Relaxing Type Restrictions of Around Advice in Aspect-Oriented Programming

Hidehiko Masuhara
Graduate School of Arts and Sciences
University of Tokyo
masuhar@acm.org

1 Advice Mechanism in AspectJ

Advice mechanism in aspect-oriented programming (AOP) languages is a powerful means of modifying behavior of a program without changing the text of the program. AspectJ[2] is one of the most popular AOP languages that support advice mechanism, and is widely used for modularizing crosscutting concerns, such as logging, profiling, security and optimization.

One of the unique features of the advice mechanism is *around advice*, which can change parameter and return values of join points (i.e., specific kinds of events during a program execution including method calls, constructor calls and field accesses). With around advice, it becomes possible to define aspects that directly affect values passed in the program such as caching results, pooling resources, sanitizing parameters and return values, and so forth. For simplicity, the rest of the abstract discusses only return values. It however applies to parameter values as well.

2 Restriction on Types of Around Advice

Though powerful and useful, the around advice mechanism in AspectJ has a restriction on types. It can be explained in the following way:

> When there is a join point whose static return type is T, AspectJ requires any around advice applied to the join point to have static return type less than or equals to T.

For example, evaluation of `new FileOutputStream(name)` creates a join point whose static return type is `FileOutputStream`. Therefore, a piece of around advice that applies to the join point is required to have a static return type `FileOutputStream` or its subtype.

Even though it looks reasonable for ensuring type safety, the restriction prohibits around advice from doing what we can do by directly editing the program text. For example, in the following method definition, we can textually replace the expression `new FileOutputStream(name)` with `System.out`.

```
void store(String fileName, Data d) {
  OutputStream s = new FileOutputStream(fileName);
  BufferedOutputStream output = new BufferedOutputStream(s);
  output.write(d.toString());
  output.close();
}
```

This is because the return value from the expression is merely used as the argument of the `BufferedOutputStream` constructor, which requires an object of type `OutputStream`. Since `System.out` is a field of type `PrintStream`, and `OutputStream` is a common supertype of `FileOutputStream` and `PrintStream`, the edit is type safe.

However, a piece of around advice that returns `System.out` cannot be applied to `new FileOutputStream(name)` because of the abovementioned restriction.

3 Type Relaxed Weaving

In this work, we propose an improved weaving algorithm, called *type relaxed weaving*, that resolves the restriction without breaking type safety. The restriction in terms of types with the new weaving algorithm become as follows:

> When there is a join point whose return values are used at least of type T in the method, it requires that the around advice applied to the join point to have static return type is less than or equals to T.

With the new restriction, since the return value from `new FileOutputStream(fileName)` is used as a value of type `OutputStream` in the `store` method, a piece of around advice that returns `System.out` is now allowed.

We believe this minor changes to the type restriction will substantially expands the application area of around advice that could not have been written in current AspectJ language. The expanded applications would include aspects that introduce proxy objects and that wraps anonymous objects.

4 Current Status

As an initial assessment, we carried out an experiment in order to estimate the number of join point shadows (i.e., source code locations to which advice can be applied) that can benefit from the type relaxed weaving. The experiment revealed that in five medium-sized Java programs, namely ANTLR, Javassist, jEdit, JHotDraw and Xerces, there are approximately 23 percent among 18 thousand join point shadows can benefit from the type relaxed weaving.

One of the our next plans is to develop a type system for proving type safety. We believe that the problem can be generalized to programming language mechanisms that replace certain values in a type safe manner, including method call interception[3] and type-safe update programming updates[1].

Another plan is to develop a practical AOP language implementation with type relaxed weaving. Since we only modified the type restrictions in the weaving algorithm, we presume that we can implement by modifying existing AspectJ compilers without major difficulties.

References

[1] Martin Erwig and Deling Ren. Type-safe update programming. In *ESOP 2003*, LNCS vol. 2618, pp.269–283, 2003.

[2] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In *ECOOP'01*, LNCS vol. 2072, pp.327–353. 2001.

[3] Ralf Lämmel. A semantical approach to method-call interception. In *AOSD'02*, pp.41–55. 2002.