MARKO ROSENMÜLLER · THOMAS LEICH · SVEN APEL · GUNTER SAAKE

Von Mini- über Micro- bis zu Nano-DBMS: Datenhaltung in eingebetteten Systemen

Funktionsumfang und Komplexität von Datenmanagementsystemen (DMS) nehmen fortwährend zu. Die tatsächlich benötigte Funktionalität wird dabei oft außer Acht gelassen und für verschiedenste Anwendungsszenarien die gleiche Software verwendet. Aufgrund unterschiedlicher Hardwarearchitekturen und Ressourcenbeschränkungen führt dies insbesondere in eingebetteten Systemen zu Problemen. So müssen oftmals für unterschiedliche Plattformen und Anwendungsprofile Datenmanagementlösungen von Grund auf neu entwickelt werden. In diesem Artikel diskutieren wir aktuelle Probleme der eingebetteten Datenhaltung sowie Techniken zur Strukturierung und Modularisierung von Software, die die Erstellung maßgeschneiderter DMS erlauben. Dabei gehen wir insbesondere auf das Datenmanagement in eingebetteten Systemen ein, da hier maßgeschneiderte Lösungen dringend benötigt werden.

1 Einführung

Das Gebiet der eingebetteten Systeme ist eines der stark expandierenden Teilgebiete der Informationstechnologie [BCC 2000]. Etwa 98% aller weltweit hergestellten Rechnersysteme sind als eingebettete Systeme im Einsatz [Turley 2002]. Mit Szenarien wie *Pervasive* und *Ubiquitous Computing* wird dieser Trend fortgesetzt [Weiser 1993]. Als eingebettete Systeme werden Rechnersysteme bezeichnet, die mithilfe einer zugehörigen Software ihre Aufgabe in einem (übergeordneten) Produkt verrichten[1]. Häufig erledigen sie dabei ihre Aufgaben für den Endnutzer vollkommen transparent.

Eine zentrale Aufgabe dieser Systeme ist die effiziente Datenverarbeitung und -verwaltung. Das Datenaufkommen, das von eingebetteten Systemen zu bewältigen ist, steigt seit Jahren kontinuierlich an. Folglich benötigen auch diese teilweise sehr leistungsschwachen Geräte ein adäquates Datenmanagement.

Beispielanwendungen aus dem Bereich der Sensornetzwerke [Akyildiz et al. 2002, Ganesan et al. 2003, Woo et al. 2004] und aus dem Automobilbau [Nyström et al. 2002] zeigen, dass derartig eingebettete Rechnersysteme stark variierende Elemente klassischer Infrastruktursoftware zur Datenhaltung benötigen. Eine Adaption vorhandener Mehrzwecksystemlösungen für diesen Bereich des Datenmanagements aus dem Großrechner- oder dem PC-Bereich ist aufgrund der Heterogenität der Hard- und Software sowie der extremen Ressourcenbeschränkungen nicht möglich. Existierende Analysen [Nyström et al. 2004] vorhandener Forschungsprototypen und kommerzieller *Datenbankmanagementsysteme* (DBMS) bzw. Datenhaltungskomponenten für den Bereich der eingebetteten Systeme zeigen, dass der Großteil dieser Systeme auf Forschungsergebnissen der späten 80er- bzw. frühen 90er-Jahre basiert. Die Berücksichtigung aktueller Entwicklungen der Softwaretechnik oder der Programmiersprachen hat seitdem nur in Ansätzen stattgefunden. Resultat ist die in der Praxis anzutreffende wiederkehrende Entwicklung ähnlicher Datenhaltungskomponenten als Teil der Anwendungssoftware. Diese sind auf den konkreten Anwendungsfall zugeschnitten und verbrauchen somit keinen unnötigen Speicher oder Rechenzeit. Solche Neuentwicklungen sind nicht nur sehr kosten- und zeitintensiv und verhindern eine schnelle Markteinführung, sondern führen zudem häufig zu Qualitätseinbußen. Die Hauptgründe hierfür liegen in den eingesetzten Programmiersprachen und -konzepten (Assembler und C), die nachweislich Defizite bezüglich einer systematischen Wiederverwendung besitzen [Booch 1990].

Die Verwendung von DMS als Infrastruktursoftware ist in allen Bereichen der Softwareentwicklung anzutreffen. Während datenintensive Anwendungen oft klassische DBMS und eine Anfragesprache wie SQL benötigen, existieren auch Anwendungen, deren Anfragen von vornherein feststehen und so z.B. auf SQL und zugehörige Komponenten wie Anfrageoptimierer verzichten können. Wiederum andere Anwendungen benötigen sehr geringe Datenmanagementfunktionalität.

Eine Gliederung dieser sehr unterschiedlichen DMS zeigt Abbildung 1. Die verwendeten Bezeichnungen wurden entsprechend dem Funktionsumfang gewählt, den die einzelnen Datenmanagementlösungen bieten. So werden Systeme als *Macro-DMS* bezeichnet, die einen großen Teil der verfügbaren Funktionalität der DMS-Domäne bieten und so z.B. den gleichzeitigen Zugriff auf den Datenbestand von mehreren Clients mit einer Anfragesprache ermöglichen. *Nano-DMS* hingegen bieten nur geringe Funktionalität wie Persistenz einzelner Datenstrukturen.

Typische Anwendungen aus dem Macro-Bereich sind klassische DBMS, wobei hier aber eine Abgrenzung zu DMS vorgenommen wird, da im Micro- bis Nano-Bereich die nach Codd definierten Eigenschaften nicht zutreffen. Bei der Zuordnung zum jeweiligen Bereich fällt zudem auf, dass einige Systeme (wie z.B. Sybase Anywhere) mehreren Bereichen zugeordnet werden können.

Neben einer Einordnung der DMS entsprechend angebotener Funktionalität findet sich in Abbildung 1 auch eine Zuordnung zum primären Anwendungsgebiet. So werden im Bereich einge-

1. Neben eingebetteten Systemen wird der Begriff eingebettetes Datenmanagement verwendet, der die Integration des Datenmanagements in eine Anwendung bezeichnet.

	Datenmodelle/ Speicherformen	Anfragen	Einsatzgebiet	Typische Systeme
Macro	objektrelationales-, multidimensionales-, und relationales Datenmodell	SQL 3	Serversysteme	Oracle, IBM, MS SQL-Server
Mini	relationales Datenmodell	SQL 1 (Ad-hoc-Anfragen möglich)	Desktop, Laptop	MySQL, Oracle Lite
Micro	persistente Tabellen	1-Relationen Zugriff meist über eine API	PDA, Smartphones	Berkeley DB, RDM-Embedded, COMET DBMS
Nano	einfache persistente Speicherstrukturen (Array, Hash-Map,..)	API	(tief) eingebettete Systeme, Smartcards	Prevayler

Abb. 1: Von Macro- bis zu Nano-Datenmanagementsystemen

betteter Systeme Datenbanken aus dem Nano- bis hin zum Mini-Bereich verwendet.

2 Datenhaltung in ressourcenbeschränkten Umgebungen

Die Entwicklung des Datenmanagements im Bereich eingebetteter Systeme erfolgte in den letzten Jahren ohne Verwendung einer einheitlichen Architektur und ausreichender Wiederverwendung. Da aber gerade in eingebetteten Systemen die Diversität der Anwendungen sehr groß ist, werden wir im Folgenden auf Probleme und Anforderungen an das Datenmanagement in diesem Bereich eingehen.

2.1 Eingebettete Systeme

Heutzutage sind Datenmengen im Tera- und Petabyte-Bereich Herausforderungen für die etablierten Datenbankhersteller. Bei eingebetteten Systemen liegen die zu verarbeitenden Datenmengen maximal im Megabyte-Bereich. Ein weiterer Unterschied zu herkömmlichen DBMS zeigt sich bei der Analyse der notwendigen Funktionalität. Während SQL, Replikation und Transaktionsverwaltung in Macro-DMS Standard sind, werden diese Funktionalitäten in eingebetteten Systemen oft nicht benötigt. Effiziente Speicherstrukturen, wie z.B. T-Bäume in In-Memory-Datenbanken [Lehman & Carey 1986], und einfache Synchronisation sind hier oft schon ausreichend.

Klassische, universell einsetzbare DBMS, so wie sie in den 70er-Jahren des vergangenen Jahrhunderts von Codd definiert wurden, werden also in eingebetteten Systemen nicht benötigt. Stattdessen ist abhängig vom Anwendungskontext hoch spezialisierte Datenmanagementfunktionalität notwendig. Das breite Anwendungsspektrum einerseits und die ressourcenbedingte Spezialisierung andererseits verlangen nach flexibler, skalierbarer und anpassbarer Datenmanagementfunktionalität bei gleichzeitigem sparsamem Umgang mit den vorhandenen Ressourcen [Chauduri & Weikum 2000, Kersten et al. 2003, Nyström et al. 2004, Stonebraker & Cetintemel 2005].

