Ein merkmalsorientierter Speichermanager für eingebettete Systeme

Thomas Leich und Sven Apel
{*leich*|*apel*}*@iti.cs.uni-magdeburg.de*
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg

Zusammenfassung

Der Bereich der eingebetteten Systeme ist ein bedeutender Markt. Einsatzgebiete sind beispielsweise Autosteuerungen oder Sensornetzwerke. Häufig benötigen derartig eingebettete Rechnersysteme Infrastruktursoftware zur Datenhaltung. Auf Grund der Heterogenität der Hard- und Software, sowie der extremen Ressourcenbeschränkungen, ist eine Adaption der klassischen Mehrzwecksysteme aus Großrechnern oder dem PC-Bereich nicht möglich. Eine Maßschneiderung der Datenmanagementfunktionalität auf den jeweiligen Anwendungskontext, sowie auf die Hard- und Softwareumgebung ist unumgänglich. Die hierfür benötigten Komponenten müssen leichtgewichtig, minimal und feingranular sein. Eine Kombination von Komponententechniken und merkmalsorientierter Programmierung bietet Möglichkeiten zur Überwindung der genannten Probleme. Im Folgenden werden der Entwurf und die Implementierung eines Speichermanagers im Sinne einer Programmfamilie mittels merkmalsorientierter Domänenanalyse und merkmalsorientierter Programmierung beschrieben. Als Bewertungsgrundlagen dienen die Anzahl der Variationspunkte und die potentielle Anzahl sinnvoller Systemvarianten.

1 Einleitung und Motivation

Der Markt für eingebettete Systeme wächst stark [4]. Bei der Entwicklung eingebetteter Rechnersysteme sind die Kosten für Hardware auf Grund der zumeist hohen Stückzahlen ein sehr wichtiger Faktor. Daher kommt es bei der Softwareentwicklung darauf an, den Ressourcenbedarf zu minimieren, um so preisgünstige Hardware einsetzen zu können. Das Ergebnis ist typischerweise die Entwicklung von Spezialzwecksoftware, die für den konkreten Anwendungsfall zugeschnitten ist und somit keinen unnötigen Speicher oder Rechenzeit verbraucht. Dies erschwert die systematische Wiederverwendung, Wartung, Erweiterung und Anpassung der Software enorm. Heutige Infrastruktursoftware zur Datenhaltung, die üblicherweise im Großrechner- und PC-Bereich eingesetzt wird, kann auf die Heterogenität der Hard- und Software und auf die starken Ressourcenbeschränkungen nicht adäquat reagieren. Neue Softwaremethoden aus dem Bereich der Komponententechniken, Programmfamilien und der merkmalsorientierten Entwicklung können helfen diese Schwierigkeiten zu überwinden ohne die Vorteile eine Spezialzwecksoftware zu verlieren [5]. Dieser Beitrag zeigt erste Ergebnisse aus dem Entwicklungsprozess eines merkmalsorientierten Speichermanagers (SM) in Form einer Programmfamilie. Aus der SM-Familie können speziell für den Anwendungskontext zugeschnittene leichtgewichtige Datenhaltungskomponenten abgeleitet werden. Die Ergebnisse in Hinsicht auf eine feingranuale Maßschneiderbarkeit des SM sollen an einem Beispielszenario aus dem Bereich der Sensornetzwerke demonstriert werden. Der Beitrag ist wie folgt gegliedert: Abschnitt 2 erläutert ein Beispielszenario aus dem Bereich der Sensornetzwerke und arbeitet unterschiedliche Anforderungen an einen SM in diesem Bereich heraus. In Abschnitt 3 werden die verwendeten Softwaretechniken vorgestellt. Der folgende Abschnitt präsentiert den Entwurf und die Implementierung des merkmalsorientierten SM. Abschnitt 5 diskutiert und vergleicht die erreichten Ergebnisse. Eine Zusammenfassung und einen Ausblick gibt Abschnitt 6.

