Language-independent aspect weaving

Bram Adams

SEL, INTEC, Ghent University, Belgium
`bram.adams@ugent.be`

Abstract. Building an aspect weaver right into the heart of the GCC compiler not only improves efficiency of weaving process and woven code in the context of C programs. Exploiting GCC 4.0's new intermediate representation also enables us to advise a C application with code written in e.g. Fortran or Java. This way, crosscutting concerns can be expressed in the most suitable language for their purpose. Issues like language interaction, programming conventions as well as a concrete implementation still need to be looked at.

1 Introduction

Aspect Oriented Programming (AOP) is a recent programming paradigm, introducing the notion of crosscutting concerns. As more research is being done on Aspect Oriented Software Development (AOSD), the (initial) promise that the ideas involved were *not* limited to Object Oriented (OO) systems, is getting confirmed. That's why more and more non-OO aspect languages are being designed and implemented. This paper is about an effort called Aspicere, an aspect language for C.

2 Aspicere

Started as a side-project of Cobble [3] (an aspect language for Cobol), and originally based on AspectC [2], Aspicere[1] has now evolved into an aspect language with pointcut expressions written as Prolog predicates [1].

Just like the currently unmaintained AspectC, it features a static weaver, whose architecture is outlined in Fig. 1. As can be seen there, our current weaver acts as a source-to-source preprocessor prepended to the GCC compiler. So, there's quite some redundant work involved, because there are actually two parse phases, there's a lot of file IO, . . .

If we'd like to extend our work to C++ or even Java, we'd face two main problems: offering a suitable pointcut language for each particular language and finding an efficient way to effectively weave these aspects. The former could be handled by expressing frontend predicates containing the right abstractions for the intended language in terms of a common set of basic predicates. The latter issue would force us to invent new XML-representations (and corresponding parsers) for all the desired base languages or, much more challenging, to design one uniform formalism for all of them. And what about the efficiency of either the woven code and the weaver itself?

[1] http://allserv.ugent.be/∼kdschutt/aspicere/

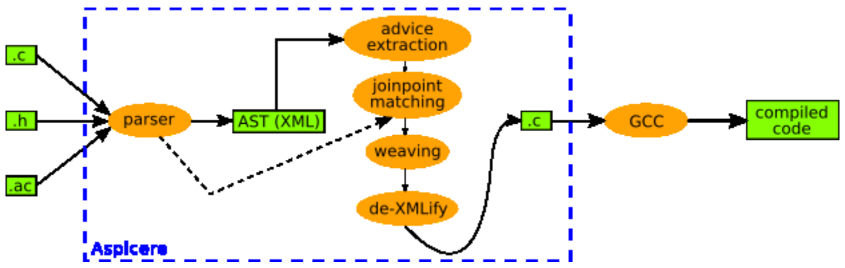

Fig. 1. Outline of our current weaver's internals.

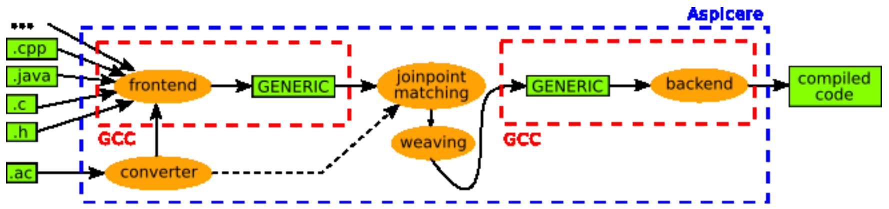

Fig. 2. Proposed architecture for aspect weaver built into GCC 4.0. The converter filters genuine source code from advice's pointcut expressions to avoid modifications to GCC's parsers.

3 Our proposition

3.1 GCC 4.0

The primary compiler on Unix/Linux systems is the GNU Compiler Collection (GCC), supporting a significant number of programming languages (C, C++, Objective-C, Fortran, Java, and Ada). Moreover, GCC has been ported to numerous computer platforms and architectures.

GCC 4.0 features two new intermediate representations: GENERIC and GIMPLE trees. Basically, each different language frontend produces a forest of GENERIC trees, which are then turned into GIMPLE for optimizations and eventually fed to the backends. As GENERIC is a higher-level representation containing control flow and all needed language constructs without language dependencies, it's in fact a uniform model for all of GCC's supported languages' ASTs.

3.2 Aspicere revisited

Combining Aspicere and GCC 4.0 as shown in Fig. 2, tackles section 2's problems. Indeed, there's only one parse pass left. Once GENERIC trees have been built for the base program and aspects, GCC's workflow is temporarily suspended to match joinpoints and advice using a Prolog engine and rules derived from the pointcut expressions. Then, the real weaving process starts by modifying the GENERIC trees accordingly and finally, GCC's remaining activities are resumed.

Section 2's second problem has also been solved, because GENERIC trees offer a uniform interface to ASTs of every supported programming language. As the weaver's semantics are expressed in terms of GENERIC trees, it only needs to be built once. What's more, there's the intriguing side-effect of "heterogeneous advice". From the beginning, most AOP-tools have opted to write advice in the same language as the base program. AOP's underlying ideas, however, do *not* dictate this. Implementing crosscutting concerns in the programming language best suited for them, could open up interesting opportunities. Of course, issues involved in (inter-paradigm) language interaction need to be treated in a proper way.

4 Conclusion and future work

GCC 4.0's GENERIC trees allow us to write "heterogeneous advice", leading to language-independent aspect weaving. Granted, a lot depends on the disciplined language-independence of the GENERIC trees. Likewise, further investigation is needed on semantically connecting language constructs in different programming languages. Restricting their interaction by using conventions is an option.

References

1. B. Adams and T. Tourwé. *Aspect Orientation for C: Express yourself.* 3rd Software-Engineering Properties of Languages and Aspect Technologies Workshop (SPLAT), AOSD 2005.
2. Y. Coady, G. Kiczales, M. Feeley and G. Smolyn. *Using AspectC to improve the modularity of path-specific customization in operating system code.* SIGSOFT Softw. Eng. Notes, 26(5):88-98, 2001.
3. R. Lämmel and K. De Schutter. *What does aspect-oriented programming mean to Cobol?* Proceedings of Aspect-Oriented Software Development, AOSD 2005.