Moore'sches Gesetz. Nach dem Moore'schen Gesetz verdoppelt sich alle 18 Monate die Rechenleistung. Unter dieser Voraussetzung sollte eigentlich davon ausgegangen werden, dass sich die Ressourcenprobleme in eingebetteten Systemen im Laufe der Zeit von selbst lösen. Allerdings sollte dieses Argument nicht unreflektiert übernommen werden. Moderne Prozesse in der Chipfertigung führen ständig zu Leistungssteigerungen von Prozessoren. Gleichzeitig mit der Leistung steigen aber auch Energieverbrauch und Wärmeentwicklung. In vielen Bereichen wie z.B. im Automobilbau werden teilweise über 100 solcher eingebetteten Systeme in einem Produkt verwendet, wodurch der Energieverbrauch eine kritische Größe erreicht. Es zeigt sich seit Jahren, dass auch die Leistungen im Bereich der eingebetteten Systeme steigen, aber zurückhaltender als etwa im Bereich der Desktop-Rechner.

Ein zweiter wesentlicher Faktor bei der Betrachtung der Leistungsentwicklung sind die Kosten eingebetteter Systeme. Diese werden durch die dramatisch ansteigenden Stückzahlen verursacht, die unter anderem mit der Etablierung des Ubiquitous Computing entstehen.

Leistungsschwächere Rechner bedeuten daher aufgrund ihres Preises einen deutlich geringeren finanziellen Aufwand und auch einen geringeren Energieverbrauch. Hinzu kommt der mit komplexerer Software einhergehende Anstieg der Fehleranfälligkeit.

Kleine und ressourcenschonende Software wird daher trotz Moore langfristig ein Wettbewerbsvorteil sein.

2.2 Eingebettete Systeme im Automobilbau

Zur Verdeutlichung der Probleme im Datenmanagement eingebetteter Systeme werden wir als Beispiel den Automobilbau betrachten. Viele Funktionen im Umfeld von Automobilen werden durch eingebettete Systeme unterstützt oder sogar vollständig gesteuert. Seit vielen Jahren zeichnet sich zudem der Trend ab, dass nach und nach mechanische Steuerungssysteme durch elektroni-

sche informationsverarbeitende Systeme ersetzt werden. Auf diese Weise soll zum einen Gewicht gespart werden und zum anderen Komfort, Sicherheit und Leistungsfähigkeit im Fahrzeug gesteigert werden.

Komplexitätsprobleme. Eines der größten Probleme der Automobilhersteller ist die zunehmende Komplexität eingebetteter Systeme. Durch die fortschreitende Übernahme von Aufgaben im Automobil werden nicht nur die einzelnen Systeme komplexer, sondern auch deren Zusammenspiel. In modernen Fahrzeugen werden z.B. Sensordaten von Fahrerassistenzsystemen, Konfigurationsparameter für Sensoren und Aktoren sowie Fehlerprotokolle über zahlreiche eingebettete Mikrocontroller verteilt, gespeichert und verarbeitet. Jedes einzelne Subsystem hat dabei unterschiedliche Anforderungen an das Datenmanagement. Hauptursache für die immer wiederkehrenden Probleme durch fehlerhafte Software ist mangelnde Wiederverwendung erprobter Softwarebausteine. Dennoch wird auch in der Zukunft eines der dominierenden Themen im automobilen Umfeld die Entwicklung softwaregestützter informationsverarbeitender Systeme sein.

Datenverwaltung. Einhergehend mit der steigenden Funktionalität, die durch eingebettete Systeme gesteuert und überwacht wird, wächst auch das Datenaufkommen, das von diesen Systemen verarbeitet und verwaltet werden muss. In einer Studie von Volvo wurde ermittelt, dass das zu verarbeitende Datenaufkommen in einem Fahrzeug jährlich um 7-10% steigt [Nyström et al. 2002]. Vergleicht man die in einem Automobil anfallenden Datenmengen mit einer typischen Datenbank in einem Unternehmen, so sind diese Datenmengen jedoch sehr gering. Die wirkliche Herausforderung wird erst sichtbar, wenn die zur Verfügung stehende Rechenleistung zu Vergleichszwecken mit herangezogen wird. So dominieren in modernen Automobilen 8- und 16-Bit-Systeme und nur einige wenige Systeme sind mit 32-Bit-Prozessoren ausgerüstet.

Die Aufgaben, die durch Datenmanagementfunktionalität abgedeckt werden müssen, sind sehr unterschiedlich. Techniken, bei denen Daten losgelöst von anderen Systemen in internen Speicherstrukturen abgelegt und manipuliert werden, sind bei der zunehmenden Vernetzung der Systeme nicht mehr akzeptabel. Vielmehr besteht immer häufiger die Notwendigkeit, dass unterschiedliche Systeme gleichzeitig einen konsistenten, effizienten Zugriff auf die Daten benötigen.

Abbildung 2 zeigt ein Beispiel eines eingebetteten Systems, das von Nyström et al. in Zusammenarbeit mit Volvo konzipiert wurde [Nyström et al. 2002]. In dieser Studie wurde eine Umsetzung zur Validierung von Daten in der Datenhaltung eingebetteter Systeme analysiert. Dabei kamen mehrere Datenspeicher z.B. für die Speicherung von Zuständen und Warnungen zum Einsatz (siehe *HWDb, WoE Db* etc. in Abb. 2). Es wurde festgestellt, dass Funktionalität existierender DBMS benötigt wird, womit aber eine Reihe unnötiger Funktionalität eingeführt wird. Diese erhöht den Ressourcenbedarf der notwendigen Systeme signifikant, sodass auch hier die Verwendung maßgeschneiderter Datenhaltungskomponenten notwendig ist.

2.3 Eingebettetes Datenmanagement

Neben den klassischen serverbasierten DBMS entwickelt sich seit Jahren der Bereich der eingebetteten DMS. Diese sind nicht mit eingebetteten Systemen zu verwechseln, wenngleich dies eines der Haupteinsatzgebiete ist. Im Gegensatz zu klassischen DBMS wird die Datenmanagementfunktionalität mithilfe von Bibliotheken in die Anwendung integriert. In solchen Systemen kann so z.B. der Overhead einer Client-Server-Architektur vermieden

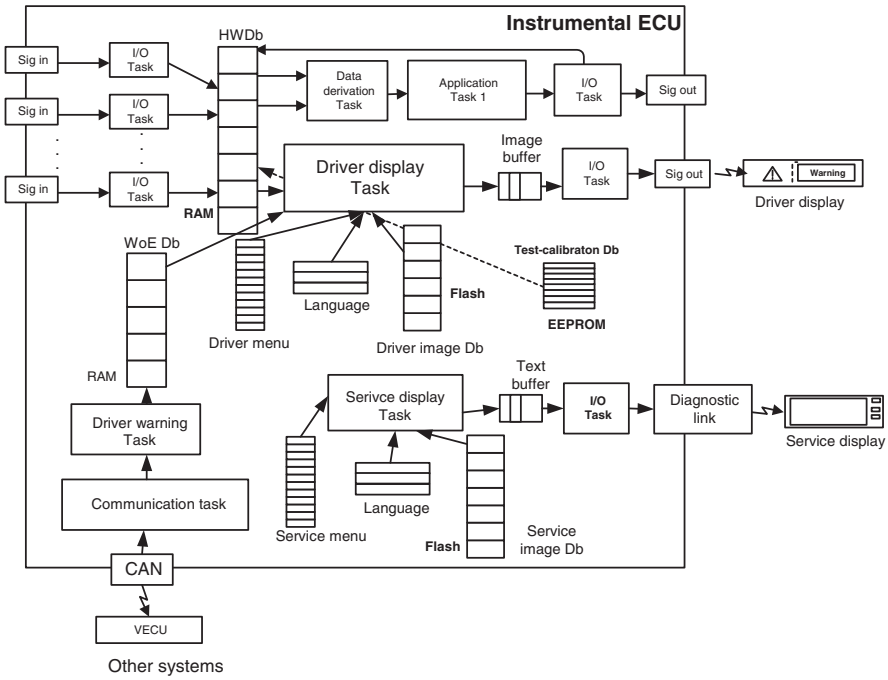


Abb. 2:
Eingebettetes System eines Radladers (aus [Nyström et al. 2002])

werden [Stonebraker & Cetintemel 2005]. Die Verwendung einer Anfragesprache ist in den meisten Fällen ebenfalls nicht notwendig, da alle Informationen in Bezug auf Anfragen bereits zur Entwicklungszeit vorliegen und stattdessen Funktionsaufrufe der API verwendet werden können.

