Implementing and Testing Performance of Readers-Writers Problems using Aspect Oriented Programming

Avinash Chandra Pandey, Dr.S.C.Chand, Deep Shikha Shukla

Abstract— Readers-Writers problem is a classical synchronization problemin the field of computer science.It can easily be implemented using any object oriented language.However, the implementationof 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 cross cutting expressions that encapsulate each concern in one place. With this there is there is minimal or no codes scattering and tangling.This paper aims to resolve concrete aspect and implement the synchronizationof readers-writers problem based on AOP. The execution time of AOPand 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.

Index Terms— Aspect Oriented Programming(AOP),AJDT, Code tangling,Eclipse,Lock,Object Oriented Programming, Readers-Writers Problem,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[1] 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[3] using AOP. Though much work has been done over aspect oriented methodology, there is less work on the readers-writers problems using AOP. As the readers-writers problem is a representative problem in synchronization, the solution will help in various areas where synchronization is required.

2 Opp and AOP

2.1 Object Oriented Programming Solution:

Many object-oriented programming languages have supported the synchronization and can implement the readers-writers problem. For example, java programming implements the synchronization through the 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 notify. All three methods can also 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.
- notifyALL( ): Wakes up all the threads that called wait( ) on the same object.The highest priority thread will run first.

2.2 Problem with the Object Oriented 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 considered 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

- *Avinash Chandra Pandey is currently working as Assistant Professor in CS department of Sri Ramswaroop Memoril University,Lucknow,India, E-mail: avish.nsit@gmail.com*
- *Dr.S.Chand is currently working as Professor in CS department of NetajiSubhas Institute of Technology,New Delhi, E-mailsatish@nsit.ac.in.*
- *Deep Shikha Shukla is currently working as Assistant Professor in EC department of Sri Ramswaroop Memorial University,Lucknow,India, E-mail:shukla_deepshikha@yahoo.com*

not beneficial for the development and maintainence of the software.

3 Aspect Oriented Programming: A Better Solution

Aspect- Oriented Programming(AOP) was first proposed 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 cross cutting concerns.The cross cutting concerns include security, logging, exception handling and synchronization etc.Many distinguish work has been done to deal with the dis-crete aspect. Aspects can contain several entities unavailable to standard classes. These are:

3.1 Inter-type Declaration:

Allow to add methods, fields etc to existing classes from with-in the aspect. This example adds an accept Visitor method to the Point class:

```
aspectVisitAspect
  {
voidPoint.acceptVisitor(Visitor v)
{v.visit(this); }
  }
```

3.2 Pointcuts:

Allow to specify join points which are well defined moments in the execution of a program, like method call, object instan-tination, 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’:

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

3.3 Advice:

Allows to specify code to run at a join point.The action can be performed *before, after,* or *around* the specified join point.eg:

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

4 Description of Reader-Writer’s Problem

In computer science, the first and second readers-writers prob-lems are examples of a common computing problem in con-currency. 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 con-straint that no process may access the share for reading or writing to it(in particular, it is allowed for two or more readers to access the share at the same time). A readers-writers 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 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 the mutex, in which case clearly no thread can access the data at the same time as another writer. However, this solution is sub-optimal, because it is possible that a reader R_1 might have the lock, and then another reader R2 request access. It would be foolish for R2 to wait until R1 was done before starting its own read operation; instead, R2 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.

4.2 Second Reader-Writer’s Problem:

Suppose we have a shared memory area protected by mutex, as above.This solutional is sub-optimal, because it is possible that a reader R_1 might have the lock, a writer W would 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 would 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.

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 environ-ment 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 sepa-rately 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 1. As shown in Table 1, the execution time of AOP is very close to that of OOP and almost better than 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 imple-mented program was bettering off as compared to the OOP implemented program.

6 Conclusion

The main contribution of this paper is that the reader and writer problem is implemented using AOP and the execution time of AOP is compared with that of OOP. The result shows that AOP is the supplement of OOP.AOP can obtain the sepa-ration of concerns and make the function parts more reusable and functional cohesion much better without losing efficiency.

TABLE 1
COMPARISON OF OOP AND AOP EXECUTION TIME

With OOP	With AOP
Please Enter integer number=2	Please Enter integer number=2
Please enter readers number=2	Please enter readers number=2
Please Enter writers number=2	Please Enter writers number=2
Reader 0 starts reading 2	Reader 1 starts reading 2
Reader 0 stops reading	Reader 1 stops reading
Time taken by Reader:688	Time taken by Reader:197
Writer 1 starts writing.2	Writer 1 starts writing.2
Writer 1 stops writing	Writer 1 stops writing
Time taken by Writer:663	Time taken by Writer:510
Reader 1 starts reading 3	Writer 1 starts writing.3
Reder 1 stops reading	Writer 1 stops writing
Time taken by Reader:947	Time taken by Writer:1402
Writer 0 starts writing.3	Reader 1 starts reading.4
Writer 0 stops writing	Reader 1 stops reading
Time taken by Writer:229	Time taken by Reader:603
Writer 0 starts writing.4	Reader 0 starts reading.4
Writer 0 stops writing	Reader 0 stops reading
Time taken by writer:2933	Time taken by Reader:318

REFERENCES

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

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

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

[4] G.Kiczales, E.Hilsdale, J.Hugunin, M.Kersten, J.Palm, W.G.Grisworld," Getting started with Aspect J",Communications of the ACM, 2001, Vol 44,No.10,pp 59-65