Implementation of Readers-Writers Problem using Aspect Oriented Programming

Kamal Kant Sharma
MTU,Noida
KIET Ghaziabad,
U.P. India.

Neha Garg
MTU,Noida
KIET Ghaziabad,
U.P. India.

Neha Yadav
MTU,Noida
KIET Ghaziabad,
U.P. India.

Sunita Kanaujiya
MTU,Noida
KIET Ghaziabad,
U.P. India

ABSTRACT

Readers-Writers problem is a classical synchronization problem in the field of computer science. It can easily be implemented using any object oriented language. However, the implementation of object oriented programming often leads code to be tangled between functional codes and synchronization codes, which are easy to lead code-scattering and code-tangling. Aspect-oriented programming (AOP) is a programming paradigm which isolates secondary or supporting functions from the main program's business logic. It aims to increase modularity by allowing the separation of cross-cutting concerns. All AOP implementations have some crosscutting expressions that encapsulate each concern in one place. With this there is minimal or no code scattering and tangling. This paper aims to resolve concrete aspect and implement the synchronization of readers-writers problem based on AOP. The execution time of AOP and OOP based solutions are measured which shows that AOP can almost get the same execution time as of object-oriented programming, but with better modularization than OOP.

Keywords

Readers-Writers Problem, Object Oriented Programming(OOP), Aspect Oriented Programming(AOP), Synchronization.

1. INTRODUCTION

In the field of computer science, the readers-writers problem is a classical example of the multi-process synchronization problem. Synchronization is an important and familiar problem in the design and development of the software. When multiple processes or threads access a common critical resource, synchronization is required. Here we have to make sure that the access to data is properly controlled so that no data loss happens.

Using OOP for solution leads to code tangling and scattering. Aspect-Oriented Programming (AOP) was first proposed in [2] as a programming technique for modularizing concerns that cross-cut the basic functionality of programs and hence reduce the limitations with OOP solution technique. The producer and consumer problem has been solved[8] using AOP.

Though much work has been done over aspect oriented methodology, there is less work on the readers-writers problem using AOP. As the readers-writers problem is a representative problem in synchronization, the solution will help in various areas where synchronization is required.

This paper takes the classical readers-writers problem as an example to provide the solution to the synchronization using AOP. In section III, the implementation of readers-writers problem is presented. In section IV, the comparison of execution time is done between OOP and AOP. Section V concludes the paper.

1.1 Object Oriented Programming Solution

Many object-oriented programming languages have supported the synchronization and can implement the readers-writers problem. For example, Java programming language implements the synchronization through the keyword synchronized, as the prefix of the method, that allows only one thread enters the synchronized code at the same time and avoid abusing the critical resource. Java can also control the communication among the thread by the methods: wait(), notify() or notifyAll(). All three methods can be called only from within a synchronized method. Although conceptually advanced from a computer science perspective, the rules for using these methods are actually quite simple: ---

- wait( ) : Tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).
- notify( ) : Wakes up the first thread that called wait( ) on the same object.
- Notify All( ) : Wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.

2. PROBLEMS WITH THE OOP SOLUTION

The implementation of OOP leads the code to be tangled between the function codes and non-functional codes, which are easy to lead code-scattering and code-tangling. Scattering is when similar code is distributed throughout many program modules. This differs from a component being used by many other components since it involves the risk of misuse at each point and of inconsistencies across all points. Changes to the implementation may require finding and editing all affected code. Tangling is when two or more concerns are implemented in the same body of code or component, making it more difficult to understand. Changes to one implementation may cause unintended changes to other tangled concerns. It is not beneficial to the development and maintenance of the software.

3. ASPECT ORIENTED PROGRAMMING: A BETTER SOLUTION

Aspect-Oriented Programming (AOP) was first proposed in as a programming technique for modularizing concerns that cross-cut the basic functionality of programs. The aim of AOP is to resolve the code-scattering and code-tangling and modularize the crosscutting concerns. The crosscutting concerns include security, logging, exception handling and synchronization etc. AOP support independent concerns like resource sharing, synchronization, debugging or distribution in a module[5].

Many distinguish work has been done to deal with the discrete aspect. Aspects can contain several entities unavailable to standard classes. These are Inter-type declaration , Pointcuts and Advice.

3.1 Inter-type declaration

Allow to add methods, fields etc to existing classes from within the aspect. This example adds an acceptVisitor method to the Point class:---

```
aspect VisitAspect
        {
          void Point.acceptVisitor(Visitor v)
            { v.visit(this); }     }
```

3.2 Pointcuts

Pointcuts allow to specify join points which are well-defined moments in the execution of a program, like method call, object instantiation, variable access etc. For example, this point-cut matches the execution of any instance method in an object of type Table whose name begins with you:----

```
pointcut set() : execution(* you*(..) )     &&
this(Table);
```

3.3 Advice

Advice allows to specify code to run at a join point. The actions can be performed *before*, *after*, or *around* the specified join point.Eg :--

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

4. IMPLEMENTATION OF READERS-WRITER PROBLEM

This section presents the implementation of readers-writers problem.

