

Downsizing Data Management for Embedded Systems

Gunter Saake¹, Marko Rosenmüller¹, Norbert Siegmund¹, Christian Kästner¹, Thomas Leich²

¹School of Computer Science, University of Magdeburg, Germany
{saake, rosenmue, siegmund, ckaestne}@ovgu.de

²METOP Research Institute, Magdeburg, Germany
leich@metop.de

Abstract

Data management functionality is not only needed in large scale database management systems, but also in embedded systems that are the predominant form of computing systems today. However, resource restrictions and heterogeneity of hardware complicate the development of data management solutions. In current practice, this typically leads to redevelopment of data management solutions since existing applications cannot be customized sufficiently. In this paper we describe our vision of tailor-made data management, based on a software product line approach, where data management software can be tailored to satisfy special requirements. We illustrate how such a software product line for data management functionality can be derived by downsizing existing database systems.

1 Introduction

Traditionally, data management functionality is discussed in the context of large scale *database management systems (DBMS)* like Oracle, IBM DB2, or Microsoft SQL Server. Modern challenges often arise in the area of very large database systems; however, in recent years database management has also shown increasingly important for embedded devices.

Today, 98 % of all computing systems are embedded systems [37]. They are used in cars, cell phones, washing machines, TV sets, and many other things of daily use. Visions of pervasive and ubiquitous computing [40] emphasize the importance of embedded systems also for the future. What makes these systems special and challenging for data management are two factors. First, embedded devices provide only few resources. They usually have a comparably low computing power and restricted memory to save production costs and energy consumption. Second, embedded systems are strongly heterogenous, meaning that they differ severely

in hardware and software. Software for these systems is usually implemented specifically for a single system.

At the same time, embedded systems have to deal with an increasing amount of data to process information collected by sensors, cameras, or microphones. Applications for these systems have different requirements on data management, ranging from simple data storage over stream processing to complex data management, e.g., using transactions, recovery, and replication. A general data management infrastructure is the basis for separating data management and application logic [21]. There are several challenges for data intensive applications in embedded systems as we will illustrate by the example of automotive systems. For new application scenarios data management is often reinvented to satisfy computational and memory restrictions as well as new requirements [11]. This practice leads to an increased time to market, high development costs, and poor quality of software. Considering the limited resources and special requirements on data management, traditional DBMS are not suited for embedded environments. The main reasons are memory requirements and limited customizability [34, 10]. Thus, new techniques have to be employed to support the development of highly customizable data management applications that can be used in resource constrained environments and allow reuse of developed code in different solutions.

In this paper, we present means to model and implement tailor-made data management using *software product lines (SPL)*. We also show how an approach for downsizing existing data management systems can be applied. We illustrate first results and a perspective for further work on tailor-made data management.

2 Data Management in Resource Constrained Environments

In the following, we present a short overview of the special data management requirements of embedded systems

to motivate the resulting need for tailor-made data management solutions.

2.1 Embedded Systems

There are many important issues that limit the computational power of embedded systems. While Moore's Law says that the number of transistors doubles every 2 years and computation power is commonly assumed to double every 18 months, at the same time power consumption increases with an increasing number of transistors. In automobiles this is a more severe problem since the total number of electronic control units per car increases and so does the overall power consumption of automobiles. In different environments additional problems arise, for example, sensor networks might be powered by batteries which also demand minimal power consumption. Another important factor are costs of embedded systems especially in mass production. Both, power consumption and production costs, encourage to use the smallest and cheapest possible computation units in growing environments like automobiles despite the positive effects of Moore's Law.

The importance of data management increases in many domains where embedded systems are used. The resource constraints of embedded systems today are comparable to the constraints in desktop or server systems many years ago. These also provided only limited memory but there was still a need for data management functionality. At the moment this trend is continued with ubiquitous computing [40] and in the future perhaps with smart dust [39]. It is only limited by physical laws [18] and can be formulated as the *law of scale invariance of data management*:

There will be always small computing devices that operate with very constrained resources, and independent of the size of these systems there is a need for dedicated data management functionality.

Thus data management for resource constrained devices will also be needed in the future.