In ressourcenbeschränkten eingebetteten Systemen gewinnt die Verwendung solch standardisierter Datenmanagementfunktionalität immer mehr an Bedeutung [Nyström et al. 2004, Stonebraker & Cetintemel 2005], da Vorteile sowohl beim Speicherverbrauch als auch bei der Performance bestehen.

3 Maßgeschneidertes Datenmanagement

Auch im Bereich herkömmlicher DBMS ist man bereits seit Jahren bestrebt, Schwierigkeiten in Bezug auf Erweiterbarkeit zu überwinden. In dieser Zeit entstanden zahlreiche Architekturen und Techniken zur Entwicklung erweiterbarer bzw. anpassbarer DBMS [Dittrich & Geppert 2001]. Im Folgenden werden wir einige dieser Ansätze diskutieren.

3.1 Monolithische Architekturen

Die Anpassung von DBMS an spezielle Einsatzbedingungen der Anwendung ist vor allem dann notwendig, wenn die Standardfunktionalität für den Anwendungsfall nicht ausreichend ist oder aufgrund von Ressourcenbeschränkungen der Einsatz von Mehrzwecksystemen einen zu großen Overhead an Funktionalität mit sich bringt. Das Hauptproblem bei den derzeitig verfügbaren Mehrzweck-DBMS ist die monolithische Struktur bzw. die *monolithische Architektur* der Systeme. Erweiterungen und Modifikationen der Funktionalität sind so nur schwer möglich. Die Gründe hierfür sind vielfältig. Die Basisstrukturen der heutigen kommerziell verfügbaren Systeme sind teilweise schon vor mehr als 20 Jahren entstanden und die damals vorhandenen technischen Möglichkeiten entsprachen hinsichtlich Variabilität, Verständlichkeit und Modularität nicht den heutigen Erkenntnissen der Softwaretechnik. Aber auch heute wird bei Erweiterungen und sogar bei Neuimplementierungen auf vorhandene Modularisierungstechniken (wie z.B. objektorientierte Programmierung (OOP)) oft nicht zurückgegriffen. So verwenden viele der heute entwickelten Systeme (z.B. PostgreSQL, Berkeley DB, MySQL) großteils strukturierte Programmierung, wie z.B. die Programmiersprache C, und nehmen damit bewusst bekannte Nachteile in Kauf.

3.2 Anpassbare und erweiterbare DBMS

Zur Überwindung der diskutierten Probleme entwickelten sich verschiedene Ansätze anpassbarer und erweiterbarer DBMS. Eine Einordnung existierender Ansätze liefern Dittrich und Geppert in [Dittrich & Geppert 2001]. Im Folgenden werden wir diese Ansätze kurz vorstellen und einige Vor- und Nachteile diskutieren.

Kernsysteme. Kernsysteme stellen eine definierte Grundfunktionalität von Datenbanksystemen bereit. Bezüglich der Basisfunktionalität ist ein Kernsystem nicht variabel und stellt kein funktionstüchtiges DBMS dar. Durch die Implementierung anwendungsspezifischer Funktionalität auf Basis einer vorgegebenen Schnittstelle wird das Kernsystem zu einem lauffähigen System erweitert. Das Festlegen der zum Kern gehörigen Funktionalität entscheidet bei der Entwicklung über Flexibilität des Kernsystems, aber auch über den Aufwand der Implementierung eines darauf basierenden DBMS.

Echte erweiterbare Systeme. Echte erweiterbare Systeme sind im Gegensatz zu Kernsystemen lauffähige DBMS. Die Anzahl der Variationspunkte ist beschränkt und fest in die Architektur integriert. Die häufigsten Erweiterungsmöglichkeiten beziehen sich auf das Hinzufügen neuer Datentypen, spezieller Operationen auf diesen Datentypen oder auf Indexstrukturen.

Anpassbare Systeme. Anpassbare Systeme sind ebenfalls vollständige DBMS, die bei Bedarf an den Anwendungskontext angepasst werden können. Im Gegensatz zu den echt erweiterbaren Systemen sind hier nur sehr eingeschränkte Anpassungen wie z.B. die Übergabe von Parametern möglich. Die häufigsten Anwendungsgebiete lassen sich im Bereich der Anfrageverarbeitung oder des Speichermanagements finden, indem z.B. Transformationsregeln für die Anfrageoptimierung hinzugefügt werden.

Toolkit-Systeme. Toolkit-Systeme stellen alternative Implementierungen für Funktionen einer Datenbank in Form einer Bibliothek zur Verfügung. So werden Lösungen in einem Baukasten bereitgestellt, mit deren Hilfe eine Erweiterung oder Anpassung einfacher zu realisieren ist. Häufig werden gerade Toolkit-Systeme mit anderen Ansätzen wie echten erweiterbaren DBMS vereint.

Generatorsysteme. Generatoren ermöglichen aus einer gegebenen Spezifikation von Datenbankfunktionalität die Generierung von Subsystemen eines DBMS. Hierzu wird ein Modell definiert und entsprechend dem Anwendungskontext parametrisiert. Ein Generator erzeugt im Anschluss die einzelnen Subsysteme. Die vollständige Generierung von kompletten DBMS ist aufgrund der Komplexität derzeit nicht realisierbar. Häufig werden deshalb Generatorenansätze zusammen mit Toolkit-Systemen verwendet.

Frameworks. Die konsequente Anpassung der Ansätze der Kernsysteme und der Toolkit-Systeme an die objektorientierte Programmierung bilden Frameworks. Sie bieten einen festen Kern an Datenmanagementfunktionalität und Variationspunkte zur Erweiterung und Anpassung in Form unterschiedlich realisierter Klassen.

3.3 Komponentensysteme

Eine Vielzahl von Ansätzen zur Zerlegung von DBMS favorisiert eine auf Komponenten basierende Architektur [Geppert et al. 1997, Chauduri & Weikum 2000, Nyström et al. 2004]. Diese auch als CDBMS [Dittrich & Geppert 2001] bezeichneten Systeme versuchen alle wesentlichen Merkmale eines DBMS in Komponenten zu fassen.

Eines der Hauptprobleme bei der Zerlegung von DBMS in Komponenten besteht in der Trennung der Funktionalität einer Komponente von den übrigen Teilen der Software. So ist etwa eine Transaktionsverwaltung in vielen DBMS notwendig, diese jedoch in eine losgelöste Komponente zu fassen, ist nahezu unmög-

lich. Die feste Integration in den Programmcode ist aber dennoch keine Alternative, da in einigen Systemen auf eine vollständige Transaktionsverwaltung verzichtet werden kann [Stonebraker & Cetintemel 2005].

Die meisten der komponentenbasierten Ansätze verwenden sehr große Komponenten. Chauduri und Weikum [Chauduri & Weikum 2000] hingegen schlagen Komponenten vor, die den RISC-Prozessoren nachempfunden sind, also nur eine geringe Funktionalität besitzen und dadurch einfacher zu entwickeln, zu testen und zu handhaben sind. Aufgrund des dabei entstehenden Aufwands für die Kommunikation der Komponenten untereinander empfehlen sie, die Größe der Komponenten nicht zu klein zu wählen, um größere Performance-Einbußen zu verhindern.

3.4 Diskussion

Die meisten der vorgestellten Ansätze konnten sich bis heute nicht oder nur in geringem Maße durchsetzen. Die Ursachen hierfür sind vielfältig.

Zum einen ist die Variabilität und Maßschneiderbarkeit der Lösungen begrenzt (z.B. anpassbare Systeme), zum anderen ist die notwendige Implementierungsarbeit für die Entwicklung eines DBMS sehr umfangreich (z.B. Kernsysteme). Dabei erfordert oft eine höhere Flexibilität auch einen erhöhten Aufwand bei der Implementierung, wie es bei den Kernsystemen besonders deutlich wird. Hinzu kommt, dass für die Vervollständigung eines DBMS umfangreiches Wissen über dessen Aufbau notwendig ist (z.B. Kernsysteme, Frameworks). Die Erweiterung durch einen nicht auf DBMS spezialisierten Anwendungsentwickler ist daher nur schwer möglich.

Eine weitere Ursache für den mäßigen Erfolg früherer Ansätze war die unzureichende Softwaretechnik, mit der vor zwanzig Jahren keine adäquate Modularisierung möglich war. Dies führte dazu, dass aufgrund von Anforderungen an die Performance und Fehlerfreiheit bewusst auf umfangreiche systematische Modularisierung verzichtet wurde.