2 Beispielszenario

Im Bereich der drahtlosen Sensornetzwerke konzentrierte sich die Forschung der letzten Jahren vor allem auf Anfragebearbeitung in Netzwerken und (Vor-) Aggregationsalgorithmen [6]. Zentralisierte Datenhaltung und Analyse ermöglichen den Einsatz preiswerter Sensorenknoten mit geringem Ressourcenverbrauch. Oft sind Sensornetzwerke für den Langzeitbetrieb entwickelt und somit auf Grund der knappen Ressourcen in der Kommunikation stark eingeschränkt [6]. Voraggregationen von Rohdaten auf den Sensorknoten selbst sollen helfen den Speicherverbrauch zu minimieren und die Kommunikationsbeschränkungen zu überwinden. Dies ist nur möglich, wenn die zu untersuchenden Merkmale a priori bekannt sind. Gerade in wissenschaftlichen Anwendungsbereichen wie der Biotopüberwachung ist dieses nicht immer möglich [6]. In Anlehnung an [12, 6] wird im Folgenden ein Versuchsaufbau zur Überwachung eines Biotops vorgestellt. Einfache *Sensorknoten (1)* messen und speichern Daten wie z. B. Temperatur oder Lichtintensität. Die jeweilige Record-Länge ist für einen Sensorknotentyp fest. Des Weiteren benötigen die Sensoren eine einfache Speicherstruktur mit Einfüge- und Suchfunktionalität. Eine Löschfunktion ist z. B. nicht notwendig, da die im Hauptspeicher gehaltenen Daten periodisch mit einem „Reset“ zurücksetzt werden. *Datenkollektoren (2)* bilden die zweite Gerätegruppe, welche die Daten verschiedener Sensoren je nach Bedarf der Wissenschaftler „zusammentragen“, und Informationen für interne und externe Analysen bereitstellen. Für eine sichere und dauerhafte Speicherung wird eine Sekundärspeicherung verwendet, welche durch ein Buffermanagement optimiert wird. Die Speicherstruktur muss für die jeweilige Analyse geeignet sein und entsprechende Operationen anbieten. Des Weiteren werden Integritätschecks benötigt. Verfügbare Datenmanagementdienste können ein solches Spektrum unter Berücksichtigung der limitierten Ressourcen nicht abdecken. Einige Dienste bieten eingeschränkte Adaptionsmöglichkeiten. Diese sind nicht ausreichend, um auf die extremen Ressourcenbeschränkungen zu reagieren.

3 Softwaretechnischer Hintergrund

Für die Umsetzung des SM in Form einer Programmfamilie wurden die Konzepte der merkmalsorientierten Domänenanalyse (engl. Feature-Oriented Domain Analysis (FODA)), der schrittweisen Verfeinerung und der merkmalsorientierten Programmierung (engl. Feature-Oriented Programming (FOP)) angewendet. Für die konkrete Implementierung wurden Mixin Layer verwendet.

Merkmalsorienierte Domänenanalyse: Ziel der FODA ist die Eingrenzung der Zieldomäne und die Identifizierung der gemeinsamen und unterschiedlichen Merkmale der Zielapplikationen [5]. Die Ergebnisse werden in einem Baumdiagramm zusammengefasst. Diese abstrakte und implementierungsunabhängige Darstellung gibt einen Überblick über die Variationspunkte der Zielanwendungen. Optionale *(C,D)* und zwingende *(B)* Merkmale werden durch einen leeren bzw. gefüllten Kreis graphisch gekennzeichnet. Des Weiteren können Merkmale zu Oder- (G,H) und Alternativ- (E,F) Gruppierungen zusammengefasst werden, welche durch leere bzw. gefüllte Halbbögen dargestellt werden.

