$ and two subsets $\mathcal{SK}_1$ and $\mathcal{SK}_2$ of $\mathcal{K}$, such that:

(i) $\mathcal{K} = \mathcal{SK}_1 \cup \mathcal{SK}_2$ and $\mathcal{SK}_1 \cap \mathcal{SK}_2 = \emptyset$;

(ii) $|\mathcal{SK}_1| = t$ and $|\mathcal{SK}_2| = p - t$;

(iii) $\forall\ k, k \in \mathcal{SK}_1$, $|V_k| = 0$ and $\tau(k) = 1$;

(iv) $\forall\ k, k \in \mathcal{SK}_2$, $|V_k| = 1$ and $\tau(k) = 0$.

As $NCCC \neq \emptyset$, it must exist at least one $k \in \mathcal{K}$ such that $k \cap NCCC \neq \emptyset$, which implies that $\tau(k) = 1$ (from (9)). We can deduce that:

$$|\mathcal{SK}_1| \neq 0 \text{ and } t \geq 1. \quad (16)$$

From definition of V_k and (iv) it follows that each cluster from $\mathcal{SK}_2$ contain elements from only one crosscutting concern. As the elements of a crosscutting concern can appear in two different clusters, we have that:

$$q \leq p - t. \quad (17)$$

Using (16) we have that:

$$p - t < p. \quad (18)$$

From (17) and (18) it follows that: $q \leq p - t < p$, but $p \leq q$ (from the lemma's hypothesis), which is a contradiction. It follows that our hypothesis $DIV(CCC, \mathcal{K}) = 1$ is false. Consequently, $0 \leq DIV(CCC, \mathcal{K}) < 1$, and this concludes our proof. □

Theorem 4 gives the necessary and sufficient conditions for a partition of a software system to be optimal with respect to its set of crosscutting concerns.

Theorem 4 ([7, 8]) *If $\mathcal{K}$ is a partition of the software system S and CCC is the set of crosscutting concerns in S, then $\mathcal{K}$ is an* ***optimal partition*** *iff $DISP(CCC, \mathcal{K}) = 1$ and $DIV(CCC, \mathcal{K}) = 1$.*

3 Formal Model For Partitioning Based Aspect Mining

The results obtained by a clustering based aspect mining technique $\mathcal{T}$, as the technique introduced in [12], do not contain empty clusters, but if other methods are used to divide (split) the software system, it is possible to obtain empty clusters. That is why, in this section we extend the formal model for clustering based aspect mining, considering the possibility to obtain empty clusters in a partition. For this purpose, we introduce the notion of **division** of a software system S, that lessens the conditions imposed for a partition.

Definition 6 ***Division of a system*** S***.***
The set $\mathcal{D} = \{D_1, D_2, ..., D_r\}$ is called a ***division*** *of the system $S = \{s_1, s_2, \ldots, s_n\}$ if the following conditions hold:*

(1) $1 \leq r \leq n$

(2) $D_i \subseteq S,\ \forall i, 1 \leq i \leq r$

(3) $S = \bigcup_{i=1}^{r} D_i$

(4) $D_i \cap D_j = \emptyset,\ \forall\ i, j,\ 1 \leq i, j \leq r, i \neq j.$

A division is similar to a partition, but it may contain empty elements.

It can be proved that for any division $\mathcal{D}$ of a software system S, a partition can be obtained. Lemma 5 given below indicates the way to obtain a partition of a software system from a division of it.

Lemma 5 *If $\mathcal{D}$ is a division of the software system S, then $\langle\mathcal{D}\rangle = \{d \mid d \in \mathcal{D},\ d \neq \emptyset\}$ is a partition of S.*

Proof: From the definition of a division $\mathcal{D}$ and the definition of $\langle\mathcal{D}\rangle$ we have the following:

(i) $\langle\mathcal{D}\rangle \subseteq \mathcal{D}$ implies that $|\langle\mathcal{D}\rangle| \leq |\mathcal{D}|$. It follows that $1 \leq |\langle\mathcal{D}\rangle| \leq r$, and as $r \leq n$, we have $1 \leq |\langle\mathcal{D}\rangle| \leq n$.

