Feingranular konfigurierbare Transaktionssysteme für eingebettete Systeme

Mario Pukall
Fakultät für Informatik, Universität Magdeburg
{*pukall*}*@iti.cs.uni-magdeburg.de*

24. April 2007

Zusammenfassung

Eingebettete Rechnersysteme gewinnen immer mehr an Bedeutung. Viele dieser Rechnersysteme benötigen Datenmanagementfunktionalität, als Folge des oftmals hohen Datenaufkommens. Die durch Ressourcenknappheit, z.B. bezüglich Speicherplatz, Rechenleistung und Energieversorgung, gekennzeichnete Domäne der eingebetteten Systeme erfordert die Bereitstellung von anwendungsfallspezifisch minimalen Systemen. Dem wird gegenwärtig durch individuelle Neuentwicklungen Rechnung getragen, die mit hohen Entwicklungskosten verbunden sind. Um die Entwicklungskosten so gering wie möglich zu halten, ist es erforderlich, die Wiederverwendbarkeit einzelner Teile der Datenmanagementfunktionalität zu verbessern. Zu diesem Zweck werden die Funktionen des Systems in Komponenten organisiert, wodurch sie Bestandteil vieler verschiedener anwendungsfallspezifischen Konfigurationen werden können.

Der in dieser Arbeit beschriebene Modularisierungsansatz adressiert einen Teilbereich von Datenmanagementsystemen - die Transaktionsverwaltung. Während in bestehenden Systemen die Konfigurierbarkeit von Speicherverfahren oder Indexstrukturen teilweise vorgesehen wird, ist eine feingranulare Konfiguration der Transaktionsverwaltung problematisch. Die starke Verzahnung von Transaktionsverwaltung mit der übrigen Datenmanagementfunktionalität erschwert ihre Kapselung mittels herkömmlicher kombonentenbasierter Techniken. Im Folgenden wird beschrieben, wie eine feingranulare Konfiguration der Transaktionsverwaltung erreicht werden kann. Kern unseres Modularisierungsansatzes ist die Anwendung moderner Programmierparadigmen, wie der Merkmals- und Aspektorientierten Programmierung, auf die Domäne der Transaktionsverwaltung.

Erklärung. Es handelt sich um eine Vorarbeit, bei der einige Passagen und Ergebnisse mittlerweile in [7] veröffentlicht wurden.

1 Einführung

Die Verbreitung von eingebetteten Rechnersystemen nimmt ständig zu. So werden beispielsweise viele Funktionen im Umfeld von Automobilen durch eingebettete Rechnersysteme kontrolliert oder sogar vollständig gesteuert. Durch die starke Spezialisierung der einzelnen Rechnersysteme auf ein Anwendungsgebiet, werden unterschiedliche Anforderungen an das Datenmanagement gestellt. Aktuelle Datenbankmanagementsysteme stellen meist Lösungen dar, deren Funktionsumfang kaum anpassbar ist. Dies gilt insbesondere für die Transaktionsverwaltung, da diese aufgrund der starken Verflechtung mit anderen Komponenten des Datenmanagements nur schwer modularisierbar und damit konfigurierbar ist. Dieser Artikel zeigt in einer Studie, wie mit Hilfe der kombinierten Merkmals- und Aspektorientierten Programmierung[1] Teile einer Transaktionsverwaltung modularisiert werden können.

2 Softwaretechnischer Hintergrund

Die Modularisierung von Programmfunktionalität ist seit Jahrzehnten Gegenstand von Forschungsarbeiten im Bereich der Softwaretechnik. Derzeitig werden FOP und AOP in diesem Zusammenhang intensiv diskutiert. Beide setzen auf dem Paradigma der Objektorientierten Programmierung auf und erweitern die vorhandenen Mechanismen zur Strukturierung von Software auf unterschiedliche Arten.

[1] engl.: *feature oriented programming - FOP*, *aspect oriented programming - AOP*

Merkmalsorientierte Programmierung. Ziel der Merkmalsorientierten Programmierung ist die Modularisierung von Software auf Basis ihrer Merkmale. Merkmale führen neue Funktionen ein oder erweitern vorhandene [3]. Die Änderungen können über mehrere Programmklassen verteilt sein. Merkmale addressieren in erster Linie heterogene quer schneidende Belange [10], also verschiedenartigen Quellcode, der nicht auf Code-Replikation beruht. Merkmale werden in Merkmalsschichten (auch: *Mixin Layers*) organisiert (Abb. 1 Feature A - C), welche Klassen oder Klassenerweiterungen beinhalten, die das Merkmal implementieren [11].