Die Generierung vollständiger DBMS ist bisher aufgrund ihrer Komplexität ebenfalls nicht möglich. Die Verwendung von OOP kam insbesondere mit den Framework-Ansätzen auf, konnte sich aber nicht in großem Umfang durchsetzen. Ursachen sind bereits vorhandene, auf strukturierter Programmierung basierende Implementierungen, aber auch Bedenken bezüglich der Performance.

Die Entwicklung der erweiterbaren Architekturen beschränkte sich fast ausschließlich auf traditionelle DBMS. Der Bereich der eingebetteten Systeme blieb dabei nahezu unberücksichtigt. Neben der Vielfalt an Hardwarearchitekturen und der damit verbundenen Ressourcenbeschränkungen sind die vielfältigen Anforderungen der Anwendungen an das Datenmanagement als mögliche Ursachen zu nennen. Anders als in großen datenzentrierten Anwendungen wird in eingebetteten Systemen oft nur ein sehr kleiner Teil vorhandener Funktionalität benötigt. Die dafür notwendige Konfigurierbarkeit und Modularisierbarkeit ist mit den vorgestellten Ansätzen nicht zu erreichen. Benötigt werden daher Lösungen, die idealerweise eine Skalierung von kleinsten Lösungen aus dem Nano- und Micro-DMS-Bereich bis hin zum Mini-DMS-Bereich erlauben, ohne Verständlichkeit und Ressourcenverbrauch negativ zu beeinflussen.

4 Moderne Softwaretechnik als Ausweg

Die fehlende Wiederverwendung und Erweiterbarkeit von Software ist schon seit Jahrzehnten ein zentraler Forschungsschwerpunkt der Softwaretechnik und wurde erstmals 1968 unter dem Begriff der Softwarekrise zusammengefasst [Naur & Randell 1969]. Die Ursachen sind oftmals fehlende Rücksicht der Entwickler in Bezug auf Wiederverwendung. Zunehmend wird aber deutlich, dass auch die etablierten Softwaretechniken (z.B. OOP) keine Lösung bieten.

4.1 Produktlinien

Neue Ansätze aus der Softwaretechnik, insbesondere aus den Bereichen der *Produktlinien,* versprechen Hilfe bei der Überwindung der beschriebenen Schwierigkeiten, ohne die Vorteile spezialisierter Software (z.B. in Bezug auf Laufzeiteffizienz und Speicherverbrauch) zu verlieren, indem sie die Möglichkeit bieten, maßgeschneiderte Software zu erstellen bzw. zu generieren. Allerdings wurden einige dieser Methoden bislang nur in sehr begrenztem Maß an Infrastruktursoftware für eingebettete Systeme erprobt [Lohmann et al. 2006, Nyström et al. 2002].

Das Erstellen von Software ist im Idealfall das Zusammenfügen gewünschter Komponenten, die so bei möglichst wenig Anpassung wiederverwendet werden. Von dieser Idealvorstellung ist die Realität in vielen Bereichen der Softwareentwicklung weit entfernt, so auch im Datenmanagement. Die Produktlinienentwicklung basiert auf allgemeingültigen Softwaretechniken, um diese Probleme zu beheben. Die Softwareerstellung folgt dabei den Anforderungen des Anwenders und berücksichtigt die für ihn wichtigen Kriterien, die Merkmale der zu entwickelnden Software. Die Merkmale werden damit zu grundlegenden Elementen und finden sich in jedem Bereich der gesamten Softwareentwicklung wieder. Diese merkmalsorientierte Softwareentwicklung (*featureoriented software engineering* – FOSE) [Kang et al. 1998] gliedert sich in die Bereiche Domänenanalyse, Entwurf und Implementierung, wobei in jedem Bereich die einzelnen Merkmale identifiziert werden können.

Wie auch in anderen Industriezweigen üblich, werden zur Erstellung einer konkreten Software aus einer Menge der modularisierten Merkmale die benötigten ausgewählt. So lässt sich eine Reihe ähnlicher Programme, eine Produktlinie oder Programmfamilie, entwickeln [Czarnecki & Eisenecker 2000]. Jedes enthaltene Produkt erfüllt dabei verschiedene Anforderungen der Anwender. Die dafür notwendige Untersuchung der relevanten Merkmale einer Domäne ist Aufgabe der Domänenanalyse.

4.2 Domänenanalyse

Am Beginn des Softwareentwicklungsprozesses steht die Analyse der Problemdomäne, in unserem Falle die Domäne des Datenmanagements. Man spricht auch von *featureorientierter Domänenanalyse* (FODA) [Kang et al. 1990]. Ziel ist das Extrahieren der für den Anwender relevanten Merkmale und die Analyse ihrer

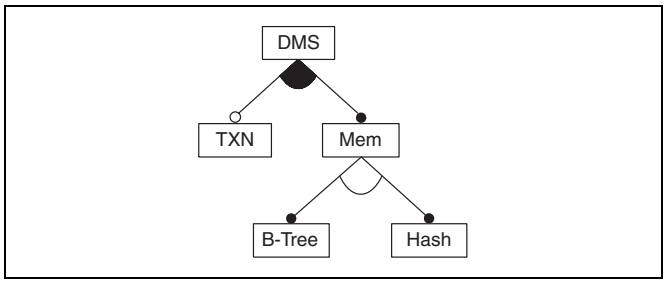
Beziehungen und Abhängigkeiten untereinander. Wesentliche Merkmale einer Software sind für einen Domänenexperten intuitiv beschreibbar und leicht verständlich. Auch die Abhängigkeiten zwischen den Merkmalen lassen sich von ihm leicht bestimmen.

Wichtig ist an dieser Stelle die Unterscheidung zwischen herkömmlicher Softwareentwicklung und der Softwareentwicklung für die Wiederverwendung [Czarnecki & Eisenecker 2000]. Während bei herkömmlicher Softwareentwicklung lediglich die Merkmale von Interesse sind, die in einem konkreten DMS zum Einsatz kommen, müssen bei der Berücksichtigung von Wiederverwendung alle Merkmale einbezogen werden, die in einer Produktlinie bzw. Problemdomäne (z.B. DMS) relevant sind. Dies ist Kern der Produktlinienentwicklung und von FODA.

Bei der Anwendung auf die Domäne des Datenmanagements stehen deren wesentliche Merkmale wie Transaktionsverwaltung, Replikation oder Indexstrukturen im Vordergrund. So muss vom Domänenexperten erörtert werden, welche Merkmale existieren, optional sind oder welche Varianten der Merkmale möglich sind (z.B. verschiedene Indexstrukturen wie B-Baum, Bitmap, Hash etc.).

Featurediagramme. Zur Darstellung und Beschreibung der analysierten Merkmale werden im Rahmen von FODA *Featurediagramme* verwendet. Abbildung 3 zeigt ein sehr einfaches Featurediagramm für DMS mit den Merkmalen Transaktionsverwaltung (TXN), Speichersystem (Mem), B-Tree und Hash. Es beschreibt sehr grobgranular die Domäne der eingebetteten DMS. Optionale Merkmale werden durch leere Kreise an den Endpunkten der Verbindungen gekennzeichnet (z.B. TXN) und obligatorische durch ausgefüllte Kreise (z.B. Mem). Des Weiteren existieren Darstellungen, die z.B. die Kennzeichnung alternativer Merkmale erlauben [Czarnecki & Eisenecker 2000]. So schließen sich in Abbildung 3 die Merkmale B-Tree und Hash gegenseitig aus, was durch einen nicht ausgefüllten Kreisbogen zwischen den Merkmalen dargestellt wird.

Abb. 3: Featurediagramm



Merkmale, wie sie in Abbildung 3 abgebildet sind, können beliebig in Untermerkmale zerlegt und diese wiederum im Featurediagramm dargestellt werden. Die Granularität der Zerlegung kann dabei beliebig gewählt werden und ist abhängig von softwaretechnischen Aspekten, wie Verbesserung der Verständlichkeit, sowie den Anforderungen an die Software, wie Maßschneiderbarkeit. Ein Teil eines komplexeren Beispiels aus [Leich et al. 2005] ist in Abbildung 4 dargestellt. In dieser Konzeptstudie konnten 93 Merkmale einer Speicherverwaltung ermittelt und zum Teil implementiert werden. So wurde gezeigt, dass FOSE im Datenmanagementbereich verwendet werden kann, um eine sehr feingranulare Konfigurierbarkeit zu erreichen. Die Skalierbarkeit des Ansatzes auf ein vollständiges DBMS muss in weiteren Studien untersucht werden.

