Reengineering for Separation of Concerns

Elizabeth A. Kendall
Sun Microsystems Chair of Network Computing
Monash University
School of Network Computing
McMahons Rd., Frankston, VIC 3199
AUSTRALIA
email: kendall@infotech.monash.edu.au

1. INTRODUCTION

Our investigation of aspect- oriented programming has involved determining the extent that AOP can be used to improve software development and maintenance, along the lines discussed by Bayer in [3]. We have found that AOP can be used to reduce code complexity and tangling; it also increases modularity and reuse.

In this case study, existing object- oriented designs for role models are used as the starting point, and they are reengineered with aspect- oriented techniques. We therefore concentrate on *reengineering*, complementing the research in *engineering* reported in [3]. However, the benefits that we achieved through reengineering should provide additional motivation for the use of separation of concerns during software development.

2. ORIGINAL OBJECT- ORIENTED DESIGN

The role models discussed here involve five FIPA (Foundation for Intelligent Physical Agents) protocols [5]. In the protocols, the agents can request and receive services in five different ways. The first two protocols involve requesting and receiving i) services and ii) information. In the other three protocols, an agent negotiates for services with multiple potential suppliers via iii) a contract net, iv) an iterated contract net, or v) an auction.

The State pattern [7] version of the Role Object pattern [2] has been employed in the object- oriented design because the protocols must follow a finite state machine. Once an initial role is assigned to an object, further roles must progress according to the state transitions found in the protocols. As many other FIPA protocols must be supported, the Owner- Driven Transitions pattern [6] was used to place state transitions in the AgentCore or owner class. As in any finite state machine, exception handling is very important.

The original design features an AgentCore class, a FIPA Role superclass, and 24 subclasses for the protocol roles or states. There are approximately 115 methods.

3. CODE TANGLING

The role objects and the role transitions were not considered in this study (see [12]). Beyond the behavior of the role models, the 115 original methods addressed six separations of concern (SOC). They are listed below with the number of methods involved.

SOC 1. **Interagent communication**. Sending messages to another agent. *17 methods.*

SOC 2. **Exception handling.** Incorrect messages or sequences. *44 methods*

SOC 3. **Failed conversations**. A conversation has ended due to failure. *12 methods*

SOC 4. **Successful conversations**. A conversation has ended. *4 methods*

SOC 5. **Negotiation strategies**. Behavior involved in competitive bidding. *8 methods.*

SOC 6. **Iterative protocols**. Auctions and iterated contract nets. *6 methods.*

Some of the code tangling is depicted in Figure 1. The numbers on the figure correspond to the number for the separation of concern. Only 10 of the subclasses of FIPA Role and a subset of the methods are shown. Figure 1 is a pictorial view of the features that we were able to identify and select in this case study. We would be interested in utilising the Feature Selection tool discussed in [15].

4. OBJECT- ORIENTED TECHNIQUES

Object- oriented techniques can alleviate some of the code tangling; behavior can be promoted to a superclass or delegated to a component. Promoted behavior decreases cohesion. Delegated behavior adds additional components, and it can lead to object schizophrenia. Further, interface maintenance is complicated unless the contained object and its interface are publicly accessible.

Two of the separations of concern were promoted to the FIPA Role superclass: SOC 3 (Failed Conversations) and SOC 4 (Successful Conversations). These two types of behavior were encoded in two new methods to the FIPA Role superclass. This was acceptable because all failed and successful FIPA conversations can be concluded in the same way.

Interagent communication was addressed via delegation. In the simplest design, the interagent communication was

delegated by the role object back to the Agent. This normally would require that the interface for sending messages be duplicated in the Agent object, adding 17 methods to the overall design. (They were protected methods of the FIPA Role class in the original design.) However, reflection was used to provide general purpose transmission or sending. With reflection, only one new method had to be added for transmission and one for reception. However, additional runtime overheads were incurred.

