

# Aspect Mining from a Modeling Perspective

Jing Zhang\*, Jeff Gray, Yuehua Lin, and Robert Tairas

Department of Computer and Information Sciences,  
University of Alabama at Birmingham,  
1300 University Blvd., 126 Campbell Hall, Birmingham, AL 35294, USA  
E-mail: {zhangj, gray, liny, tairasr}@cis.uab.edu

\*Corresponding author

**Abstract:** Aspect mining aims at identifying, analyzing, and refactoring crosscutting concerns throughout a legacy system for the purpose of improving software modularization. Current research on aspect mining prevails at the implementation level as applied to source code. However, an aspect-oriented approach can be beneficial at various levels of abstraction and at different stages of the software lifecycle. This paper presents our investigation into raising the benefits of aspect mining to a higher level of abstraction through application of aspect mining algorithms to domain-specific models. A key contribution of the approach is a capability to identify crosscutting concerns early in development, which assists in modularizing a design through aspects before proceeding to the implementation level. Furthermore, our experience has led us to believe that aspects are easier to identify at the modeling level because much of the accidental complexities of implementation concerns are not present in the corresponding modeling abstractions.

**Keywords:** Aspect-Oriented Software Development; Aspect Mining; Model-Driven Engineering; Domain-Specific Modeling.

**Biographical notes:** Jing Zhang is a PhD candidate in the Department of Computer and Information Sciences at the University of Alabama at Birmingham (UAB) and a member of the Software Composition and Modeling (SoftCom) Laboratory. Her research interests include techniques that combine model transformation and program transformation to assist in evolving large software systems. She received an MS in computer science from UAB.

Jeff Gray is an Assistant Professor in the Department of Computer and Information Sciences at UAB, where he co-directs research in the SoftCom lab. His research interests include model-driven engineering, aspect-orientation, and generative programming. He received a PhD in computer science from Vanderbilt University.

Yuehua Lin is a PhD candidate in the Department of Computer and Information Sciences at UAB and a member of the SoftCom lab. Her technical interests are focused on model transformation and supporting tools. She received an MS in computer science from Auburn University.

Robert Tairas is a PhD student in the Department of Computer and Information Sciences at UAB and a member of the SoftCom lab. His research interests include clone detection and aspect mining. He received an MS in computer science from UAB.

## 1 INTRODUCTION

In poorly modularized software, numerous concerns are often tangled within the boundary of a single module, which leads to cohesion problems. In other cases, a single concern may be scattered across several different modules, which introduces strong coupling among modules. The occurrence of tangling and scattering offers challenging maintenance problems when a software system needs to evolve to address changing requirements. Throughout programming language history, new language constructs have been offered to address the issues of scattering and tangling. For example, the hierarchical decomposition provided by object-orientation assists in localizing common behaviour in superclasses, such that the same behaviour is not repeated in multiple places in subclasses.

Although scattering and tangling have been an important focus of software engineering and language design for decades, a new type of concern has emerged, which is crosscutting in nature. *Crosscutting* represents a relationship property between two concerns such that traditional hierarchical composition is not capable of modularizing each concern in a separate unit. Aspect-Oriented Software Development (AOSD) (Filman et al., 2004) offers a powerful technology for supporting the separation of such concerns, whereby the crosscutting is explicitly specified as an aspect.

For legacy software to benefit from AOSD, it is necessary to analyze the existing implementation to discover the crosscutting concerns and refactor them into aspects. *Aspect mining* refers to the identification and analyses of non-localized crosscutting concerns throughout an existing legacy software system (Bruntink et al., 2005). The ultimate goal of aspect mining is to support aspect-oriented refactoring to improve software comprehensibility, reusability and maintainability.

### 1.1 Key challenges of aspect mining

The challenges of aspect mining are focused along three separate phases:

- **Aspect Identification:** This phase is concerned with an analysis task that leads to identification of a suggested set of candidate aspects. This phase may require user interaction to provide initial seed information, or to assist in sifting through false positive noise (i.e., suggested aspects that are not really representative of a crosscutting concern).
- **Aspect Extraction:** After a set of candidate aspects has been identified, the crosscutting concern must be extracted from the existing representation (i.e., all of the locations in the legacy software where the aspect appears must be removed).
- **Aspect Refactoring:** After extracting the crosscutting concerns from the base representation, an equivalent aspect must be codified in an aspect language in order to preserve the initial functionality. The result is

improved modularization (as captured in the new aspect), with no change in functional behaviour.

With respect to these three phases of aspect mining, there appear to be no reports in the research literature on individual tools that perform all three of the above challenges successfully. Most aspect mining research tools are focused on one phase of aspect mining, with the majority of work (as summarized in Section 6) focused on the aspect identification phase. Regarding extraction and refactoring, a technique that uses program slicing to perform these two phases has been presented by Ettinger and Verbaere (Ettinger and Verbaere, 2004).