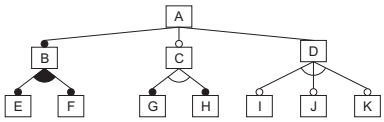

Abbildung 1: Merkmalsbaum

Programmfamilien und Schrittweise Verfeinerung: Den Begriff der Programmfamilie führte Parnas [9] ein. Familienmitglieder sollen aus einer gemeinsamen Basis abgeleitet werden. Eine Umsetzung kann durch *schrittweise Verfeinerung* [10] erfolgen. Beginnend mit einer minimalen Basis wird durch Hinzufügen von Schichten die Funktionalität inkrementell erweitert. Durch ein einfaches Hinzufügen, Austauschen und Weglassen einzelner Schichten wird Erweiterbarkeit, Wartbarkeit und Anpassbarkeit ermöglicht. Batory et al. [10] überführten diesen Ansatz in die Objekt-Orientierte Welt und erkannten, dass neue *Software-Merkmale* oft eine Erweiterung oder Modifikation mehrerer Klassen nach sich zieht. Basierend auf diesen Beobachtungen erkannten sie, dass Merkmale *Kollaborationen von Klassen und/oder Objektfragmenten* sind.

Abb. 2 zeigt Schichten von Kollaborationen. Klassen werden vertikal (c_1 – c_3) und Kollaborationen horizontal (f_1 – f_3) angeordnet. Verschiedene Merkmale bilden als Schichten von Kollaborationen die gewünschte Software. Im Sinne von FOP [10] implementiert eine Kollaboration von Objekten ein Merkmal [10][1]. *Mixin Layer* stellen eine mögliche Implementierungstechnik für die Umsetzung der Merkmale in Form von Kollaborationen dar [10]. Sie kapseln Fragmente verschiedener Klassen statisch. Dadurch wird ein hoher Grad an Modularisierung und ein einfaches Zusammensetzen der Schichten gewährleistet. AHEAD [1] stellt z. B. eine Spracherweiterung für Java bereit, um Klassen in Form von Mixins zu organisieren.

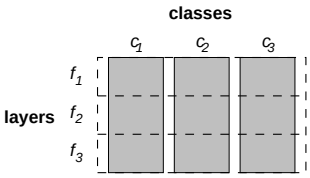

Abbildung 2: Kollaborationsschichten

4 Analyse, Entwurf und Implementierung

Im Folgendem wird die Analyse, der Entwurf und die Implementierung des SM beschrieben.

FODA: Abb. 3 zeigt einen Ausschnitt des Merkmalsdiagramms, welches sich aus der Domänenmodellierung ergibt. Die Merkmale in den grauen Boxen können weiter verfeinert werden. Der SM teilt sich in vier zwingende Merkmale: 1. *Data Type (DT)* repräsentiert die unterstützten Datentypen , 2. *Buffer Management (BM)* organisiert die primäre bzw. sekundäre Speicher- und die Freispeicherverwaltung, 3. *Storage Organisation (SO)* übernimmt die Zugriffsverwaltung der Daten und 4. *Record (Rec)* repräsentiert den internen Aufbau der Daten. Optional können verschiedene Integritätschecks *Integrity Checks (IC)* und Dateitypen *File Types (FT)* ausgewählt werden. Der B^**-Baum* mit seinen Operationen als Submerkmalen ist ein Beispiel für den feingranularen Entwurf. Nur dieser feingranulare Entwurf erlaubt eine Maßschneidung für stark ressourcenbeschränkte Umgebungen. Eine genauere Untersuchung würde hunderte Merkmale hervorbringen und führt schnell zu unübersichtlichen Merkmalsbäumen. Ein weiteres Problem ist, dass Merkmalsdiagramme in ihrer Aussagekraft beschränkt sind. So benötigen Änderungsoperationen eine Sucheoperation. Hierfür gibt es unterschiedliche Erweiterungen der Merkmalsdiagramme. Für den hier vorgestellten Speichermanager wurden diese Abhängigkeiten durch das Konzept der *Design Rule Checks (DRC)* [3] modelliert. DRC werden auf Implementierungsebene eingesetzt. Das gesamte Merkmalsmodell der SM-Familie umfasst 93 Merkmale. Auf eine Untersuchung von speziellen Datentypen, Transaktionsmechanismen oder Mehrbenutzerverfahren wurde verzichtet. Zur Bestimmung der Anzahl möglicher Varianten, die aus der SM-Familie entstehen können, werden die in Tab. 1 beschriebenen Annahmen getroffen. So werden z. B. durch die SM-Familie vier unterschiedliche Datentypen unterstützt. Mit Hilfe einer GenVoca-Grammatik [2] können für die präsentierte SM-Familie