(ii) $\forall d, d \in \langle\mathcal{D}\rangle$ we have that $d \in \mathcal{D}$ and $d \neq \emptyset$. As $\forall d, d \in \mathcal{D}, d \subseteq S$ it follows that $\forall d, d \in \langle\mathcal{D}\rangle, d \subseteq S$ and $d \neq \emptyset$.

(iii) $\forall D_i, D_j \in \langle\mathcal{D}\rangle \Rightarrow D_i, D_j \in \mathcal{D} \overset{Definition\ 6}{\Longrightarrow} D_i \cap D_j = \emptyset$.

From (i), (ii) and (iii) it follows that the conditions (1), (2) and (4) from Definition 1 are satisfied. Now we just have to prove that $S = \bigcup_{d \in \langle\mathcal{D}\rangle} d$.

From Definition 6 we have:

$$S = \bigcup_{d \in \mathcal{D}} d = (\bigcup_{d \in \mathcal{D},\ d \neq \emptyset} d) \cup (\bigcup_{d \in \mathcal{D},\ d = \emptyset} d). \quad (19)$$

But

$$\bigcup_{d \in \mathcal{D},\ d=\emptyset} d = \emptyset \tag{20}$$

and

$$\{d \mid d \in \mathcal{D},\ d \neq \emptyset\} = \langle \mathcal{D} \rangle \tag{21}$$

From Equations (19), (20) and (21), it follows that $S = \bigcup_{d \in \langle \mathcal{D} \rangle} d$, meaning that condition (3) from Definition 1 is also satisfied and this concludes our proof. □

The following definitions classify the divisions obtained by an aspect mining technique considering the efficiency condition (Equation (1)).

Definition 7 ***Good division of a system*** S**.**
Being given a division $\mathcal{D} = \{D_1, D_2, ..., D_p\}$ of the system S, $\mathcal{D}$ is called a ***good division*** *of the system S with respect to the set $CCC = \{C_1, C_2, ..., C_q\}$ of its crosscutting concerns, iff $\langle \mathcal{D} \rangle$ is a good partition of the system S.*

Definition 8 ***Optimal division of a system*** S**.**
Being given a division $\mathcal{D} = \{D_1, D_2, ..., D_p\}$ of the system S, $\mathcal{D}$ is called an ***optimal division*** *of the system S with respect to the set $CCC = \{C_1, C_2, ..., C_q\}$ of its crosscutting concerns, iff $\langle \mathcal{D} \rangle$ is an optimal partition of the system S.*

In the following, the quality measures **DISP** and **DIV** from Subsection 2.2 are redefined for divisions. We denote by CCC the set of crosscutting concerns from the software system S.

Definition 9 ***DISPersion of crosscutting concerns in a division-*** $DISP_{DV}$**.**
The dispersion of the set CCC in a division $\mathcal{D}$ of a software system S, denoted by $DISP_{DV}(CCC, \mathcal{D})$, is defined as

$$DISP_{DV}(CCC, \mathcal{D}) = \frac{1}{|CCC|} \sum_{i=1}^{|CCC|} disp_{DV}(C_i, \mathcal{D}). \tag{22}$$

In (22) *$disp_{DV}(C_i, \mathcal{D})$ is the dispersion of a crosscutting concern C_i and is defined as:*

$$disp_{DV}(C_i, \mathcal{D}) = \frac{1}{|D_{C_i}(\mathcal{D})|}, \tag{23}$$

where

$$D_{C_i}(\mathcal{D}) = \{d \mid d \in \mathcal{D} \text{ and } d \cap C_i \neq \emptyset\}. \tag{24}$$

D_{C_i} is the set of elements from $\mathcal{D}$ that contain elements which are also in C_i.