### 1.2 Aspect mining earlier in the lifecycle

Much of the current research on aspect mining focuses solely on the implementation as applied to source code. However, an aspect-oriented approach can be beneficial at various levels of abstraction and at different stages of the software lifecycle. For instance, aspect-oriented analysis and design (Clarke and Baniassad, 2005) is a new development approach that unites AOSD with requirements and design models. Likewise, the concepts of feature-oriented programming are also being applied to design models (Batory, 2006). Research in Aspect-Oriented Modeling (AOM Workshop) has the potential to help define common characteristics (which are encapsulated within aspects) from a perspective that is at a more abstract level. For existing models to benefit from AOSD, it is indispensable to perform reengineering techniques, such as aspect mining, at many different stages throughout the development lifecycle.

This paper presents our investigation into raising the benefits of aspect mining to a higher level of abstraction through application of aspect mining algorithms to domain-specific models. Specifically, the paper describes our approach to the aspect identification problem as applied to models, rather than source code. A key contribution of the approach is a capability to identify crosscutting concerns early in development, which assists in modularizing a design through aspects before proceeding to implementation. Furthermore, our experience in performing both manual and automated aspect mining suggests that aspects are easier to identify at the modeling level because the accidental complexities of implementation concerns are absent in the corresponding modeling abstractions.

The remainder of this paper is structured as follows. Section 2 introduces the basic idea of domain-specific modeling and describes how aspects emerge in such models. This section also initiates the motivation of applying an aspect identification technique at the modeling level. Sections 3 and 4 propose two different approaches for aspect identification, i.e., pattern matching and clone detection. Section 5 offers a case study using clone detection to identify crosscutting concerns in a modeling language for embedded systems. The last two sections discuss the related work, conclusions and future work.

## 2 ASPECTS IN DOMAIN-SPECIFIC MODELING

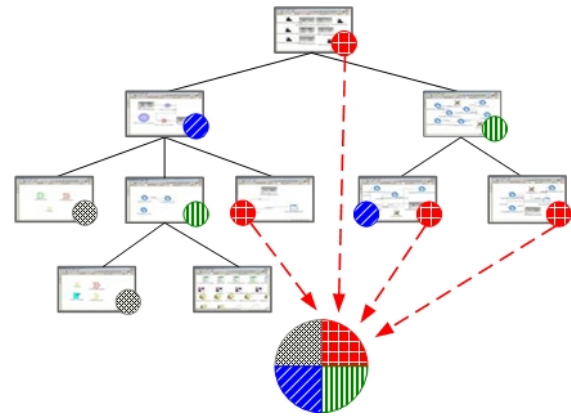
Model-Driven Engineering (MDE) (Schmidt, 2006) is an emerging paradigm supporting the development of computer-based systems. The principles of MDE have been applied successfully in many domains, but have exhibited specific contributions in the domain of embedded control software, such as avionics and automotive control systems (Lédeczi et al., 2003). An important characteristic of MDE is the use of Domain-Specific Modeling (DSM) techniques by which software products are derived from models that directly relate to the problem domain (Gray et al., 2006b). Meta-configurable domain-specific modeling environments provide support for customization of modeling tools that enable domain experts to construct models in notations that are familiar to them. Such tools also offer the capability to generate, or synthesize various artifacts from models. The ability to describe properties of a system at a higher level of abstraction, and in a technology-independent notation, can protect intellectual assets from technology obsolescence.

The Generic Modeling Environment (GME) (Karsai et al., 2004) is a metamodeling environment that can be configured and adapted from meta-level specifications that describe a domain. The GME supports a set of generic modeling concepts to represent entities, relationships and attributes. An *atom* is the most basic type of entity that cannot have any internal structures. A *model* is another type of entity that can contain any other modeling types. A *connection* represents the relationship between two entities. *Attributes* are used to record state information and are bound to atoms, models, and connections.

In our previous work (Gray et al., 2001), we made the observation that crosscutting concerns emerge in domain-specific models, as shown in Figure 1. It is often the case that the metamodel forces a specific type of decomposition, such that the same concern is repeatedly applied in many places, usually with slight variations at different nodes in the model. Abstractly, Figure 1 shows four different concerns that are spread about a model hierarchy. Examples of crosscutting modeling aspects include constraints (Gray et al., 2001), concurrency and state management (Gray et al., 2004), and pre/post conditions (Gray et al., 2006a).

The crosscutting concerns scattered across the models lead to several impediments to system comprehension and maintenance:

- Discovering or understanding a specific concern representation that is spread over the model hierarchy is difficult, because the concern is not localized in one single module. This limits the ability to reason analytically about such a concern.
- Changing a concern requirement is also difficult and time-consuming, because the model engineers must go into each relevant model and modify the specific elements one by one. This requires much typing and mouse clicking, which affects productivity and correctness (Gray et al., 2006a).



**Figure 1** Crosscutting concerns throughout a model hierarchy

Manual inspection of models to discover potential aspects is a laborious task. Performing automated aspect mining to existing non-aspectized models can offer insight into the identification of emergent aspects. Aspect mining from a modeling perspective allows the designer to locate the places in a model that must be changed when a particular concern needs to be modified. The identification of aspects earlier in the software lifecycle allows crosscutting concerns to be managed and understood before details of the implementation are planned.

