



The lifecycle starts here.

Software Development

April 2004

Show Me the Structure

Aspect-oriented programming is all about crosscutting structure. Understand that, and it's crystal clear what direction AOP tools need to follow.

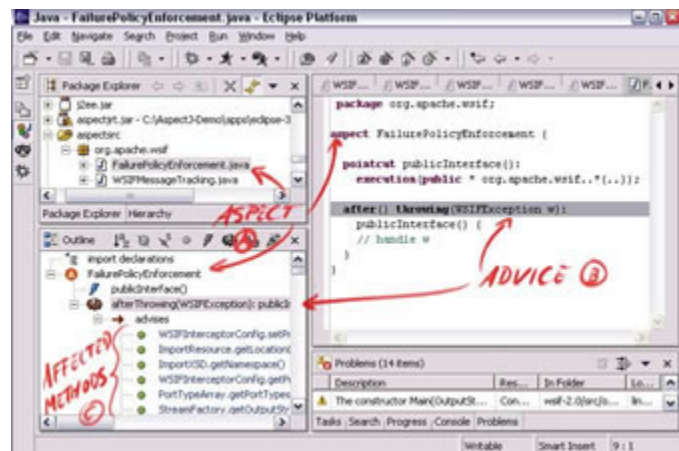
By Gregor Kiczales and Mik Kersten

Modern integrated development environments (IDEs) understand more about your programs than ever before. They feel “smarter”—and as a result, make you much more productive. Integrating aspect-oriented programming with these object-oriented IDEs is a major challenge for AOP proponents. While AOP looks promising for enterprise application development, you won't want to use it unless your IDE is not only compatible with AOP, but also smart about it. An IDE should provide facilities for structure browsing, smart editing, refactoring, building, testing and debugging. The next step is to extend these features to work with AOP.

Structure browsing means that the tools will allow you to view a system's object-oriented structure. For example, you can use Eclipse's type hierarchy view to explore the inheritance in the code you're working on. JBuilder's UML support provides a more abstract view of the associations between classes. Structured searches also make it easier to navigate call graphs.

Smart editing means that the editor knows enough about your code to help you complete your thoughts; instead of typing and navigating the code, you can select your edits from lists of links and hints.

These smarts are all rooted in the IDE's deep knowledge of the program's structure. The structure browser navigates it, the editor uses it to autocomplete what you were about to type, and the debugger needs it to show you where you are in the execution. For example, the latest Visual Studio .NET pop-ups are aware of generics, and IDEA provides automatic correction of syntax errors. Eclipse's refactoring support lets you perform most of the Fowler refactorings without directly editing code. Build support understands object-oriented dependency structure, so compilers can incrementally build the system on-the-fly as you edit. Integrated Ant tools comprehend how your project structure can be packaged into deployable units. JUnit integration automatically builds test skeletons



[\[click for larger image\]](#)

IDE Support for AspectJ

Since aspects are like a special kind of class, they show up in the same places as classes (A). Advice shows up as an aspect member in the editor and outline view (B). The structure browser makes it easy to navigate from advice to the affected join points (C).

from the class structure. The debuggers use object structure for presenting dynamic views of the running system.

OOP has inheritance and encapsulation, so the IDEs understand the structural relationships of classes, members and references. AOP adds a new kind of structure: advice and introduction on join points. The existing IDE features must be extended to understand this cross-cutting structure. In addition, new IDE tools are needed for AOP-specific tasks such as browsing the large-scale crosscutting of some aspects.

State of the Art

A good way to find out what we currently know about how IDEs must integrate support for AOP is to use the Eclipse AspectJ plug-in, which aims to make working with AspectJ as easy as working with Java. Say that you want to use AOP to implement an exception-handling policy in a Java project. After building the system with the aspect, you'll immediately see the effect of the advice you wrote as crosscutting structure links in the outline view. You can use these links to navigate to the advised join points, and inspect the list to ensure that your aspect behaves as expected.

The crosscutting aspect structure is shown in the same views that Eclipse uses for showing class structure. When you make a new method, you expect Eclipse to inform you that it over-rides another. Similarly, the AspectJ support shows you where your program is crosscut by an aspect. For example (see "Editor Annotations"), you can see when a call site is advised in the editor's inline annotations display (1). Tree node links do the same for members in the outline view (2). The aspect visualizer provides a big-picture view of the structure, showing how the aspects crosscut the classes of a package (3).

These AOP-aware IDE tools make crosscutting structure explicit, just as the OOP tools make inheritance and encapsulation structure explicit. They need to fully integrate working with aspects. For example, it must be as easy to set a breakpoint on advice as it is on a method. If a documentation view shows program structure, as Javadoc does, it must also show the crosscutting aspect structure.

This perspective on AOP clarifies how to think about IDE support. Tool builders must ask which IDE features help programmers understand a system's structure. Those are the first features to make AOP-aware. The challenge is to integrate these extensions seamlessly so that the object-oriented tooling is still intuitive and easy to use. Once that's done, providing new crosscutting-centric views and features will meet the needs of the growing number of programmers who find AOP indispensable.

2005 Forecast

The tool builders have much more work to do. First, the kind of tool support we now have for AspectJ



[\[click for larger image\]](#)

Editor Annotations

You can see when a call site is advised in the editor's inline annotations display (1). Tree node links do the same for members in the outline view (2). The aspect visualizer provides a big-picture view of the structure, showing how the aspects crosscut the classes of a package (3).

needs to be provided for other approaches, such as AspectWerkz. The dynamic nature of advice selection in JBoss makes this a little more difficult, but no less important.

Second, we need to catch up with the advanced features that are becoming standard in object-oriented IDEs. For example, Java developers already take refactoring support for granted, and soon .NET developers will, too. Developers will also want these refactoring tools to help them refactor from tangled code to aspects as well as refactor existing aspects.

By 2005, we can expect to see smarter editing with code assist and refactoring support, better crosscutting views and UML notation, and generics- and metadata-aware compilers and tools, as well as debuggers and testing tools that understand the join point model. Early adopters already have it better than their OOP counterparts, who struggled with the first batch of C++ tools. The next round of improvements in AOP tool support should aim to make aspects as accessible and easy-to-use as objects are today.

Next month: AOP and attributes.

Gregor Kiczales led the Xerox PARC teams that developed aspect-oriented programming and AspectJ, and is a leading AOP evangelist based in Vancouver. This month, Gregor is joined by **Mik Kersten**, who built the IDE tools for AspectJ.