4.3 Entwurf und Implementierung

Derzeitig verwendete Softwaretechniken im Bereich des Datenmanagements basieren auf strukturiertem Design und strukturierter Programmierung (z.B. Programmiersprache C). Zum Teil erfolgt bereits der Einsatz von OOP, jedoch ist dies oft auf die Programmiersprache Java beschränkt, die aufgrund der hohen Anforderungen an die Ressourcen nicht überall verwendet werden kann. Ein weiterer Grund für die Verwendung strukturierter Programmierung besteht in den bereits vor Jahren begonnenen Entwicklungen einzelner Lösungen, die optimierten Code enthalten, der nicht ohne Weiteres in eine objektorientierte Programmiersprache überführt werden kann.

Die Programmiersprache C wird aber auch heute noch häufig der objektorientierten Sprache C++ vorgezogen. Oft werden dafür Vermutungen als Argumente angeführt. So wird nicht selten beim Vergleich von C und C++ davon ausgegangen, dass bei der Verwendung von C++ grundsätzlich Einbußen in Bezug auf Performance oder Leistungsfähigkeit in Kauf genommen werden müssen. Wie aber Stroustrup feststellte, ist dies unbegründet [Stroustrup 2002]:

»Contrary to popular myths, there is no more tolerance of time and space overheads in C++ than there is in C. The emphasis on runtime performance varies more between different communities using the languages than between the languages themselves. In other words, overheads are found in some uses of the languages rather than in the language features.«

Für die Entwicklung effizienter Anwendungen ist also vielmehr Sorgfalt bei Entwurf und Implementierung geboten, wie es für jede Programmiersprache (einschließlich C) der Fall ist. So müssen z.B. bei der Verwendung von C++ virtuelle Funktionen so eingesetzt werden, dass sie nicht zu wesentlichen Einbußen bei Laufzeit oder Speicherbedarf führen. Beispiele hierfür lassen sich ebenso bei Verwendung der Programmiersprache C finden[2]. Die Vorteile von OOP, wie bessere Wiederverwendbarkeit, Erweiterbarkeit und Wartbarkeit [Booch 1990], könnten also ohne Einbußen bzgl. des Ressourcenbedarfs in die Entwicklung von DMS einfließen. Dennoch kann selbst mit OOP keine ausreichende Konfigurierbarkeit erzielt werden [Biggerstaff 1994].

FODA bildet die Grundlage für die Entwicklung von DMS als Produktlinie. Ziel ist nicht die Entwicklung eines konkreten DMS, sondern die Implementierung aller Merkmale der Produktlinie in modularisierter Form, aus denen während der *Konfigurierung* ein konkretes DMS erstellt wird.

2. Die Verwendung von Funktionspointern innerhalb von structs ist ein Beispiel hierfür.

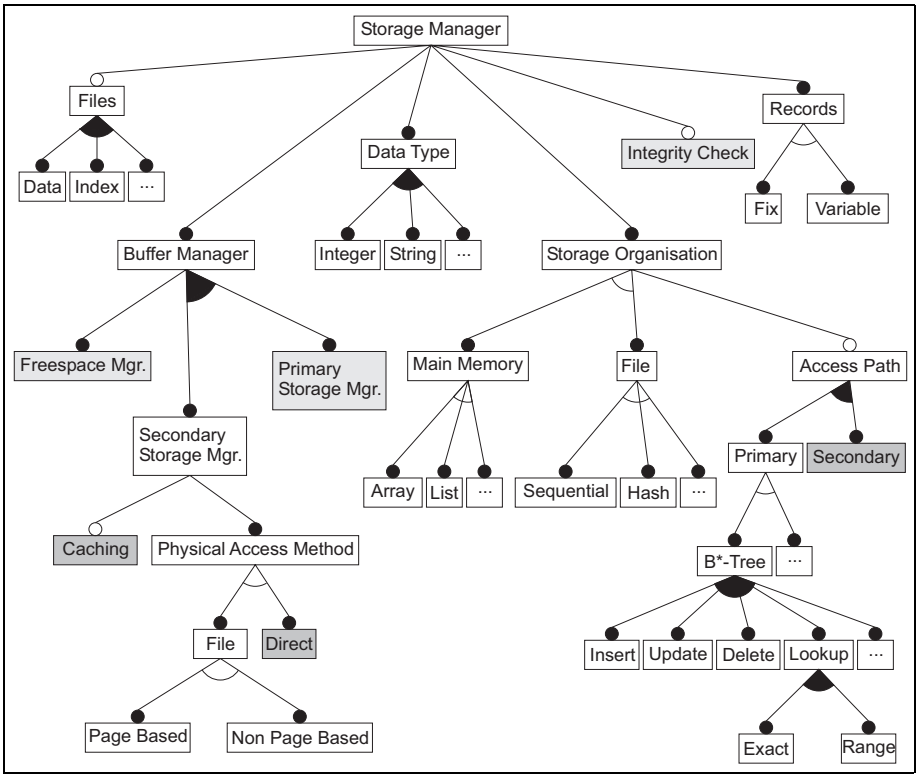


Abb. 4:
Featurediagramm eines Speichermanagers [Leich et al. 2005]

Eine Umsetzung dieses Ansatzes bieten die jüngeren Programmierparadigmen *aspektorientierte Programmierung* (AOP) [Kiczales et al. 1997] und *featureorientierte Programmierung* (FOP) [Prehofer 1997, Batory et al. 2004]. Beides sind Ansätze, die OOP erweitern und bereits auf der Ebene der Programmiersprache eine Modularisierung der Merkmale von Software unterstützen. Dabei ist auch die Modularisierung von Merkmalen möglich, die bei Verwendung herkömmlicher Softwaretechniken über den gesamten Programmcode verteilt (*code scattering*) und mit anderen Merkmalen verschränkt (*code tangling*) vorliegen. Diese sogenannten *crosscutting concerns* bilden das Hauptproblem bei der Zerlegung einer Software [Batory et al. 2003, Tarr et al. 1999, Kiczales et al. 1997] und sind zum Teil Ursache der Probleme von Frameworks und komponentenbasierten Ansätzen.

Sowohl AOP als auch FOP vermögen diese Probleme zu beheben und erlauben eine sehr feingranulare Zerlegung von Software [Batory et al. 2003, Kiczales et al. 1997]. Da beide Ansätze unterschiedliche Herangehensweisen verwenden, ergeben sich unterschiedliche Szenarien, in denen ein Paradigma Vorteile gegenüber dem anderen aufweist. Die Entscheidung, welcher Ansatz in einem speziellen Fall zu verwenden ist, ist vom zu modularisierenden Merkmal abhängig und wurde in [Apel & Batory 2006] genauer untersucht. Neben der Entscheidung für AOP oder FOP besteht zudem mit *Aspectual Mixin Layers* (AML) [Apel et al. 2006] die Möglichkeit der Kombination von AOP und FOP, um die Vorteile beider Ansätze nutzen zu können. Umsetzungen von AOP und FOP bestehen z.B. als Erweiterungen für die Programmiersprachen C++ [Spinczyk et al. 2002, Apel et al. 2005] und Java [Kiczales et al. 2001, Batory et al. 2004], was die Anwendung im Bereich des Datenmanagements ermöglicht.

5 Herausforderungen an die Datenbankforschung

Im Gebiet der Softwaretechnik entstanden in den letzten Jahren mit FODA, AOP und FOP Ansätze zur Entwicklung von Produktlinien, die den Weg für die Entwicklung maßgeschneiderter DMS bereiten. Die Techniken allein sind jedoch nicht ausreichend, um erfolgreich DMS-Produktlinien zu entwickeln. Mit der Verwendung von FOSE ändert sich der bisher bekannte Softwareentwicklungsprozess. Es geht nicht länger um die Implementierung eines konkreten DMS, sondern um die Softwareentwicklung mit Wiederverwendung als zentralem Ziel. Für die Domäne des Datenmanagements in eingebetteten Systemen ergeben sich damit neue Herausforderungen, die wir im Folgenden skizzieren.

Anwendbarkeit auf DMS. Die Anwendbarkeit und der Nutzen der Produktlinientechnologie wurden vielfach analysiert. Untersuchungen zur Anwendbarkeit im Bereich des Datenmanagements existieren hingegen nur wenige [Tesanovic et al. 2004, Leich et al. 2005], die zudem keine vollständige Umsetzung eines DMS untersuchen. Studien aus anderen Bereichen haben Vorteile der besprochenen Ansätze aufzeigen können [Apel & Batory 2006, Batory et al. 2002, Lopez-Herrejon & Batory 2006, Trujillo et al. 2006, Colyer & Clement 2004, Coady & Kiczales 2003, Garcia et al. 2003, Hunleth & Cytron 2002, Zhang & Jacobsen 2004], jedoch sind für die Anwendung im Bereich der DMS Vergleiche zu herkömmlichen Implementierungen notwendig. Für die Daten-

haltung in eingebetteten Systemen muss dies insbesondere unter Berücksichtigung von Ressourcenverbrauch und Performance erfolgen.