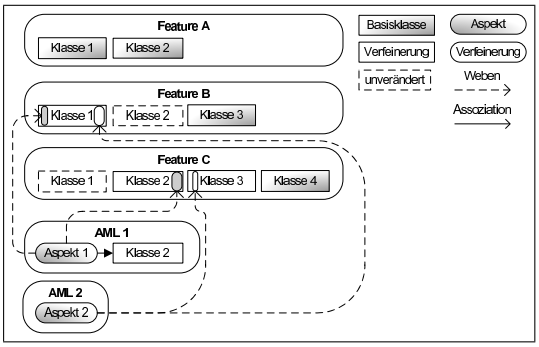

Abbildung 1: Merkmalsschichten und Aspekte

Aspektorientierte Programmierung. Aspekte eignen sich für die Kapselung von *homogenen quer schneidenden Belangen* d.h. von wiederkehrendem Programmcode, also Code der repliziert wird und an verschiedenen Stellen im Programm seine Arbeit verrichtet [9]. Das Einfügen des Aspektcodes an den vordefinierten Programmstellen wird *Weben* genannt (Abb. 1).

Aspectual Mixin Layer. Die Bestandteile eines komplexen Programms adressieren in den wenigsten Fällen entweder homogene oder heterogene quer schneidende Belange, sondern beide Formen. Deshalb ist eine kombinierte Anwendung von FOP und AOP sinnvoll. *Aspectual Mixin Layers* (kurz: AML) vereinen die beiden Techniken [9]. Abbildung 1 zeigt den *AML 1*, welcher sowohl einen Aspekt kapselt, als auch eine Klassenverfeinerung.

3 Verwandte Arbeiten

In der Vergangenheit gab es nur wenige Versuche den Modularisierungsgedanken im Bereich der Transaktionsverwaltung zu konkretisieren. Mit *KIDS* zeigen Geppert et. al in [1] einen Ansatz, bei dem die Belange von DBMS in Subsystemen organisiert werden, welche nach der Konfiguration zu kompletten DBMS zusammengefügt werden. Nyström et. al [4] beschreiben mit COMET ein toolkit-basiertes DBMS, dessen Elemente funktionale Software-Einheiten (Komponenten) sind, die für den konkreten Anwendungsfall (re-)kombiniert werden können. Insbesondere die transaktionsverwaltungsspezifischen Funktionalitäten werden in diesem Ansatz in Aspekten gekapselt. Das PLENTY-System, welches von Hasse entwickelt wurde, zeigt den kernel-basierten Ansatz eines konfigurierbaren DBMS [2]. Alle genannten Ansätze können bezüglich der Transaktionsverwaltung nur unzureichend konfiguriert werden. Das Portfolio der funktionalen Einheiten umfasst jeweils nur wenige, unspezifische Module, die kaum Spielraum für die Entwicklung maßgeschneiderter Lösungen lassen.

4 Studie einer konfigurierbaren Transaktionsverwaltung

Auf der Basis von FOP und AOP erstellte Komponenten sollen es zukünftig ermöglichen, umfassend konfigurierbare Transaktionsverwaltungssysteme zu erstellen. Dies erfordert die Identifizierung der zentralen Eigenschaften der Transaktionsverwaltung.

4.1 Identifikation der Module

Einen Auszug identifizierter Merkmale und Aspekte der Transaktionsverwaltung zeigt Abbildung 2. Die vollständige Beschreibung ist [6] zu entnehmen. Die initiale Merkmalsschicht zeigt Ausschnitte eines Speichermanagers (SM), der als minimales DBMS ohne Transaktionsverwaltung angesehen werden kann. Dieser Speichermanager soll die Verflechtung der Transaktionsverwaltung mit weiteren Teilen eines DBMS andeuten. Der Speichermanager als minimales DBMS kann auf der Basis von FOP und AOP jederzeit funktional erweitert werden. Beispielsweise wurde der Speichermanager um Funktionen zum Sperren der Datensätze ergänzt, als für die

Transaktionsverwaltung das Merkmal des 2-Phasen-Sperrprotokolls eingeführt wurde (Abb. 2 Merkmalsschicht *2PLP*).

