Alternatives to Aspect-Oriented Programming?

David Bruce Nick Exon

Distributed Technology Group
QinetiQ ltd
St Andrews Road, MALVERN WR14 3PS, UK

dib@QinetiQ.com njexon@QinetiQ.com

Abstract. In the interest of stimulating debate, we present a broad, all-inclusive view of aspect-oriented programming (AOP) and related initiatives. In particular, we contrast the compelling vision used to 'sell' the idea of AOP with the more finicky nature of its realizations to date. We outline a scenario that we feel AOP ought at least aspire to address, and ask whether the direction it is currently following is likely to take us as far as we really need to go.

Separation of concerns: there is no alternative!

In the general software world, there would appear to be an irresistible trend towards the widespread use of components. There are countless articles, both in the technical literature and the computing press, that expound various reasons for this, and we have neither the space nor the inclination to re-iterate them all here. We will simply note that much the same forces are at work in computer simulation, our core research focus in recent years. Indeed, amidst the inevitable debates on the desirability or otherwise of 'federating' simulation components for large-scale simulations, Richard Weatherly in particular has noted on several occasions that "there is no alternative!".

The view of many researchers is broadly similar for aspect-oriented programming (AOP) [1,2]. The limitations of traditional, time-honoured but fundamentally quite basic techniques for software construction are increasingly making themselves felt, most notably through our inability to construct and evolve programs at the pace demanded by modern times. Proper separation of concerns [3,4] plays a vital rôle here, but the abstraction and composition mechanisms we have today far from suffice. What AOP and related initiatives can offer are exciting glimpses of how we might be able to articulate and encapsulate hitherto-elusive concepts in qualitatively new ways. Thereby springs a much-needed sense of hope; surely there can be no alternative?

An anecdotal non sequitur

Back in autumn 2000, the first author gave an internal talk to our group, entitled "Aspect-oriented programming and AspectJ". This talk first examined the nature of computer software, and some of the fundamental problems caused by inadequate

separation of concerns. It went on to present the vision motivating aspect-oriented programming, that one could provide independent specifications for each distinct concern and then 'weave' them together to build the resulting system. The talk finally touched on some of the prototype AOP systems/tools that were available at that time. In particular, it outlined Xerox PARC's AspectJ [5,6,7] and the sorts of things that that language lets you do — specify program 'pointcuts', add or modify functionality through before/after/around advice, extend classes using introduction, and associate 'aspect' classes with objects, pointcuts, etc.

Several of our colleagues pointed out — some there and then, others later — that the early part of this talk was fine, as was the later part, but the two seemed a void apart. The idea of AOP was great, they said, and AspectJ made perfect sense in its own right … but the former was a grand conceptual vision while the latter focussed on low-level details.

Since then, the authors and various other members of our group have experimented with AspectJ from time to time. This continued exposure to AspectJ has done little to bridge the gap, however; if anything, it's reinforced it! *(We should note that we're not picking on AspectJ in particular here; it's just the most prominent example, and the one that we've had most experience with. Other AOP and related systems seem broadly comparable in this regard. More on this later.)*

So, maybe our colleagues' gut-reactions were founded; maybe there is something missing? But what? Could it just be that we're being dumb? We'd like to think not! The Xerox PARC team behind AspectJ acknowledge that its documentation often lags behind their implementation efforts, but it would be churlish as well as disingenuous to suggest that that's the problem. Aspect-oriented programming is, of course, a relatively new field, so it is only natural that the community at large will take some time to learn and communicate good design principles; perhaps we just need to wait? (It is worth noting here that excellent tutorials such as [7] are now starting to emerge.) One final option remains: it could be that AspectJ et al. really are too low-level for our ambitions — or, turning that around, that we're guilty of expecting too much.

A multi-dimensional functionality thought experiment

We have spoken of aspirations, but given few details. What sort of thing do we have in mind?

One way to articulate such matters is by means of a 'thought experiment' — in this case taking inspiration from the military simulation domain. Our intent is to show the breadth of multi-dimensional functionality in what for that domain is a relatively simple problem, and to stimulate thought about how software construction techniques influence its subsequent evolution.

Consider a computer generated forces (CGF) assault on an enemy position. The requirement is for a simulation (component) to plan and execute an operation:

- according to some specified scenario (location, time, military resources, opposition, …)
- using a given form of reasoning process (broad agents, rule based, scripted, …)
- following particular doctrinal principles (tactics, rules of engagement, …)

- inside certain computational resource limits (time, memory, …)
- implementing a particular style of simulation (training, analytic, predictive, …)
- visualized as required (immersive VR, plan view display, statistical summary, …)
- within acceptable validity tolerances
- …

Each of the above represents a design decision that could — in principle at least — be changed independently. To get a sense for how hard it is to plan ahead for all possible eventualities, think about how you might go about coding such an example. What abstraction mechanisms would you use to structure it? How well could your approach cope with this range of changes, and with other possible variations that you can think of?