2.2 The Need for Tailor-made Data Management

The data management functionality that is needed for embedded systems varies for nearly every application scenario. As a motivating example, consider a modern passenger car which nowadays has over 100 electronic control units installed. The tasks of these systems strongly vary, from navigation systems, over a driver's logbook and total distance recorder, to devices that simply measures the number of revolutions. In all these cases we need some data management functionality, e.g., we need to store data persistently. However, we do not always need the full functionality of a large scale database management system like

Oracle. In Table 1 we present some possible applications and the needed data management functionality.

What we observe is that data management comes in many variants. We often need typical data management components like storage management, transactions, query processing, or recovery with equivalent or similar implementations. However, we rarely ever need all functionality in one system and we sometimes need different implementations for the same functionality optimized for different purposes, e.g., optimized for performance, low energy consumption, low working memory, or minimal footprint size.

Considering the diverse hardware and special application scenarios, the needed data management functionality differs for most systems. A solution can be tailor-made software that supports high variability which is provided by a product line approach, as known from other industries.

3 Software Product Lines

Most data management systems have a monolithic architecture to meet the high requirements on performance [11, 17]. Consequently, appropriate techniques are needed to develop tailor-made DBMS that do not degrade performance in exchange for customizability. This is even more important for embedded devices where resource restrictions do not allow any functionality overhead. Customization of software can be achieved by using *software product lines (SPL)* which allow for composing software solutions tailored to a specific problem.

The basic idea of SPLs is to develop software based on their *features*. These features represent functional requirements on a software that are of interest to the stakeholders [23]. Software development based on features was applied successfully in different domains [1, 5, 7, 6, 29, 15, 13, 19, 38, 41]. These case studies show that customizability of software can be achieved with the SPL approach.

3.1 Domain Modeling

Typically, the SPL development process consists of domain analysis, design, implementation, and configuration [16]. The first step, *Feature-Oriented Domain Analysis* [23], concentrates on the analysis of features of a domain to support their reuse. Different relevant concerns related to an SPL are integrated into one domain or *feature model*. A *feature diagram* is a hierarchical representation of all features of a feature model. Features in an SPL can be mandatory or optional [23] and may have relations to other features, e.g., two features can be alternative or conjunctive. Products of an SPL are *derived* in the configuration phase by selecting a subset of the available features and variants

System	Persistence	Recovery	Consistence	Queries	Granularity
navigation system	x	x		SQL	database
driver's logbook	x	xx	x	cursor	tables
total distance recorder	x	x	xx	fetch	tuple
number of revolutions recorder (min, max)	x			int	

Table 1. Data management in a car

of features. To enforce correctness of a configured product, there have to be additional constraints to overcome the restrictions of the hierarchical representation [25].

Figure 1 depicts a sample feature diagram of an SPL of a DBMS. The diagram consists of the base concept DBMS as the root node and additional nodes that represent the features of the product line. Features are arranged in a tree-like form and can have subfeatures to specialize their parent features or can be children of a grouping feature. The feature diagram shows that only the STORAGE MANAGER has to be configured for every product. The OPTIMISTIC transactions feature cannot occur in the same product with PESSIMISTIC transactions. Feature TRANSACTION furthermore requires an optional sub-feature of STATISTICS that further decreases the number of possible variants of the SPL.

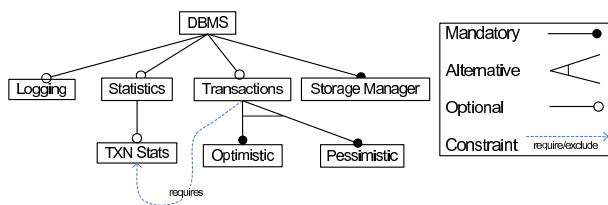


Figure 1. Feature diagram of a DBMS product line.

3.2 Implementing Software Product Lines

Currently, there are different approaches to implement software product lines. The most prominent are the use of preprocessor statements, as found in C/C++, and composition of product lines using components and frameworks. Both approaches have benefits but also some deficiencies.

