StrongRelaxAJ: integrating adaptability of RelaxAJ and expressiveness of StrongAspectJ

Tomoyuki Aotani
Graduate School of Arts and Sciences
University of Tokyo
aotani@graco.c.u-tokyo.ac.jp

Manabu Toyama
Graduate School of Arts and Sciences
University of Tokyo
touyama@graco.c.u-tokyo.ac.jp

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

ABSTRACT

A sketch of StrongRelaxAJ is presented. StrongRelaxAJ is an extension to AspectJ with a type system for around advice that integrates the ones in RelaxAJ and StrongAspectJ. In other words, StrongRelaxAJ employs the type-relaxed weaving mechanism in RelaxAJ for better adaptability of around advice, and supports type variables and explicit signatures of proceed for better expressiveness without relying on dangerous and annoying dynamic casting on the return values from `proceed`.

Categories and Subject Descriptors

D.3.3 [**PROGRAMMING LANGUAGES**]: Language Constructs and Features—*Polymorphism*

General Terms

Design, Languages

Keywords

Aspect-Oriented Programming, Around Advice, Type-Relaxed Weaving, AspectJ, RelaxAJ, StrongAspectJ, StrongRelaxAJ

1. INTRODUCTION

Around advice is one of the unique and powerful features of the pointcut and advice mechanism. It allows programmers not only to replace the operations with others without directly modifying the source code but also to change parameters and return values of operations by using `proceed`.

Defining a good type system for around advice is one of the challenges in statically typed aspect-oriented programming (AOP) languages that employ the pointcut and advice mechanism. Because a type system conservatively accepts "safe" programs, it constrains the adaptability of around advice.

In Proceedings of Foundations of Aspect-Oriented Langauges (FOAL2010), pp. 1-4, March 2010, published as Technical report CS-TR-10-04, School of Electrical Engineering and Computer Science, University of Central Florida.

Recent studies revealed and solved problems of type-safety and expressiveness in AspectJ [3, 5], which is one of the widely used statically typed AOP languages.

RelaxAJ [6], which is an extension to AspectJ, improves the adaptability of AspectJ's around advice by relaxing the restriction on its return type. While a piece of around advice and its *target join point* on which the advice is executed must have the same return type for type safety in AspectJ, this is not required any more in RelaxAJ. Instead, it guarantees type safety by ensuring that the return values of around advice are safely used within the program.

StrongAspectJ [2] is another extension to AspectJ, which supports type-safe generic around advice. A piece of around advice in StrongAspectJ is safely evaluated on each target join point. Intuitively, it is achieved by ensuring that a piece of around advice always returns the return values of `proceed`.

This position paper points out the problems of expressiveness in RelaxAJ as well as the problems of adaptability in StrongAspectJ, and proposes *StrongRelaxAJ* as our solution. StrongRelaxAJ integrates the adaptability of RelaxAJ and genericity of StrongAspectJ. In other words, it solves the problems of expressiveness in RelaxAJ as well as solves the problems of adaptability in StrongAspectJ. In the position paper, we roughly explain its syntax and type checking rules by using a concrete example. Its formalization and implementation are left for future work.

The rest of the paper is organized as follows. Section 2 gives a brief overview of RelaxAJ and StrongAspectJ. Section 3 presents examples that cannot be achieved by either RelaxAJ or StrongAspectJ, and Section 4 shows a sketch of StrongRelaxAJ. After discussing related work in Section 5, Section 6 concludes the position paper and lists our future work.

2. BACKGROUND: RELAXAJ AND STRONGASPECTJ

This section presents brief overviews of RelaxAJ and StrongAspectJ along with a code fragment that implements a popup window.

2.1 Base code: creating a popup window

Suppose we have an image editor in which one can manipulate images by applying various filters (e.g., Gaussian blur filters) and see a preview of the filter's effect in a popup window. Listing 1 shows the method `showPreview` that creates a popup window for previewing.

