

Illustrations of Crosscutting

Cristina Lopes,[†] Erik Hilsdale,[‡] Jim Hugunin,[†] Mik Kersten,[†] Gregor Kiczales[‡]

Introduction

Crosscutting poses some challenges in terms of clean separation of concerns. Aspect-oriented programming [2] is intended to address those challenges. This position paper presents a couple of examples of situations that involve crosscutting. The first example illustrates some basic crosscutting that all technologies for advanced separation of concerns should be able to handle gracefully. We show how this example is handled using AspectJTM 0.6 [1]. The second example shows a more complex crosscutting situation, and we present how it might be handled in a future version of AspectJ.

Example 1: An Asymmetric Multi-Object Protocol

Description of the Situation

Consider a telecom application in which the objects are customers, calls and connections. The class diagram and basic operations of a simplified version of this application are shown in Figure 1.

One concern of such a system is timing. The timing feature can be described as follows:

1. It is related to a connection and to a customer, the caller of that connection.
2. When the connection is established, a timer is started.
3. When the connection drops, that same timer is stopped and the time is accumulated in the caller.
4. The accumulated time in the caller Customer can be read/used by other parts of the system.

Using basic OOP to program this feature, we need to add a number of elements to the diagram in Figure 1. The increment in the resulting diagram is shown in Figure 2.

The timing feature is a good example of crosscutting. As Figure 2 shows, this feature spans two classes, and involves state (the timer and the accumulated time in Customer), behavior (the timing-related operations in Customer and in Timer) and protocol (the interaction among the objects).

The AOP community is well aware of the consequences that crosscutting has in programs. One of those consequences is code tangling. In Java, the

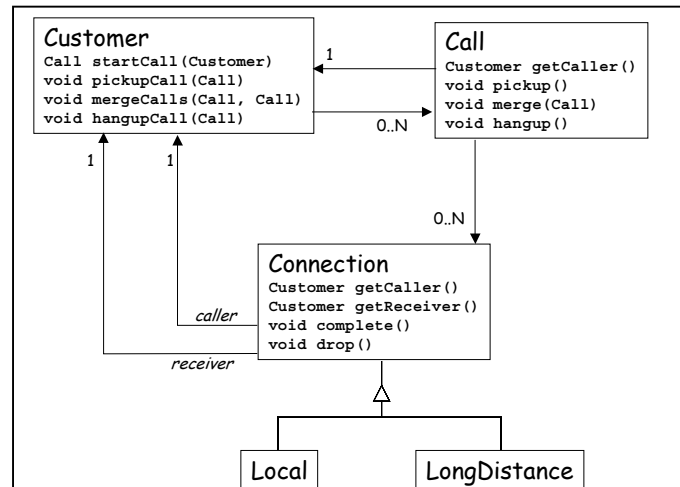


Figure 1. Basic class diagram for Example 1.

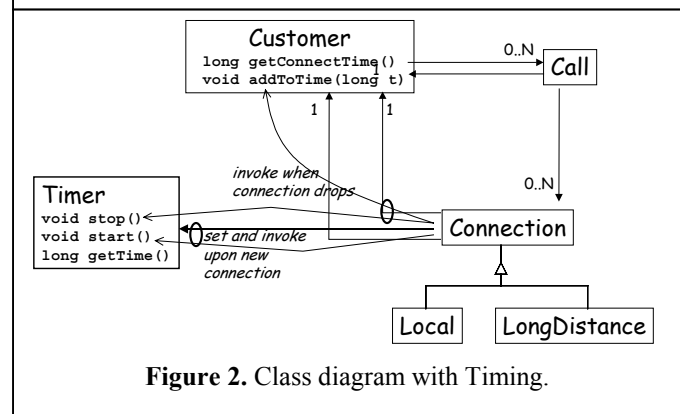


Figure 2. Class diagram with Timing.

[†] Xerox PARC (lastname@parc.xerox.com).

[‡] Indiana University (eh@acm.org)

[‡] University of British Columbia (gregor@cs.ubc.ca)

implementation of a method such as the `drop()`, of `Connection`, will look like this:

```
void drop() {
    ...implementation of drop...
    timer.stop();
    caller.addToConnectTime(timer.getTime());
}
```

Code tangling manifests itself in many different ways. In this example, it shows in lines of code inside the methods (as illustrated above), in the declaration of variables and methods in `Customer`, and in the declaration of the class `Timer`.

