Chapter 5

Unification of AOP and FOP in Model Driven Development

In this chapter, AOP and FOP have been explored to analyze the similar and different characteristics. The main objective is to justify that AOP and FOP are not competitive approaches but complementary approaches. It has been done from the design perspective, and, in a later chapter it has been done from architectural meta-programming perspective. Models proposed in the previous chapters are modified and integrated in lieu of the present prevailing approaches, which support the integration of AOP and FOP. Techniques of Software Product Line Development have been analyzed. The techniques applied in one of the models used features and aspects simultaneously. A case study on SNTPL has been done to validate the necessity of unification of aspect and feature. This integration of aspect modules with the feature modules enhances the efficiency of MDD.

5.1 Background

Aspect-Oriented and Feature-Oriented Programming are closely related to each other. FOP is a design methodology and tool for program synthesis whereas AOP is a special design methodology for handling cross-cutting concerns. Cross-cutting concerns are inevitable in complex programs. Large-scale compositional programming needs FOP by using feature modularity in product lines. Such programs are complex and hence cross-cutting incidents occur. Some frequently needed unanticipated modifications and extensions of evolving software cause code tangling and code scattering. FOP is not able to modularize these crosscutting concerns appropriately.

Though AOP and FOP originated as two different programming paradigms but present research works have shown that they are complementary to each other [10].
Software designing, maintenance and reusability can be benefitted on unifying them in language and design level. This motivates further to explore unification of AOP and FOP in modular design level.

5.1.1 Analysis of existing approaches to combine Aspects and Features

Several works have been carried out to combine AOP and FOP both on language and design level. It has been observed from all the studies that both paradigms have limitations and strengths [10, 46]. Thus, the following three popular ways have been suggested by Apel et al. [107] for combining AOP and FOP, by introducing AOP concepts in FOP.

(a) Multi Mixins
(b) Aspectual Mixin Layers and
(c) Aspectual Mixins

The characteristics of the three methods have been analyzed in this section. This is to identify the most suitable method as per the advantages and the disadvantages provided by each of them.

Multi Mixins:

(a) Supports homogeneous crosscuts
(b) Prevents excessive hierarchy confirming refinement, by refining all the parent mixins and not refining one mixin with another.

Advantages of Multi Mixins:

(a) Refining is not complex
(b) Wild cards (%) are used to refine all the parent mixins (either classes or methods)

Disadvantages of Multi Mixins:

(a) Does not support heterogeneous and dynamic crosscuts

Aspectual Mixin Layers:
(a) Integration of aspects and mixins are done in design level by placing aspects and mixins in all the mixin layers
(b) A mixin in one layer refines other mixins in different layers
(c) Aspects refine the parent mixins by adding method calls

Advantages of Aspectual Mixin Layers:

(a) Mixins implement heterogeneous crosscuts while aspects implement homogeneous crosscuts
(b) Mixins give hierarchy conforming crosscutting while aspects give non-hierarchy conforming crosscutting
(c) Aspects provide advanced dynamic crosscutting

Disadvantages of Aspectual Mixin Layers:

(a) Designing phase of Software Development Life Cycle (SDLC) can be done while not implementation as, it is not supported by any high level language

Aspectual Mixins:

(a) Integration of aspects and mixins are done in language level
(b) FeatureC++ supports defining and refining mixins
(c) Pointcuts and advices are provided into the definitions of mixins

Advantages of Aspectual Mixins:

(a) An instantiation of a mixin automatically triggers the instantiation of an aspect
(b) In SDLC, both designing and implementation can be achieved

Disadvantages of Aspectual Mixins:

(a) An aspect cannot be instantiated alone
(b) FeatureC++ supports in a limited way, which needs to be enhanced by providing the concepts of multi mixins and aspectual mixin layers
As per the above analysis, it can be said that one can go for aspectual mixin layers when the requirement is for designing only. However, for designing as well as for implementation one has to use aspectual mixins. In any case, multi mixins are not preferred because of the limitations mentioned above. But in future, as indicated by the authors, all the three approaches will be applied in real world case studies [107]. Hence, the second approach is preferred, i.e. aspectual mixin layer, which is suitable in a modular approach where designing of modules are required.

