Managing Aspect Orderings
to Support Multiple Quality Concerns

Bedir Tekinerdoğan
Bilkent University
Dept. of Computer Engineering
06800 Bilkent, Ankara, Turkey

bedir@cs.bilkent.edu.tr

Ersin Er
Hacettepe University
Dept. of Computer Engineering
06800 Beytepe, Ankara, Turkey

ersin.er@cs.hacettepe.edu.tr

ABSTRACT

When multiple aspects are composed undesired behavior may emerge due to the interference of aspects. Different interference management approaches have been proposed including detection and resolution of the conflicting aspects. It appears that the majority of the existing approaches have basically focused on functional correctness, whereby orderings of aspects are evaluated with respect to assumed contractual specification. Although functional correctness is an important quality concern also other quality concerns such as evolvability, reuse and reliability can demand a specific ordering. As such, the resulting possible set of orderings might need to be further reduced. In this paper we discuss the impact of other quality concerns than functional correctness, on the required orderings of aspects. Based on a domain analysis of existing approaches we provide a feature model and complementary to this a metamodel for defining aspect interference management approaches for multiple quality concerns.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques.

General Terms

Design, Documentation, Performance, Verification

Keywords

Aspect interference, metamodeling, aspect ordering, quality concerns

1. INTRODUCTION

Aspect-Oriented Software Development (AOSD) provides abstractions to separate and modularize crosscutting concerns into aspects and compose these later in the base code [1]. If multiple aspects are composed, aspects can interact in the base code. This does not pose a problem if the aspects are orthogonal to each other, that is, if their order of processing does not impact the behavior of the interacting aspects. It has been shown, though, that the interaction of aspects can lead to undesired behavior due

to the so-called interference of aspects [9][12]. Aspects interfere with each other if the order of processing the aspects is semantically relevant for the final result [5].

In recent years several studies have been carried out to address the aspect interference problem and a relatively broad insight has been gained on this topic. To cope with this problem several aspect interference detection and resolution approaches have been proposed. The proposed techniques for solving the aspect interference problem are usually dependent on the type of interference [12]. For example, static analysis techniques have been proposed to identify potential shared join-points of the aspects. To detect indirect interaction of aspects through data variables, dataflow analysis and tracing techniques can be applied. To detect interference based on semantic properties contractual specifications of aspects together with model checking techniques can be applied [12].

Despite the different interpretations of interference and the different techniques that are proposed, we can observe that all these approaches aim to *order* the composition of aspects explicitly to prevent undesired interference. A further analysis of the literature shows that the majority of the existing aspect interference approaches have mainly focused on functional correctness as a key motivation for aspect ordering. This means that either implicitly or explicitly it is assumed that an aspect has to adhere to some contractual specifications that define the semantic properties of the aspect.

Yet, although functional correctness is an important concern, if not the most important one, it appears that the ordering of aspects might also be of importance for different quality concerns such as evolvability, reuse, availability and performance. This means that aspects might (also) need to be ordered or reordered for these quality concerns. The main theme of this paper, as such, is that for ordering aspects not only functional correctness but each relevant quality concern should be *explicitly* and *separately* addressed. We think that this observation can further support the research on aspect composition, aspect interference and aspect interference detection and resolution problems.

Based on the existing research on aspect interference problem we first provide a domain model that defines the space of the aspect interference problem and the proposed techniques. The domain model will be presented as a feature model and does not only summarize existing work but may also help to detect new problems and aspect interference management approaches. Further, using a case study and a set of example scenarios we will define the motivation for ordering aspects for multiple quality

concerns. Finally, we provide a metamodel and a generic process for defining aspect interference management approaches for multiple quality concerns.

The outline of the paper is structured as follows: In section 2 we provide the feature model for aspect interference problem and the related techniques. In section 3 we show the impact of aspect ordering on concerns other than functional correctness. In section 4 we define the metamodel for ordering aspects for multiple quality concerns, and define the process for applying the metamodel. Section 5 will provide the related work. Finally, section 6 provides the conclusions.