Die zweite Merkmalsschicht (BS) umfasst einen Scheduler, der grundlegende Funktionen zur Transaktionsausführung bereitstellt und als minimales Transaktionsverwaltungssystem angesehen wird. In den folgenden Schichten wird das minimale Transaktionsverwaltungssystem um Funktionen, die z.B. den Mehrbenutzerbetrieb und das Recovery ermöglichen, erweitert. Da es sich bei den identifizierten Merkmalen (SM - BRM) um heterogene quer schneidende Belange handelt, empfiehlt sich die Modularisierung der entsprechenden Funktionen durch FOP.

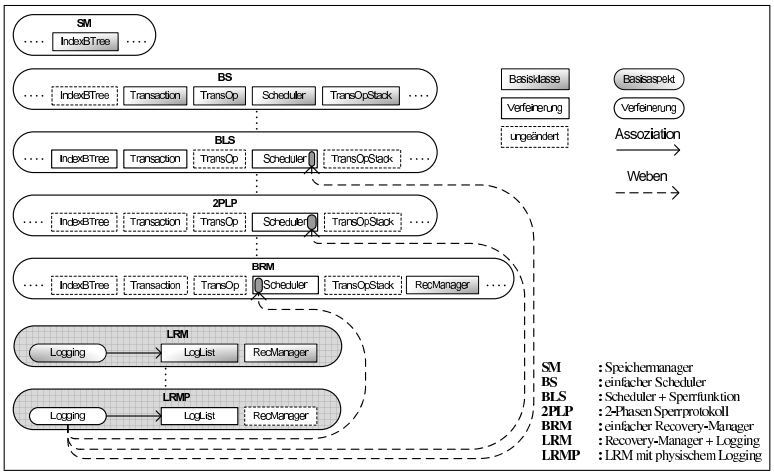

Abbildung 2: Auszug aus Schichtenmodell des Prototypen

Die beiden unteren Schichten (*Aspectual Mixin Layer*) kapseln die Ausführung eines Logbucheintrags. Der Aufruf zum Logbucheintrag muss an verschiedenen Stellen in Klasse *Scheduler* erfolgen, wodurch der Quellcode des Aufrufs repliziert wird. Diese Replikate sind der Klasse der homogenen quer schneidenden Belange zuzuordnen, die mittels AOP modularisiert werden sollten. Die AML *LRMP* stellt eine Verfeinerung des initialen Logging-Aspekts (LRM) dar. Der Aspektcode der beiden AML greift auf die (erweiterten) Methoden der Klassen *LogList* und *RecManager* zu.

```
1  class Scheduler {
2    public:
3      /*Scheduler-Konstruktor*/
4      Scheduler();
5      /*Input-Schedule*/
6      TransOpStack* getInput()
7      void setInput(TransOpStack*);
8      void addInput(TransOp*);
9      /*erzeugt Transaktionsvektor*/
10     vector<Transaktion>* parseOps();
11 };
```

Abbildung 3: Scheduler (BS)

Insgesamt wurden für diese Fallstudie einer Transaktionsverwaltung 10 Merkmale identifiziert und implementiert. Ergänzt werden die Merkmale durch 3 Ausprägungen des Logging-Aspekts (organisiert in 3 Aspectual Mixin Layer).

4.2 Umsetzung

Während der Identifikation der Module wurde festgestellt, dass die kombinierte Nutzung von FOP und AOP ein geeignetes Mittel zur Modularisierung der Transaktionsverwaltung darstellt, weil die transaktionsverwaltungsspezifischen Funktionen sowohl heterogene, als auch homogene quer schneidende Belange beinhalten. Die prototypische Implementierung der Transaktionsverwaltung erfolgt gemäß des Schichtenmodells (Abb. 2) und wird nachfolgend beispielhaft erläutert.

```
1  refines class Scheduler {
2    public:
3      /*2Phasen-Sperrprotokoll*/
4      void _2PL();
5  };
```

Abbildung 4: Scheduler (2PLP)

Merkmale. Eine der zentralen Elemente der prototypischen Implementierung des Transaktionsverwaltungssystems ist die Klasse *Scheduler*. In Abbildung 3 ist ein Ausschnitt der Klassendefinition des Basis-Schedulers zu erkennen. Neben der Konstruktordeklaration sind verschiedene Methodendeklarationen dargestellt, die das Einfügen und Parsen von Transaktionsoperationen betreffen.

In einem der folgenden Verfeinerungsschritte (siehe Abb. 2 2PLP) wird dem Scheduler die Funktionalität des 2-Phasen-Sperrprotokolls hinzugefügt. Abbildung 4 zeigt die zugehörige Klassendefinition.