5.1.2 Architectural Symbiosis of Aspect and Feature

As Aspectual Mixin Layers are one of the mechanisms to integrate AOP and FOP, the whole composition of Aspectual Mixin Layers (AML) has been demonstrated. AML extends the notion of mixin layers by encapsulating both mixins and aspects. Figure 5.1 depicts the Aspectual Mixin Layer in terms of class, mixin, aspect etc [86].

![Figure 5.1: Aspectual Mixin Layers with class, mixin, aspect](image)

There are two layers in Figure 5.1. The bottom layer consists of an aspect and also a mixin. The top layer comprises few classes. The classes, aspects and mixins are inter-related by weaving, association, inheritance and mixin-based inheritance. These are depicted in Figure 5.1. An AML (bottom layer) encapsulates both collaborating classes and aspects that contribute to a feature.

An AML may refine a base program in the following two ways.

(a) by using common mixin-composition, or
(b) by using aspect-oriented mechanisms, in particular pointcuts and advice
The most important contribution of AML is that the programmers may choose the appropriate technique – mixins or aspects – that best fits a given problem.

A detail diagram (Figure 5.2) is now presented to show the positions of FOP and AML in a layered structure and the association of them [86].

**Figure 5.2:** Feature and Aspect in a layered structure

In Figure 5.2, classes that are not refined are explicitly displayed as rectangles with dotted lines (e.g., Class 1 in Feature Z). The Figure demonstrates that there are three features X, Y and Z. There are two aspects, Aspect1 and Aspect2 in AML1 and in AML2 respectively. There are also a few classes, where Class 1 and 2 are base classes in Feature X and Class 3 is a base class in Feature Y. The Figure shows the relationships among classes, features, aspects and AMLs with operations like weaving, association, refinement etc. Aspects are weaved into classes or mixins and can be associated with a class. The refinement takes place in a class which appears in two consecutive layers. Figure 5.2 also depicts that the layers are the features. Classes, mixins and aspects are member entities in each feature. Weaving, association, inheritance and refinements integrate all the components in the different layers. This reveals that the features encapsulate aspects and mixins. Weaving, association etc. are
required for the interactions among the features and their entities. This concept has been demonstrated in our STNPL case study, in the later section.

5.1.3 Duality of AOP and FOP

It has been observed that different mechanisms available in AOP and FOP are similar although they look different. The similarities are the following.

(a) The Inter Type Declaration (ITD) or Introduction mechanism of AOP is equivalent to the Mixins mechanism of FOP. Both mechanisms extend OOP classes with additional methods and fields.

(b) The advice execution mechanism of AOP is equivalent to the Method Extension mechanism of FOP. Both mechanisms extend the body of a method for all subsequent calls.

This analysis of duality mechanism [11, 44, 98] helps to co-relate AOP and FOP from the design and the modular levels and also assists in understanding the relationship between AOP and FOP, which is essential for unifying AOP and FOP.

5.1.4 Features and Aspects in Software Product Line Development

The effectiveness of a software product line approach directly depends on how well feature is implemented and managed throughout the development lifecycle, that is, from early analysis through maintenance and evolution. Features are separated in models not only for better abstraction but also because the model transformations support the transition from problem to solution domain. In [85], software product line development has used aspect-oriented techniques for variability management. It is thought that feature-oriented techniques also have significant contributions in variability management. They also have contribution towards the flexibility to adapt to different product requirements that are captured in core features. In fact, variability management comprises activities like identifying, designing, implementing and tracing flexibility in software product lines (SPLs). In traditional SPL approaches, variability management and tracing are also handled using architectural modelling [85].

The implementation of software product line is done by integrating both model-driven software development (MDSD) and by aspect-oriented and feature-oriented
software development (AOSD and FOSD). For building product lines, MDSD, AOSD and FOSD can further increase productivity because of the following reasons [85].

(a) Variability can be described more concisely in model level
(b) The mapping from problem to solution domain can be formally described as model-to-model transformations
(c) Aspect-oriented techniques enable the explicit expression and modularization of crosscutting variability in model
(d) Fine grained traceability is supported since tracing is done on model element level rather than on the level of code artefacts

Based on the work of Voelter et al. [85], an attempt has been made to design the models for login and array handling and then to integrate the login model (based on AOP approach) with the array handling model for building a SPL. In FOSD, each feature is mapped to a module. Hence, it is necessary to design the features and then map the features to modules i.e., mapping a problem to the module. This is known as structural mapping.