2. ASPECT INTERACTION

In recent years, several researchers have focused on the aspect interference problem, and this has resulted in several approaches and herewith a better understanding of the problem. To depict the space of the problem we will define feature diagrams for *Aspect Composition, Relation among Aspects, Aspect Interference* and *Inference Detection*. A feature diagram is a tree in which the root represents the domain concept being described and the remaining nodes denote features. Features of a concept can be *mandatory*, *alternative*, or *optional* [2]. The feature diagrams that we present in the following are the result of a thorough domain analysis to the existing aspect interference management approaches.

The first and top level feature diagram is the one for *Aspect Composition,* which is depicted in Figure 1. *Aspect Composition* concept can be characterized using the features *Composition Time, Composition Scheme,* and *Composition Result.* Composition Time refers to the time of weaving the aspect, which can be basically at compile time, load time and run-time. *Composition Scheme* refers to whether the aspects are composed sequentially or concurrently [4]. *Composition Result* defines whether the aspects interact or not. The interaction might be direct on shared joinpoints or indirect through data members. Further, the interaction might lead to interference.
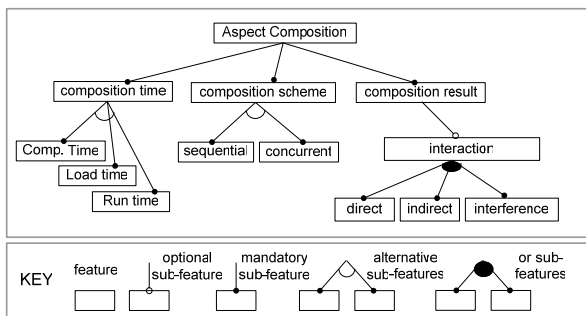

Figure 1. Top-level feature diagram of aspect composition

When aspects interact with each other it is important to know the possible relations among these aspects. Figure 2 defines the feature diagram for *Aspectual Element Relations. Aspectual Element* is either an aspect or advice. This is to denote that several interference management approaches consider composition of aspectual elements either at the granularity level of aspects and/or at the granularity level of advices. Aspectual elements have a relation with the base code (application), which can be *spectative*, *regulative* or *invasive* [8]. Aspects are *spectative* if they only query the state of the base system but do not change it. Aspects are *regulative* if they can alter the control flow of the base system. Finally, *invasive* aspects can alter both the control flow and the

state of the base system. Invasive aspects can be further characterized by *augmentation, narrowing,* or *replacement* [11]. This means that aspectual elements can either augment the base code or narrow its functionality or replace it all together.
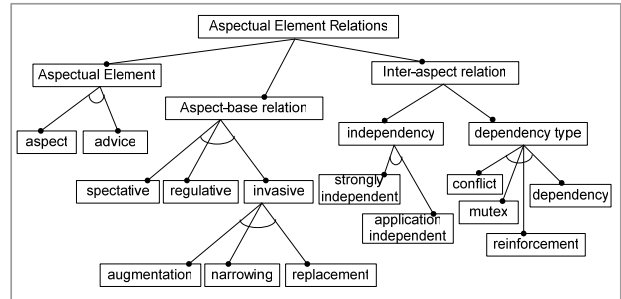

Figure 2. Feature diagram for aspect relations

The feature *Inter-Aspect Relation* defines the relation among aspects in an interaction. Aspectual elements can be dependent or independent from each other. In case of *independency* we can distinguish between *strong independency* and *application independency* [3]. In case of strong independency the aspects are independent for all programs. Application independence relates to independence of aspects for a given particular program. This implies that the same set of aspects could be dependent for a different program.

The feature *Dependency Type* defines the way aspects depend on each other. In principle four different dependency types can be distinguished, *conflict, mutex, reinforcement,* and *dependency* [12]. Aspects conflict with each other if they negatively affect each other's behavior. *Mutex* implies that the aspects cannot be composed together. *Reinforcement* occurs if an aspect positively influences the functionality of another aspect. *Dependency* implies that an aspect requires being composed together with another aspect.
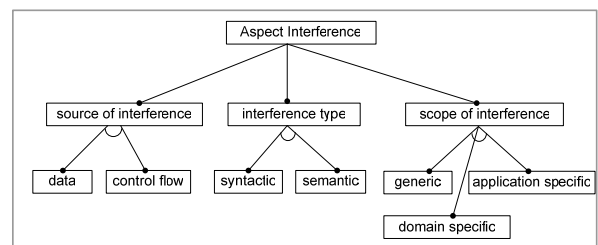

Figure 3. Feature diagram of aspect interference

Figure 3 represents the feature diagram for *Aspect Interference* which can be characterized using three sub-features *source of interference type, interference type,* and *scope of interference. Source of interference* can be either data or control flow related [7][9]. *Interference type* is either syntactic or semantic. *Scope of interference* can be *generic, domain specific* or *application specific* [5].

