Towards Reusable and Modular Aspect-Oriented Concurrency Control

Sérgio Soares
Software Productivity Group
Computing Systems Department
Pernambuco State University
Recife, Pernambuco, Brazil
sergio@dsc.upe.br

Paulo Borba
Software Productivity Group
Informatics Center
Federal University of Pernambuco
Recife, Pernambuco, Brazil
phmb@cin.ufpe.br

ABSTRACT

Information systems based on the World Wide Web increased the impact of concurrent programs. Such increase demands the definition of methods for obtaining safe and efficient implementations of concurrent programs, since the complexity of implementation and tests in concurrent environments is bigger than in sequential environments. This work defined guidelines to restructure object-oriented software in order to modularize concurrency control using aspect-oriented programming. Those guidelines are supported by a concurrency control implementation that guarantees system correctness without redundant concurrency control, both increasing performance and guaranteeing safety. We define abstract aspects that constitute a simple aspect framework that can be reused to implement concurrency control in other applications. The achieved modularization makes the concurrency control easy to evolve and decreases the complexity of other parts of the software, such as business and data management modules, by decoupling concurrency control code from them.

Categories and Subject Descriptors

D.1 [**Software**]: Programming Techniques—Aspect-Oriented Programming; D.2 [**Software**]: Software Engineering; D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*Object-oriented programming*; D.3.2 [**Programming Languages**]: Language Classifications—AspectJ

General Terms

Languages, Standardization, Experimentation

Keywords

Aspect-oriented programming, framework, AspectJ, concurrency control, web information systems, Implementation approaches and techniques

SAC'07 March 11-15, 2007, Seoul, Korea

1. INTRODUCTION

The advent of information systems based on the World Wide Web increased the impact of concurrent programs. In fact, concurrent environments increase the complexity of the implementation and test activities, since subtle implementation errors lead to incorrect program executions that may be difficult to detect. This scene indicates that we need more adequate ways to implement concurrent programs, such as using a method definition [6] to support concurrency control, avoiding controls based on the programmer intuition. Such intuitive controls may have a negative impact on efficiency and may not guarantee system safety in concurrent environments. Intuition based controls also may add new race conditions that lead to invalid executions in concurrent environments. Besides guaranteeing safety, the method's definition documents and standardizes the control to be applied, favoring system extensibility, improving its maintainability.

We used AspectJ [3], a general purpose aspect-oriented [1] extension to Java, to evolve a previous work [6] on object-oriented concurrency control to derive an aspect-oriented concurrency control. This aspect-oriented concurrency control is tailored to a specific software architecture. Despite being specific, this software architecture can be used to implement several kinds of software, as addressed in the next section. The defined aspects complement the concurrency control implementation method for object-oriented software by modularizing concurrency control, which was not possible to achieve without an aspect-oriented approach. In fact, concurrency control is a good example of crosscutting concern that is hard to modularize using object-oriented programming and design patterns [2]. Moreover, some generated guidelines and aspects are also a complement to another work [7] that shows how to implement some persistence and distribution concerns using aspect-oriented programming. The composition of these works derived an aspect-oriented implementation method that deals with persistence, distribution, and concurrency control [5].

The contributions of this work are:

- A real example of crosscutting concern hard to modularize in with object-oriented programming, and therefore, is spread over several modules of a software;
- An aspect-oriented version of a concurrency control method that is modular, guarantees software safety, and is defined in a way that it can be precisely applied to an application that complies with an specific software architecture;

- Guidelines that allow restructuring object-oriented software to use the aspect-oriented concurrency control;
- An aspect framework that can be reused to implement concurrency control in other applications. The framework implements three different concurrency control approaches that are more adequate to different situations.