Aspekte. Der implementierte Aspekt (siehe Abb. 2 LRM) des Prototypen kapselt die Ausführung eines Log-Eintrags. In der Klasse *Scheduler* ist immer genau dann ein Eintrag in das Logbuch notwendig, wenn eine Transaktionsoperation ausgeführt wurde.

```
1  aspect Logging {
2   virtual void logExecution(TransOp* n,
3                  Scheduler* s, TransOpStack* t){
4    /*Aufruf der LogList-Methode*/
5   }
6   /*Pointcut*/
7   pointcut log(TransOp* n, Scheduler* s,
8                  TransOpStack* t)
9                  =execution
10                 ("% Scheduler::schedule_pop()")
11                &&result(n)&&that(s)&&args(t);
12  /*Advice*/
13  advice log(n, s, t):after
14                       (TransOp* n, Scheduler* s,
15                       TransOpStack* t){
16    logExecution(n, s, t);
17  }
18 };
```

Abbildung 5: Aspekt Logging

Jedes Protokoll (sperrenbasiert, etc.), das zur Ausführung eines Schedules verwendet wird, muß daher den Logging-Aufruf implementieren. Abbildung 5 zeigt einen Ausschnitt der Implementierung des Basisaspekts. Die Zeilen 7-11 definieren die Stelle (auch: *Pointcut*), an welcher der erweiternde Code (Advice - Zeile 16) in das Programm eingewebt wird.

5 Evaluierung

Ziel der prototypischen Implementierung ist die Verbesserung der Konfigurierbarkeit und Wiederverwendbarkeit von Transaktionsverwaltungssystemen. Zu diesem Zweck wurde eine feingranulare Modularisierung der transaktionsverwaltungsspezifischen Funktionen auf der Basis von FOP und AOP vorgenommen. Die resultierenden Module können so (re-)kombiniert werden, dass anwendungsfallspezifisch minimale/maßgeschneiderte Transaktionsverwaltungssysteme entstehen. Nachfolgend werden die Vorteile dieser Lösung evaluiert.

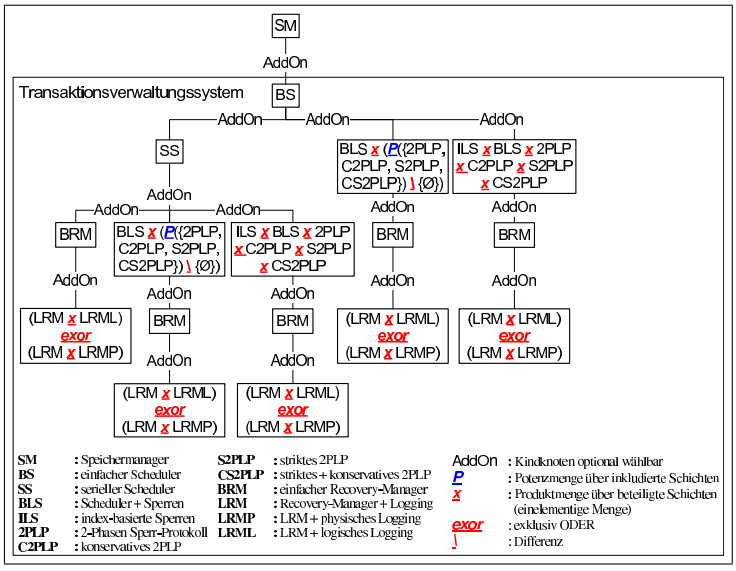

Abbildung 6: Kombinationsbaum

Konfigurierbarkeit und Wiederverwendbarkeit. Die entwickelten Module können zu Transaktionsverwaltungssystemen von unterschiedlichstem Funktionsumfang (re-)kombiniert werden. Wobei durch die Angabe von Entwurfsregeln die Menge der möglichen Kombinationen auf die Menge der sinnvollen Kombinationen reduziert werden kann. Abbildung 6 zeigt einen Kombinationsbaum, der die Menge der zulässigen Konfigurationen des Prototypen zeigt. Dem Speichermanager (SM) können weitere Module (Schichten) hinzugefügt werden, sodass jeder Pfad im Baum eine funktional einzigartige Konfiguration darstellt. Die Berechnung der Anzahl gültiger und sinnvoller Programmkompositionen führt zu folgender Gleichung:

$$Kombinationen_T = 2 + (4) + (15 * (3)) + (4)) + (15 * (3)) + (4) = 104$$

Ausgehend von 13 wählbaren Modulen, stellt dieser Wert ein beachtliches Ergebnis dar und bescheinigt dem entwickelten System ein hohes Maß an Konfigurierbarkeit. Unter Verwendung der entwickelten Module kann auf nahezu jedes Anwendungsszenario mit einem maßgeschneiderten Transaktionsverwaltungssystem reagiert werden.