Die Konfigurierbarkeit vorhandener Lösungen ist nur ein Vorteil, der durch die Zerlegung und Wiederverwendung entsteht. So ist auch bessere Verständlichkeit des Programmcodes sowie verbesserte Qualität und Erweiterbarkeit der entwickelten DMS zu erwarten.

Domänenanalyse. Die für die Verwendung von FODA im Hinblick auf Wiederverwendung notwendigen Überlegungen können nur vor dem Hintergrund detaillierten Domänenwissens vorgenommen werden und daher nicht aus dem Bereich der Softwaretechnik erwartet werden. Folgende Punkte sind dabei von besonderem Interesse und benötigen das Wissen eines Domänenexperten aus dem DMS-Bereich:

- Entwicklung einer datenhaltungsspezifischen Ontologie, um die Domänenanalyse zu vereinheitlichen und zum besseren Verständnis der Domäne beizutragen.
- Bewertung der Kombinierbarkeit einzelner Merkmale. So ist z.B. die Verwendung einer Hashstruktur in Kombination mit Bereichsanfrage nicht möglich. Ein weiteres Beispiel ist die Kombination von Prefetching und Tupel-IDs.
- Festlegung der Granularität für die Zerlegung in Merkmale.
- Analyse von Ressourcenbeschränkungen, da dies im Bereich der Softwaretechnik oft keine Beachtung findet.

Architekturen für die Datenhaltung in eingebetteten Systemen. Basierend auf einer detaillierten Analyse der Domäne können Architekturen für DMS in eingebetteten Systemen entworfen werden. Die Entwicklung einer angemessenen Architektur ist jedoch kein trivialer Schritt und bedarf weiterer Studien, die Aspekte wie Performance oder Ressourcenbedarf berücksichtigen.

Die in Abschnitt 3.2 untersuchten Lösungen zur Konfigurierbarkeit können dabei als Anhaltspunkt dienen. Sie bieten zum Teil bereits Lösungen für die Zerlegung in einige wenige Komponenten. Zusätzlich ist nun aber eine weitere, deutlich feingranularere Zerlegung notwendig, um den Anforderungen des eingebetteten Datenmanagements gerecht zu werden. Es muss dabei die Berücksichtigung der gesamten Domäne erfolgen und nicht nur eines konkreten DMS.

Zudem ist zu untersuchen, inwieweit eine Vereinheitlichung der Architektur möglich ist und diese im Bereich eingebetteter Systeme skaliert, bzw. für welche Hardwarearchitekturen und Anwendungsgebiete unterschiedliche DMS-Architekturen notwendig sind. Bei der Untersuchung der Skalierbarkeit ist insbesondere Handhabung und Konfigurierbarkeit einer sehr hohen Anzahl von Merkmalen von Interesse, aber größere Studien [Apel & Batory 2006, Batory et al. 1995, Batory et al. 2002, Batory & O'Malley 1992, Lopez-Herrejon 2006, Lopez-Herrejon & Batory 2006, Trujillo et al. 2006, Xin et al. 2004] lassen vermuten, dass eine Skalierung möglich ist.

Ebenso ist die Anwendung auf die gesamte Domäne des Datenmanagements möglich. Hier fehlt nach wie vor eine ausreichende Maßschneiderbarkeit [Stonebraker & Cetintemel 2005, Chauduri & Weikum 2000] und aktuelle Ansätze, wie komponentenbasierte DBMS, stoßen auf Probleme. Sollte sich FOSE im Bereich der eingebetteten Systeme als erfolgreich erweisen, so spricht dies sicher auch für die Untersuchung der Anwendbarkeit im Bereich herkömmlicher DBMS.

Anwendungsanalyse. Je breiter das Anwendungsspektrum einer Produktlinie, desto mehr Merkmale fließen bei ihrer Entwicklung ein. Eine feingranulare Zerlegung zur Erhöhung der Flexibilität und die Entwicklung verschiedener Varianten eines Merkmals erhöhen die Anzahl der Merkmale weiter. Dies führt zu einem komplizierten Konfigurationsprozess bei der Erstellung eines konkreten DMS. Dieser Prozess muss so weit vereinfacht werden, dass er auch ohne detailliertes Wissen über die Implementierung einzelner Merkmale möglich ist. Hierbei kann die Anwendungsanalyse hilfreich sein. Sie kann verwendet werden, um Eigenschaften einer Produktlinie zu ermitteln oder aber um notwendige Merkmale eines DMS aus der Analyse einer Anwendung zu bestimmen. Dabei unterscheiden wir folgende Analyseprozesse:

- Analyse einer Produktlinie zur Automatisierung der Bestimmung notwendiger Merkmale eines DMS auf Grundlage einer unvollständigen Spezifikation.
- Bestimmung emergenter Eigenschaften[3] einer Produktlinie bzw. einzelner Merkmale durch die Analyse existierender Anwendungen dieser Produktlinie.
- Automatisierte Herleitung notwendiger Merkmale eines DMS durch die Analyse einer gegebenen Anwendung.

Aufbauend auf der Bestimmung emergenter Eigenschaften wie z.B. Hauptspeicherverbrauch lassen sich zudem mögliche und notwendige Merkmale eines DMS aus der Vorgabe von Ressourcenbeschränkungen ermitteln und so die zu verwendenden Merkmale eingrenzen. Ebenso können Vorgaben von Eigenschaften als Ziele weiterer Optimierung benutzt werden (z.B. Speichereffizienz vs. Energieeffizienz).

Die Ergebnisse dieser Analysen helfen dem Anwender, Entscheidungen zur Auswahl von Merkmalen zu treffen, oder nehmen ihm diese Entscheidungen ab. Sie sind daher ein wesentlicher Schritt auf dem Weg zur Automatisierung des Entwicklungsprozesses.

FOSE und Entwurf von Datenbankanwendungen. Derzeit existieren Werkzeuge, die den Entwurf von Datenbanken erleichtern oder diesen Prozess bereits teilweise automatisieren. Eine Erweiterung dieser Methoden unter Einbeziehung von FOSE ist daher wünschenswert. FOSE ist im erweiterten Kontext der modellbasierten Softwareentwicklung (z.B. IBMs *Model-Driven Development* [Booch et al. 2004] und OMGs *Model-Driven Architecture* [Kleppe et al. 2003]) gleichermaßen eine Theorie sowie eine Technik, um Produkte einer Produktlinie auf Basis von Modellkomposition zu synthetisieren und dann in ausführbare Programme zu transformieren [Trujillo et al. 2007, Batory 2006]. Dadurch wird die durchgängige Unterstützung bei Spezifikation, Entwurf

3. Emergente Eigenschaften bezeichnen die Eigenschaften eines Systems, die durch Wechselwirkung der Komponenten entstehen und nicht explizit festgelegt werden können, wie z.B. Ressourcenbedarf oder Laufzeit.

und Implementierung von datenintensiven Anwendungen möglich. Als Ergebnis des modellbasierten und werkzeuggestützten Datenbankentwurfs könnten z.B. Vorschläge für notwendige Merkmale des verwendeten DMS stehen, die so eine (Teil-)Automatisierung der Implementierung ermöglichen.

Untersuchungen zur Automatisierbarkeit von Spezifikation und Modellierung von DMS und deren Unterstützung durch Werkzeuge müssen daher unter Berücksichtigung von FOSE und modellbasierter Softwareentwicklung erfolgen.

SQL und Produktlinien. Neben der Anwendung von FOSE auf Domänenanalyse und Programmierung ist eine Anwendung auf beliebige Elemente der Softwareentwicklung möglich.

Im Bereich des Datenmanagements ist so auch die Anwendung auf Anfragesprachen wie SQL denkbar. Auf diese Weise könnte die Komplexität der unterstützten Sprache entsprechend der Anwendung verringert und so deren Verwendung vereinfacht werden. Ein weiterer Vorteil entsteht auch hier in Bezug auf den Ressourcenbedarf, da z.B. SQL-Parser und Optimierer entsprechend der verwendeten Syntax einfacher gestaltet oder sogar entsprechend den Vorgaben generiert werden können.

Dies ist aufgrund der Komplexität des zugehörigen Programmcodes und der üblicherweise über den gesamten Programmcode verstreuten Funktionalität sehr umfangreich, dennoch bietet z.B. FOP die Möglichkeit, solche quer schneidenden Merkmale zu modularisieren. Neben der dadurch erreichten Konfigurierbarkeit kann so auch Verständlichkeit und Übersichtlichkeit dieses sehr komplexen Merkmals verbessert werden.

