IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Review - Aspect-Oriented Programming

Ankur Singh Bist
Govind Ballabh Pant University of Agriculture & Technology, India
ankur1990bist@gmail.com

Abstract

Grid computing is a term referring to the association of computer assets from multiple administrative domains to This paper presents various features of aspect oriented programming and tries to analyze the effect of aspect oriented methodology on c and java domain respectively called ASPECTC and ASPECTJ .We also include the various security crosscutting concerns with the implementation approaches of aspect oriented approach that has been used for various modifications .

Keywords: Grid computing, AspectsJ.

Introduction

Object oriented programming is widely used in present scenario and this approach is properly involved in the whole flow of software tools . Xerox PARC effort in the development of programming methods has made it possible the origin and development of programming named aspect oriented programming that overcomes the situations where object oriented programming get failed.

There are various concerns at various levels one of these concern is core concern and other are system level concerns , object oriented programming tries to synchronize this situation but to localize these all concerns becomes very heavy and complex task , these all concern collectively called as security concern includes logging , security and other issues associated .

OOP added a new chapter of encapsulation and inheritance similarly AOP give its contribution to solve and deal with various issues related to crosscutting .

ASPECTJ , HYPERJ ,DJ , ASPECTC all these are aspect oriented extension . ASPECTJ is one of the most popular extension of AOP so java program is appropriate AspectJ program .

This paper is divided in such a way that it starts with formal introduction of Aspect oriented programming and traverse the complete path that includes the other essential issues of crosscutting and at last we try to give merits and demerits of AOP .

AspectsJ

AOP has several direct antecedents: reflection and metaobject protocols, subject- oriented programming , Composition Filters and Adaptive Programming.
Gregor Kiczales and colleagues at Xerox PARC developed the explicit concept of AOP, and followed this with the AspectJ AOP extension to Java. IBM's research team pursued a tool approach over a language design approach and in 2001 proposed Hyper/J and the Concern Manipulation Environment , which have not seen wide usage. EmacsLisp changelog added AOP related code in version 19.28. The examples in this article use AspectJ as it is the most widely known AOP language.

The Microsoft Transaction Server is considered to be the first major application of AOP followed by Enterprise JavaBean .

ASPECTJ *is a small and well-integrated extension to Java . Java is a general-purpose OO language and it uses freely available implementation utilizes compiler that is Open Source it also includes IDE support like emacs, JBuilder 3.5, Forte 4J[1] .*

ASPECTJ includes following attributes-

1. Aspects

An aspect is a module for handling crosscutting concerns .Aspects are defined in terms of pointcuts, advice, and introduction .Aspects are reusable and inheritable .

2. Joinpoints – Points of execution of java program. it further includes ----

. constructor call , method call , exception handler execution , Exception handler execution .

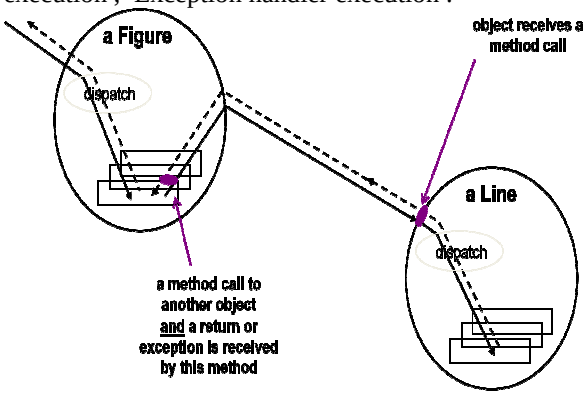

Fig. 1

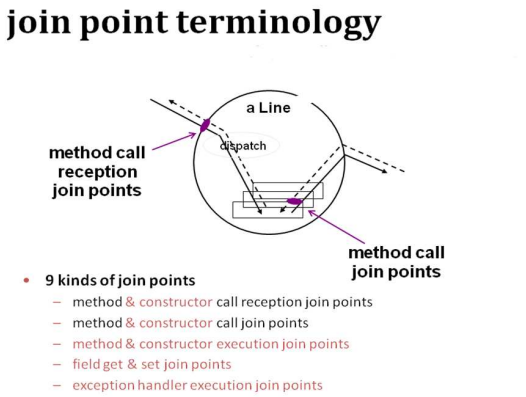

Fig 2 [1]

3. Pointcuts
   A pointcut is a group of join points .

4. Advice
   Advice is code that is executed at a pointcut . Introduction modifies the members of a class and the relationships between classes .