Lemma 6 *If S is a software system, CCC is the set of crosscutting concerns from S, and $\mathcal{D}$ is a division of S, then*

$$DISP_{DV}(CCC, \mathcal{D}) = DISP(CCC, \langle \mathcal{D} \rangle).$$

Proof: In order to prove that $DISP_{DV}(CCC, \mathcal{D}) = DISP(CCC, \langle \mathcal{D} \rangle)$, we will prove that:

$$\forall C,\ C \in CCC,\ D_C(\mathcal{D}) = D_C(\langle \mathcal{D} \rangle). \tag{25}$$

From the definition of $\langle \mathcal{D} \rangle$ we have that $\langle \mathcal{D} \rangle \subseteq \mathcal{D}$. From Equations (4) and (24) it follows that

$$\forall C, C \in CCC,\ D_C(\langle \mathcal{D} \rangle) \subseteq D_C(\mathcal{D}).$$

But, from the definition of $\langle \mathcal{D} \rangle$ we also have that $\forall d, d \in \mathcal{D} \setminus \langle \mathcal{D} \rangle,\ d = \emptyset$. We deduce that $\forall d, d \in \mathcal{D} \setminus \langle \mathcal{D} \rangle,\ d \notin D_C(\mathcal{D})$ (as $d \cap C = \emptyset$), $\forall C \in CCC$. So, Equation (25) is proven.

From Equations (3), (23), and (25) we have that:

$$disp_{DV}(C, \mathcal{D}) = disp(C, \langle \mathcal{D} \rangle)\ \forall C \in CCC. \tag{26}$$

From Equations (2), (22) and (26) it follows that

$$DISP_{DV}(CCC, \mathcal{D}) = DISP(CCC, \langle \mathcal{D} \rangle),$$

and this concludes our proof. □

Definition 10 ***DIVersity of a division -*** DIV_{DV}**.**
The diversity of a division $\mathcal{D}$ with respect to the set CCC, denoted by $DIV_{DV}(CCC, \mathcal{D})$, is defined as

$$DIV_{DV}(CCC, \mathcal{D}) = \frac{1}{|\langle \mathcal{D} \rangle|} \sum_{i=1}^{|\mathcal{D}|} div_{DV}(CCC, D_i). \tag{27}$$

$div_{DV}(CCC, d)$ is the diversity of an element of the division $d \in \mathcal{D}$ and is defined as:

$$div_{DV}(CCC, d) = \begin{cases} \frac{1}{|V_d| + \tau(d)} & \text{if } d \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

where V_d and $\tau(d)$ are defined as in Definition 5.

Lemma 7 *If S is a software system, CCC is the set of crosscutting concerns from S, and $\mathcal{D}$ is a division of S, then*

$$DIV_{DV}(CCC, \mathcal{D}) = DIV(CCC, \langle \mathcal{D} \rangle).$$

Proof: In order to prove Lemma 7, based on Equations (6) and (27), we have to prove the following equality:

$$\sum_{d \in \langle \mathcal{D} \rangle} div(CCC, d) = \sum_{i=1}^{|\mathcal{D}|} div_{DV}(CCC, D_i). \tag{29}$$

We have that:

$$\sum_{i=1}^{|\mathcal{D}|} div_{DV}(CCC, D_i) = X + Y \qquad (30)$$

where:

$$X = \sum_{d \in \mathcal{D},\ d \neq \emptyset} div_{DV}(CCC, d)$$

and

$$Y = \sum_{d \in \mathcal{D},\ d = \emptyset} div_{DV}(CCC, d).$$

From (28) we deduce that $div_{DV}(CCC, d) = div(CCC, d)$ if $d \neq \emptyset$. It follows that:

$$\sum_{d \in \mathcal{D},\ d \neq \emptyset} div_{DV}(CCC, d) = \sum_{d \in \mathcal{D},\ d \neq \emptyset} div(CCC, d).$$