This paper describes the first investigation (with corresponding tool support) into aspect identification at the modeling level. The next two sections discuss different approaches that we have investigated to realize aspect identification on models.

## 3 PATTERN MATCHING FOR ASPECT MODEL MINING

The pattern matching process is conducted by a human designer who suspects the existence of aspects in a model. The designer has to comprehend the domain information contained in a model and provide a “seed” pattern to indicate properties of potential aspects. Such a seed serves as the starting point for discovering all matched concerns. There are two different representations of the seed for aspect mining of models through pattern matching. One representation is based on textual expressions, and another kind of seed is described by graphical models.

### 3.1 Textual-based pattern matching

In our past work (Sudarsan and Gray, 2006), we used XPath expressions as the pattern description to search for properties within domain-specific models. The underlying model search engine parses the XPath expression and traverses the internal representation of a model to compare the user-defined pattern with every model entity. This modeling search technique can be adapted to perform aspect mining. Although this technique is easily implemented and provides lightweight search power for

simple textual pattern expressions, it lacks the capability to specify complex patterns intuitively and efficiently (e.g., a collection of sub-models that involve heterogeneous model elements and sinuous relationships among them).

### 3.2 Graphical-based pattern matching

Another approach to identify crosscutting modeling concerns is to represent a pattern in a graphical notation. As an example of this type of pattern matching, GReAT (Agrawal et al., 2003) defines a graph pattern specification language to express complicated patterns with a fixed and variable cardinality. This graph notation complements the shortcomings of textual pattern expression and supports complex and dynamic pattern matching. We did not explore graphical-based pattern matching in our investigation.

## 4 CLONE DETECTION FOR ASPECT MODEL MINING

Pattern matching techniques assist users in efficiently locating predefined crosscutting concerns. However, users of pattern matching are required to have a considerable amount of knowledge about the domain and overall model structure (e.g., users must input a particular format of the seed so that the aspect mining process can be automated partially). Moreover, pattern matching cannot explore unknown classes of crosscutting concerns (i.e., those for which no seed is known) and will often result in missing some desirable aspects. In order to overcome the deficiencies of pattern matching, we developed a clone detection technique for aspect mining applied to models.

Various clone detection techniques (as summarized in Section 6) have been investigated to detect duplicated source code. The intention of applying clone detection for aspect mining is to reveal the unknown crosscutting concerns through full automation of the aspect mining process. In terms of modeling, clone detection identifies the similar (clone) model fragments throughout the model hierarchy. The similarity of elements of sub-models is determined based on one of the three levels of similarity among metamodeling concepts.

In the context of metamodeling, an atomic modeling element (e.g., an atom in GME) is defined by a combination of its type, name, and set of attributes. Correspondingly, a model consists of a set of elements, including atoms, sub-models or connections. Three levels of similarity are defined based on the type, name, and attribute of the model elements (see Table 1):

- Level 1 indicates the most liberal policy (i.e., two atoms are considered clones as long as they have the same type; two models are clones if they own the same type and all of their elements are correspondingly Level 1 clones).
- Level 2 represents a moderate clone detection philosophy, which is based on type and name similarity (e.g., two connections are considered

clones if their source and targets are Level 2 clones, in addition to each connection having the same type and name).

- Level 3 defines the most stringent rule (i.e., two models are considered clones only when they hold the same type, name, and attribute set; furthermore, all of their elements should be correspondingly recognized as Level 3 clones.)

**Table 1** Three levels of similarity

	Atom	Model	Connection
Level 1	• Type	• Type • Elements	• Type • Source • Target
Level 2	• Type • Name	• Type • Name • Elements	• Type • Name • Source • Target
Level 3	• Type • Name • Attributes	• Type • Name • Attributes • Elements	• Type • Name • Attributes • Source • Target

Based on the above levels of similarity, the four steps of the clone detection algorithm for models are presented below.

### Step 1. Metamodel preprocessing

We perform an initial step of evaluating the metamodel in an effort to reduce the number of model instance comparisons that will be needed in the latter steps. This involves the partitioning of the metamodel entities into different groups that need to be compared. Each group includes a set of the type pairs, such as:

$$\{\text{Type-model}\} : \{\text{Type-element}\}$$

where  $\{\text{Type-model}\}$  is a collection of types whose model instances comprise some common elements, and  $\{\text{Type-element}\}$  is the collection of model elements that  $\{\text{Type-model}\}$  share. Because  $\{\text{Type-element}\}$  is contained by more than one model, it has the potential to become one of the selected crosscutting concerns.