Preprocessors. Using preprocessor statements is a well known technique to achieve customizability and often employed when implementing software for embedded systems. In Figure 2 we show an excerpt of the C source code of Berkeley DB¹, a database engine that can be embedded

into client applications. In this example, preprocessor statements are placed within methods to achieve customizability. The use of preprocessor statements is known to degrade readability of the source code and to complicate maintenance of a software [33]. Because of missing modularization also the evolution of software and even the elimination of dead features is problematic [9]. However, preprocessor statements are often used since they do not degrade performance and allow to produce tailor-made applications.

```

1 static int __rep_queue_filedone(dbenv, rep, rfp)
2     DB_ENV *dbenv;
3     REP *rep;
4     __rep_fileinfo_args *rfp; {
5 #ifndef HAVE_QUEUE
6     COMPQUIET(rep, NULL);
7     COMPQUIET(rfp, NULL);
8     return (__db_no_queue_am(dbenv));
9 #else
10    db_pgno_t first, last;
11    u_int32_t flags;
12    int empty, ret, t_ret;
13 #ifdef DIAGNOSTIC
14    DB_MSGBUF mb;
15 #endif
16    // over 100 lines of additional code
17 #endif
18 }
```

Figure 2. Code excerpt of Berkeley DB.

Components. Also components and frameworks can be used to develop SPLs and to attain customizability of DBMS [17, 20, 34]. However, components have a problem regarding granularity and performance. They usually implement coarse-grained modules that represent multiple features. Fine-grained customization is not possible without performance degradation because of the needed overhead to manage components [11]. Furthermore, the crosscutting structure of some features (e.g., transaction management of a DBMS affects many parts of the system) prohibits the use of components in many situations [30]. Those *crosscutting concerns* are part of and interact with many other components and thus cannot be modularized in a single component. The result is an overhead not suitable for embedded systems, or a degradation of configuration possibilities and therefore a loss of customizability.

¹<http://www.oracle.com/database/berkeley-db/db>

3.3 Discussion

The analysis shows that neither preprocessors, nor components and frameworks are appropriate for implementing fine-grained SPLs for embedded systems.

Most data management systems have a monolithic architecture that meets the high requirements on performance [11, 17]. Consequently, appropriate techniques are needed to develop tailor-made DBMS that do not degrade performance when implementing customizable solutions. The techniques that are of interest for embedded systems also have to provide fine-grained customizability, clear modularization, and appropriate support for reuse and maintenance of developed software.

4 Lightweight Software Product Line Implementations

Feature-oriented programming (FOP) [8, 31] and *aspect-oriented programming (AOP)* [23] are new programming paradigms that are promising for implementing SPLs with respect to these requirements. In contrast to components, FOP and AOP also support modularization of cross-cutting features.

FOP treats the features of software as basic elements of the whole development process. It allows to compose a family of similar programs based on the features of a domain. Features are increments in functionality. Technically, features implement program transformations that add new code to programs implemented in *feature modules* [8]. By composing a base program with a set of features (successively applying the program transformations) it is possible to derive different variants of the application. It is also possible to implement variants of a single feature to later select between specialized solutions optimized for different concerns. In contrast to object-oriented software development, this combination of a modular approach and code transformation allows to freely compose feature modules.

AOP also decomposes software with respect to their features. It has successfully been applied to operating systems [14, 13, 28] and middleware [42, 15]. These studies show that AOP is appropriate to decompose infrastructure software with respect to crosscutting features. The evaluations furthermore show that AOP can be used with negligible impact on performance and resource consumption, as long as no dynamic mechanisms are employed.

Both, FOP and AOP, allow to statically compose software using specialized compilers, e.g., AHEAD² and AspectJ³. Some studies have shown that AOP and FOP are similar approaches [1, 26, 4] and both have benefits as well

as deficiencies and can be combined [3]. In the following, we concentrate on FOP since we argue that it is more appropriate for implementing SPLs [3, 1, 24]. However, most of the presented concepts apply to AOP and FOP.

4.1 FeatureC++

With *FeatureC++* [2, 3] we developed an FOP language extension for the C++ programming language. It supports static composition and allows to apply FOP to software systems intended for resource constrained environments.

FeatureC++ uses a code transformation to C++ to benefit from compilers that exist for most computing systems. C++ is often considered to provide poor performance and high resource consumption. According to Stroustrup, however, there is no evidence for this argument [35]:

Contrary to popular myths, there is no more tolerance of time and space overheads in C++ than there is in C. The emphasis on run-time 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.

FeatureC++ furthermore uses method inlining to avoid a performance degradation for methods that are decomposed with respect to features. Though, when using C++, or FeatureC++, it has to be considered that complex mechanisms (e.g., exception handling, virtual methods) can introduce an overhead. All in all, performance is equivalent to preprocessor implementations, but FeatureC++ allows to separate features and to modularize the source code.