Consequently, we have:

$$\sum_{d \in \mathcal{D},\ d \neq \emptyset} div(CCC, d) = \sum_{d \in \langle \mathcal{D} \rangle} div(CCC, d). \qquad (31)$$

From (28) we also have that $div_{DV}(CCC, d) = 0$, $\forall d, d \in \mathcal{D},\ d = \emptyset$. It follows that:

$$\sum_{d \in \mathcal{D},\ d = \emptyset} div_{DV}(CCC, d) = 0. \qquad (32)$$

From Equations (30), (31), and (32) we have that:

$$\sum_{i=1}^{|\mathcal{D}|} div_{DV}(CCC, D_i) = \sum_{d \in \langle \mathcal{D} \rangle} div(CCC, d).$$

So, equality (29) is proven, and this concludes our proof. $\square$

From the above lemmas we can conclude that the properties (characteristics) of partitions of a software system S, are also true for divisions of S, as for any division $\mathcal{D}$ of S, we can obtain the corresponding partition $\langle \mathcal{D} \rangle$.

As the problem of aspect mining can be viewed as the problem of finding a partition or a division of a software system S, we introduce the notion of *partitioning aspect mining technique*.

Definition 11 ***Partitioning aspect mining technique.*** *Let $\mathcal{T}$ be an aspect mining technique and S 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*** *$\mathcal{K}$ or a* ***division*** *$\mathcal{D}$ of S.*

Based on Definition 11, the aspect mining techniques that will be analyzed in Section 4 are all *partitioning aspect mining techniques*.

4 Applicability

In this section we describe how the formal model introduced in Section 3 can be applied to different aspect mining techniques.

In order to emphasize the generality of the proposed formal model, we are focusing only on aspect mining techniques that do not use clustering.

4.1 Aspect Mining using Event Traces

This technique was introduced in [1] and [2] and it detects crosscutting concerns in legacy systems based on dynamic analysis. We will denote this technique by $\mathcal{T}_{Events}$.

The analysis uses program traces that are generated in different program executions as underlying data pool. These traces are then investigated for recurring execution patterns based on different constraints, such as the requirement that the patterns have to exist in different calling contexts in the program trace. The authors have defined four execution relations that may exist between two methods in a program trace: *outside-before-execution relation* (denoted by $\rightharpoonup$), *outside-after-execution relation* (denoted by $\leftharpoonup$), *inside-first-execution relation* (denoted by $\in_\top$) and *inside-last-execution relation* (denoted by $\in_\bot$) and two constraints: *uniformity* and *crosscutting*. The following notations are used in this approach ([1, 2]):

- $S^{<rel>}$ denotes the multiset of all $< rel >$-relations that exist in the program trace, where $< rel > \in \{\rightharpoonup,\ \leftharpoonup,\ \in_\top,\ \in_\bot\}$.
- $U^{<rel>}$ denotes the multiset of all $< rel >$-relations that satisfy the uniformity constraint, where $< rel > \in \{\rightharpoonup,\ \leftharpoonup,\ \in_\top,\ \in_\bot\}$.
- $R^{<rel>}$ denotes the set of all $< rel >$-relations from $U^{<rel>}$ that also satisfy the crosscutting constraint, where $< rel > \in \{\rightharpoonup,\ \leftharpoonup,\ \in_\top,\ \in_\bot\}$.

The sets $R^{<rel>}$ are searched to find crosscutting concerns ($< rel > \in \{\rightharpoonup,\ \leftharpoonup,\ \in_\top,\ \in_\bot\}$).