In Figure 2, the type “ModelA” and “ModelB” share the element “AtomAB”. “ModelB” and “ModelC” both contain “AtomBC”. In this case, the partition of the illustrated fragment of the metamodel would be:

$$\begin{aligned} \{\text{ModelA}, \text{ModelB}\} &: \{\text{AtomAB}\} \\ \{\text{ModelB}, \text{ModelC}\} &: \{\text{AtomBC}\} \end{aligned}$$

The preprocessing of the metamodel partition will facilitate the desired steps of the algorithm, because only those models that have the same type or fall into the same

group will be compared. Furthermore, only the shared elements of the two models should be compared. For example, imagine that there is one instance of “ModelA” and one instance of “ModelB”. In such a case, we only need to consider whether their shared atoms (instances of “AtomAB”) are clones. Any other irrelevant elements will not be considered

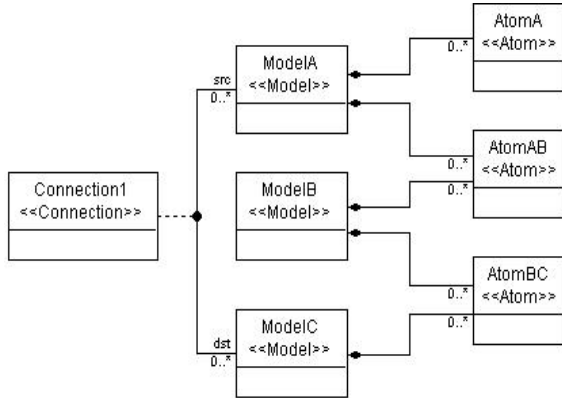


Figure 2 A metamodel fragment

### Step 2. Model fragments comparison

The second step of clone detection in models determines if the elements of a sub-model pair are clones by comparison. From the root of the model hierarchy, each sub-model is compared with the other sub-models that either have the same type or fall into the same group in step 1. As an example, suppose a comparison is to be made between sub-model instance X and sub-model instance Y:

- If X and Y are of the same type, every element inside should be compared correspondingly. The comparison is based on the choice of the level of similarity as defined in Table 1. Each time, the atoms are compared first, then the models, followed by the connections.
- If X and Y are in the same group, only their shared elements need to be compared.
- If X and Y do not have the same type, and do not fall into the same group, it means that they cannot have an intersection; thus, further comparison is not necessary.

### Step 3. Maximally similar model fragments grouping

For all of the clone elements that sub-model instance X and Y share, we group them together as a common property named  $P$ .  $P$  is considered as the maximally similar model fragments of X and Y. If  $P$  is not null, the next task is to find out whether  $P$  is already stored in the list of maximally similar fragments. An efficient way to search for commonalities on a list is to construct a hash function  $h(P)$ , which computes the number of a bucket (hash value)

based on  $P$  (Baxter et al., 1998). The hash function will always return the same bucket number given the same  $P$ . If  $P$  is not in the bucket  $h(P)$ , then X, Y, and  $P$  will be added to this bucket. If  $P$  is already in such a bucket, only X or Y will be added into the collection of the sub-models that share the same property  $P$ .

### Step 4. Aspect filtering

The maximally similar model fragments generated from the above steps (i.e., the initial result of the clones) may contain too much noise and need to be refined further (i.e., many false positives could be suggested, which can be removed on further analysis). For instance, based on our experimentation we found that if one model entity in a maximally similar model fragment group has a connection (in or out) that does not fall into the same group, then this model entity is seldom considered as an aspect and can be filtered out.

## 5 ASPECT MINING IN EMBEDDED SYSTEM MODELS

This section presents a case study that applies clone detection for aspect mining on the Embedded Systems Modeling Language (ESML) (Neema et al., 2005), which is a domain-specific graphical language for modeling real-time mission computing embedded avionics applications. The ESML has been defined within the GME and used on several DARPA funded research projects to provide the following modeling categories that allow representation of an embedded system: a) Components, b) Component Interactions, and c) Component Configurations. The primary use of the ESML is to model Boeing’s Bold Stroke, which is a product-line architecture for a variety of military aircraft written in several million lines of C++ (Sharp, 2000). There are over 20 representative ESML models for all of the Bold Stroke usage scenarios that exist. For each specific scenario within Bold Stroke, the components and their interactions are specified as ESML models.

In our previous work (Gray et al., 2004), we manually performed aspect mining on ESML models based on our own domain experience. The manual approach was a tedious process that identified crosscutting concerns such as concurrency and state management. We manually extracted these concerns from the ESML one by one in order to demonstrate aspect weaving at the modeling level, which led to the concept of model-driven program transformation. Much time was spent in understanding the ESML model ontology to support the manual process of searching the model for crosscutting concerns. In this section, we show how an automated approach to aspect identification assists in discovering some of the aspects that were previously identified manually.