Join Point Models

The advice-related component of an aspect-oriented language explains a join point model (JPM). A JPM explains three things [2]:

1. When the advice can execute . These are called *join points* because they are points in a running program where additional behavior can be joined efficiently . A join point requires to be addressable and understandable by an ordinary programmer to be useful. It should also be stable across inconsequential program changes in order for an aspect to be stable across such changes. Many Aspect oriented programming implementations support method executions and field references as join points.
2. A method to specify (or *quantify*) join points, called *pointcuts*. Pointcuts identifies whether a given join point matches. Most beneficial pointcut languages use a syntax like the base language (for example, AspectJ uses Java signatures) and promotes reuse through naming and combination.
3. A means of specifying code to execute at a join point. AspectJ calls this *advice*, and can execute it before, after, and around join points. Some implementations also promotes things like defining a method in an aspect on another class.

Join-point models can be compared based on the join points exposed, how join points are specified, the operations permitted at the join points, and the structural enhancements that can be expressed.

Join-Point Model of AspectJ

- The join points in AspectJ include method or constructor call or execution, the initialization of a class or object, field read and write access, exception handlers, etc. They do not include loops, super calls, throws clauses, multiple statements, etc [2].
- Pointcuts are specified by combinations of *primitive pointcut designators* (PCDs). "Kinded" PCDs match a particular kind of join point (e.g., method execution) and tend to take as input a Java-like signature. One such pointcut looks like this:

```
execution(* set*(*))
```

This pointcut recognizes a method-execution join point, if the method name starts with "set" and there is exactly one argument of any type.

"Dynamic" PCDs check runtime types and bind variables. For example

```
this(Point)
```

This pointcut recognizes when the currently executing object is an instance of class Point. Note that the unqualified name of a class can be used via Java's normal type lookup.

"Scope" PCDs limit the lexical scope of the join point. For example:

```
within(com.company.*)
```

This pointcut recognizes any join point in any type in the com.company package. The * is one form of the wildcards that can be used to match many things with one signature.

Pointcuts can be composed and named for reuse. For example
pointcut set() : execution(* set*(*)) && this(Point) && within(com.company.*);
This pointcut recognizes a method-execution join point, if the method name starts with "set" and this is an instance of type Point in the com.company package. It can be referred to using the name "set()".

- Advice specifies to run at (before, after, or around) a join point (specified with a pointcut) certain code (specified like code in a method). The AOP runtime invokes Advice automatically when the pointcut recognizes the join point. For example:

```
after() : set() {
  Display.update();
}
```

This effectively specifies: "if the *set()* pointcut recognizes the join point, run the code Display.update() after the join point completes."

Other Potential Join Point Models

There are many other kinds of JPMs. All advice languages can be explained in terms of their JPM. For example, a hypothetical aspect language for UML may have the following JPM:

- Join points are all model elements.
- Pointcuts are some boolean expression combining the model elements.
- The means of affect at these points are a visualization of all the matched join points.

Inter-Type Declarations

Inter-type declarations provide a way to express crosscutting concerns affecting the structure of modules. Also known as *open classes*, this enables programmers to declare in one place members or parents of another class, typically in order to combine all the code related to a concern in one aspect. For example, if a programmer implemented the crosscutting display-update concern using visitors instead, an inter-type declaration using the visitor pattern might look like this in AspectJ:

```
 aspect Display Update {
   void Point. acceptVisitor (Visitor v) {
     v.visit(this); }    // other crosscutting code...
 }
```

This code snippet adds the acceptVisitor method to the Point class.

It is a requirement that any structural additions be compatible with the original class, so that clients of the existing class continue to operate, unless the AOP implementation can expect to control all clients at all times.

Implementation

AOP programs can make change to other programs in two different ways, depending on the underlying languages and environments:

1. a combined program is produced, valid in the original language and indistinguishable from an simple program to the ultimate interpreter
2. the ultimate interpreter or environment is updated to understand and implement AOP features.