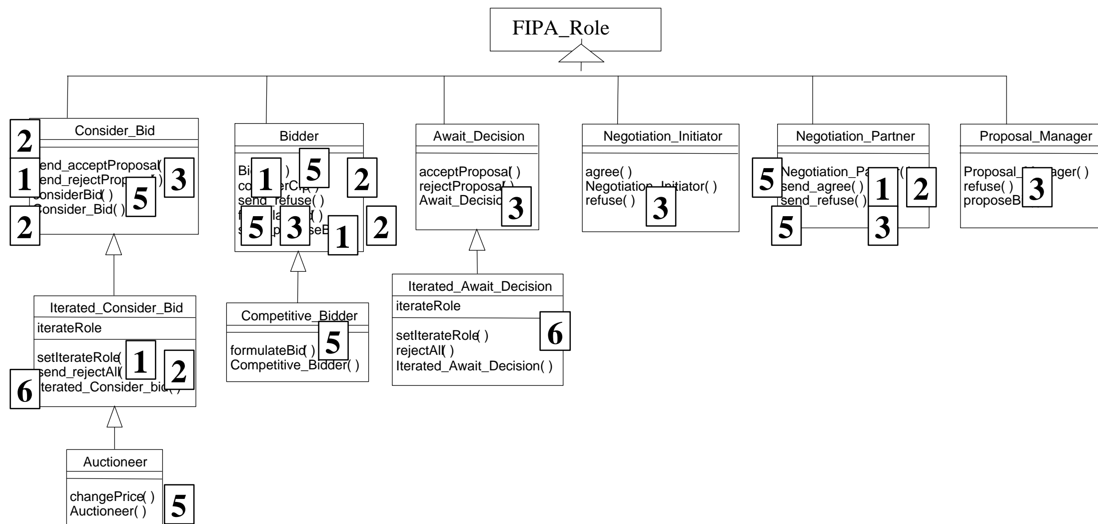

Figure 1: Indication of the Code Tangling found in the Original Design

5. ASPECT- ORIENTED TECHNIQUES FOR EXCEPTION HANDLING

Exception handling is an important separation of concern for this application as invalid transmissions and receptions have to be caught. Exceptions have to be thrown locally, at the methods where the errors occur. The behavior can not be promoted to a superclass, and it is not beneficial to delegate the behavior to another component.

The superclass FIPA Role defines the default behavior for all 28 messages in the protocols, throwing an exception if the message is invalid. Each method that sends a message to another agent (SOC 1 - 17 methods) also incorporates exception handling.

The exception handling is redundant, but it can not be delegated or promoted. AOP is therefore appropriate. Static aspects are employed because the behavior is required for all instances of the FIPA Role subclasses. Aspect Invalid State Message introduces the interface and the behavior to the FIPA Role class to throw the correct exception. Aspect SendCatcher uses advise weaves to add behavior to existing methods in some of the subclasses of FIPA Role.

6. ASPECT- ORIENTED PROGRAMMING FOR OTHER CONCERNS

Separation of concern 5 involves different strategies for competitive bidding. For example, an agent may be negotiating for services with a monopoly; on the basis of a fixed price contract; or in a joint venture. Separation of concern 6 involves restarting or resetting a protocol that is iterative. For example, an auction protocol is carried out repeatedly, but the bidding does not start from scratch. A contract net can also be iterated.

These two separations of concern are classic "mix-ins". They can not be promoted to the superclass because all of the possibilities will occur. Because only single inheritance is allowed in Java, they would lead to duplicate and redundant hierarchies. That is, two subclasses would be required for each state in the contract net protocol: one for the non-iterative and one for the iterative version.

Additionally, delegation is not an attractive option because it just adds components and indirection. As in the case of interagent communication, interfaces would have to be duplicated. That is, the interface for each of the eight methods that deal with negotiation strategies would have to be duplicated in the strategy components.