Figure 4 represents the feature diagram for *Inference Detection* which is characterized by three features *time of detection, technique* and *aspect interference.* The latter one is reused from the feature diagram as presented in Figure 3. The feature *time of detection* defines the time when the interference is detected. Basically, we can distinguish among *pre-weaving, weaving time* and *post-weaving time.* The feature *approach* defines the applied

approaches for detecting interferences. It includes three sub-features *analysis scheme, target of analysis* and *technique.* The analysis scheme can be done using *static analysis* [8] without executing the program. However, some interference problems cannot be detected statically and for these *dynamic analysis* can be applied. Hereby the interference is detected while executing the program or part of the program. The *target of analysis* could be either the interference related to data or control flow, corresponding to *dataflow analysis* and *control flow analysis* respectively. The feature *method* defines particular approaches to detect interference problems. These methods range from theorem proving to resource modeling approaches.
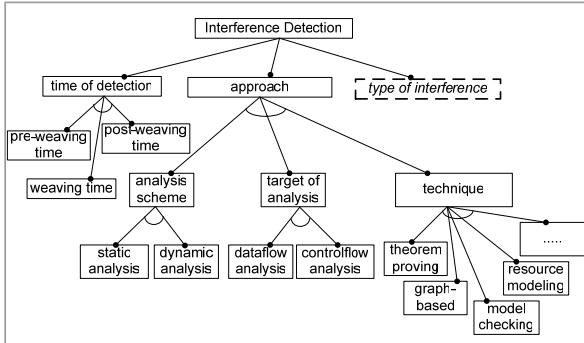

Figure 4. Feature diagram for conflict detection

3. ASPECT ORDERING FOR OTHER QUALITY CONCERNS

3.1 Quality Concerns

In the following we will show that also other quality concerns than functional correctness might demand a specific ordering. For this, let us first consider the feature diagram in Figure 5 for ordering aspectual elements for quality concerns.
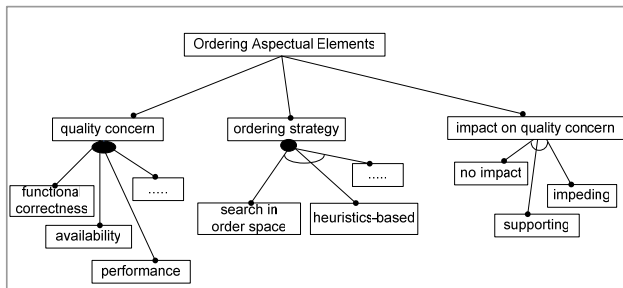

Figure 5. Feature diagram for ordering aspectual elements for quality concerns

Ordering Aspectual Elements can be done for various quality concerns such as *functional correctness, availability* and *performance*. We will discuss these in the following subsection. The aspects might need to be ordered according to some strategy. This could be through *search in a order space*, using *heuristics-based* strategy or any other strategy. In fact, every ordering of aspects can have an impact on the considered quality concerns. We distinguish the following three impacts of ordering aspectual elements on a quality concern:

- *no impact*, the ordering does not matter.

- *impeding*, the specific ordering has a negative impact on the quality concern. For example, it might result in incorrect situation (correctness), lower performance or decrease availability.

- *supporting*, the specific ordering has a positive impact on the quality concern. For example, it might result in better performance, better availability or performance.

3.2 Example Scenarios

In the following subsections we will provide examples of quality concerns that require a specific ordering of aspects.