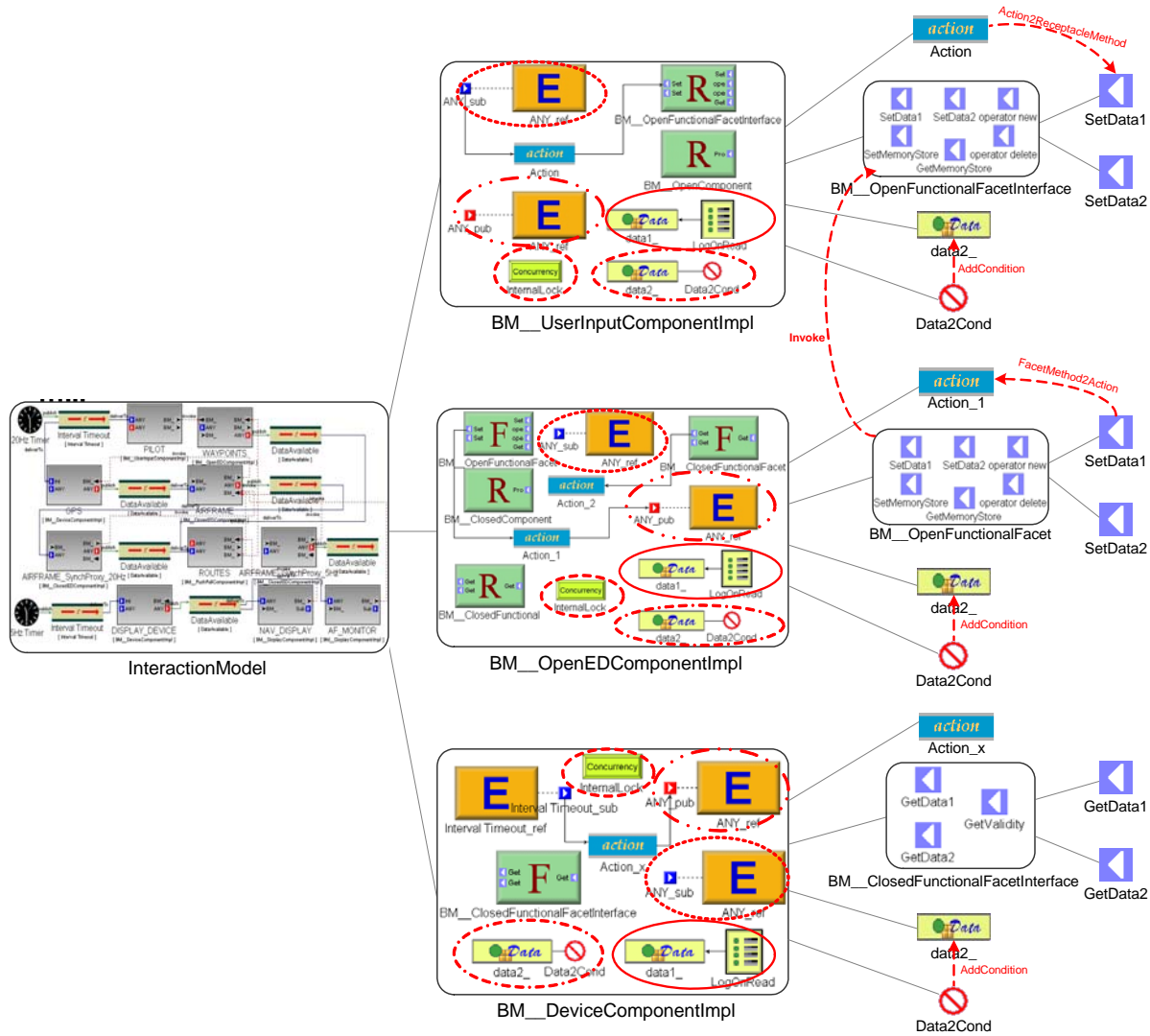


Figure 3 Sample crosscutting concerns in an ESML model

In general, an ESML model has a tree-like hierarchical structure. Figure 3 partially illustrates the internal representation of an ESML model named “InteractionModel”. The model on the first layer is the root of “InteractionModel”, which specifies a particular scenario that involves certain configurations of various sub-models. These sub-models belong to the second layer. In this figure, only three component sub-models are depicted on the second layer (e.g., “BM\_UserInputComponentImpl”, “BM\_OpenEDComponentImpl”, and “BM\_DeviceComponentImpl”). Several models and atoms representing the containment of the second layer models are depicted separately on the third layer (e.g., “BM\_OpenFunctionalFacetInterface” represents an interface for the component “BM\_UserInputComponentImpl”). The fourth layer is the last layer shown in Figure 3 (e.g., the “SetData1” atom denotes a method object that is contained by the corresponding

component interface model “BM\_OpenFunctionalFacetInterface” and “BM\_OpenFunctionalFacet”). A solid line between any two layers represents containment, and a dotted line with an arrow represents connections that may occur on the same layer or across layers.

In the case where users have no knowledge of the system (or, they have some knowledge, but not enough to express textual or graphical patterns), the clone detection technique for aspect mining may be applied to suggest possible aspects within an ESML model. The level of similarity is set to Level 2 (i.e., only compare the type and the name, without considering the attributes) for this particular case study. After applying the algorithm, the maximal similar model fragments of “BM\_OpenEDComponentImpl”, “BM\_UserInputComponentImpl”, and “BM\_DeviceComponentImpl” are:

```

{data2_, Data2Cond, AddCondition}
{data1_, LogOnRead, AddLog}
{InternalLock}
{ANY_sub, ANY_ref, EventTyping}
{ANY_pub, ANY_ref, EventTyping}

```

These concerns are circled with different patterns of lines in Figure 3, representing 5 different concerns. The last two groups both contain model entities that carry connections out of the group (e.g., “ANY\_sub” in “BM\_UserInputComponentImpl” and “ANY\_pub” in “BM\_OpenEDComponentImpl”). Therefore, these two elements (as well as their relationships in the group) should be filtered out. Thus, the algorithm identifies the resulting aspect candidates for the three component models as:

```

{data2_, Data2Cond, AddCondition}
{data1_, LogOnRead, AddLog}
{InternalLock}

```

These three concerns correspond to the same aspects (i.e., pre/post conditions, state management and concurrency) that were manually identified in our previous research on aspect weaving and model-driven program transformation (Gray et al., 2004).