The difficulty of varying environments means most implementations produce compatible combination of programs through a process known as *weaving* . It is a special case of program transformation . An aspect weaver reads the aspect-oriented code and produces appropriate object-oriented code with the aspects integrated. The same AOP language can be implemented through a variety of weaving methods, so the semantics of a language should never be understood in terms of the weaving implementation. Only the speed of an implementation and its ease of deployment are fluctuated by which method of combination is used

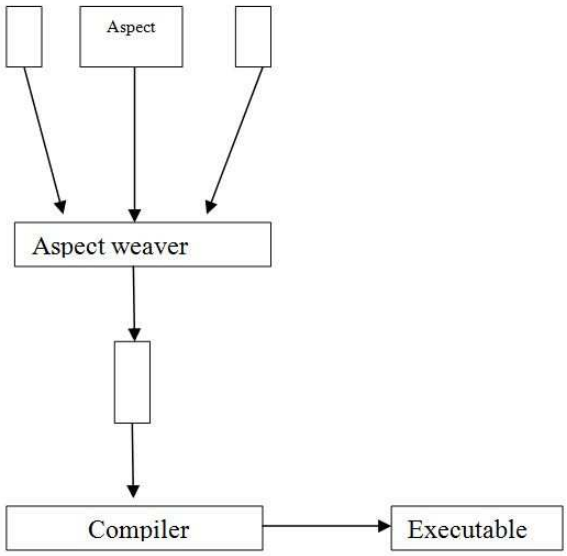

Fig 3

Aspects

RUN TIME VERIFICATION AND Aspect oriented programming have the common purpose of monitoring program execution against temporal files. A monitored program is instrumented which is obtained in with various packages like source code instrumentation tools cil for c , to drive monitors. State machine based monitoring system such as RMOR for c get developed ,it is a extension to RCAT . ASPECTC with state machines inspired by rmor resulting in XSPEC language x for crosscutting have support of aspect oriented programming , spec used for specification . Testing and fault protection are some functional activities of these framework[3].

- According to Klaus Havelund and Eric Van Wyk , there are following requirement for monitoring the execution of c program against specification :-Lexical specification language
- State machines
- Temporal languages
- Programs as specifications
- Aspect oriented programming
- Domain specific monitoring

Klaus Havelund and Eric Van Wyk also explained the design of XSPEC , programming a monitor in ASPECTC ,RMOR, XSPEC with implementation of Extensible xspec .
Collectively it can be said that XSPEC can be used as a general and relevant tool for aspect oriented programming it also glows for purpose of run time verification.

Quantification & Obliviousness

Robert E. Filman and Daniel P. Friedman , tried to explain the question what makes a language Aop ?

The two factors Quantification and obliviousness are found to be necessary ingredients for aop .

Quantified program statement have their effect on many loci in the certain code .oblivious programmers do not require additional effort to make aop mechanism work. Aop can be stated as the union of quantifications handled by oblivious programmers. Quantification have static and dynamic subtypes that further includes their specifications . As a summary it can be stated that aop is not about because oblivious quantification is independent of object oriented concepts and also aop is not beneficial for singletons. More oblivious in system make the AOP better.

Seonah Lee , focussed on the capability of aspect oriented programming as a reverse engineering tool due to its tracing capability . He also explained the modification of reflection model with aspect .

Benefits and Drawbacks of AOP

- Aop solved the code tangling and various crosscutting concern but it is not tested in planned way .
- Dima AlHadidi , Nadia Belblidia , Mourad explained some possible extension for aspectj shortcomings such as : Predicted control flow pointcut ,dataflow pointcut ,loop pointcut ,pattern matching wildcard ,type pattern modifier s,local variable get and set and synchronized block join point [4] .

Conclusion

AOP introduces a different and efficient way of programming and will be used in programming paradigms .rule based framework for implementing state machine and logic are still to be developed efficiently. More flexibility of Aop in different environment will make it more widely used and popular in near future .

References

[1] Gregor Kiczales at Xerox Parc aspect oriented programming concepts .
[2] www.wikipedia.com.
[3] Klaus Havelund and Eric Van Wyk , modelling with ASPECTC .
[4] Robert E. Filman and Daniel P. Friedman , Quantification and obliviousness in ASPECT oriented .