3.2.1 Example: Ordering for Availability

In general *availability* indicates the degree to which a system is available. When composing aspects, the order of the included aspects should not unnecessarily reduce the availability of the system. Let us discuss how the order can impede the availability. In general availability might be impeded due to failures in the system. The formula for availability is given as follows:

$$Availability = MTTF / (MTTF + MTTR)$$

MTTF and MTTR stand for the mean time to failure and the mean time to recover, respectively. To maximize the availability of the overall system, MTTF of separate modules must be kept high and MTTR must be kept low. An important heuristic might be that for having a low MTTR it is necessary to perform the recovery actions as soon as the failure is detected, and do not execute additional tasks that will be undone by recovery actions anyhow. This might imply that aspect *Recovery* needs to be invoked earlier than other aspects in order not to waste time for imposing other aspects that will be undone by the recovery aspect later on. As such, considering the orderings of aspects from availability perspective it makes sense to put the recovery aspect as early as possible. Note that other orderings where *Recovery* aspect is not put early on, might still be valid from the *functional correctness* concern perspective. The reasoning about *Availability* will further restrict the orderings. This is not only of importance to increase availability but has also a practical benefit in dramatically reducing the number of feasible ordering alternatives.

3.2.2 Example: Ordering for Resource Usage Optimization

Very often resource usage plays an important role in programs, and effort is spent to optimize the programs to that the resources are used efficiently. In this context, the ordering of aspects might have a direct impact on the resource usage. For this example let us consider the two aspects, *Transaction Management* and *Authorization*. *Transaction Management* ensures consistency of a system before and after the execution of certain operations which are considered as atomic units of work. *Authorization* is a process that establishes whether an authenticated user has sufficient permissions to access certain resources. Both *Transaction Management* and *Authorization* aspects can be considered as *spectative aspects* as defined in Figure 2, and as such do not change state variables in the system. Moreover, unless it is explicitly stated for a system, they do not interfere in any way from a functional correctness perspective. So the order of execution of *Transaction Management* and *Authorization* do not lead to a conflicting situation with respect to correctness criteria. However, when we consider *Resource Usage Optimization* as a

quality concern, then it may be more appropriate to execute *Authorization* aspect first. The reason for this is that *Authorization* aspect can conditionally interrupt the execution of the aspect (ordering) chain in case of unauthorized access. For such a case, if *Authorization* is executed after *Transaction Management*, the system resources will be wasted by initiating an unnecessary transaction.

3.2.3 Example: Ordering for Robustness and Evolvability

In general every system has to cope with evolutionary requirements. One of the key concerns in such situations is to anticipate on the changing requirements and define a robust system. In aspect-oriented systems we can encounter both the evolution of base code and the evolution of aspects. If we are dealing with a single aspect, then the need for evolution of base code, might require the change of the pointcuts of the corresponding aspect. This is usually referred to as the *fragile pointcut* problem. Several techniques have been proposed to support the robustness of the pointcuts and to avoid the fragility of aspects. However, if we are dealing with a composition of aspects then this problem might have a larger impact. In case one or more aspects in the aspect ordering evolves, then this might have a further impact on the robustness of pointcuts of other aspects in the chain. In this perspective, we can term the problem as *fragile ordering of aspects*, which can be defined as the situation in which the aspect ordering needs to be broken due to evolution of either the base code or the aspect code. This is not an imaginary problem, because as we have seen before, aspects might have a dependency relation with each other. To release or relieve this problem, aspects might need to be reordered.

4. METAMODEL FOR ORDERING ASPECTS

In the previous sections we have provided a domain model for aspect compositions and the interference problems and approaches. In addition we have indicated the need for considering multiple quality concerns. In this section we provide a metamodel that can be used to define aspect interference management approaches and the generic process for applying it.

4.1 Metamodel

The metamodel is depicted in Figure 6. Hereby, *AspectualElement* is an implementation of crosscutting unit, which may be of different granularity such as an *aspect* or an *advice*. *AspectualElements* are managed by *AspectualElementManager* which adds, removes or updates aspectual elements to the *AspectualElementList*. The *AspectualElementList* is ordered by the *OrderManager* to manage the interference of aspectual elements. *OrderManager* considers one or more *QualityConcerns*, which is modeled by *QualityModel*. *OrderManager* uses *InterferenceModel* and *AspectualModel* to order *AspectualElements*. *AspectualModel* represents the abstraction of *AspectualElement* with respect to *QualityModel*. While *QualityModel* represents a general abstraction of *QualityConcern*, *AspectualModel* represents a specific abstraction of *AspectualElement* with respect to the general *QualityModel*. *InterferenceModel* uses *QualityModel* and *AspectualModel* to define either the feasible or the conflicting orderings of *AspectualElements* with respect to the considered *QualityConcerns*. For this *InterferenceModel* can use different