This technique can be formalized using the proposed formal model as follows:

1. The elements of the software system S are execution relations from the program trace:

   $$S = S^{\leftharpoonup} \cup S^{\rightharpoonup} \cup S^{\in_\top} \cup S^{\in_\bot}.$$

2. In our model, $\mathcal{T}_{Events} =$

   $$(divide_{Events}, select_{Events}, order_{Events}),$$

   where:

- $divide_{Events}(S) = \mathcal{D}$, where $\mathcal{D} = \{R^{\leftharpoonup}, R^{\rightharpoonup}, R^{\in\top}, R^{\in\bot}, RT\}$. $R^{\leftharpoonup}, R^{\rightharpoonup}, R^{\in\top}, R^{\in\bot}$ are the sets of execution relations that satisfy the *crosscutting* constraint, and $RT = S \setminus (R^{\leftharpoonup} \cup R^{\rightharpoonup} \cup R^{\in\top} \cup R^{\in\bot})$. The result of the $divide$ function is a division as some (or all) sets $R^{\leftharpoonup}, R^{\rightharpoonup}, R^{\in\top}$, $R^{\in\bot}$ may be empty.
- $select_{Events}(S, \mathcal{D}) = \{R^{\leftharpoonup}, R^{\rightharpoonup}, R^{\in\top}, R^{\in\bot}\}$
- $order_{Events}(\mathcal{D}) = 1_{\mathcal{D}}$, where by $1_{\mathcal{D}}$ we denote the identity function. No particular order is specify for the analysis of the results.

3. The variation limits for the quality measures are as follows:

$DISP_{DV}$
: $\frac{1}{5} \leq DISP_{DV}(CCC, \mathcal{D}) \leq 1$, because $D_C(\mathcal{D}) \in \{1, 2, 3, 4, 5\}, \ \forall C \in CCC$.

DIV_{DV}
: The interval remains the same, with one remark. If $|CCC| \geq 5$, then $DIV_{DV}(CCC, \mathcal{D}) < 1$, based on Lemma 3, and an optimal division cannot be obtained.

4.2 Aspect Mining using Fan-in Analysis

Fan-In Analysis ([6]) is another technique that can be formalized using the formal model proposed in Section 3. We will denote this technique by $\mathcal{T}_{FanIn}$.

The considered technique is based on determining methods that have a high *fan-in* value, in order to identify candidate aspects in a number of open-source Java systems. The *fan-in* value of a method m is defined as the number of distinct method bodies that invoke m ([4]).

$\mathcal{T}_{FanIn}$ can be formalized using the proposed formal model as follows:

1. The elements of the software system S are methods from it:

$$S = \{m_1, m_2, \dots, m_n\},$$

where m_i is a method from the system.

2. In our model, $\mathcal{T}_{FanIn} =$

$$(divide_{FanIn}, select_{FanIn}, order_{FanIn}),$$

where:

- $divide_{FanIn}(S) = \mathcal{D}$, where $\mathcal{D} = \{D_1, D_2\}$, $D_1 = \{m \in S \mid fan\text{--}in(m) \geq threshold\}$, $D_2 = \{m \in S | fan-in(m) < threshold\}$. The result is a division, as D_1 or D_2 may be empty if the value of the $threshold$ is not properly chosen. The developers of the technique consider 10% of $|S|$ as an appropriate value for the $threshold$.
- $select_{FanIn}(S, \mathcal{D}) = \{D_1\}$.
- $order_{FanIn}(\mathcal{D}) = \{D_1, D_2\}$, and the elements of D_1 are analyzed in descending order by their *fan-in* value.

3. The variation limits for the quality measures are as follows:

$DISP_{DV}$
: $\frac{1}{2} \leq DISP(CCC, \mathcal{D}) \leq 1$ as $|D_C(\mathcal{D})| \in \{1, 2\}, \ \forall C \in CCC$.

DIV_{DV}
: $0 < DIV(CCC, \mathcal{D}) \leq \frac{1+|CCC|}{2 \cdot |CCC|}$. Based on Lemma 3, if $|CCC| \geq 2$, then $DIV(CCC, \mathcal{D}) < 1$ and an optimal division cannot be obtained.