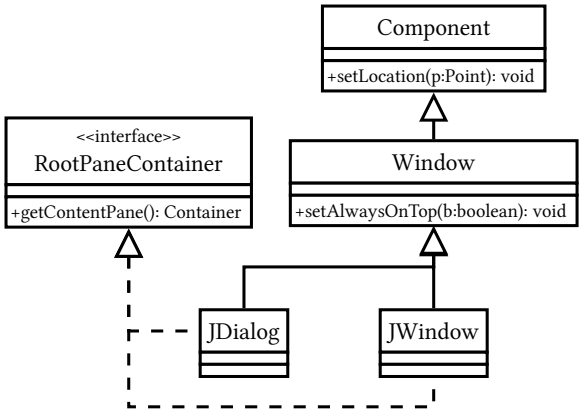

Figure 1: Relationship between `Component`, `Window`, `JDialog`, `JWindow` and `RootPaneContainer`

```
1 void showPreview(JFrame mainWin, MyImage image){
2   JWindow popup = new JWindow(mainWin);
3   MyCanvas canvas = new MyCanvas(image);
4   JButton closeButton = new JButton("close");
5   popup.getContentPane().add(canvas);
6   popup.getContentPane().add(closeButton);
7 }
```

Listing 1: Popup window for previewing

It takes two parameters, namely `mainWin`, which is the main window of the application, and `image`, which is the processed image the user will see. `MyCanvas`, which is a subclass of `JPanel`, draws the processed image. The popup window `popup`, which is an instance of `JWindow`, contains a canvas and button. If the button is clicked then the popup window is closed.

2.2 RelaxAJ

RelaxAJ is an extension to AspectJ, which has a novel type-checking rule for the return types of around advice. A piece of around advice in RelaxAJ can have a return type that is not a subtype of the target join points' on which the advice is executed. In AspectJ, the return type instead must be a subtype of the target join points'.

Of course the return type cannot be any type in RelaxAJ. It must be consistent with the types as which the return values are used in the program.

Assume we want to create a modal dialog instead of a simple popup window in the above example so as not to leave previews outdated. One of the easiest ways to achieve it is to use `JDialog` with the modal flag, instead of `JWindow`.

Listing 2 shows a piece of around advice that implements

```
1 RootPaneContainer around(Frame frame)
2   :call(JWindow.new(Frame))&&args(frame){
3   return new JDialog(frame, true);
4 }
```

Listing 2: Around advice that replaces JWindow with JDialog

```
1 <T extends Component>
2 T around(Frame frame)
3   :call((JDalog||JWindow).new(Frame))&&args(frame)
4   :T proceed(Frame){
5   T popup = proceed(frame);
6   popup.setLocation(DEFAULT_LOC);
7   return popup;
8 }
```

Listing 3: Around advice that specifies the location of the popup window

the idea. It simply creates a new modal `JDialog` object and returns it when a `JWindow` object is to be created.

Note that the return type, which is `RootPaneContainer`, is a supertype of the target join point's return type, which is `JWindow`. It is not valid in AspectJ because it requires the return type is a subtype of the return types of its target join points.

RelaxAJ accepts the advice when it is applied only to the line 2 in Listing 1 because its return value is used only as `RootPaneContainer` within the program and thus the replacement is safe.

2.3 StrongAspectJ

StrongAspectJ is another extension to AspectJ, which supports type-safe generic around advice. The genericity and type-safety are achieved by using (bounded) type variables to declare the return types of around advice and also `proceed`. Although AspectJ implicitly decides the return type of `proceed`, StrongAspectJ does not. It is given by the programmer through a dual advice signature.

Assume a popup window is an instance of either `JDialog` or `JWindow`, and we want to specify the location where the window appears. This can be achieved by calling `setLocation` that is defined in `Component`.

Listing 3 is a piece of the around advice in StrongAspectJ that catches the popup window object and calls `setLocation` on it. Line 1 declares the type variable `T` whose upper bound is `Component`. It is used as the return type of the advice. Line 4 is the dual advice signature that declares the return and argument types of `proceed`: here its return type is `T` and its argument type is `Frame`.

If the base program is type safe, the woven program is also type safe. This is because (1) the return type of each target join points (`JDialog` or `JWindow`) is always a subtype of `Component` (see Figure 1) so that no type error occurs within the advice, and (2) the return types of the advice and `proceed` are always the same so that the values returned by the advice can be used safely as the original values.

3. EXAMPLES THAT NEED AN INTEGRATED LANGUAGE

By integrating RelaxAJ and StrongAspectJ, we can implement more adaptive and interesting aspects. This section presents two examples that cannot be achieved in either RelaxAJ or StrongAspectJ alone but can be achieved in the integrated language.