4.1 Description Of The Reader-Writer's Problem

In computer science, the first and second readers-writers problems are examples of a common computing problem in concurrency. The two problems deal with situations in which many threads must access the same shared memory at one time, some reading and some writing, with the natural constraint that no process may access the share for reading or writing while another process is in the act of writing to it. (In particular, it is allowed for two or more readers to access the share at the same time.) A readers-writer lock is a data structure that solves one or more of the readers-writers problems.We have following two variants of the problem :---

4.1.1 First Reader-Writer's Problem : Suppose we have a shared memory area with the constraints detailed above. It is possible to protect the shared data behind a mutex, in which case clearly no thread can access the data at the same time as another writer. However, this solution is suboptimal, because it is possible that a reader R_1 might have the lock, and then another reader R_2 request access. It would be foolish for R_2 to wait until R_1 was done before starting its own read operation; instead, R_2 should start right away. This is the motivation for the first readers-writers problem, in which the constraint is added that no reader shall be kept waiting if the share is currently opened for reading. This is also called readers-preference.

4.1.2 Second Reader-Writer's Problem : Suppose we have a shared memory area protected by a mutex, as above. This solution is suboptimal, because it is possible that a reader R_1 might have the lock, a writer W be waiting for the lock, and then a reader R_2 request access. It would be foolish for R_2 to jump in immediately, ahead of W; if that happened often enough, W would starve. Instead, W should start as soon as possible. This is the motivation for the second readers-writers problem, in which the constraint is added that no writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is also called writers-preference.

4.2 Implementation Of First Readers-Writers Problem

In this mode, we allow multiple readers to read the shared data if currently a read operation is in execution. In the implementation we have a class readwriteAOP that allows user to enter the number of Reader and Writer threads. After getting the values we instantiate the readers with Reader class and writers with Writer class. These classes have a run function that is executed when the Reader or Writer thread is executed. Next, we have an aspect named synreadwrite that defines pointcuts before and after the execution of the run function in Reader and Writer. Before executing the run function of either class the synchronization constraints are checked i.e. for reader we check weather a writer is accessing data if so we ask the reader to wait or else it is given access, for writer we check whether there is any other process accessing data if so we ask writer to wait or else we give it access. If a writer has finished executing it will notify other threads waiting, while when a reader finishes executing it will notify to other waiting Writer Threads[1]. The Coding used is as follows :---

THE SYNREADWRITE ASPECT

```
public aspect synreadwrite {
        long starttime,endtime;
        private int readers;
        public synreadwrite()
        {
                this.readers = 0;
        }
        pointcut syncRead():call (void Database.read());
        before(): syncRead()
```

```
    {
                synchronized(this)
                {
                        this.readers++;
                starttime=System.currentTimeMillis();
                         System.out.println("Reader " +
  number + " starts reading.");
                 }
}
after() returning:syncRead()
{
                synchronized(this)
                {
    endtime=System.currentTimeMillis();
System.out.println("Reader " + number + " stops
  reading.");
                System.out.println("Time Taken By Reader
  : " + (endtime-
    starttime));
this.readers--;
                        if (this.readers == 0)
                        {
                                this.notifyAll();
                        }
                }


}
pointcut syncwrite() : call  (void Writer.run());
before() : syncwrite()
{               while (this.readers != 0)
                {
                   try
                         {
    this.wait();
                        }
                   catch (InterruptedException e) {}
                }
    starttime=System.currentTimeMillis();
}
after() returning : syncwrite()
{               endtime=System.currentTimeMillis();
                System.out.println("Writer " + number + "
  stops writing.");
                System.out.println("Time Taken By Writer
  : " + (endtime-starttime));
                this.notifyAll();
    } }
```

The programming tools that we employed is Eclipse 3.6 and AspectJ for executing the program. The AspectJ plugin AJDT(Aspect J Development Tools) is used to collaborate with Eclipse to get the result[3]. AspectJ is an extension to Java, where the

form of an aspect is similar to the form of a class[6].

4.2.1 Implementation Of First Readers-Writers Problem Using Object Oriented Programming

In the solution using OOP the classes used are the same as in above section (section 2). Here we don't have the aspect functionality to capture the cross cutting synchronization concern, so here we add the synchronization functionality to the database class.

5. EXPERIMENTATION

The execution time is compared between AOP and OOP. In the experiment we include four threads: two reader threads and two writer threads. The hardware and software environment is as following.

In the aspect of hardware, the frequency of CPU is Intel Core™ 2 Duo T5600 2.00GHz and the capacity of memory is 4GB.

In the aspect of software, operating system is Windows 7, and the software uses Eclipse 3.6 and AspectJ's Eclipse plug-in AJDT(Aspect J Development Tools). We separately test the execution time according to the OOP and AOP implementation. The result of a sample execution time of both AOP and OOP implemented program is shown in TABLE I.