The following mappings are defined in order to achieve structural mappings [95].

(a) Basically, each feature is mapped to a module. If a feature should not be mapped, it has to be explicitly excluded from the mapping
(b) Alternatively, mandatory or optional features have the following relationship with their super features
   (i) is-a relation
   (ii) part-of relation

   In is-a relationship the alternative, mandatory or optional features etc are mapped to the super feature. In part-of relationship the alternative, mandatory or optional features etc. are not directly mapped to the super feature but are mapped to a feature that is nested inside the super feature.
(c) An influences-relationship between a feature and another feature has crosscutting nature and is mapped to an adaptation between modules. The influencing module has to adapt the module which it influences. If a feature has more than one influences-relationship, the corresponding module has to adapt all
modules which it influences. The adaptation is realized by class and method bindings.

A proper structural mapping can be obtained by organizing the internal structure of the feature. This internal structure becomes a part of the feature model [95]. Hence, it can say that before structural mapping, the internal structure of the features must be well organized so that a feature model can be composed. As a part of the structural mapping, the feature model for proposed security module has been designed in the later section.

5.1.5 Scope of Unifying AOP and FOP in Language Level

There are different approaches which combine AOP and FOP in language level. Mezini et al. proposed Caesar [46] as a combined approach. Similar to FeatureC++, Caesar supports dynamic crosscutting using pointcuts. Instead of FeatureC++, the focus of Caesar is on aspect reuse and on-demand remodularization. Aspectual Collaborations proposed by Lieberherr et al. [66] encapsulate aspects into modules, with expected and provided interfaces. The objective behind this approach is similar to Caesar. Colyer et al. proposed the principle of dependency alignment [45] as a set of guidelines for structuring features in modules and aspects with regard to program families. They distinguish between orthogonal and weak-orthogonal features/concerns. Loughran et al. support the evolution of program families with Framed Aspects [67]. They combine the advantages of frames and AOP, to serve unanticipated requirements. The present work supports the AML approach since it is found to be a powerful mechanism because of the simplicity of the approach. Moreover, very limited support has been provided for aspect refinement as mentioned in [84]. All the existing softwares are not capable of handling lifters, which is a very powerful mechanism for feature composition.

5.2 Unifying AOP and FOP in the Proposed Model

An Aspectual Mixin Layer is now demonstrated in our proposed scheme of array handling with login security. The array handling activities are designed in software which enables few basic operations of array. The software is named as Array Product Line (APL) which will be described in details in the later chapters. The array operations
provided in previous chapter through FOP will be further modified in this work and will be termed as APL. The term APL will be used, in the subsequent sections which consist of some limited activities in array. Figure 5.3 demonstrates the APL and the login security in Aspectual Mixin Layers.

Figure 5.3: Aspectual Mixin Layers comprising Array & Login operations

Figure 5.3 shows a stack of Mixin Layers that implement some array functionality. The layers are the features named as Base, Allocation, Synchronization etc. The descriptions of the layers are as follows.

(a) **Base Layer:** The layer named Base consists of array and size classes (mixins). The former describes an array and the later describes a counter mechanism to manipulate the size of the array.

(b) **Allocation Layer:** The next layer is Allocation layer which contains array and Noperation classes. The layer Allocation facilitates array allocation in memory and Noperation/Operation disables or enables the array operations. This layer is exclusively providing allocation of array and disabling or enabling operations in array.

(c) **Synchronization Layer:** The Synchronization layer synchronizes the insert and traverse operations with array, size and nooperation facilities.
These three features (Base, Allocation, & Synchronization) are implemented as common Mixin Layers. It is also assumed that the other features are placed in the successive layer which is indicated by the dots.