Analog zu den Programmfamilien bei der Programmierung ergibt sich auf diese Weise eine Familie von SQL-Dialekten. Die Anwendbarkeit des Ansatzes auf Grammatiken konnte bereits für Java gezeigt werden [Batory et al. 2004].

6 Zusammenfassung

Das Datenmanagement in eingebetteten Systemen wird zunehmend umfangreicher. Dennoch kommt es häufig zu Neuentwicklungen und von Wiederverwendung wird selten Gebrauch gemacht.

Entwicklungen erweiterbarer und anpassbarer DBMS der vergangenen zwanzig Jahre erlauben keine ausreichende Anpassbarkeit und erfordern oft ein hohes Maß an Implementierungsarbeit, um ein maßgeschneidertes DBMS zu entwickeln. Sie bieten daher für den Bereich der eingebetteten Systeme aufgrund hoher Variabilität in den Anforderungen an das Datenmanagement sowie extremer Ressourcenbeschränkungen keine adäquaten Lösungen.

Die Produktlinientechnologie als neuere Entwicklung aus dem Bereich der Softwaretechnik erlaubt mit FODA, AOP und FOP die Entwicklung hoch konfigurierbarer Software. Ein Einsatz dieser Techniken sollte auch für den Bereich des Datenmanagements möglich sein. Einige Studien zeigen bereits, wie hohe Konfigurierbarkeit erreicht werden kann, doch sind detaillierte Analysen insbesondere in Bezug auf eingebettete Systeme und den damit verbundenen Ressourcenbeschränkungen notwendig. Die Anwendbarkeit auf die gesamte DMS-Domäne verspricht im Erfolgsfall weitere Vorteile.

Mit der Produktlinientechnologie als Grundlage ist die Automatisierung des gesamten Softwareentwicklungsprozesses als eine weitere Herausforderung an die Datenbankforschung zu sehen. Hierbei ist insbesondere Unterstützung bei der Erstellung eines konkreten DMS unter Berücksichtigung der Anforderungen der Anwendung und beschränkter Ressourcen notwendig.

Die Anwendung der Produktlinientechnologie auf andere Elemente der Softwareentwicklung bietet einen weiteren Ansatzpunkt für die Forschung. So kann z.B. die Entwicklung einer Familie von SQL-Dialekten erfolgen, um auch die Komplexität der Anfragesprache an den jeweiligen Anwendungskontext anzupassen.

Literatur

[Akyildiz et al. 2002] *Akyildiz, I.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E.:* A Survey on Sensor Networks. IEEE Communications Magazine, 40(8):102-114, 2002.

[Apel & Batory 2006] *Apel, S.; Batory, D.:* When to Use Features and Aspects? A Case Study. In: Proceedings of International Conference on Generative Programming and Component Engineering. ACM Press, 2006, S. 59-68.

[Apel et al. 2005] *Apel, S.; Leich, T.; Rosenmüller, M.; Saake, G.:* FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming. In: Proceedings of International Conference on Generative Programming and Component Engineering. Springer-Verlag, 2005, S. 125-140.

[Apel et al. 2006] *Apel, S.; Leich, T.; Saake, G.:* Aspectual Mixin Layers: Aspects and Features in Concert. In: Proceedings of International Conference on Software Engineering. ACM Press, 2006, S. 122-131.

[Batory & O'Malley 1992] *Batory, D.; O'Malley, S.:* The Design and Implementation of Hierarchical Software Systems with Reusable Components. ACM Transactions on Software Engineering and Methodology, 1(4):355-398, 1992.

[Batory et al. 1995] *Batory, D.; Coglianese, L.; Goodwin, M.; Shafer, S.:* Creating Reference Architectures: An Example from Avionics. In: Proceedings of the Symposium on Software Reusability. ACM Press, 1995, S. 27-37.

[Batory et al. 2002] *Batory, D.; Johnson, C.; MacDonald, B.; v. Heeder, D.:* Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study. ACM Transactions on Software Engineering and Methodology, 11(2):191-214, 2002.

[Batory et al. 2003] *Batory, D.; Liu, J.; Sarvela, J. N.:* Refinements and Multi-Dimensional Separation of Concerns. In: Proceedings of International Symposium on Foundations of Software Engineering. ACM Press, 2003, S. 48-57.

[Batory et al. 2004] *Batory, D.; Sarvela, J. N.; Rauschmayer, A.:* Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering, 30(6):355-371, 2004.

[Batory 2006] *Batory, D.:* Multi-Level Models in Model Driven Development, Product-Lines, and Metaprogramming. IBM Systems Journal, 45(3):527-540, 2006.

[BCC 2000] *Business Communications Company (BCC):* Future of Embedded Systems Technology, 2000. BCC Press release on market study RG-229.

[Biggerstaff 1994] *Biggerstaff, T.:* The Library Scaling Problem and the Limits of Concrete Component Reuse. In: Proceedings of International Conference on Software Reuse. IEEE Computer Society Press, 1994, S. 102-109.

[Booch 1990] *Booch, G.:* Object Oriented Analysis and Design with Applications. Benjamin Cummings, 1990.

[Booch et al. 2004] *Booch, G.; Brown, A.; Iyengar, S.; Rumbaugh, J.; Selic, B.:* The IBM MDA Manifesto. The MDA Journal, Meghan-Kiffer Press, 2004.

[Chauduri & Weikum 2000] *Chaudhuri, S.; Weikum, G.:* Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In: Proceedings of International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 2000, S. 1-10.

[Coady & Kiczales 2003] *Coady, Y.; Kiczales, G.:* Back to the Future: A Retroactive Study of Aspect Evolution in Operating System Code. In: Proceedings of the International Conference on Aspect-Oriented Software Development. ACM Press, 2003, S. 50-59.

[Colyer & Clement 2004] *Colyer, A.; Clement, A.:* Large-Scale AOSD for Middleware. In: Proceedings of the International Conference on Aspect-Oriented Software Development. ACM Press, 2004, S. 56-65.

[Czarnecki & Eisenecker 2000] *Czarnecki, K.; Eisenecker, U.:* Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.

[Dittrich & Geppert 2001] *Dittrich, K. R.; Geppert, A.:* Component Database Systems: Introduction, Foundations, and Overview. In: Compontent Database Systems. dpunkt.verlag, 2001, S. 1-28.

[Ganesan et al. 2003] *Ganesan, D.; Greenstein, B.; Perelyubskiy, D.; Estrin, D.; Heidemann, J.:* An Evaluation of Multiresolution Storage for Sensor Networks. In: Proceedings of the ACM SenSys Conference. ACM Press, 2003, S. 89-102.

[Garcia et al. 2003] *Garcia, A.; Sant'Anna, C.; Chavez, C.; Silva, V.; v. Staa, A.; Lucena, C.:* Separation of Concerns in Multi-Agent Systems: An Empirical Study. In: Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications, Volume 2940 of Lecture Notes in Computer Science. Springer-Verlag, 2003.

[Geppert et al. 1997] *Geppert, A.; Scherrer, S.; Dittrich, K. R.:* KIDS: Construction of Database Management Systems based on Reuse. Bericht ifi-97.01, Department of Computer Science. University of Zurich, 1997.

[Hunleth & Cytron 2002] *Hunleth, F.; Cytron, R.:* Footprint and Feature Management Using Aspect-Oriented Programming Techniques. In: Proceedings of Joint Conference on Languages, Compilers, and Tools for Embedded Systems & Software and Compilers for Embedded Systems. ACM Press, 2002, S. 38-45.

[Kang et al. 1990] *Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson, A.:* Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report CMU/SEI-90-TR-21, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1990.

[Kang et al. 1998] *Kang, K. C.; Kim, S.; Lee, J.; Kim, K.; Shin, E.:* Feature-Oriented Software Engineering for the Electronic Bulletin Board System (EBBS) Domain. In: 3rd World Conference on Integrated Design and Process Technology, 1998, S. 142-153.

[Kersten et al. 2003] *Kersten, M. L.; Weikum, G.; Franklin, M. J.; Keim, D. A.; Buchmann, A. P.; Chaudhuri, S.:* A Database Striptease or How to Manage Your Personal Databases. In: Proceedings of International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 2003, S. 1043-1044.

[Kiczales et al. 1997] *Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M.; Irwin, J.:* Aspect-Oriented Programming. In: Proceedings of European Conference on Object-Oriented Programming. Springer-Verlag, 1997, S. 220-242.

[Kiczales et al. 2001] *Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. G.:* An Overview of AspectJ. In: European Conference on Objectoriented Programming. Springer-Verlag, 2001, S. 327-355.

[Kleppe et al. 2003] *Kleppe, A.; Warmer, J.; Bast, W.:* MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley, 2003.

