Aspektorientierte Ansätze zur Konstruktion schlanker objektorientierter Betriebssystemsoftware

Olaf Spinczyk
(olaf@ivs.cs.uni-magdeburg.de)

Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, 39106 Magdeburg

7. August 2001

1 Einführung

Im Jahr 2000 hatten 8 Bit μ-Controller noch immer einen Marktanteil von 58% aller produzierten Stücke. Die darauf basierenden kleinen Rechnersysteme besitzen im allgemeinen nur wenige tausend Bytes Hauptspeicher und eine im Vergleich zu heutigen PC Systemen winzige Rechenleistung. Besonders stark ist ihre Verbreitung in der Domäne der "tiefst eingebetteten Systeme", d.h. eingebettete Systeme, bei denen wegen der großen Stückzahl die Hardwarekosten des einzelnen Produkts auf ein Minimum reduziert werden müssen.

Aus der Betriebssystemsicht ist das Besondere an diesem Bereich, dass ein tiefst eingebettetes System ein Spezialzwecksystem ist. D.h., die Aufgabe, für die es konstruiert wurde, ändert sich über seine Lebenszeit hinweg nicht. Anwendung, Betriebssystem und Hardware müssen als eine Einheit begriffen werden, die mit möglichst geringen Kosten diese Aufgabe erfüllt. Das Betriebssystem muss dabei die Anwendung möglichst optimal unterstützen und darf gleichzeitig keine ungenutzten Funktionen enthalten, um den eigenen Ressourcenverbrauch und damit die Hardwarekosten zu minimieren. Aus diesem Grund hielt man bis vor wenigen Jahren die *Programmierung auf der Maschinenebene* für die einzig

praktikable Lösung oder verzichtete sogar völlig auf die Unterstützung durch ein Betriebssystem.

Gleichzeitig ist es aber auch unwirtschaftlich, für jedes Produkt das Rad immer wieder neu zu erfinden. Daraus resultiert für die Konstruktion von Betriebssystemen, dass man in starkem Maße auf Konzepte wie Betriebssystembaukästen oder Betriebssystemfamilien setzen muss. Ziel sollte es sein, aus einem Satz von flexiblen, wiederverwendbaren und lose gekoppelten Bausteinen *anwendungsangepasste Betriebssysteme* zu generieren.

2 Lösungsansatz

Die objektorientierte Betriebssystemfamilie PURE [1] ist eine Entwicklung, die diesem Konzept folgt. Ziel ist es dabei, den Mehraufwand im Vergleich zu handoptimierten Spezialzweckimplementierungen zu minimieren. Um dieses Ziel zu erreichen, ist es notwendig, Techniken zu finden, die den Konflikt zwischen der Spezialisierung und der Wiederverwendbarkeit von Betriebssystemkomponenten lösen. Ein naheliegender Ansatz ist hierzu die Verwendung der statischen Konfigurierung, also der Anpassung der Systemkomponenten zur Übersetzungszeit.

Mit konventionellen Präprozessoren, die bedingte Übersetzung und Makroersetzungen erlauben, wie z.B. dem C Präprozessor oder GNU m4, gelangt man jedoch wegen der enormen Zahl von Konfigurationspunkten bei PURE schnell an die Grenzen der Wartbarkeit. Stattdessen werden Implementierungstechniken eingesetzt, die dem Konzept der *aspektorientierten Programmierung* [4] (kurz AOP) folgen.

AOP ist eine neue Form des Programmierens, die es erlaubt, das Prinzip *separation of concerns* auch dort konsequent zu verfolgen, wo es mit den üblichen von Programmiersprachen gebotenen Kompositionsmechanismen nicht möglich ist. Solche Situationen sind daran zu erkennen, dass Code, der aus einer bestimmten Entwurfsentscheidung resultiert und so einen bestimmten *Aspekt* des Programms implementiert, nicht gekapselt werden kann, sondern weit über das gesamte System verstreut werden muss. Das führt zu Problemen bei Wiederverwendbarkeit und Wartbarkeit. Die aspektorientierte Lösung des Problems besteht darin, durch ein Werkzeug (den sogenannten *Aspektweber*) den Aspektcode mit dem restlichen Code des Programms automatisch verbinden zu lassen. Auf diese Weise entsteht ein hohes Maß an Modularität im Code.

