AOP considered harmful

Constantinos Constantinides* Therapon Skotiniotis† Maximilian Stoerzer‡

August 23, 2004

Abstract

In his famous letter "`Go To` statement considered harmful" Dijkstra started a discussion finally resulting in banning of most unstructured control flow statements from modern high level programming languages.

To overcome limitations of todays programming languages, aspect oriented programming has been proposed. Unfortunately language elements used by many AO languages are in a way similar to the `Go To` statment, so we ask the provocative: AOP considered harmful?

1 `Go To` considered harmful

In his famous letter "`Go To` statemnt considered harmful" [2] Dijkstra argued that programmers should always be able to create a sound mapping from their variables to values only based on a 'coordinate system', which of course should not depend on program values but on external values.

For a sequential program including conditional and branch statements this for example is a single textual index (the line number). Adding loops introduces an additional loop counter but nevertheless maintains the ability for the programmer to reason about variable values. Even procedures do not destroy this ability, although the necessary coordinate system now also comprises the call stack.

However, with unstructured control flow as represented by the `Go To` statment, such a coordinate system does no longer exist.

As a conclusion, Dijkstra claimed that – even though he did not claim that the list of programming language elements covered by his article was complete – any programming language construct has to maintain the property that it does not destroy the coordinate system for the programmer.

*Concordia University, Montreal, Quebec, Canada, cc@cse.concordia.ca

†Northeaster Univ., Boston, USA, skotthe@ccs.neu.edu

‡Univ. of Passau, Germany, stoerzer@fmi.uni-passau.de

2 A Coordinate System for Advice?

The question now is if a sound coordinate system can be constructed if also pointcuts and advice are included. On the one hand, advice is very similiar to procedures, on the other hand there are two important differences:

- obliviousness of application and
- non-certainty of application.

By *obliviousness of application* we want to express that at the source code location describing an affected join point (i.e. a method call, field access, ...) the adivce which will be executed is not visible, in contrast to a method call.

As a result just looking at the source code line, loop counters or the call stack is not enough to deduce a variable value – a piece of advice might have changed it inbetween, invisible for the programmer.

Although tool support can lighten this problem by showing applying advice, the language alone does no longer allow to create a coordinate system for the programmer. From that point of view pointcuts and advice violate Dijkstras postulated property.

So in a way advice here is even worse than `Go To` as the `Go To` statement is at least visible in the code, but advice is not. This can be compared to the `Come From` statement [1] which has been proposed as a way to avoid `Go To` – of course only as an April Fools joke. However, it's semanics look surprisingly familiar.

If you consider the example in figure 1 (left), when control reaches line 10, control it transfered to the `Come From` statement in line 20. A similiar formulation with advice is shown to the right.

The second difference outlined above – the *non-certainty of application* refers to pointcuts where

```
 5 input x
10 print 'result is :'
15 print x

20 come from 10
25      x = x * x
30 return
```

```
main() {
    input x
    print(result(x))
}
int result(int x) { return x }
around(int x): call(result(int)) && args(x) {
    int temp = proceed(x)
    return temp * temp
}
```

Figure 1: Comparing advice and `Come From`

the set of matching joinpoints cannot be evaluated *statically* but depends on runtime values.

In this case, even if a tool shows potential advice matching, the *actual matching state* in general *is not known*, i.e. the programmer cannot deduce from the source code alone weather a piece of advice is applied at a certain joinpoint or not. So any coordinate system including advice and dynamic pointcuts also has to consider runtime values – a clear violation of Dijkstras statments.

3 Dynamic dispatch compared to `Go To`

Object-orientation or more specifically the dynamic dispatch also in a way violate Dijkstras demand for a coordinate system, as here also the target of a method call depends on the runtime type of the callee object. This "feature" of object-oriented programs can – is misused – also lead to decreased comprehensibility of programs.

However, the situation of AOP differs in two ways from OO. First, as AOP is orthogonal to the OO paradigm (but can add AOP features to OO) the problem cannot be dismissed by claiming that OO is problematic but nonetheless successful. So excusing the breaking of Dijkstras coordinate system with some AOP language constructs with OO is no option.

Second, to counter the effect of the uncertain method call target the OO community has developed a set of *rules how to use method overriding and how not to*. Informally an overriding method should only expect less and provide more, by maintaining all invariants, so each overriding method ideally has (more or less) the same semanics. As a consequence, the programmer can reason about the semantics of a method call even if the actually called method is statically unknown. This is still missing for AOP.

4 Conclusion

In this position statement we tried to show that the AOP pointcut and advice mechanism has some of the problems associated with the `Go To` statment, namely that it does not allow to provide a sound mapping from variables to values based only on a coordinate system of non-program values.

We compared these AOP constructs also to dynamic dispatch in OO, but argued that for OO inheritance rules lighten the problem. For AOP similar rules are still missing and thus in our opinion AOP is considerably more dangerous.

Dijkstra in his letter observed: " ... that the quality of programers is indirectly proportional to the amount of `Go To` statemets they use in their programs.". As currently most AOP research is not about methodolgy but about more dynamicity in the future this might be rephrased to " ... indirectly proportional to the amount of advice they use in their programs."

References

[1] Lawrence R. Clark. A linguiistic contribution of goto-less programming. *DATAMATION*, December 1973.

[2] Edsger W. Dijkstra. Letters to the editor: Go to statement considered harmful. *CACM*, 11(3):147–148, March 1968.