Clearly, some of the flexibility that we and our customers demand can be accommodated using conventional methods (e.g., parametrization of scenario). Aspects as we currently know them might well serve for others (e.g., at least for some forms of visualization). However, entirely new techniques would also seem to be required (e.g., for separating out elements of 'intelligent' behaviour such as doctrine — especially if this is to be in some way independent of the reasoning approach).

So what's the point?

Having started by arguing the case for aspect-oriented programming, we conclude by turning about-face to knock our strawman down — if only on a technicality.

Although there may be no alternative but to pursue such mechanisms, that does not make the future entirely predestined and inevitable. We actually have a lot of choice. It is not the choice of whether to adopt something like AOP, but the more exacting choice of how best to adopt it. This might not be the choice we thought that we had, but it's actually a pretty good one; being so wide-ranging and open-ended, it gives plenty of room for manoeuvre.

In other words, the principle seems sound, but the practice still needs a good deal of refinement. The real question is whether the mechanisms that we know about now (e.g., those in AspectJ, in other variations on the theme such as HyperJ [8], or even those investigated in related initiatives such as Minsky's law-governed regularities [9] or Microsoft's intentional programming [10,11]) suffice to satisfy the aspirations that we already have, and those that we are going to formulate over the coming years.

Our honest answer is that we don't know, but on balance we remain sceptical.

We therefore challenge the research community to join us in looking for new forms of program abstraction, composition and transformation — which may or may not end up resembling (or being called) aspect-oriented programming — that address both the precisely formed targets of academic fascination and the less easily characterized problems that software developers really face. Bridging the gulf between conceptual levels, and exploring the full life-cycle viability of aspect-oriented programs, are but two of the more interesting that immediately spring to mind.

Aspect-oriented programming as we know it now is doing a grand job of exploring interesting territory; we simply urge that the research community widen its horizons, to see what else remains uncharted.

References

1. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopez, John-Marc Loingtier and John Irwin, "Aspect-Oriented Programming", pp. 220 ff. in Proc. *11th European Conference on Object-Oriented Programming* (Jyväskylä, Finland, 9–13 June 1997) — published by *Springer-Verlag* as *Lecture Notes in Computer Science* no. 1241 (Mehmet Akşit and Satoshi Matsuoka, editors).
2. "Aspect-Oriented Programming".
   Xerox PARC website, URL: http://www.parc.xerox.com/csl/projects/aop/.
3. D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", pp. 1053–1058 in *Communications of the ACM*, Vol. 15, No. 12, December 1972.
4. Edsger W. Dijkstra, "A discipline of programming", *Prentice-Hall*, 1976.
   [See in particular "In Retrospect" (chapter 27; pp. 209–217), and also "note 1" (p. 203).]
5. "AspectJ: Crosscutting Objects for Better Modularity".
   Xerox PARC website, URL: http://aspectj.org/.
6. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold, "An Overview of AspectJ", pp. 327 ff. in Proc. *15th European Conference on Object-Oriented Programming* (Budapest, Hungary, 18–22 June 2001) — published by *Springer-Verlag* as *Lecture Notes in Computer Science* no. 2072 (Jørgen Lindskov Knudsen, editor).
7. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold, "Getting Started with AspectJ", tutorial article submitted for a special theme section on Aspect-Oriented Programming to appear in *Communications of the ACM*, Vol. 44, No. 10, October 2001.
   (Available on-line at URL: http://aspectj.org/doc/gettingStarted/index.html.)
8. "HyperJ™: Multi-Dimensional Separation of Concerns for Java™".
   IBM Research website,
   URL: http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm.
9. Naftaly H. Minsky, "Law-Governed Regularities in Object Systems. Part I: An Abstract Model", pp. 283–301 in *Theory and Practice of Object Systems*, Vol. 2, No. 4, 1996.
10. Charles Simonyi, "The Future Is Intentional", pp. 56–57 in *IEEE Computer*, Vol. 32, No. 5, May 1999. [One of nine contributions to "Software [R]evolution: A Roundtable", Kirk L. Kroeker (editor), pp. 48–57 of that issue.]
11. "Intentional programming".
    Microsoft Research website, URL: http://www.research.microsoft.com/ip/.

Acknowledgements

One of the authors has twice had the privilege of seeing Gregor Kiczales' excellent presentations on aspect-oriented programming. Our colleague David Allsopp offered some particularly thoughtful observations. This work has been supported by the UK Ministry of Defence under Corporate Research TG10 project 5.4.4, "Re-usable Simulation Components for Synthetic Environments".