Figure 2 depicts the aspect- oriented solution. Aspect instances are utilized (as indicated by diamonds in Figure 2), and an aspect instance is required for each negotiation strategy. In the figure, aspect instances are shown for fixed price contract, monopoly, and joint venture. With aspect instances, strategies are only added when needed; each of these aspects adds or advises behavior to the required methods. Seven of these methods are shown in the figure, along with a strand that connects them to the Fixed Price Contract aspect instance.

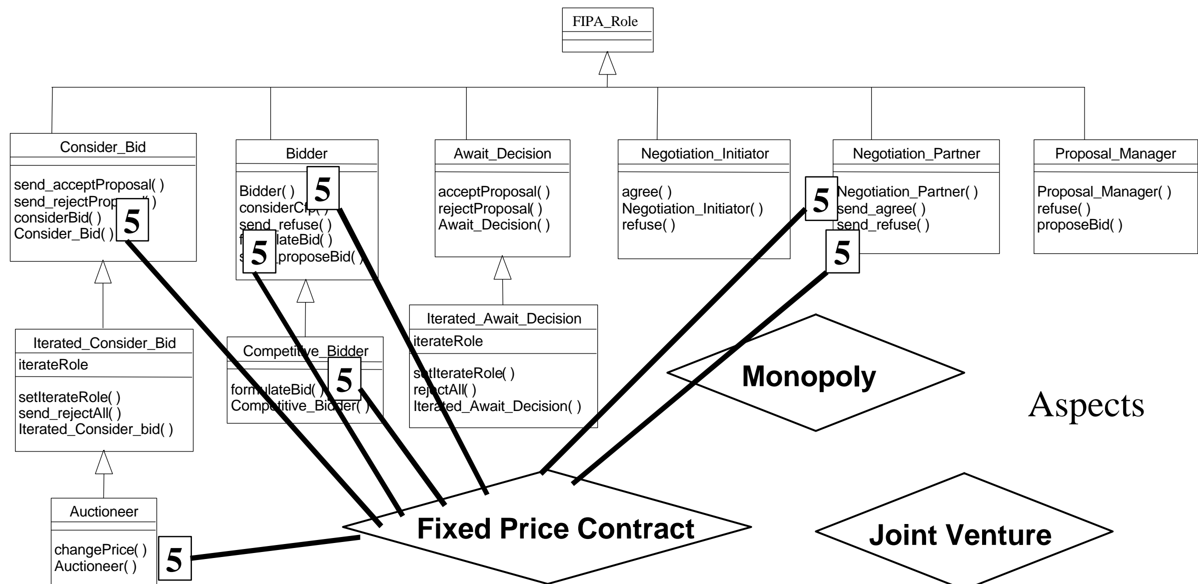

Figure 2: Negotiation Strategy Aspects

7. CODE TANGLING SUMMARY

The use of AOP in this application reduced the overall module (class and method) and lines of code (LOC). In exception handling, the Invalid State Message aspect with one introduce weave (listing all relevant methods) replaced the 28 methods, reducing the module count by 26. The Send Catcher aspect replaced 51 (17 * 3) lines of code with five, saving 46 LOC. The Iterative protocol aspect replaced six methods with one aspect and one weave. Lastly, the Negotiation Strategies aspect reduced the code for strategies from 40 * 3 (8 methods * 5 SLOC * 3 strategies) to 21 (7 * 3). These results are summarized in Figure 3.

Aspect	Method Reduction	LOC Reduction (source)
Invalid State Message	- 26	
Send Catcher		- 46
Iterative Protocol	- 4	
Negotiation Strategies		- 99
Total AOP	**- 30**	**- 145**

Figure 3: Impact of Aspect- oriented Programming on the Application

8. DISCUSSION: FINDINGS AND LINKS TO OTHER WORKSHOP PAPERS

Although our findings are preliminary, it appears that AOP is a promising approach to modelling, representing, and integrating separations of concern, reducing module count and lines of code (pre- woven) for cross- cutting behavior

Many other questions remain, including AspectJ™ constructs for various aspect- oriented designs and implementations, and interfaces between aspects and objects. Additional work is also required in appropriate metrics for comparisons between aspect- oriented and object- oriented designs and implementations. Lines of code and module count are only two limited metrics.