As the additional concerns that were identified automatically by our algorithm, consider the interface models “BM\_OpenFunctionalFacetInterface” and “BM\_OpenFunctionalFacet”, whose maximal similar model fragments are:

```

{SetData1}
{SetData2}
{operator new}
{operator delete}
{SetMemoryStore}
{GetMemoryStore}

```

In this example, “SetData1” will be removed in the filtering process because it has connections coming into or going out from the group. Consequently, the rest of the five atoms indicate the clone methods in the interface models and can be regarded as the potential aspect candidates (as a matter of fact, these five atoms appear in 7 different interface models).

---

## 6 RELATED WORK

---

The topics of clone detection and aspect mining have received considerable attention in the research literature. In fact, there are workshops that are dedicated to discussing these research issues (TEAM, 2006; CLONES, 2003). However, the existing research literature has focused on clone detection and aspect mining at the source code level. To our knowledge, no other research has been presented that focuses on the implications of aspect mining from a modeling perspective. Our approach can be distinguished

from all of the related work summarized below by the simple observation that we have applied clone detection to search for aspects at the model level. The primary benefit our approach offers over the existing techniques is that modularization of a design through aspects is done even before proceeding to the implementation level.

### 6.1 Related work in clone detection

Various clone detection techniques have been developed and implemented. Baker (Baker, 1995) applies a token-based analysis to locate the duplication in large software systems. CCFinder (Kamiya et al., 2002) is a tool that also uses a token-based representation of source code to find clones. Mayrand et al. (Mayrand et al., 1996) use metrics that are calculated from the source fragment to find clones. Similarity analyses based on metrics and dynamic programming are used by Kontogiannis et al. (Kontogiannis et al., 1996) to search for clones.

Baxter et al. (Baxter et al., 1998) use the abstract syntax tree (AST) representation of a source program to find clones through the discovery of similar sub-trees. Our approach is similar to Baxter’s technique for determining similarity by identifying shared and different elements. However, since these two approaches are working at different levels of abstraction, they differ on what are compared. Baxter’s approach determines the similarity of sub-trees based on the number of shared and different nodes of the sub-trees. Our approach determines the similarity of elements of sub-models based on one of the three levels of similarity described in Table 1. CloneDR™ is a commercially available tool based on this approach (CloneDR, 2006).

As it relates to aspect mining, three clone detection tools are evaluated by Bruntink et al. (Bruntink et al., 2005) to determine how well suited they are in detecting predetermined crosscutting concerns in a program.

### 6.2 Related work in aspect mining

Several existing aspect mining tools have been described in the literature, including a comparison of three approaches (Ceccato et al., 2005). The current state of aspect mining is represented by the collection of tools described below. All of these tools are focused on source code analysis.

The Aspect Browser (Griswold et al., 1999) enables users to enter regular expressions as patterns to identify aspects. An early contribution of Aspect Browser was an aspect visualizer that graphically conveyed a visual overview of the crosscutting effect of a specific aspect. The Aspect Mining Tool (AMT) (Hannemann and Kiczales, 2001) augments the Aspect Browser with type-based mining.

In the Prism tool (Zhang and Jacobsen, 2004), users define a fingerprint that captures a certain property of a crosscutting concern in code. The Prism advisor autonomously computes the crosscutting property of the mining target and returns all of the matches, which are called footprints.

FEAT (Robillard and Murphy, 2002) introduces the concept of a concern graph that localizes an abstracted representation of program elements contributing to the implementation of the concern. FEAT enables users to perform maintenance tasks that involve non-localized changes. Users initiate the search process by providing a seed, which is expressed through a text file using a declarative language to describe a concern. FEAT generates the concern graph automatically according to the declared concern. Users can visit the source file corresponding to each class in the concern graph.

Ophir (Shepherd et al., 2004) is a fully automatic mining and refactoring tool based on the combination of a program dependence graph (PDG) and abstract syntax tree (AST). Ophir's aspect identification algorithm starts only at specific points of each method in order to speed up the processing time. However, this approach may overlook some potential aspects.

ER-Miner (Sampaio et al., 2005) provides automated support for identifying crosscutting concerns within requirements documents by using natural language processing techniques. Although it is intended to be applied at the requirements level, our approach is performed on the design of domain-specific models.

Breu and Zimmermann (Breu and Zimmermann, 2006) use version history to mine aspect candidates. Their approach yields a high precision for large projects with a long history but suffers from the much fewer available data for small projects.