2. FROM OBJECT-ORIENTED TO ASPECT-ORIENTED CONCURRENCY CONTROL

In this section, we briefly present steps to restructure an object-oriented software by removing the concurrency control code that is tangled and spread over software units and implementing aspects to modularize this concurrency control. The-se steps are the guidelines for restructuring the object-oriented concurrency control. In fact, by ignoring these steps to remove concurrency control from the object-oriented software, the restructuring guidelines are simplified to implementation guidelines to be used during aspect-oriented software development. More details can be found elsewhere [5]

1. Identifying used concurrency control code — examples are: `synchronized` Java modify [4], `synchronized` Java block [4], calls to `wait`, `notify`, and `notifyAll` methods [4], use of the Concurrency Manager design pattern [6], and use of a timestamp technique [6]. Typical information system implementing four layers architecture would implement concurrency control in the following places: Facade class has to implement transactions if in persistence environment, Business classes might have some synchronized methods, Data storage classes might have synchronized methods and some additional SQL commands for implementing specific approaches, and some entity classes might have new fields and methods to implement the timestamp technique.
2. Removing the concurrency control code — After identifying where the concurrency control code is, we must remove it and document where and which kind of concurrency control the software has in order to implement it later with aspects;
3. Implementing concurrency control aspects — This step uses our defined framework [5] to implement concurrency control. The framework has abstract aspects that allow aspect reuse. There are aspects to synchronize join points similar to methods synchronization and to implement timestamp techniques. Both have optimistic and pessimistic approaches for performance improvements.

3. CONCLUSION

Concurrent environments have a great complexity inserted by their non-determinism that can turn the system to an inconsistent state in abnormal interference. By using aspect-oriented programming we can separate concurrency control from other concerns, such as business rules, data management and user interface. This separation makes easier to change concurrency control policies, since concurrency control code is not tangled with other concerns code. Therefore, besides making programs more modular, this separation also decreases their complexity, since there is no concurrency control to reason about when implementing business, data management and user interface.

Some defined aspects are abstract and constitute a simple aspect framework that can be used to implement concurrency control in other applications. Those reusable aspects are quite simple and concise if compared with the object-oriented solution, but are very useful, since they implement the most used concurrency controls, including the Java concurrency control approach. The others aspects are application specific but different implementations might follow the same aspect pattern, which allows automatic aspects generation. We are currently working into a code generation tool in order to support a more general implementation method that considers, besides concurrency control, data management and distribution aspects [5].

We defined guidelines to restructure an object-oriented to an aspect-oriented concurrency control. Despite being originally defined to restructuring existing object-oriented software, the guidelines can also be used during aspect-oriented software development. Those guidelines are also tailored to the specific software architecture, which allows more precise definitions, also giving better support to programmers. Although specific, the software architecture can be used to implement several kinds and families of software.

The aspect-oriented restructuring/development guidelines are complementary to a concurrency control implementation method [6]. This method is also tailored to the same specific software architecture and provides guidelines to implement concurrency controls in object-oriented software. In fact, the restructuring guidelines provide an aspect-oriented implementation to the concurrency controls defined by the implementation method and use the method analyses to identify which and where each concurrency control mechanism is to be used.

4. REFERENCES

[1] Tzilla Elrad, Robert Filman, and Atef Bader. Aspect–Oriented Programming. *Communications of the ACM*, 44(10):29–32, October 2001.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[3] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. Getting Started with AspectJ. *Communications of the ACM*, 44(10):59–65, October 2001.

[4] Doug Lea. *Concurrent Programming in Java*. Addison–Wesley, second edition, 1999.

[5] Sérgio Soares. *An Aspect-Oriented Implementation Method*. PhD thesis, Informatics Center, Federal University of Pernambuco — CIn-UFPE — Brazil, October 2004.

[6] Sérgio Soares and Paulo Borba. Concurrency Control with Java and Relational Databases. In *Proceedings of 26th Annual International Computer Software and Applications Conference*, pages 834–849, Oxford, England, August 2002. IEEE Computer Society Press.

[7] Sérgio Soares, Paulo Borba, and Eduardo Laureano. Distribution and Persistence as Aspects. *Software: Practice & Experience*, 36(7):711–759, 2006. John Wiley & Sons.