Moving Code

Johan Fabry
Johan.Fabry@vub.ac.be*

Johan Brichau
Johan.Brichau@vub.ac.be†

Tom Mens
Tom.Mens@vub.ac.be

April 13, 2001

1 Introduction

In general, the goal of Advanced Separation of Concerns (ASoC) technologies is to provide encapsulations for all kinds of concerns, with a specific focus on cross-cutting concerns. An application is formed by composing these different concerns using some composition mechanism, specifically focused on concern integration and composition. In some cases, however, a specific concern needs to be decomposed in the resulting application. In this paper, we identify a need for decomposition in ASoC technologies, using a case dealing with code movement. We will present an example problem in which adding new code at specific join-points in the code does not suffice to implement a cross-cutting concern dealing with 'distribution'.

1.1 The case: a Library Application

Consider the following example: an application for libraries. This application will be used to keep track of the libraries' books: it contains a database of books and library card holders, and keeps track of which book is lent out to which person. There are two front-ends for the application: a 'public' front-end, which can be used by anyone to search through the books database, and an 'employee' front-end, which is used to register loans and to manage the databases of books and card holders.

The application runs on one (central) computer, but can be accessed remotely by redirection of the programs' in- and output which is done using X[1]. To allow for different front-ends the program is structured in two parts: a server which manages the database, and two different clients, one for each front-end. The clients communicate with the server using Inter-Process Call (IPC) facilities provided by the operating system. This architecture is illustrated in figure 1.

*Author funded by a doctoral grant of the Flemish Institute for the advancement of scientific-technological research in the industry (IWT)

†Research Assistant of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.)

[1] X allows for an application to be run on one computer, and its' full user interface to be redirected to a different machine. The application is completely unaware of this redirection.

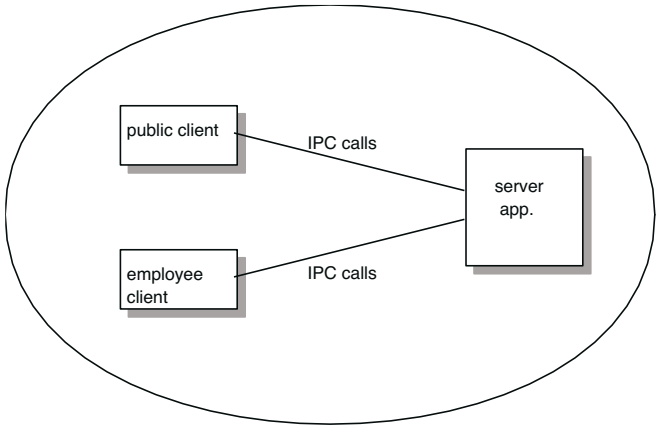

Figure 1: The Application's Architecture

1.2 The problem: Minimizing Network Traffic

This application is built and successfully deployed in different libraries, but soon a performance issue develops: the user-interface becomes progressively more sluggish whenever an extra user is online. This is traced to the relative inefficiency of X, combined with the low-quality network used in the libraries. When using X, all screen updates are sent over the network, which severely increases the network load due to the frequent transfers of a large volume of data.

The decision is made to change the application to a distributed system: every client will now run on the same machine as its user interface (which we will call the UI Client computer), while the database server remains on the central machine. The clients will now connect to the server over the network, using a Remote Procedure Call (RPC) mechanism instead of using IPC.

However, extra issues arise due to the distribution of the clients: not only are the UI Client computers not powerful enough to perform some of the clients' operations, but also some database operations will need to be performed on the database server to further lower the network load. This means that the clients will have to be changed in such a manner that a part of the clients' computation is performed on the UI Client computer, and that another part will have to be performed on the database server.

2 Tentative Solution Approaches

2.1 Rewriting the application

After determining which of the clients' operations are to be run where, the application is rewritten to the new specification. Both clients are effectively split in two: an UI Client and a Computation Client, glue code is written, using RPC to let the two halves talk to each other, using RPC to let the UI Client

communicate with the server, and using IPC to let the Computation Client work with the server. It is clear that this entails severely changing the code of both the server and the two clients by hand, and that these changes cross-cut the entire system, while the core functionality of the application effectively remains unchanged. Indeed, we are introducing a new distribution concern for which we need to change the entire application. What is even worse is that we effectively split a single encapsulated concern in two encapsulations because of a non-functional requirement. The architecture of the rewritten application is illustrated in figure 2.

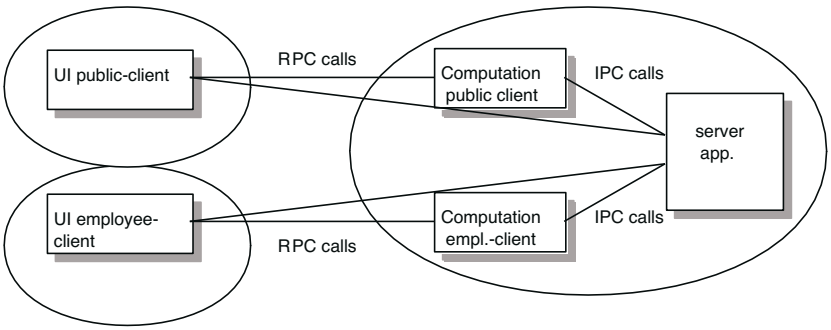

Figure 2: The Application's Architecture after the rewrite

2.2 An ASoC Solution

For the above reasons, AOP and related technologies are suggested as a possible technique to ease the burden of rewriting the application. Current techniques for separations of concerns can help reasonably well to separate the new code dealing with the RPC mechanism from the base functionality of the system. However, the supplementary requirement to move some client computations to the server raises some problems.

