Aspect Oriented Programming

There are already a lot of robust programming methodologies like "Object Oriented Programming", "Procedural Programming" and "Distributed OOP". So, is it really a necessity to have another programming methodology named "Aspect Oriented Programming"? Is "Aspect Oriented Programming" is a substitute for "Object Oriented Programming"? What is an aspect? Many questions come in mind and the mystery gets more puzzled when Googling in the internet. This article gives a brief idea of this new programming paradigm coined by Gregor Kiczales.

ASHOK KU. HOTA
Technical Director
NIC Orissa
ak.hota@nic.in

ER. NILADRI B. MOHANTY
Scientific Officer
NIC Peren
niladri.mohanty@nic.in

OBJECT Oriented Programming (OOP) has created a revolution by simplifying the computing problems through visualization of a system as a group of entities and interaction between those entities. But this revolutionary model is very static in nature and any changes in the requirement regarding the security, logging or exception handling can seriously affect the development timeline. In the OOP many classes and methods used to contain some codes, which are not the primary responsibility of that class/method. These kinds of codes are used to be called as tangle codes.

For Example: In an e-Governance application, one form has been developed to take employee code as an input and produce the service book of the employee with options to print and send through e-mail. In this application the class "get_emp_details" supposed to be responsible for only to process the input i.e. employee code and provide the output. But due to the static nature of OOP the class is forced to sanitize the input from malicious character to avoid Sql Injection attack, call the "print()" method of another class to print the service book as well as calls the "send_mail()" method to send service book through e-mail. These extra codes are called as tangles codes and their entry points are called as cross cuts. This kind of codes actually dilutes the real concepts of modularity. In this case, the primary responsibility of the class "get_emp_details" is to fetch the service book from database by taking the employee code as input. The secondary responsibility is to sanitize input parameters and call "print()" as well as "send_mail()" methods.

The additional codes required to fulfill the secondary responsibility of the "get_emp_details" class can be kept in a single location rather than its redundant use by different classes at different time. To complement the OOP, Aspect Oriented Programming (AOP) allows the developer to dynamically modify the static Object Oriented Model to create a system that can grow as and when new requirement arrives.

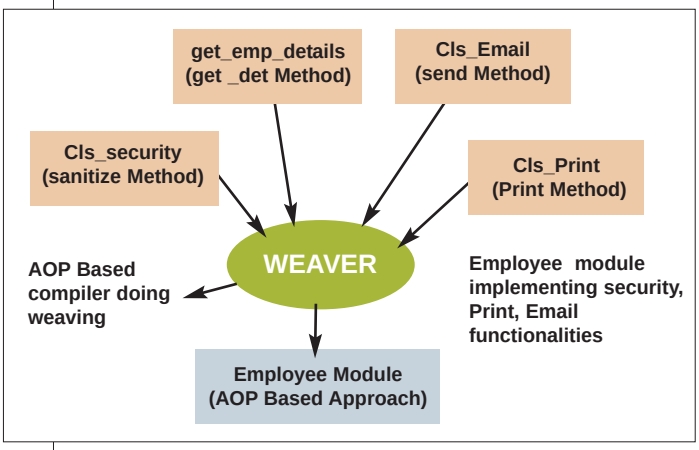
"Aspect Oriented Programming is a methodology to separate cross cut code across different modules in a software system."

AOP provides the solution for Tangle Codes by separating primary codes and secondary codes in separate modules and then feed both the modules to the compiler. AOP does not replace existing programming

```
public void get_det(empcode as string)
{
        Cls_security sec = new Cls_security (empcode);
        Sempcode =sec.sanitize();
        ////////////////////////////////////
        // This method fetches the service book
        //
        // Adding code will go here
        ////////////////////////////////////
        // After fetching from database email is sent
        Cls_Email Obj_email = new Cls_Email ();
        Obj_email.Send ();
        // After sending email its printed
        Cls_Print Obj_Print = new Cls_Print();
        Obj_Print.Print();
}
```

Primary Code

Secondary codes/Tangle Code -This Dilute the core responsibility of the method



paradigms and languages; instead, it works with them to improve their expressiveness and utility. It enhances our ability to express the separation of concerns, necessary for a well-designed, maintainable software system. AOP supported compilers generates single executable module after the compilation of the both Primary & secondary codes. This process of compiling core and cross cut concern together by the AOP supported compiler is to be known as weaving.

Types of AOP Compilers

- **Compile Time Weaving:** In this weaving the core concern (Primary code) and cross cutting concern (Secondary code) is weaved by the AOP compiler at compile time and then feed in to the main compiler.
- **Link time Weaving:** This type of AOP compiler should weave the core concern and cross cutting concern after the generation of intermediate code at the linker level.
- **Run time Weaving:** Here the core concern and cross cutting concern are used to be detected and executed at run time.

Terminologies used in AOP

Cross cuts: A program logic is used to be consists of many distinct parts called as concerns. Few methodologies like procedures, modules and classes are used to separate, group and encapsulate different concerns of the program logic. Still there are some kinds of concerns which are not feasible to be implemented through the above mentioned methodology as this concern cuts across multiple abstraction in a program. Logging exemplifies a crosscutting concern because a logging strategy necessarily affects every logged part of the system. Logging thereby crosscuts all logged classes and methods. Exception Handling, Security and fault tolerance codes may also be considered as crosscutting concerns in the core module.

Advice: The "advice" is an extra code required to fulfill the secondary requirements of the existing model. Code to implement logging, security, exception handling etc. can be considered as an advice in the perspective of AOP. It defines what needs to be applied and when in a particular system. In AOP there are different types of Advice as Before Advice, After Advice, Around Advice and Throws Advice.

Joint Points: These are the points before and after the method execution where the Advice needs to be applied. This is the term given to the point of execution in the application at which cross-cutting concern needs to be applied. The combination of different 'Joint points' where the advice need to be applied is called 'Point cuts'.

Aspect: Aspect is the combination of Pointcuts and Advice. In a better way it can be said that, the act of appling Advice at the "Point Cuts" is called Aspect.

In the context of above "employee module" implementation, let us see the logging features through AOP based approach.

Security code, exception handling, Self Healing System implementation, fault tolerance codes, logging features etc are becoming more and more popular in today's software development to achieve quality, security and robustness in the software. The AOP concept has given an opportunity for the developer to adopt a modular release approach for the software and make the application "attack proof" with out changing the core classes repeatedly. As realists, we acknowledge that no one process, technique, language, or platform is good for all situations and AOP is not out of the box. Specifically AOP may not be suitable when code review is extensively in used for security audit and other testing purposes as, we do not know whether the code might be either augmented by an advice from some aspect or completely replaced by such advice at runtime. To be able to reason about an application's code, we must be able to look at the code from each class as well as the code for any aspects that might affect the class's behavior. However the AOP is a best tool for migration of a legacy system and modify their existing functionalities without affecting their core codes so much.

```
public aspect EmployeeChangeLogger { pointcut
employeeUpdates(Employee e) : call( public void
Employee.update*Info()) && target(e);
pointcut employeeFinanceUpdates(Employee e) :
call( public void update*Info(Employee)) &&
args(e);
after(Employee e) returning :
employeeUpdates(e) || employeeFinanceUpdates(e)
{ System.out.println("\t>Employee : "
+e.getName() + " has had a change ");
System.out.println("\t>Changed by " +
thisJoinPoint.getSignature()); } }
```

Pont cuts
Pont cuts
Advice
Aspect