3.1 Specifying return type of proceed in type-relaxing advice

```
1 RootPaneContainer around(Frame frame)
2   :call(JWindow.new(Frame))&&args(frame){
3   if(POPUP_MODAL) return new JDialog(frame, true);
4   else{
5     JWindow popup=(JWindow)proceed(frame);
6     JOptionPane.showMessageDialog(popup,ALERT);
7     return popup;
8   }
9 }
```

Listing 4: Using dynamic casting to use return values of `proceed` as a `JWindow`

Suppose that we want to make the popup window modal only if `POPUP_MODAL` is true. Otherwise, we show a message dialog that warns danger of out-of-date previews along with the original popup window. It can be achieved in RelaxAJ by defining a piece of around advice shown in Listing 4

The problem here is the use of a cast operator at line 5. It is necessary because RelaxAJ simply adapts AspectJ's typing rule for `proceed`.

The return type of `proceed` is the same to the one of the around advice, that is, `RootPaneContainer`. On the other hand, to set `popup` as the parent window of the message dialog (`JOptionPane`) through `showMessageDialog`[1], its static type must be a subtype of `Component`, which is incompatible with `RootPaneContainer`.

We should be able to omit dynamic casting because it is obvious that `proceed` always returns a `JWindow` object. One way to achieve it is to add StrongAspectJ's dual advice signature and the typing rules to RelaxAJ. StrongAspectJ allows programmers to declare the return type of `proceed`. `JWindow` is a valid return type here because it is (1) a supertype of the return type of the target join points (`JWindow` itself) and also (2) a subtype of the return type of the advice (`RootPaneContainer`).

Of course StrongAspectJ does not accept such advice because its return type is invalid: it must be a subtype of the return types of the target join points in StrongAspectJ, but here `RootPaneContainer` is a supertype, not a subtype, of `JWindow`.

3.2 Abstracting the return type of around advice by using type variables

RelaxAJ provides no way to write a piece of around advice whose return value of is used as two or more types incompatible with each other. In other words, the return type of a piece of around advice must be one type in RelaxAJ.

It is natural to control orders of windows in GUI programs. Listing 5 extends the base program (Listing 1) so that the popup window stays above all other windows. Line 7 is added where `popup` is used as a `Window` because `setAlwaysOnTop`, which is an instance method defined in `Window`, is called on it.

The around advice declaration in Listings 2 and 4 cannot be compiled with the above extended program. This is because the return type cannot be relaxed to `RootPaneContainer`. As mentioned before, the return value is used as not only a `RootPaneContainer` but also a `Window`.

Modifying the return type is not a solution because there is no such a type that is a subtype of `RootPaneContainer` and `Window` and a supertype of `JDialog` and `JWindow`.

[1] `showMessage(Component,Object)` is a static method in `JOptionPane`

```
1 void showPreview(JFrame mainWin, MyImage image){
2   JWindow popup = new JWindow(mainWin);
3   MyCanvas canvas = new MyCanvas(image);
4   JButton closeButton = new JButton("close");
5   popup.getContentPane().add(canvas);
6   popup.getContentPane().add(closeButton);
7   popup.setAlwaysOnTop(true);
8 }
```

Listing 5: Create a popup window that stays above all other windows

```
1  <T extends RootPaneContainer & Window>
2  T around(Frame frame)
3    :call(JWindow.new(Frame))&&args(frame)
4    :JWindow proceed(Frame){
5    if(POPUP_MODAL) return new JDialog(frame, true);
6    else{
7      JWindow popup=proceed(frame);
8      JOptionPane.showMessageDialog(popup,ALERT);
9      return popup;
10   }
11 }
```

Listing 6: Around advice in StrongRelaxAJ with an explicit signature of proceed and type variable

4. STRONGRELAXAJ

We propose StrongRelaxAJ, which is a hybrid of RelaxAJ and StrongAspectJ. StrongRelaxAJ has two additional language features, namely *explicit signature of proceed* and *type variables* to the type-relaxed weaving mechanism in RelaxAJ. An explicit signature of proceed helps us to omit dynamic casts shown in Section 3.1. Type variables are used to define a piece of around advice whose return values are used as two or more incompatible types shown in Section 3.2.

This section first explains how the StrongRelaxAJ around advice looks by showing an example. Then it explains about explicit signature of proceed and type variables.