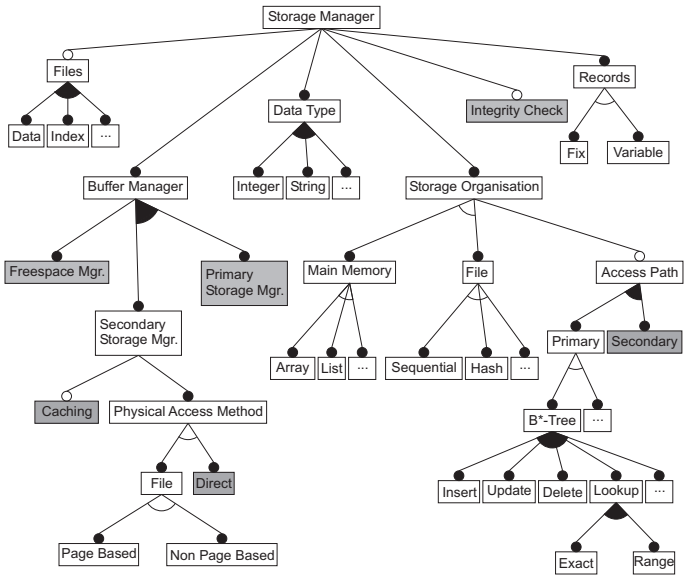

Abbildung 3: Merkmalsdiagramm des SM

[1] Im Folgenden werden Schichten und Merkmal deshalb synonym verwendet.

Parameter	Merkmal	# für Berechnung
d	Data Types	4
f	File Type	2
m	Main Memory Organisation	2
s	Data File Structures	2
a	Access Structure	2
o	B^* Tree	6

Tabelle 1: Parameter und Werte für die Berechnung

8.164.800 Varianten berechnet werden. Die Unterschiede zwischen einzelnen Varianten sind minimal, so dass die Forderung nach einer feingranularen Konfiguration erfüllt ist.

$$\#SM = \underbrace{(2^f - 1)}_{FT} * \underbrace{(15)}_{BM} * \underbrace{(2^n - 1)}_{DT} * \underbrace{((m + s) * (6 * a * (2^o - 1)))}_{SO} * \underbrace{(2)}_{IC} * \underbrace{(2)}_{Rec}$$

Entwurf und Implementierung: Zu Evaluierungszwecken wurde der SM mit Hilfe der *AHEAD-ToolSuite* umgesetzt. Abb. 4 zeigt einen Ausschnitt der implementierten Schichten. Beginnend mit der Basis wird der SM in jeder Schicht erweitert bzw. modifiziert. Dies geschieht durch das Hinzufügen von Klassen und/oder das Verfeinern bereit existierender Klassen. Für den SM wurden 36 Klassen und wenige Hilfsklassen implementiert. Im Durchschnitt wurden 3 Klassen pro Schicht verfeinert. Ein Beispiel hierfür ist das Merkmal *Page Based Storage*. In dieser Schicht werden die Klassen: `File`, `FreeSpaceMgr`, `SecStorageMgr` verfeinert und die Klasse `Page` hinzugefügt. Die AHEAD-ToolSuite generiert aus den in Abb. 4 dargestellten Klassen eine Vererbungshierarchie.