(d) **Security Layer:** The last layer is login validation which embeds aspects and this feature comprises of only aspect modules. This layer is the Aspectual Mixin Layer as it embeds the aspect modules in this layer. The Login feature is implemented as a separate layer (Aspectual Mixin Layer), with two aspects LogAspect and LogAPL. It consists of a login aspect and a login to the software named APL. The LogAPL prints out the communication messages and is implemented using a common Mixin. The LogAspect captures a set of methods that is refined with logging code (dashed arrows). The LogAspect is derived from LogAPL which is mentioned in the figure as an arrow.

The above mentioned Aspectual Mixin Layers need modifications as the LogAspect module has methods for accessing the modules like array allocation, array size and noperation but the modules like insert and traverse of the Synchronization Layer has no relation with LogAspect. Hence, we refine the Aspectual Mixin Layer of Figure 5.3. The refinement of AML is shown in Figure 5.4, which removes the drawbacks of the previous layered diagram (Figure 5.3).
The two AML at the bottom are Log and Exit Log. The aspect modules are same in both the layers. They differ in the method calls. The Log APL is same as the previous which serves as the parent aspect of the LogAspect module. The LogAspect module of Exit Log layer (feature) is thus involved with method calls. This is done for terminating operations like insert, traverse or Noperations on array and exit from the APL software.

The advantage of the AML is that there are the core features (which are non-crosscutting concerns) in Mixin Layers and the crosscutting concerns in Aspectual Mixin Layers. The layers in both the cases can be increased as per situations. In order to implement the aspect modules there is a need to weave the AMLs with the core features. This technique allows one to integrate AOP and FOP.

The above example also shows that Aspectual Mixin Layers are used and not Aspectual Mixins, as the later allows aspect codes in mixins. The aspect modules are implemented by separate layers.
Both approaches provide solutions for problems of FOP with crosscutting modularity by supporting the following features.

- homogeneous and heterogeneous crosscuts
- dynamic crosscutting (pointcuts, join-point etc.)
- non-hierarchy-conforming refinements
- preventing excessive method extensions
- handling method interface extensions

AML has been used because of software availability. At the same time, the method extension cannot be replaced with lifters, which have been used earlier for feature composition because of lack of support from the existing software for feature handling.

5.2.1 Model Driven Development of Security Aspect

In any software accessing, security implementations are required which are prone to crosscutting code. Authentication is tangled with authorization if the latter is present. In the implementation of authentication, login mechanisms and logout mechanisms are tangled [95]. Hence, security aspects need to be developed by using some latest MDD techniques.

The security aspect in context of code designing has already been explained (Chapter 3). A modular abstraction of the Feature model is now being produced and the mapping of features from problem domain to solution domain is suggested. Abstraction in Feature model helps in problem domain mapping. This is known as structural mapping and is done in the next section.

5.2.2 Proposed Feature Model of the Security Component

The model of security in array handling by using feature models with some modifications has designed [95]. The security component is intended to provide access control by user authentication and authorization. A user management stores login names and passwords. For authorization, it is assumed that access restrictions for each user can
be specified. Authentication and authorization are the functional requirements. Authentication is the minimal function required for access control. Authorization is dependent on authentication.

In the feature model (given in Figure 5.5), this is expressed by modelling authentication as a mandatory feature (filled dot above feature node) while authorization is an optional feature (empty dot). If authentication is only requested for, then the authenticated user has complete access to the application. The minimal requirement for authentication is the login feature. The login is carried out through a user interface. This has been depicted in Figure 5.5.

Figure 5.5: Feature Model of Login to APL

In order to explicitly logout from the system, an optional logout feature is given. Logout automatically takes place at the time of shutdown of the application APL. A new user or the same user can login through re-login feature. The feature ‘Second attempt’ either gives a successful login or provides for automatically logout after a certain lapse of time without user’s interaction. The above model is a generalized feature model for Login to APL. Authorization controls two categories of users. They are the end-users and the developers.
The relevance of the feature model is justified in Figure 5.5. This abstract feature model can now be used for structural mapping as the designed feature model possesses the following virtues.

(a) The designed 10 features need to be mapped

(b) Optional features are Authorization, Logout and Second Attempt. Super feature of Authentication is Authorization, as authorization is followed by authentication. Similarly, Logout and Second Attempt have is-a relation to their super feature Login.

(c) Re-login is a non-alternative sub-feature and is in a part-of relation to the super feature Login. So, re-login should be placed inside the Login so that it is mapped when Login is mapped.