Using AspectJ

Using AspectJ 0.6, we can define an aspect that encapsulates the Timing feature:

```
abstract class Timing {

    introduction Connection { Timer timer = new Timer(); }

    introduction Customer { long totalConnectTime = 0; }

    public static long getCustomerTotalConnectTime(Customer c) {
        return c.totalConnectTime;
    }

    static advice (Connection c): c & void complete() {
        after {
            c.timer.start();
        }
    }

    static advice (Connection c): c & void drop(){
        after {
            c.timer.stop();
            c.getCaller().totalConnectTime += c.timer.getTime();
        }
    }
}
```

Example 2: Propagating Context

Description of the Situation

Consider a server application in which the server ends up delegating the requests to some other parts of the system, the workers, possibly through many levels of delegation. Figure 3 illustrates the situation.

One concern of such a system is the charge applied to the client for the actual work that is done. This feature requires the original client to be charged for each item of work that each worker does.

There are many possible designs for accommodating this feature. A typical solution involves the client explicitly passing itself to the server, and then the server passing the client down through all the levels to the workers. This typically involves adding an extra argument to each method along the way. Or we can decide not to pass the client and, instead, pass a request id that the workers use to notify the server

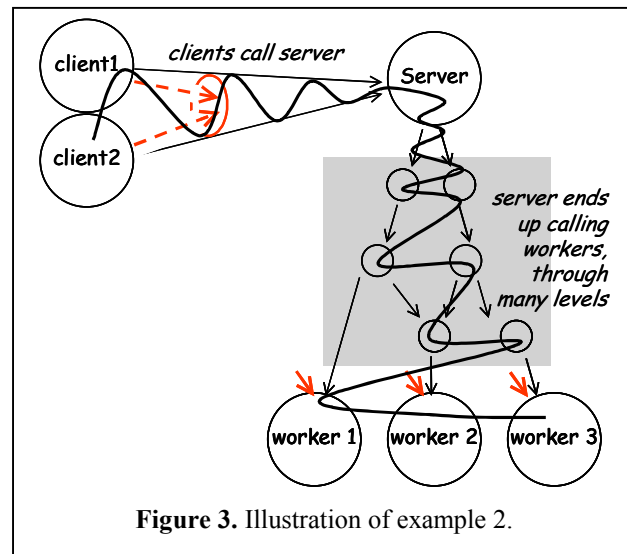


Figure 3. Illustration of example 2.

when they're done with the request, so that the server can compute the charge. In either case, we must also pass some object that accumulates the charge for each piece of work involved.

This ChargeBack feature is also a good example of crosscutting. It spans a large number of objects and operations in the system, and it involves behavior (doing the work) that happens in one context (the workers) but must communicate with another context (the original client).

Using AspectJ

What follows is a solution to encapsulating this context-passing situation in a modular way, without requiring the declaration of a Client argument in all the methods along the call graph. This solution uses an extension that is currently being considered for AspectJ. We expect AspectJ to be able to support this kind of crosscutting in time for the workshop.

```
class HandleChargebacks {

    crosscut entryPoints(): Server & (void doService1(Object) | void doService2());

    crosscut workPoints():
        (ServiceHelper1 & void doWorkItemA()) |
        (ServiceHelper2 & void doWorkItemB()) |
        (ServiceHelper3 & void doWorkItemC());

    // calls captures message sends, on the caller side
    crosscut invocations(Client c): c & calls(entryPoints());

    // cflow captures all the execution control that flows from the given crosscut
    crosscut clientWork(Client c): cflow(invocations(c)) & workPoints();

    static advice (Client c): clientWork(c) {
        before { c.chargeback(); }
    }
}
```

Contributions for the Workshop

The major goal of this paper is to describe the situations involving crosscutting concerns. By doing so, we hope to contribute to the clarification of what crosscutting is about, to demonstrate how it can be identified and to identify the requirements for a systematic approach towards improved support for separation of concerns.

We would also like to propose AspectJ as a technology to be used for solving these and other problems during the breakout session.

REFERENCES

1. AspectJ web site - <http://aspectj.org>
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 (ECOOP '97)*, Finland, Springer-Verlag, 1997, pp. 220-242.