The implementation of this requirement using current ASoC tools does not provide as great a relief as expected. The best solution, using existing tools, is to write the two different clients as two different aspects. Each aspect introduces the code for the computation client into the server's application code. This approach is illustrated in figure 3.

It is clear that, while this solution provides some relief, there is a fundamental error somewhere. The programmers are misusing the aspect tools to achieve the required results: the two different clients are not aspects, they should remain base components. The real aspect here is *where the code should be run.* The programmmers should be able to specify separately, for both clients, that the code for the computation client should be run on the server. However, using current ASoC tools this is not possible, because these tools only allow for extra functionality to be added to the components, and do not allow existing components to be modified.

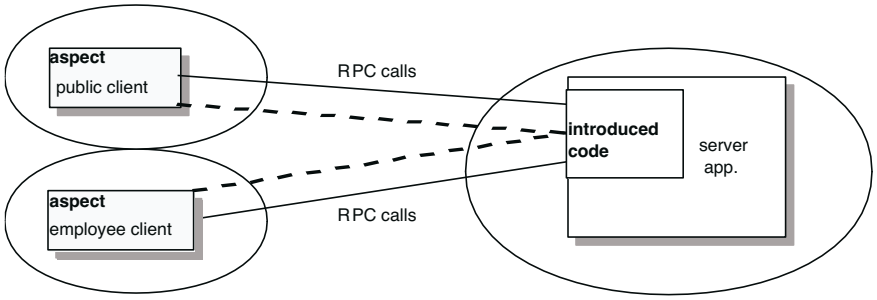

Figure 3: The Application's Architecture using an AspectJ aspect.

3 Moving Code

Imagine an ASoC technology that does provide support for a code moving aspect. We would be able to keep the code of both clients, without any modifications, and just define, in the code moving aspect, that some parts of the code should be taken out and placed somewhere else. Some proxy code will take care of redirecting calls to the moved code, but this is the responsibility of the aspect weaver.

Note that we are not proposing explicit meta-programming or refactoring abilities of aspect languages. Consider the 'introduction' possibility of the AspectJ language. It allows the aspect developer to introduce a new method in a certain class. We imagine a similar facility to 'move' a certain existing method from one class to another class. We envision that using ASoC technologies, we could implement an architecture as illustrated in figure 4.

The difference between the existing ASoC solution presented in the previous section and the approach using code movement is that in the latter case, the distribution concern can be separated from the concern implementing the functionality of the client.

4 Conclusion

Our position is that, in ASoC techologies, we do require more than the ability to compose different concerns (cross-cutting or not) together, effectively composing their functionalities. There is also the need to make changes to existing components or to 'decompose' concerns. In our example case, we need to be able to move code around because of a non-functional requirement. In this case, development would be signifantly eased if ASoC technologies would allow for some form of decomposition, which could be used to implement the concern of *where the code should be run.*

References

[asp] The aspectj language specification. Xerox Corporation, Palo Alto, http://www.aspectj.org.

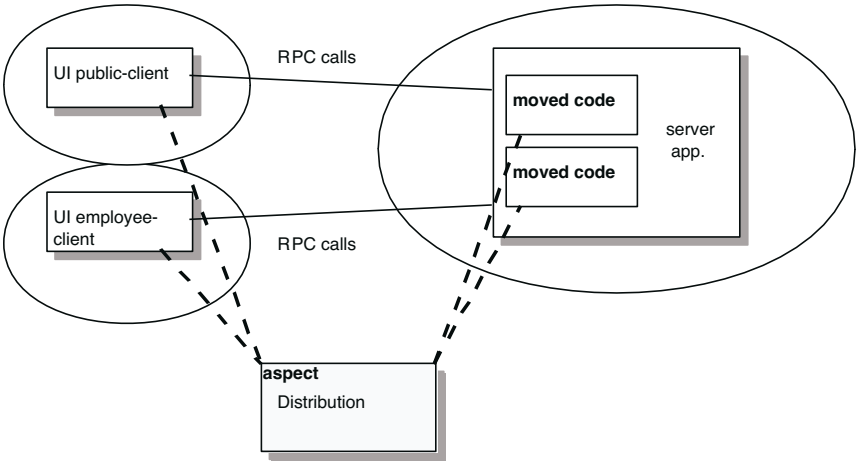

Figure 4: The Application's Architecture using an aspect with code movement abilities

[KLM+97] G. Kickzales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object Oriented Programming*. Springer Verlag, June 1997.

[OT00] H. Ossher and P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluwer, 2000. To appear.