Abstract of the dissertation

"Operating System Construction by Aspect Orientation and Program Families"

Olaf Spinczyk

Very small (often called "deeply") embedded systems are forced to operate under extreme resource constraints. Typical hardware configurations in this domain are based on 8 bit microcontrollers equipped with only a few Kbytes of RAM. To get on with these hard limitations an operating system, which targets this special domain, must be specifically tailored to the application. Such tailoring is possible, because typical embedded computer systems run only a single application that will not change until the whole product, in which it is embedded, is replaced.

For the construction of such highly configurable operating systems the **Program Family** concept plays a very important role. A program family allows to model the commonalities and variabilities of the system variants and, thus, helps to find a module structure for the implementation that provides a high degree of code reuse.

However, in some cases the edge of reusability is reached. This happens if the implementation of a certain system feature or functionality *must* be spread widely across the source code. Synchronization code in operating system families is an example for this phenomenon. In common operating system implementations nearly every device driver, the scheduler, and many other subsystems contain synchronization code. This results in a fixed synchronization strategy (e.g. granularity). In a program family such strategy should be configurable according to the needs of the application and, thus, other implementation techniques must be found.

A solution to this problem is promised by the new concept of **Aspect-Oriented Programming** (AOP). With AOP programming language elements called "aspects" are available, which are able to implement the crosscutting nature of features like the synchronization strategy (called "crosscutting concerns") in a modular way.

Until now the realization of an aspect-oriented operating system was virtually impossible due to the lack of adequate programming language support, even though AOP seems to be very promising especially in the operating systems domain. For example, strategic design decisions like the granularity of interrupt synchronization or the association of threads to system components could be implemented in a modular way with AOP. In the context of a family these features can be easily configured according to the application scenario.

The goal of the doctoral research presented in this dissertation was to provide all necessary methodologies and tools to gather experience about the approach with concrete case studies. In these case studies aspect-oriented language features should be applied to implement several crosscutting concerns in an operating system family. To achieve this goal the author first presented an aspect-oriented extension to the model of functional hierarchies, which are normally used to design program families. Furthermore a whole suite of so called aspect weavers was designed and implemented. These weavers are based on a source code transformation tool for C++ code developed by the author as well. They can merge the modular aspect code that implements a crosscutting concern with the regular component code in an aspect-oriented manner. The two special purpose aspect weavers SOSP and COMA can be used to implement a global function inling strategy and the class relations inside of system components in a modular way. They can be used for an application-oriented streamlining of the system code, which has a significant positive impact on the code size and run-time efficiency.

Besides the two special purpose weavers a general purpose aspect weaver was developed. It is a translator for AspectC++, an aspect-oriented language extension to C++. The syntax and semantic of this language is closely related to AspectJ, which is a widely-known aspect language developed at Xerox PARC. However, AspectC++ provides several unique features. One of its most important properties is that AspectC++ requires no fixed runtime system and that the (on demand) generated code is also highly efficient. At the moment AspectC++ is the only general purpose aspect language that is feasible to be used in the context of deeply embedded systems.

The case studies themselves where conducted in the context of the operating system family PURE. They where used to show that the expected advantages of the approach to combine AOP and the program family concept for the construction of application-specific operating systems really exist. For instance, it was shown that only a few aspects were needed to express the interrupt synchronization strategy of PURE. Calls to synchronization primitives, which were originally scattered across 15 classes, could now be modularized in a single aspect. In another case study aspects were used to implement the architecture of device drivers. Now the same basic driver code can be used to implement procedure-oriented or message-oriented drivers with different degrees of concurrency. By modularizing these concerns the strategic decision about the driver architecture can now be made on a higher level of abstraction and its implementation can be easily configured.

These results led to the conclusion that the design and implementation of program families can significantly benefit from aspect orientation. Concerning future work the vision of a fully aspect-oriented operating system family was sketched, which should be constructed from completely architecture neutral system components.