# Position Summary: Aspect-Oriented System Structure

Yvonne Coady, Gregor Kiczales, Michael Feeley,
Norman Hutchinson, Joon Suan Ong and Stephan Gudmundson
*University of British Columbia*

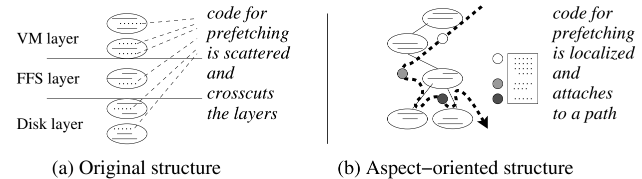
Operating system structure is important – it leads to understandable, maintainable, 'pluggable' code. But despite our best efforts, some system elements have been difficult to structure. We propose a new analysis of this problem, and a new technology that can structure these elements.

Primary functionality in system code has a well defined structure as layered abstractions. Other key elements naturally defy these structural boundaries – we say that they *crosscut* the layered structure. For example, prefetching for mapped files involves coordinated activity at three levels: predicting the pattern of access and allocating pages in the VM layer, determining the contiguity of blocks in the disk layer, and reconciling the costs of retrieval in the file system layer. Because of its inherent crosscutting structure, the implementation of prefetching is scattered through the primary functionality in each of the layers involved (Figure 1a).

In FreeBSD v3.3, prefetching for mapped files is approximately 265 lines of code, grouped into 10 different clusters, scattered over 5 functions from VM and FFS alone. Dynamic context, such as flagging VM-based requests, is passed as parameters from high level functions down through lower ones. Portions of prefetching code violate layering by accessing high level abstractions from lower level functions, such as freeing and page-flipping VM pages from within FFS. In this form, there is no structure to the implementation of prefetching – it is hard to understand, hard to maintain, and certainly hard to unplug.

Aspect-oriented programming (AOP) [3, 2] uses linguistic mechanisms to support the separation of crosscutting elements, or *aspects* of the system, from primary functionality. Aspects declare code to execute *before*, *after* or wrapped *around* existing primary function calls, within the execution flow of other function calls, and with access to specific parameters of those calls. AOP improves the comprehensibility of crosscutting elements in two ways: it allows small fragments of code that would otherwise be spread across functions from disparate parts of the system to be localized; and it makes the localized code more coherent, because interaction with primary functionality is declared explicitly and within shared context.

We have developed a proof-of-concept AOP implementation of prefetching in FreeBSD [1]. In our implementation, we have been able to modularize prefetching. The internal structure of the prefetching code and its interaction with the overall VM and FFS activity are explicitly defined as a sequence of activities that occur at well defined points *along a page-fault path*, rather than being broken into layers (Figure 1b). The AOP implementation is designed to allow us to see precisely how low-level prefetching code acts in service of high-level prefetching code. Primary page fault handling functionality no longer includes prefetching code, nor does it explicitly invoke prefetching functionality.



(a) Original structure (b) Aspect–oriented structure

Figure 1: Prefetching and primary functionality.

In the AOP implementation, one aspect captures how prefetching plays out over page-fault handling for sequentially accessed mapped files: first the page map is locked and pages are pre-allocated according to a prediction, then these and possibly other pages are synchronously brought into the file buffer cache and page-flipped where appropriate, and finally further pages may be asynchronously prefetched into the cache. We can clearly see how this differs from the prefetching aspect for the non-sequential case, where pages may be de-allocated if it is not cost-effective to retrieve them, the file buffer cache is not involved, and further asynchronous prefetching is not applied. Structured this way, prefetching gains context, is more tractable to work with, and is even unpluggable.

We believe that other key elements of operating systems are crosscutting and that their unstructured implementation is excessively complex. We are currently developing AspectC, and plan to use it to further explore the structure of elements such as paging in layered system architectures, consistency in client-server architectures, and scheduling in event-based architectures.

## References

[1] AspectC. www.cs.ubc.ca/labs/spl/aspects/aspectc.html.

[2] AspectJ. www.aspectj.org.

[3] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII '01)
1530-1621/01 $10.00 © 2001 **IEEE**

IEEE COMPUTER SOCIETY