[Lehman & Carey 1986] *Lehman, T. J.; Carey, M. J.:* A Study of Index Structures for Main Memory Database Management Systems. In: Proceedings of International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 1986, S. 294-303.

[Leich et al. 2005] *Leich, T.; Apel, S.; Saake, G.:* Using stepwise refinement to build a flexible lightweight storage manager. In: Proceedings of the East-European Conference on Advances in Databases and Information Systems. Springer-Verlag, 2005, S. 324-337.

[Lohmann et al. 2006] *Lohmann, D.; Scheler, F.; Tartler, R.; Spinczyk, O.; Schröder-Preikschat, W.:* A Quantitative Analysis of Aspects in the eCos Kernel. In: Proceedings of EuroSys. ACM Press, 2006, S. 191-204.

[Lopez-Herrejon 2006] *Lopez-Herrejon, R.:* Understanding Feature Modularity. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2006.

[Lopez-Herrejon & Batory 2006] *Lopez-Herrejon, R.; Batory, D.:* From Crosscutting Concerns to Product Lines: A Function Composition Approach. Technical Report TR-06-24, Department of Computer Sciences, The University of Texas at Austin, 2006.

[Naur & Randell 1969] *Naur, P.; Randell, B. (Hrsg.):* Software Engineering: Report of the Working Conference on Software Engineering, Garmisch, Germany, 1969. NATO Science Committee, 1969.

[Nyström et al. 2002] *Nyström, D.; Tesanovic, A.; Norström, C.; Hansson, J.; Bankestad, N.-E.:* Data Management Issues in Vehicle Control Systems: A Case Study. In: Proceedings of Euromicro Conference on Real-Time Systems. IEEE Computer Society Press, 2002, S. 249-256.

[Nyström et al. 2004] *Nyström, D.; Tesanovic, A.; Nolin, M.; Norström, C.; Hansson, J.:* COMET: A Component-Based Real-Time Database for Automotive Systems. In: Proceedings of the Workshop on Software Engineering for Automotive Systems. IEEE Computer Society Press, 2004, S. 1-8.

[Prehofer 1997] *Prehofer, C.:* Feature-Oriented Programming: A Fresh Look at Objects. In: Proceedings of European Conference on Object-Oriented Programming. Springer-Verlag, 1997, S. 419-443.

[Spinczyk et al. 2002] *Spinczyk, O.; Gal, A.; Schröder-Preikschat, W.:* AspectC++: An Aspect-oriented Extension to the C++ Programming Language. In: Proceedings of International Conference on Tools Pacific. Australian Computer Society, 2002, S. 53-60.

[Stonebraker & Cetintemel 2005] *Stonebraker, M.; Cetintemel, U.:* One Size Fits All: An Idea Whose Time Has Come and Gone. In: Proceedings of International Conference on Data Engineering. IEEE Computer Society Press, 2005, S. 2-11.

[Stroustrup 2002] *Stroustrup, B.:* C and C++: Siblings. The C/C++ Users Journal, 14(11), 2002.

[Tarr et al. 1999] *Tarr, P.; Ossher, H.; Harrison, W.; Sutton, S. M.:* N Degrees of Separation: Multi-Dimensional Separation of Concerns. In: Proceedings of International Conference on Software Engineering. IEEE Computer Society Press, 1999, S. 107-119.

[Tesanovic et al. 2004] *Tesanovic, A.; Sheng, K.; Hansson, J.:* Application-Tailored Database Systems: A Case of Aspects in an Embedded Database. In: Proceedings of International Database Engineering and Applications Symposium. IEEE, 2004, S. 291-301.

[Trujillo et al. 2006] *Trujillo, S.; Batory, D.; Diaz, O.:* Feature Refactoring a Multi-Representation Program into a Product Line. In: Proceedings of the International Conference on Generative Programming and Component Engineering. ACM Press, 2006, S. 191-200.

[Trujillo et al. 2007] *Trujillo, S.; Batory, D.; Diaz, O.:* Feature Oriented Model Driven Development: A Case Study for Portlets. In: Proceedings of International Conference on Software Engineering, ACM Press, 2007, to appear.

[Turley 2002] *Turley, J.:* The Two Percent Solution. Embedded Systems Programming, 2002, *www.embedded.com.*

[Weiser 1993] *Weiser, M.:* Hot Topics: Ubiquitous Computing. IEEE Computer, 26(10):71-72, 1993.

[Woo et al. 2004] *Woo, A.; Madden, S.; Govindan, R.:* Networking support for Query Processing in Sensor Networks. Communications of the ACM, 47(6): 47-52, 2004.

[Xin et al. 2004] *Xin, B.; McDirmid, S.; Eide, E.; Hsieh, W. C.:* A Comparison of Jiazzi and AspectJ for Feature-Wise Decomposition. Technical Report UUCS-04-001, School of Computing, The University of Utah, 2004.

[Zhang & Jacobsen 2004] *Zhang, C.; Jacobsen, H.-A.:* Resolving Feature Convolution in Middleware Systems. In: Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications. ACM Press, 2004, S. 188-205.

Marko Rosenmüller
studierte Informatik an der Otto-von-Guericke-Universität Magdeburg. Von 2000 bis 2006 war er bei der icubic AG in Magdeburg als Softwareentwickler tätig. Seit 2006 ist er wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Datenbanken und Informationssysteme an der Universität Magdeburg.

Thomas Leich
ist Geschäftsbereichsleiter im An-Institut METOP der Otto-von-Guericke-Universität Magdeburg. Seit der Beendigung seines Wirtschaftsinformatikstudiums 2002 forscht er an der Universität Magdeburg im Bereich der eingebetteten Datenbanken.

Sven Apel
ist seit 2003 wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Datenbanken und Informationssysteme der Otto-von-Guericke-Universität Magdeburg. Seit 2007 ist er Oberassistent am Lehrstuhl für Programmierung der Universität Passau. Sven Appel ist Autor von über 30 wissenschaftlichen Publikationen.

Gunter Saake
ist Professor für Datenbanken und Informationssysteme an der Otto-von-Guericke-Universität Magdeburg und forscht unter anderem auf den Gebieten: Datenbankintegration, maßgeschneidertes Datenmanagement, objektorientierte Informationssysteme und Informationsfusion.

Dipl.-Inf. Marko Rosenmüller
Dipl.-Inf. Sven Apel
Prof. Dr. Gunter Saake
Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Universitätsplatz 2
39106 Magdeburg
{rosenmueller, apel, saake}@iti.cs.uni-magdeburg.de
http://wwwiti.cs.uni-magdeburg.de/iti_db/

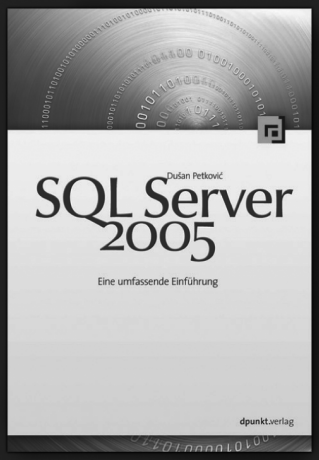
Dipl.-Wirtsch.-Inf. Thomas Leich
METOP GmbH
Geschäftsbereich Angewandte Informatik
Sandtorstr. 23
39106 Magdeburg
thomas.leich@metop.de
www.metop.de

dpunkt.SQL Server

Dusan Petkovic

SQL Server 2005
Eine umfassende Einführung

2006, 640 Seiten, Festeinband
€ 49,00 (D) · ISBN 978-3-89864-367-2



Dieses Buch behandelt ausführlich die neue Version des SQL Servers von Microsoft. Im Mittelpunkt stehen die Verwendung der Transact-SQL-Sprache, Performance Tuning sowie Sicherheitsaspekte des SQL-Server-Systems. Ein Schwerpunkt liegt auf den Neuerungen des SQL Servers 2005, wie Common Language Runtime (CLR), Common Table Expressions (CTE), SQL-OLAP und Reporting. Hierdurch wird den Lesern der Zugang zu den neuen Möglichkeiten erleichtert.

Ausführlich wird die Anwendung von Analysis Services als Data-Warehouse-System und der Einsatz vom SQL Server als natives XML-Datenbanksystem beschrieben. Neben der umfassenden Einführung in die Arbeit mit dem SQL Server bietet das Buch grundlegendes Wissen zu Themen wie Data Warehousing, XML und Volltextsuche.

dpunkt.verlag
Ringstraße 19 B
D-69115 Heidelberg
fon: 0 62 21 / 14 83 40
fax: 0 62 21 / 14 83 99
e-mail: bestellung@dpunkt.de
www.dpunkt.de