Ressourcen. Konfiguration für den konkreten Anwendungsfall bezeichnet das Wählen von benötigten Modulen und das Weglassen von nicht benötigten Modulen. Je weniger Funktionen/Module benötigt werden, desto weniger Speicherplatz belegt das resultierende Transaktionsverwaltungssystem. Beispielsweise belegt ein Transaktionsverwaltungssystem, welches nur die serielle Transaktionsausführung erlaubt 57,8 Kilobyte Speicherplatz, während das funktional vollständige System 273 Kilobyte Speicherplatz beansprucht.

Performance. Im Bereich der Datenbankenprogrammierung hat sich die Verwendung von objektorientierter Programmierung und deren Erweiterung (hier: FOP, AOP) noch nicht durchgesetzt, weil Einbußen in der Programmgeschwindigkeit befürchtet werden. Rosenmüller et. al [8] zeigen, dass die Verwendung der objektorientierten Programmiersprache C++ gegenüber der imperativen Programmiersprache C keine Performanceeinbußen in der Datenbankenprogrammierung mit sich bringt. Kuhlemann et. al [5] stellen fest, dass die Performance von Software, welche auf der Basis von FOP implementiert wurde, vergleichbar ist mit der Performance von objektorientiert entwickelter Software.

6 Zusammenfassung

Im Blickpunkt dieser Arbeit standen die Möglichkeiten der Modularisierung der Transaktionsverwaltung als Teil komplexer DBMS, unter dem Gesichtspunkt maximaler Konfigurierbarkeit und Wiederverwendbarkeit. Es wurde gezeigt, dass die kombinierte Anwendung von FOP und AOP eine feingranulare Modularisierung und Konfigurierung der Transaktionsverwaltung ermöglicht. Darüber hinaus konnte die Eignung der modularisierten Transaktionsverwaltung für einen Einsatz im Bereich der eingebetteten Systeme nachgewiesen werden.

Literatur

[1] A. Geppert and S. Scherrer and K.R. Dittrich. KIDS: Construction of Database Management Systems based on Reuse. Technical report, 1997.

[2] C. Hasse. *Inter- und Intratransaktionsparallelität in Datenbanksystemen: Entwurf, Implementierung und Evaluation eines Datenbanksystems mit Inter- und Intratransaktionsparallelität.* PhD thesis, Departement Informatik, ETH Zürich, 1995.

[3] D. Batory. A Tutorial on Feature Oriented Programming and the AHEAD Tool Suite. *Summer school on Generative and Transformation Techniques in Software Engineering*, 2005.

[4] D. Nyström and A. Tešanović and C. Norström and J. Hansson. The COMET Database Management System. Technical report, Mälardalen University, 2003.

[5] M. Kuhlemann and S. Apel and T. Leich. Streamlining Feature-Oriented Designs. In *Software Composition*, 2007.

[6] M. Pukall. FAME-DBMS: Entwurf ausgewählter Aspekte der Transaktionsverwaltung. Diplomarbeit, Fakultät für Informatik, Universität Magdeburg, 2006.

[7] M. Pukall and T. Leich and M. Kuhlemann and M. Rosenmüller. Highly Configurable Transaction Management for Embedded Systems. In *AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, 2007.

[8] M. Rosenmüller and T. Leich and S. Apel. Konfigurierbarkeit für ressourceneffiziente Datenhaltung in eingebetteten Systemen am Beispiel von Berkeley DB. In *Datenbanksysteme in Business Technologie und Web – Workshop Proceedings.* Verlag Mainz, 2007.

[9] S. Apel and T. Leich and G. Saake. Aspectual mixin layers: aspects and features in concert. In *Proceeding of the 28th international conference on Software engineering (ICSE '06)*, 2006.

[10] S. Apel and T. Leich and M. Rosenmüller and G. Saake. Combining Feature-Oriented and Aspect-Oriented Programming to Support Software Evolution. In *Proceedings of the 2nd ECOOP Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'05)*, 2005.

[11] Y. Smaragdakis and D. Batory. Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs. *ACM Transactions on Software Engineering and Methodologies (TOSEM)*, 11(2):215–255, 2002.