4.3 A Graph Based Approach in Aspect Mining

In ([11]) a graph based approach in Aspect Mining is introduced. Let us denote this technique by $\mathcal{T}_{Graph}$. $\mathcal{T}_{Graph}$ can also be formalized using the proposed model.

The basic idea of this technique is to determine methods that are similar. The similarity between two methods is computed using different distance metrics and the vector space model, where each method m is characterized by a l-dimensional vector with real values: $m = (a_1, a_2, \dots, a_l)$. The approach is to construct a graph (denoted by $\mathcal{MG}_S$) 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.

$\mathcal{T}_{Graph}$ can be formalized using the proposed formal model as follows:

1. The elements of the software system S are methods from it:

$$S = \{m_1, m_2, \dots, m_n\},$$

where m_i is a method from the system.

2. In our model, $\mathcal{T}_{Graph} =$

$$(divide_{Graph}, select_{Graph}, order_{Graph}),$$

where:

- $divide_{Graph}(S) = \mathcal{K}$, where $\mathcal{K} = \{K_1, K_2, \ldots, K_r\}$, $r \leq n$ and K_i is a connex component of the graph $\mathcal{MG}_S$, $\forall i$, $1 \leq i \leq r$. The result of this function is a partition because a connex component of a graph cannot be empty.
- $select_{Graph}(S, \mathcal{K}) = \{K_{i_1}, K_{i_2}, \ldots, K_{i_z}\}$. $z \leq r$, and the distance between K_{i_j} and 0_l is greater than a given threshold, $\forall j, 1 \leq j \leq z$. Here 0_l represents the l-dimensional vector having all the components 0.
- $order_{Graph}(\mathcal{K}) = \{K_{i_1}, K_{i_2}, \ldots, K_{i_r}\}$. The obtained clusters are analyzed in descending order by their distance to the 0_l point.

3. The limits for the quality measures remain unmodified (as given by Lemma 1 and Lemma 2), as the number of clusters obtained by the technique depends on the graph $\mathcal{MG}_S$.

5 Conclusions and Further Work

In this paper we have presented a new formal model for partitioning aspect mining. Two new quality measures for evaluating the results of partitioning aspect mining techniques were defined. The proposed quality measures are used to evaluate the optimality degree of the results obtained by a partitioning aspect mining technique.

In order to show its generality, the proposed formal model was applied to three different aspect mining techniques ([2], [6], [11]).

Further work can be done in the following directions:

- To generalize the formal model to consider the case when two crosscutting concerns have common elements.
- To identify other possible measures for evaluating partitioning approaches in aspect mining.
- To study the applicability of our model to other aspect mining techniques ([9], [13]).

References

[1] Breu, S. Aspect Mining Using Event Traces. Master's thesis, University of Passau, Germany, March 2004.

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

[3] Bruntink, M., van Deursen, A., van Engelen, R., and Tourwe, T. On the Use of Clone Detection for Identifying Crosscutting Concern Code. *IEEE Transactions on Software Engineering*, 31(10):804–818, 2005.

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

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

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

[7] Moldovan, G. S. and Serban, G. A Formal Model for Clustering Based Aspect Mining. In *Proceedings of 8th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering (MMACTEE '06)*, pages 70–75, 2006.

[8] Moldovan, G. S. and Serban, G. Clustering Based Aspect Mining Formalized. *WSEAS Transactions on Computers*, 6(2):199–206, February 2007.

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

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

[11] Serban, G. and Moldovan, G. S. A Graph Algorithm for Identification of Crosscutting Concerns. *Studia Universitatis Babes-Bolyai, Informatica*, LI(2):3–10, 2006.

[12] Serban, G. and Moldovan, G. S. A new k-means based clustering algorithm in aspect mining. In *Proceedings of 8th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06), Timisoara, Romania*, pages 69–74. IEEE Computer Society Press, 2006.

[13] Tonella, P. and 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)*, pages 112–121, November 2004.