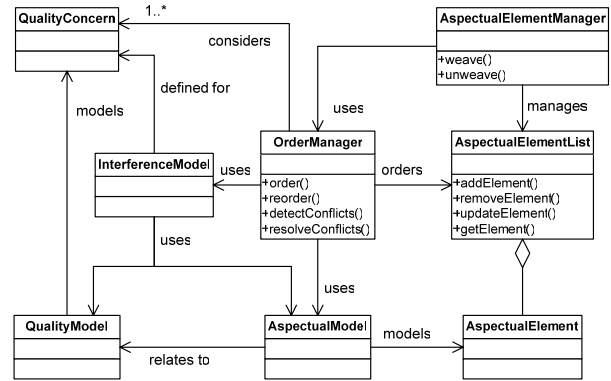techniques such as rules, predicates, regular expressions, automata, or temporal logic [5].

Figure 6. Metamodel for Ordering Aspects based on multiple quality concerns

4.2 Process for applying metamodel

The metamodel of Figure 6 defines the general concepts that can be used to define multiple aspect interference approaches. Obviously, to define a particular approach each concept should be specified concretely. In this paper we do not elaborate on defining a specific approach. Instead, we will define a generic process that can be instantiated to define a concrete aspect interference approach. The process is depicted in Figure 7.
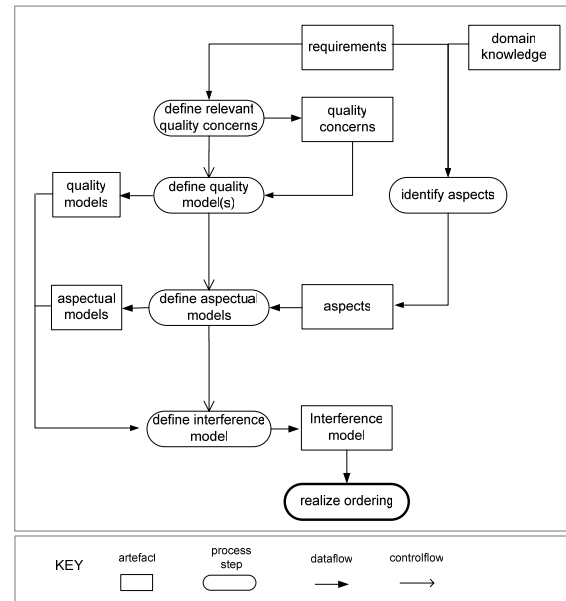

Figure 7. Generic process for aspect interference management for multiple quality concerns

The first step in the process is the identification of quality concerns which are derived from the requirements. Based on the identified quality concerns, quality models are defined. The step *define aspectual models* takes as input the quality models and the aspects and as such result in aspectual models. Aspects are derived from the step *identify aspects* which uses requirements and domain knowledge. The step *define interference model* takes as input the quality models and the aspectual models and provides

16

an interference model. The interference model can be used for realizing the ordering of aspects in the application.

Note that this generic process aims also to cover the existing aspect interference approaches in the literature. In addition we can define a specific process based on this generic process. For this we need to select the specific features from the feature diagram in section 2. For example we could define an interference management system for the quality concerns correctness and availability. We could consider different kind of aspects, adopt specific interference modeling techniques, use different ordering strategies etc.

5. RELATED WORK

As a result of a domain analysis process in this paper we have provided feature models that represent the key features of aspect interference management approaches. To the best of our knowledge this has not been defined before. In the literature we could identify several surveys on aspect interference management approaches such as defined in [7]. We have analyzed these and other publications on specific approaches to define the domain model.

To define the aspect orderings it is very important to know the properties of aspects. In this way the analysis on the interaction of aspects will be supported. For analyzing the aspects the Network of Excellence on AOSD deliverable "A domain analysis of key concerns – known and new candidates" [10] provides an invaluable resource of key aspects. The deliverable includes both traditionally known crosscutting concerns (persistence, security, context awareness and mobility) and more recent candidate crosscutting concerns (agent technology and coordination).