Aspect Browser, AMT, Prism, and FEAT all require user interaction. Users must understand the application domain and provide the pattern seed from their knowledge of the code. This limitation is in addition to the fact that these tools only look for source code level aspects. A further shortcoming is shared by our own work in that the first phase of aspect identification is the primary focus, with the challenges of extraction and refactoring given less attention.

---

## 7 CONCLUSION AND FUTURE WORK

---

From our experience, it is advantageous to perform reengineering techniques, such as aspect mining, at different stages throughout the software development lifecycle and on software artifacts other than source code. This article presented our initial investigation into aspect identification on domain-specific models.

We investigated two approaches to aspect identification - pattern matching is useful for identifying the location of pre-known aspects, and clone detection assists in identifying unknown aspect candidates. The pattern matching technique is useful only when the users are able to offer a concern pattern (i.e., the "seed"), but the clone detection technique is more powerful because it can suggest multiple unknown aspects with little human interaction.

There are several areas that need additional investigation to further the maturity of model-driven aspect mining:

**Noise Filtering:** The result of the clone detection is usually adulterated with too much undesired noise. Currently, we only use one filter layer that is based on model connections. We are considering other metrics that will be integrated into the filtering analysis.

**Visualization of Modeling Aspects:** An aspect mining tool enables identification of the potential aspects and often provides the capability to visualize the various locations affected by an aspect. Traditional aspect mining techniques work on the source code level, thus their corresponding visualization tools are based on a graphical notation that is particular for line-oriented software statistics (Griswold et al., 1999). Because a model is a containment hierarchy of entities, it is necessary to develop a specific means to visualize the crosscutting aspects over different levels of models. Our future visualization tool will use a tree structure to display the model hierarchy natively with potential aspects highlighted across the whole structure. Users will have the option to expand or collapse any level of a specific model.

**Aspect Extraction and Refactoring:** With respect to general model refactoring, we have already implemented a model refactoring browser in GME by means of a model transformation engine (Zhang et al., 2005). The research on aspect-oriented refactoring is still under investigation, which aims to extract the mined crosscutting concerns into the separately described aspects. For instance, these aspects can be represented by aspect-oriented model transformation rules written in the Embedded Constraint Language (ECL) (Gray et al., 2006a).

---

## ACKNOWLEDGEMENTS

---

This project was previously supported by the DARPA Program Composition for Embedded Systems (PCES) program and is currently supported by the National Science Foundation under CSR-SMA-0509342.

---

## REFERENCES

---

- Agrawal, A., Karsai, G., and Lédeczi, A. (2003), 'An End-to-End Domain-Driven Software Development Framework', *Domain-Driven Development track, 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Anaheim, CA, October, pp. 8-15.
- AOM Workshop, *International Workshop on Aspect-Oriented Modeling*, <http://aspect-modeling.org>.
- Baker, B. S. (1995), 'On Finding Duplication and Near-Duplication in Large Software Systems', *Second Working Conference on Reverse Engineering*, Toronto, Ontario, July, pp. 86-95.
- Batory, D. (2006), 'Multi-Level Models in Model-Driven Development, Product-Lines, and Metaprogramming', *IBM Systems Journal*, Vol. 45, No. 3, pp. 527-540.