4.1 Around advice in StrongRelaxAJ

The syntax of around advice in StrongRelaxAJ is similar to StrongAspectJ. A piece of around advice in StrongRelaxAJ has declarations of type variables and the signature of proceed.

Listing 6 is a piece of around advice in StrongRelaxAJ. It is a modified version of the advice in Listing 3.1 that works with the extended base code shown in Listing 5.

Line 1 declares the type variable `T` whose upper bounds are `RootPaneContainer` and `Window`. It is used as the return type of the advice in Line 2 instead of `RootPaneContainer`. Line 4 declares the signature of `proceed`.

Note that we does not use dynamic casting at line 7. Because the return type of `proceed` is `JWindow`, we can use its

return value as a `JWindow` object.

4.2 Explicit signature of proceed

The return type of `proceed` in a piece of around advice must be (1) a supertype of the return types of the target join points and also (2) a supertype of the return types of around advice that may be called by `proceed`. In other words, StrongRelaxAJ does not need any relationships between the return type of a piece of around advice and its `proceed` unlike AspectJ, StrongAspectJ and RelaxAJ. This does not break type-safety because `proceed` never calls the around advice that encloses it.

Let's look at the example in Section 4.1. The explicit signature of proceed on Line 4 in Listing 6 satisfies the condition. The return type `JWindow` is not a type variable, and it is clearly a supertype of `JWindow`, which is the return type of the target join points. Because there is no other pieces of advice, the second condition for non variable return types holds too.

4.3 Type variables

Type variables in StrongRelaxAJ are more expressive than the ones in StrongAspectJ. StrongAspectJ uses type variables to ensure that the return value of `proceed` is the return value of the advice. For instance, if the return type of proceed is `T`, which is a type variable, then the enclosing around advice must be `T`.

In addition to the usage, StrongRelaxAJ uses them to declare that the advice returns a value of some type that satisfies the upper bounds. Listing 6 is an example. It uses the type variable `T` to return `JDialog` and `JWindow`. Because each of them is a subtype of `Window` and `RootPaneContainer`, StrongRelaxAJ's type system accepts the return statements.

Note that the return value is used as only a `Window` and a `RootPaneContainer` within the target program, that is, Listing 5. Therefore, type safety is preserved.

5. RELATED WORK

Adding union types to Java [4] gives another solution for the situation in Section 3.2. If it is allowed to use union types, we can declare the return type of the advice as `JWindow`∨`JDialog` instead of using a type variable as in Listing 6. Then RelaxAJ with union types successfully accepts the advice because `JWindow` and `JDialog` are subtypes of `JWindow`∨`JDialog` and `RootPaneContainer` and `Window` are supertypes of `JWindow`∨`JDialog`.

6. CONCLUSIONS AND FUTURE WORK

The position paper presented a sketch of StrongRelaxAJ, which is a hybrid of RelaxAJ and StrongAspectJ. By using explicit signature of proceed, programmers can omit dynamic casting on the return values of proceed. Type variables are used not only to write generic advice but also to declare the return type of a piece of around advice whose return type cannot be described by using only one type.

Dealing with parameter types of `proceed` is one of our future work as RelaxAJ. Formalization and implementation are also our future work. Formalization could be done by extending Featherweight Java for Relaxation (FJR) [6] and StrongAspectJ [2]. Implementation would be done on top of StrongAJ compiler or the AspectBench compiler (abc) [1].

7. REFERENCES

[1] Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondrej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. abc: An extensible AspectJ compiler. In *Proceedings of AOSD'05*, pages 87–98, 2005.

[2] Bruno De Fraine, Mario Südholt, and Viviane Jonckers. StrongAspectJ: Flexible and safe pointcut/advice bindings. In *Proceedings of AOSD'08*, pages 60–71, 2008.

[3] Erik Hilsdale and Jim Hugunin. Advice weaving in AspectJ. In *Proceedings of AOSD'04*, pages 26–35, 2004.

[4] Atsushi Igarashi and Hideshi Nagira. Union types for object-oriented programming. In *Proceedings of SAC'06*, pages 1435–1441, 2006.

[5] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In *Proceedings of ECOOP'01*, pages 327–353, 2001.

[6] Hidehiko Masuhara, Atsushi Igarashi, and Manabu Toyama. Type relaxed weaving. In *Proceedings of AOSD'10*, 2010. To appear.