Obviously one of the related key domains that are relevant for the approach that we presented in this paper is the domain of quality attributes. A lot has been published about different quality concerns and its measurement. In this context, ISO 9126 is an international standard for the evaluation of software quality [6]. The standard is divided into four parts: quality model, external metrics, internal metrics and quality in use metrics. The quality model established in the first part of the standard, classifies software quality in a structured set of characteristics and sub-characteristics.

6. CONCLUSIONS

If aspects are composed together in a system, they may interact. If aspects interact, they may semantically interfere. Interference of aspects is bad because they can violate functional correctness. Despite most of the aspect interference management approaches have indeed focused on functional correctness, it appears that aspect interactions are also important for other quality concerns such as availability, performance and evolvability. Since most quality concerns are non-functional and cannot be easily expressed as functional requirements, they need to be addressed explicitly to define feasible aspect orderings. This was the main theme of this paper. When considering aspect orderings we have to take multiple quality concerns into account. To support our statement we have first provided the domain model for aspect interaction and aspect interference and showed that despite the broad knowledge on aspect interference management, the current trend is basically on functional correctness. We have used a number of example scenarios to show the requirements of different quality concerns on aspect orderings. We have given the examples for correctness, availability, resource usage and

evolvability. Obviously other quality concerns could be provided here. For this a more focused analysis to each quality concern is required. We have defined a metamodel that aims to reflect the existing approaches for managing aspect interactions and which can be used to define new aspect interference management approaches. To support the definition of specific aspect interference management approaches we have defined a generic process for realizing the metamodel.

We hope that our study paves the way for a further study towards the impact of orderings of aspect to other quality concerns than just functional correctness. In particular, the requirements of different quality concerns on aspect ordering, needs further investigation. When we are dealing with multiple quality concerns, sooner or later we have also to consider the trade-off among these quality concerns. We have not elaborated on this trade-off analysis of quality concerns yet, but we consider this as our future work. Finally, since aspect interference management is hard to define manually, we will focus on tool development based on the metamodel.

7. REFERENCES

[1] AOSD-Europe. European Network of Excellence on Aspect-Oriented Software Development. European Commission grant IST-2-004349.

[2] Czarnecki, K. and Eisenecker. U. Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000.

[3] Douence, R. and Fradet, P. A framework for the detection and resolution of aspect interactions. In GPCE: ACM SIGPLAN/ SIGSOFT Conference, GPCE 2002, Lecture Notes in Computer Science, Pittsburgh, US, October,6 2002..

[4] Douence, R., Le Botlan, D., Noyé,J., and Südholt, M. Concurrent aspects. In *Proc.of the 5th international Conference on Generative Programming and Component Engineering*, Portland, USA, October 22 - 26, 2006).

[5] Dürr, P., Staijen,T., Bergmans, L. and Aksit, M. Reasoning about semantic conflicts between aspects. In EIWAS '05: The 2nd European Interactive Workshop on Aspects in Software, Brussel, Belgium, September, 1-2, 2005.

[6] International Organization for Standardization. Software Engineering — Product Quality — Part 1: Quality Model. ISO/IEC 9126-1:2001(E), Geneva, Switzerland, 2001.

[7] S. Katz et. al. Detecting Interference among Aspects, NoE AOSD Deliverable, D116, 2007.

[8] Katz, S. A Survey of Verification and Static Analysis for Aspects, AOSD Europe Milestone M8.1, July 2005.

[9] Leavens, G. T. and Clifton, C. Foundations of aspect-oriented languages workshops. In Foundations of Aspect-Oriented Languages Workshop, AOSD, 2003-2008.

[10] Loughran, N., et al. A domain analysis of key concerns - known and new candidates, NoE AOSD Deliverable D

[11] Rinard, M., Salcianu,A. and Bugrara, S. A classification system and analysis for interactions in aspect-oriented programs. In Foundations of Software Engineering (FOSE). ACM, Oct. 2004.

[12] Sanen, F., Truyen, E., Joosen, W., Loughran, N., Rashid, A. Jackson, A. Nedos, A. and Clarke, S. Study on interaction issues. AOSD-Europe Deliverable 44. March 2006.