- Baxter, I. D., Yahin, A., Moura, L., Anna, M. S., and Bier, L. (1998), 'Clone Detection Using Abstract Syntax Trees', in *Proceedings of IEEE International Conference of Software Maintenance (ICSM)*, Bethesda, MD, November, pp. 368-377.
- Breu, S., and Zimmermann, T. (2006), 'Mining Aspects from Version History', in *Proceedings of 21<sup>st</sup> IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Tokyo, Japan, September.
- Bruntink, M., van Deursen, A., Engelen, R. v., and Tourwé, T. (2005), 'On the Use of Clone Detection for Identifying Crosscutting Concern Code', *IEEE Transactions on Software Engineering*, Vol. 31, No. 10, October, pp. 804-818.
- Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., and Tourwé, T. (2005), 'A Qualitative Comparison of Three Aspect Mining Techniques', in *Proceedings of the 13th International Workshop on Program Comprehension*, a workshop held at the *International Conference on Software Engineering (ICSE)*, St. Louis, MO, May, pp. 13-22.
- Clarke, S. and Baniassad, E. (2005), *Aspect-Oriented Analysis and Design: The Theme Approach*, Addison Wesley.
- CloneDR (2006), Clone Doctor: Software Clone Detection and Removal, Semantic Designs, Inc., <http://www.semdesigns.com/Products/Clone/>
- CLONES (2003), *International Workshop on Detection of Software Clones*, Victoria, BC, November. <http://www.iste.uni-stuttgart.de/ps/clones/>
- Ettinger, R. and Verbaere, M. (2004), 'Untangling: A Slice Extraction Refactoring', in *Proceedings of International Conference on Aspect-Oriented Software Development (AOSD)*, Lancaster, UK, March, pp. 93-101.
- Filman, R. E., Elrad, T., Clarke, S., and Aksit, M. (2004), *Aspect-Oriented Software Development*, Addison-Wesley.
- Gray, J., Bapty, T., Neema, S., and Tuck, J. (2001), 'Handling Crosscutting Constraints in Domain-Specific Modeling', *Communications of the ACM*, Vol. 44, No. 10, October, pp. 87-93.
- Gray, J., Zhang, J., Lin, Y., Wu, H., Roychoudhury, S., Sudarsan, R., Gokhale, A., Neema, S., Shi, F., and Bapty, T. (2004), 'Model-Driven Program Transformation of a Large Avionics Framework', in *Proceedings of Generative Programming and Component Engineering (GPCE)*, Vancouver, BC, October, pp. 361-378.
- Gray, J., Lin, Y., and Zhang, J. (2006a), 'Automating Change Evolution in Model-Driven Engineering', *IEEE Computer*, Vol. 39, No. 2, February, pp. 51-58.
- Gray, J., Tolvanen, J.-P., Kelly, S., Gokhale, A., Neema, S., and Sprinkle, J., (2006b), 'Domain-Specific Modeling', *Handbook on Dynamic Systems Modeling*, CRC Press.
- Griswold, W. G., Katoy, Y., and Yuan, J. J. (1999), 'Aspect Browser: Tool Support for Managing Dispersed Aspects', *First Workshop on Multi-Dimensional Separation of Concerns*, a workshop held at *OOPSLA*, Denver, CO, November.
- Hannemann, J. and Kiczales, G. (2001), 'Overcoming the Prevalent Decomposition in Legacy Code', *Workshop on Advanced Separation of Concerns*, a workshop held at the *International Conference on Software Engineering (ICSE)*, May.
- Kamiya, T., Kusumoto, S., and Inoue, K. (2002), 'CCFinder: A Multilingual Token-Based Code Clone Detection System for Large Scale Source Code', *IEEE Transactions on Software Engineering*, Vol. 28, No. 2, July, pp. 654-670.
- Karsai, G., Maroti, M., Lédeczi, Á., Gray, J., and Sztipanovits, J. (2004), 'Composition and Cloning in Modeling and Meta-Modeling Languages', *IEEE Transactions on Control System Technology*, Vol. 12, No. 2, March, pp. 263-278.
- Kontogiannis, K., DeMori, R., Merlo, E., Galler, M., and Bernstein, M. (1996), 'Pattern Matching for Clone and Concept Detection', *Automated Software Engineering*, Vol. 3, No. 1-2, June, pp. 77-108.
- Lédeczi, Á., Davis, J., Neema, S., and Agrawal, A. (2003), 'Modeling Methodology for Integrated Simulation of Embedded Systems', *ACM Transactions on Modeling and Computer Simulation*, Vol. 13, No. 1, January, pp. 82-103.
- Mayrand, J., Leblanc, C., and Merlo, E. M. (1996), 'Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics', in *Proceedings of the International Conference on Software Maintenance (ICSM)*, Monterey, CA, November, pp. 244-253.
- Neema, S., Bakay, A., and Karsai, G. (2005), 'Embedded Systems Modeling Language', Institute of Software Integrated Systems, Vanderbilt University, Technical Report. [http://www.escher.isis.vanderbilt.edu/tools/get\\_tool?ESML](http://www.escher.isis.vanderbilt.edu/tools/get_tool?ESML)
- Robillard, M. P. and Murphy, G. C. (2002), 'Concern Graphs: Finding and Describing Concerns using Structural Program Dependencies', in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, Orlando, FL, May, pp. 406-416.
- Sampaio, A., Chitchyan, R., Rashid, A., and Rayson, P. (2005), 'EA-Miner: A Tool for Automating Aspect-Oriented Requirements Identification', in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Long Beach, CA, pp. 352-355.
- Schmidt, D. (2006), 'Model-Driven Engineering', *IEEE Computer*, Vol. 39, No. 2, February, pp. 25-31.
- Sharp, D. (2000), 'Component-Based Product Line Development of Avionics Software', in *Proceedings of the Software Product Lines Conference (SPLC)*, Denver, CO, August, pp. 353-369.
- Shepherd, D., Gibson, E., and Pollock, L. (2004), 'Design and Evaluation of an Automated Aspect Mining Tool', *International Conference on Software Engineering Research and Practice*, Las Vegas, NV, June, pp. 601-607.
- Sudarsan, R. and Gray, J. (2006), 'Meta-Model Search: Using XPath to Search Domain-Specific Models', *Journal of Research and Practice in Information Technology*, Fall 2006.
- TEAM (2006), Towards Evaluating Aspect Mining, a workshop held at the *European Conference on Object-Oriented Programming (ECOOP)*, Nantes, France, July. <http://www.st.cs.uni-sb.de/TEAM/2006/>
- Zhang, C. and Jacobsen, H.-A. (2004), 'PRISM is Research In aSpect Mining', *Software Demonstration at OOPSLA 2004*, Vancouver, BC, October.
- Zhang, J., Lin, Y., and Gray, J. (2005), 'Generic and Domain-Specific Model Refactoring using a Model Transformation Engine', in *Model-Driven Software Development*, (S. Beydeda, M. Book, and V. Gruhn, eds.), Springer, pp. 199-218.