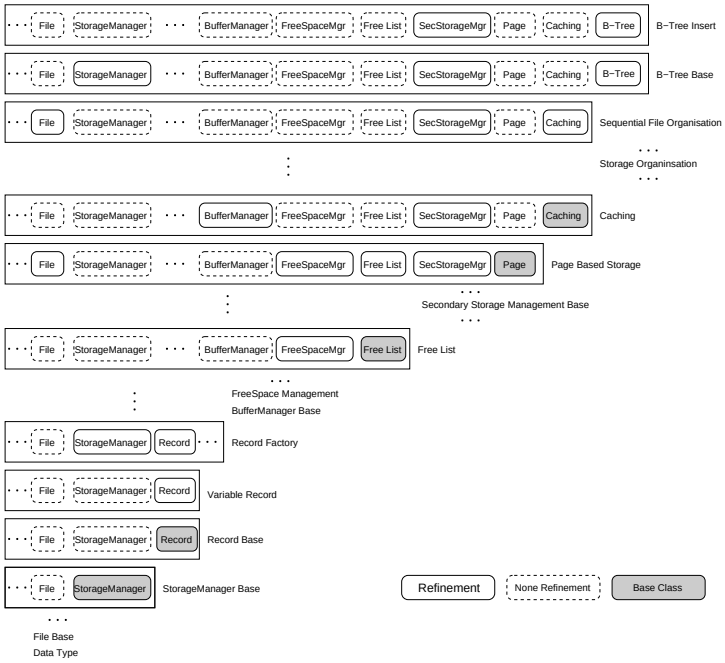

Abbildung 4: Layer Stack

5 Ergebnisse und Vergleich

Zu Evaluierungszwecken wurde für die im Abschnitt 2 eingeführten Szenarien eine Prototypenfamilie implementiert, von der verschiedene Konfigurationen abgeleitet werden können. Expemplarisch werden die Sensorknoten- und Datenkollektorenkonfiguration vorgestellt. *Sensorknoten:* Es wurden zwei Varianten konfiguriert. Beide Varianten sind Hauptspeicherdatenbanken mit einer statischen Speicherverwaltung (feste Record-Längen). In der ersten Sensorknotenvariante werden einfache Datentypen (integer, number) in einem Array gespeichert. Diese Konfiguration besteht aus 11 Schichten. Für die zweite Variante wurde statt einem Array eine Hash-Speicherstruktur verwendet. Dadurch können effiziente Such- und Änderungsoperationen unterstützt werden. Des Weiteren ist ein einfacher Integritätscheck vorhanden. Dieser SM besteht aus 16 Schichten. *Datenkollektoren:* Auf Grund der in Abschnitt 2 beschriebenen Anforderungen ist die Funktionalität der Datenkollektoren komplexer. Für die Sekundärspeicherverwaltung wurde eine seitenbasierte Speicherverwaltung konfiguriert und zur Überwindung der Zugriffslücke ein Cache-Management