As shown in TABLE below, the execution time of AOP is very close to that of OOP and almost bettering it. The execution of OOP is zero(5000 is the base) sometimes while the execution of AOP is not zero. Sometimes the execution time of AOP is zero. We repeatedly executed the program with different number of Readers and Writers, each time finding that the AOP implemented program was bettering off as compared to the OOP implemented program. In the aspect of software, operating system is Windows 7, and the software uses Eclipse 3.6 and AspectJ's Eclipse plug-in AJDT(Aspect J Development Tools). We separately test the execution time according to the OOP and AOP implementation.

The result of a sample execution time of both AOP and OOP implemented program is shown in TABLE I.

6. CONCLUSION

The main contribution of this research is that the reader and writer problem is implemented using AOP and the execution time of AOP is comparison with that of OOP. Aspect-Oriented Programming (AOP) is a paradigm proposal that retains the advantages of OOP [4, 7] ..The result shows that AOP is the supplement of OOP. AOP can obtain the separation of concerns and make the function parts more reusable and functional cohesion much better without losing efficiency. Our work will benefit to the development and maintenance of the software that related to the synchronization.

With OOP	**With AOP**
Enter Number of Readers :2	Enter Number of Readers :2
Enter Number of Writers :2	Enter Number of Writers :2
Reader 0 starts reading.	Reader 0 starts reading.
Reader 1 starts reading.	Reader 1 starts reading.
Reader 1 stops reading.	Reader 0 stops reading.
Time Taken By Reader : 5000	Time Taken By Reader : 5000
Reader 0 stops reading.	Reader 1 stops reading.
Time Taken By Reader : 5002	Time Taken By Reader : 5000
Writer 0 starts writing.	Writer 0 starts writing.
Writer 0 stops writing.	Writer 0 stops writing.
Time Taken By Writer : 5001	Time Taken By Writer : 5000
Writer 1 starts writing.	Writer 1 starts writing.
Writer 1 stops writing.	Writer 1 stops writing.
Time Taken By Writer : 5002	Time Taken By Writer : 5001
Writer 0 starts writing.	Reader 1 starts reading.
Writer 0 stops writing.	Reader 0 starts reading.
Time Taken By Writer : 5001	Reader 1 stops reading.
Writer 1 starts writing.	Time Taken By Reader : 5000
Writer 1 stops writing.	Reader 0 stops reading.
Time Taken By Writer : 5000	Time Taken By Reader : 5000 Writer 1 starts writing.
Reader 0 starts reading.	Writer 1 stops writing.
Reader 1 starts reading.	Time Taken By Writer : 5000
Reader 0 stops reading.	Writer 0 starts writing.
Time Taken By Reader : 5002	Writer 0 stops writing.
Reader 1 stops reading.	Time Taken By Writer : 5001
Time Taken By Reader : 5000	Writer 1 starts writing.
Writer 1 starts writing.	Writer 1 stops writing.
Writer 1 stops writing.	Time Taken By Writer : 5000
Time Taken By Writer : 5000	

7. REFERENCES

[1] Charles Zhang, "FlexSync: An aspect oriented approach to Java synchronization", 31st International Conference on Software Engineering, Vancouver, Canada, May, 2009.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.Lopes, J.-M.Loingtier, J. Irwin. "Aspect-Oriented Programming", in Proceedings of the 11th European Conference on Object-Oriented Programming, Finland,Springer-Verlag, 1997, pp. 220-242.

[3] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G.Grisworld. "Getting started with AspectJ". Communications of the ACM, 2001, Vol.44, No.10, pp59-65.

[4] Kiczales G., J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors. Proceedings of the 11th European Conference on Object-Oriented Programming, number 1241 in Lecture Notes in computer Science, pp.220-242, Finland, June 9-13 1997.ECCOP'97, Springer Verlag, Berlin.

[5] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C.,Lopes, C., Loingtier, J.-M. and Irwin, J. (1997) Aspectoriented programming. In ECOOP'97, Jyv¨askyl¨a, Finland,June. Lecture Notes in Computer Science, 1241, 220–242.Springer, Berlin.

[6] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm,J. and Griswold, W. G. (2001) An overview of AspectJ.In ECOOP 2001, Budapest, Hungary, June. Lecture Notes in Computer Science, 2072, 327–353. Springer, Berlin. http://aspectj.org.

[7] Lopes C., B. Tekinerdogan, W. de Meuter, and G. Kiczales. Aspect-Oriented Programming. In M. Aksit and S.Matsuoka, editors, Proceedings of the 12th European Conference on Object-Oriented Programming EC-COP'98, Springer Verlag, 1998.

[8] Yang Zhang, Jingjun Zhang and Dongwen Zhang. "Implementing and Testing Producer-Consumer Problem Using Aspect-Oriented Programming." 2009 Fifth International Conference on Information Assurance and Security.