(d) Logout and Second Attempt have an influences-relationship with Login and hence crosscutting occurs. So Login is the influencing module which has to adapt the aspect modules for encapsulating crosscutting concern.

In context of the above discussion, it is desired that a feature model, comprising several features can accommodate aspects in certain features to avoid crosscutting concerns. Login feature embeds aspectual modules for Logout and Second Attempt feature. This also supports that feature and aspect can be combined and hence they are complementary to each other. In order to further demonstrate this fact, an AspectualMixin Layers of STNPL is presented in the next section.

5.3 Case Study: Smart Telephone Network Product Line

As suggested by Apel et al. [103], Feature-Oriented Software Development aims essentially at three properties: structure, reuse and variation. In Chapter 3 and 4, the first two properties have been used in the Case study. In this section the variation properties are used by unifying AOP and FOP through aspectral mixin layers and composing varied product by selecting different features.
5.3.1 Unification of AOP and FOP in STNPL

The aspect security module can easily be integrated into the Telephone Network Product Line, which is a FOP based designing. This requires Aspectual Mixin Layers. The mixin layers consist of the features and the aspect layers. Figure 5.6 depicts the Aspectual Mixin Layers of STNPL.

![Figure 5.6: Aspectual Mixin Layers of STNPL](image)

The feature layers for call processing are Base, Fetch, Synchronization and Call Operation. The security features are present in the last two layers, Phone and Exit Phone. The details of the classes present in each layer are given below.

a) Base Layer: This layer consists of PSTN, Gateway and Server classes which contains the necessary methods for call processing.

b) Fetch Layer: The checker class detects the route availability to server. The PSTN class fetches the route and time information.
c) Synchronization Layer: This layer provides the necessary interfaces to the different classes of the layer for synchronizing the activities among the layers.

d) Call Operation Layer: The different units, apart from the units of the previous layers like connection created, receive, call end, connection closed etc. functions together for call processing.

e) Phone Layer: This layer takes care of the login process into PSTN. The aspect code is in LogAspect module.

f) Exit Phone: This layer, through LogAspect performs connection establishment, call receive, call end etc. It acts as an interface to the call operations.

5.3.2 Significance of AML in STNPL

The Aspectual Mixin Layers not only design the functions in layers but also integrates aspect layers into feature layers. The integration is done by designing the aspects which acts as interfaces. The core functions are present in the upper layers whereas the crosscutting concerns are present in the bottom layers. This designing gives a clear visualization of the core and crosscutting functions. Aspectual Mixin Layers are very useful in designing the different types of functions. In MDD, aspects and features are necessary to encapsulate cross cutting and non-crosscutting concerns. However, when the encapsulated modules (aspects or features) are presented in a layered structure then the co-ordination among them becomes more distinct. Properly designed AMLs not only unifies aspects and features, but also gives clarity of the crosscutting and non-crosscutting concerns, their relationships and the role in the system. Another advantage is the extensibility by adding additional layers either in feature layers (non-crosscutting functions) or in aspect layers (the lower layers for crosscutting concerns) which is necessary for MDD.

The designed STNPL can have additional functions like call conversation storage, billing information against call duration etc. which can be done by adding feature layers. The classes in the new feature layers must be synchronized with the server. This justifies that AML also supports software evolution.
5.4 Summary

In the proposed model, the security module with the feature module has been embedded. These are responsible for basic array operations through AML which justifies the unification of AOP and FOP. The two aspects login and logout have been concentrated upon which have been incorporated initially in a single aspectual layer and later on they have been segregated into two layers to enhance the performance. This signifies that more refinements can be obtained in a layered design which encapsulates the details of all the operations. Each layer can be mapped to different models by using suitable user interface configurations. The main objective was to design Feature models in which a cross-cutting module Login, Logout, authorization (partly) etc. are encapsulated. The role of Feature model has also been demonstrated. This is required for structural mapping in Software Product Line and is commonly used for expressing and binding variability. The case study of STNPL reveals that AML brings in more clarity in representing crosscutting and non-crosscutting functions. It also suggests evolutionary software. Hence, for efficient MDD techniques, aspects and features are not only required but their integration through AML substantially enhances the performance.