eingeführt. Die Dateien sind sequenziell geordnet. Als Zugriffstruktur wurde ein B^*-Baum ausgewählt. Im Gegensatz zu den Sensorknoten wird im Datenkollektor ein variable Record-Länge verwendet, um auf die verschiedenen Datenformate der Sensoren effizienter reagieren zu können. Der eingefügte Integritätscheck ist im Vergleich zu dem des Sensorknotens komplexer. Die gesamte Konfiguration dieses SM wird durch 38 Schichten realisiert. Die semantische Richtigkeit der Konfigurationen wird durch DRC sichergestellt. Durch DRC konnten ca. 60 % aller Konfigurationen, die aus dem Merkmalsdiagramm möglich sind, ausgeschlossen werden.
Ein Vergleich zu Berkley DB soll Unterschiede zu bisherigen Lösungen verdeutlichen. Berkley DB liegt in einer C- und Java-Implementierung vor. Das DBMS ist auf der Basis seiner 4 Teilsysteme konfigurierbar: dem Zugriffspfadmanagement, der Speicher-, Transaktions- und Sperrverwaltung. Durch Kombination bzw. Weglassen einzelner Subsysteme wird die Konfiguration ermöglicht. Dieses ist eine vergleichsweise grobe Variabilität zu dem präsentierten Ansatz. Änderungen in einem Teilsystem wie z. B. der Zugriffpfadverwaltung sind möglich, aber auf Grund der Abhängigkeiten in den Modulen sehr schwierig. Tesanovic et. al [11] bestätigten diese Aussage durch Untersuchungen im Bereich der crosscutting concerns(CC). CC sind bekannt dafür, dass sie eine Erweiterung in Softwaresystemen behindern. Durch die Auslagerung einiger CC in Aspekte konnte die Codegröße um 57 % reduziert werden. Dies zeigt die hohe zyklomatische Komplexität in DBMS und die damit verbundenen Schwierigkeiten für die Konfigurier- und Erweiterbarkeit. Im COMET-Projekt [7] wurden viele Abhängigkeiten eines SM in Module und Aspekte getrennt. Auch bei der Entwicklung mit AOP besteht ein Merkmal typischerweise aus Klassen und Aspekten. Kohäsive Module, bei denen Klassen und Aspekte ein Merkmal zusammenfassen sind laut Lopez-Herrejon et al. [8] mit AOP nicht möglich. Dies führt vor allem in großen Projekten wie z. B. bei der Entwicklung von SM zu Problemen.

6 Zusammenfassung und Ausblick

Merkmalsorienierte Softwareentwicklung und schrittweise Verfeinerung ermöglichen eine variable Implementierung von Datenmanagementdiensten für eingebettete Systeme. Hierzu wurde die Kombination von FODA, FOP und Mixin Layer zur Implementierung eine SM-Familie beschrieben. Ein kleine Anzahl an Grundfunktionen wurden analysiert und implementiert, um einen hohen Grad an Variabilität und Maßschneiderung an einem Beispielszenario aus dem Bereich der Sensornetzwerke nachzuweisen. Durch einen einfachen Konfigurationsprozess konnten drei verschiedenen Varianten des SM auf verschiedene Anwendungsszenarien zugeschnitten werden. Zukünfige Arbeiten konzentrieren sich auf eine Umsetzung des SM mit C++. Hierzu wird C++ um FOP-Funktionalitäten erweitert. Des Weiteren werden Kombinationen von FOP und AOP für die Umsetzung evaluiert.

Literatur

[1] D. Batory. Feature-Oriented Programming and the AHEAD Tool Suite.
[2] D. Batory and S. O'Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology*, 1(4):355–398, 1992.
[3] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. In *Proc. of the 25th Int. Conf. on Software Engineering*, 2003.
[4] Business Communications Company. Future of Embedded Systems Technology, 2000. BCC Press release on market study RG-229.
[5] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
[6] D. Ganesan et. al. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of the ACM SenSys Conference*, pages 89–102, Los Angeles, California, USA, November 2003. ACM.
[7] D. Nyström et. al. Comet: A component-based real-time database for automotive systems. In *ICSE'04)*, Edinburgh, Scotland, May 2004. IEEE Computer Society Press.
[8] R. Lopez-Herrejon, D. Batory, and W. Cook. Ëvaluating Support for Features in Advanced Modularization Technologies. In *(ECOOP)*, 2005.
[9] D. L. Parnas. Designing Software for Ease of Extension and Contraction. *IEEE*, SE-5(2), 1979.
[10] Y. Smaragdakis and D. Batory. Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs. *ACM TOSEM*, 11(2), 2002.
[11] A. Tešanović, K. Sheng, and J. Hansson. Application-tailored database systems: a case of aspects in an embedded database. In *IDEAS*, Coimbra, Portugal, July 2004. IEEE Computer Society Press.
[12] Alec Woo, Sam Madden, and Ramesh Govindan. Networking support for query processing in sensor networks. *Commun. ACM*, 47(6):47–52, 2004.