Für PURE wird nun die aspektorientierte Programmierung mit der statischen Konfigurierung kombiniert. Auf diese Weise kann ein einzelner Konfigurations-

punkt im Quelltext Auswirkungen auf das gesamte System haben und so zur Maßschneiderung des Systems für einen bestimmten Anwendungsfall beitragen.

3 Beispiele

Der hier angekündigte Vortrag soll vor allem dazu genutzt werden, zwei konkrete Beispiele für diesen Ansatz vorzustellen. Bei dem ersten Beispiel handelt es sich um die Restrukturierung, d.h. die Modifikation von Klassenbeziehungen, einer Systemkomponente unter Hinblick auf die effizienteste Lösung für ein gegebenes Anwendungsszenario [2]. Der Entwickler implementiert hierzu neben der Komponente selbst, die frei von Konfigurierungscode ist, ein separates, konfigurierbares Aspektprogramm. Dieses beschreibt auf der Abstraktionsebene von Klassen und Klassenbeziehungen, welche Teile der Komponente in bestimmten Konfigurationen tatsächlich benötigt werden und welche Beziehungen gegebenenfalls vereinfacht werden können. Beispielsweise kann so die in wiederverwendbaren objektorientierten Softwaresystemen übliche Nutzung abstrakter Schnittstellen vermieden werden, wenn im gegebenen Anwendungsszenario nur eine Klasse diese Schnittstelle implementiert. Das vermeidet den damit verbundenen Verbrauch von Ressourcen in Form von Speicherplatz für die virtuellen Funktionstabellen, den zusätzlichen Code für die Durchführung des dynamischen Bindens sowie Rechenzeit. Messungen haben gezeigt, dass sich so signifikante Verbesserungen erzielen lassen.

Das zweite Beispiel befasst sich mit der anwendungsabhängigen Findung von Einbettungsentscheidungen (Stichwort “inlining”) für die Funktionen des Betriebssystems [3]. Bei dem vorgestellten Ansatz implementiert ein vom eigentlichen System völlig getrenntes Aspektprogramm einen genetischen Algorithmus, der zur Übersetzungszeit ausgeführt wird. Auch dieses Beispiel soll zeigen, dass durch aspektorientierte Implementierungstechniken in Verbindung mit statischer Konfiguration Spezialisierung und Wiederverwendbarkeit von Systemkomponenten kein Widerspruch sein müssen.

Literatur

[1] D. Beuche, A. Guerrouat, H. Papajewski, W. Schröder-Preikschat, O. Spinczyk, and U. Spinczyk. The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems. 1999. Proceedings the 2nd IEEE In-

ternation Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99), St. Malo, France, Mai 1999.

[2] M. Friedrich, H. Papajewski, W. Schröder-Preikschat, O. Spinczyk, and U. Spinczyk. Efficient Object-Oriented Software with Design Patterns. In *Krzysztof Czarnecki, Ulrich W. Eisenecker (Eds.): Generative and Component-Based Software-Engineering. First International Symposium, GCSE'99*, Erfurt, Germany, Sept. 1999. Springer-Verlag. Revised Papers. LNCS 1799.

[3] A. Gal, W. Schröder-Preikschat, and O. Spinczyk. On Minimal Overhead Operating Systems and Aspect-Oriented Programming. In *Proceedings of the 4th ECOOP Workshop on Object-Orientation and Operating Systems (ECOOP-OOOSWS'2001)*, Budapest, Hungary, June 2001. ISBN 84-699-5329-X.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, June 1997.