The small case study reported here did not lead to any conflicts or overlapping concerns, as reported in [16]. However, we have encountered these in our other work [11], and we agree with M. Monga's view that linguistic constructs for composing concerns must be designed to reduce clashes.

Another issue is that during our reengineering we employed both traditional object-oriented design techniques, such as delegation and promotion, and separation of concerns. We have not yet had the opportunity to utilise the Hyper/J tool, but we feel it would be beneficial if it facilitated a hybrid approach. That is, two other case study papers [4, 15] report on experiences with Hyper/J. It doesn't seem that Hyper/J facilitates both object-oriented design and separation of concerns.

REFERENCES

1. Batury, D., “Refinements and Separation of Concerns,” Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000), June, 2000.
2. Baumer, D., D. Riehle, W. Siberski, M. Wolf, "Role Object," *Proceedings of the 4th Annual Conference on the Pattern Languages of Programs*, Monticello, Illinois, USA, September 2-5, 1997.
3. Bayer, J., “Towards Engineering Product Lines Using Concerns,” Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000), June, 2000.
4. Carver, L., “A Practical Hyperspace Application: Lessons from the Option-Processing Task,” Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000), June, 2000.
5. Dickinson, I., “Agent Standards”, Agent Technology Group, 1997. http://drogo.cselt.stet.it/fipa.
6. Dyson, P., B. Anderson, "State Patterns," in *Pattern Languages of Program Design 3*, R. Martin, D. Riehle, F. Buschmann, Ed., Addison Wesley, 1998.
7. Gamma, E.R., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.
8. Harrison, W., H. Osher, "Subject- Oriented Programming (a critique of pure objects)," in *Proceedings of the Conference on Object- oriented Programming: Systems, Languages, and Applications*, Washington, D. C. September, 1993. pp. 411 - 428.
9. IBM Research: Subject- oriented Programming Group, "Subject-oriented Programming and Design Patterns," http://www.ibm.research/sop
10. Kaplan, M., Harold Ossher, William Harrison, Vincent Kruskal, Subject-Oriented Design and the Watson Subject Compiler, Position paper for OOPSLA'96 Subjectivity Workshop, October, 1996
11. Kendall, E. A., " Role Model Designs and Implementations with Aspect Oriented Programming," Proceedings of the International Conference on Object Oriented Programming, Languages, and Applications (OOPSLA), Denver, November, 1999.
12. Kendall, E. A., "Agent Roles and Role Models: New Abstractions for Multiagent System Analysis and Design," *International Workshop on Intelligent Agents in Information and Process Management*, Germany, September, 1998
13. Kiczales, G., C. Lopes, "Aspect-Oriented Programming w/AspectJ™, " Tutorial and Primer, Xerox PARC, www.parc.xerox.com/spl/projects/aop/
14. Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. - M. Loingtier, and J. Irwin, "Aspect- oriented Programming," Xerox Corporation, 1997. www.parc.xerox.com/spl/projects/aop/
15. Lai, A., G. C. Murphy, R. J. Walker, “Separating Concerns with Hyper/J: An Experience Report,” Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000), June, 2000.
16. Monga, M., “Concern Specific Aspect-Oriented Programming with Malaj,” Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000), June, 2000.
17. Ossher, H., Matthew Kaplan, William Harrison, Alexander Katz and Vincent Kruskal, "Subject-Oriented Composition Rules," *Proceedings of 1995 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1995
18. Ossher, H., M. Kaplan, A. Katz, W. Harrison, V. Kruskal, "Specifying Subject- Oriented Composition," *Theory and Practice of Object Systems (TAPOS)*, Vol 2, No 3, 1996.
19. Tarr, P., Ossher, H., *Hyper/J User and Installation Manual*, IBM Research, 2000.