# Architecture-Neutral Operating System Components

Daniel Lohmann and Olaf Spinczyk
Friedrich-Alexander University Erlangen-Nuremberg
Martensstr. 1
D-91058 Erlangen, Germany
{lohmann,spinczyk}@informatik.uni-erlangen.de

## 1. INTRODUCTION

The development of a new operating system requires a wide range of decisions at a very early stage. These early decisions cover (and define) fundamental properties and policies of the new OS. They determine *locking strategies*, *module structure* and *interaction*, *interrupt handling* and *synchronization*, supported *hardware architectures* and more. We denote this set of fundamental properties as the *architecture* of the OS.

Almost 25 years ago, Lauer and Needham described the duality of procedure- and process-based operating system structures. They have showed that it is, in principle, possible to transform one into the other [2]. However, in real operating systems architectural properties, like locking and component interaction schemes, tend to be hard-coded into almost every component of the system, mostly for the sake of efficiency. Architectural properties are inherent *crosscutting concerns* which makes it nearly impossible to reuse components in a system with a different architecture.

## 2. ARCHITECTURE ENCAPSULATION

*Aspect-oriented programming (AOP)* has proven to be a promising way to deal with crosscutting concerns. However, until today experiments with AOP in OS code have always been restricted to aspects with a limited scope due to a lack of compiler support [1]. A practical investigation of the benefits for the design and evolution of new system software has been impossible. With AspectC++ [3], an aspect-oriented language extension to C++, our group has recently developed an important tool that enables the application of AOP concepts in the development of efficient system software. In the CiAO project[1] we have now started the development of an aspect-oriented family of operating systems which provide, based on aspects and static configuration, a full encapsulation of architectural properties. CiAO is targeted at the broad area of embedded systems, scaling from very small deeply-embedded devices up to embedded UNIX systems. Therefore, it needs to reach an unattained level of adaptability and configurability. The ambitious end goal is to provide even customization of fundamental architectural properties. For instance, a monolithic kernel, micro kernel, or library OS version should be generated from the same sources if only a specific architecture can fulfill the requirements of the application.

[1] CiAO is Aspect-Oriented

## 3. ARCHITECTURAL EVOLUTION

The encapsulation of architectural properties as aspects provides, furthermore, a promising way to cope with architectural evolution. Architectural evolution tends to be extremely hard and, thus, expensive, but also unavoidable and required on the long term. This lesson could be learned from the integration of SMP support into the Linux kernel. The first kernel release that supported SMP hardware was version 2.0. However, it still used the coarse system-wide locking scheme of earlier versions and therefore performed badly in SMP environments. The unavoidable change of the locking strategy was the trigger for a costly and still on-going process of architectural evolution. Hundreds of device drivers and other operation system components relying on the locking scheme had to be rewritten. If the locking strategy had been encapsulated, e.g. as an aspect, the process of architectural evolution had probably been much cheaper.

## 4. CONCLUSION

The architecture of an operation system is usually seen as something static. Architectural properties are often crosscutting, their implementation is spread over the whole system. This makes it difficult to deal with architectural evolution and nearly impossible to configure fundamental architectural properties on behalf of the target applications demands. We are convinced that new programming paradigms, namely AOP and static configuration, are a reasonable way to reach encapsulation (and therefore maintainability, configurability and reusability) of architectural properties.

## 5. REFERENCES

[1] Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn. Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code. In *Proceedings of the Joint European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Internation Symposium on the Foundations of Software Engineering (FSE-9)*, 2001.

[2] H. C. Lauer and R. M. Needham. On the Duality of Operatings System Structures. In *Proceedings of the Second International Symposium on Operating Systems*, IRIA, Oct. 1978. Reprinted in Operating Systems Review, 13,2 April 1979, pp. 3-19.

[3] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: An Aspect-Oriented Extension to C++. In *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002)*, pages 53–60, Sydney, Australia, Feb. 2002.