5 Downsizing Data Management

Developing downsized data management solutions for embedded systems that contain only and exactly the functionality required can be achieved using software product line concepts with FOP. Different implementations of the same features can be used to optimize the system for different concerns and to adapt the software for different hardware platforms.

There are different approaches to create an SPL for embedded data management. One possibility is to design an SPL from scratch, starting with domain analysis and implementing and testing the required features. Figure 3 shows an excerpt from a feature diagram of a storage manager (gray features contain sub-features not shown) [27]. This feature diagram makes use of a fine-grained decomposition and thus allows high customizability.

Alternatively, instead of starting from scratch, we also decomposed existing data management systems. Therefore,

²<http://www.cs.utexas.edu/~schwartz/>

³<http://www.eclipse.org/aspectj/>

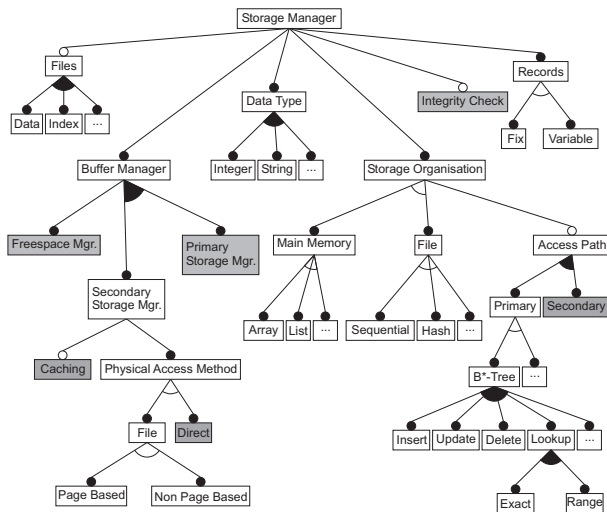
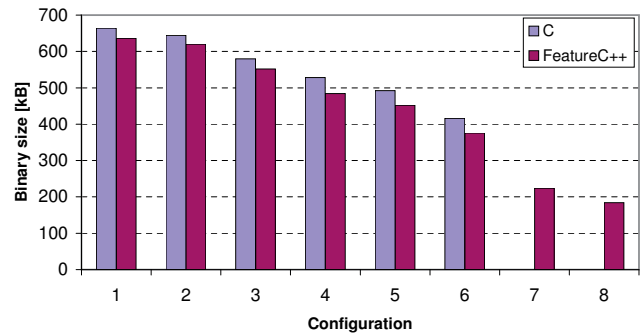


Figure 3. Feature diagram of a storage manager (excerpt).

we started with an existing, tested, and optimized data management system and incrementally decomposed features to create variability. After decomposing all relevant features, we got a stripped-down version that only contains the common functionality, while all features can be added when required for a certain use case. This approach, also known as *extractive adoption model* [12], has several advantages. First, by starting with an existing application we can reuse tested and already highly tuned code. Second, the required effort and risk are lower which makes this especially attractive for companies that want to adopt software product line technology for their products. Finally, this allows us to compare the downsized versions with the original application which makes it useful as a research benchmark.

In one case study we decomposed the C version of the embedded database engine Berkeley DB [32]. Even though Berkeley DB is already fairly small (484 KB footprint) and already allows few static configuration options using preprocessors in its original implementation, it is still too large for deeply embedded devices and contains several features like TRANSACTION MANAGEMENT that might not be required in all use cases. Therefore, we first transformed the Berkeley DB code from C to C++ and then decomposed it into features using FeatureC++. For decomposition we used behavior preserving refactorings, to maintain the original tested and tuned behavior.

Our case study has shown that the transformation from C to FeatureC++ (1) has no negative impact on performance or resource consumption, (2) successfully increases customizability so that we are able to create far more variants that are specifically tailored to a use case, and (3) successfully decreases binary size in specialized configurations by



- | | |
|--------------------------|---|
| 1 complete configuration | 5 without Queue |
| 2 without Crypto | 6 minimal C version using B-tree |
| 3 without Hash | 7 minimal FeatureC++ version using B-tree |
| 4 without Replication | 8 minimal FeatureC++ version using Queue |

Figure 4. Binary size of different C and FeatureC++ variants of Berkeley DB.

removing unneeded functionality to satisfy the tight memory limitations of small embedded systems.

The results are summarized in Figures 4 and 5. Before the refactoring the binary size of Berkeley DB embedded into a benchmark application was between about 400 and 650 KB, depending on the configuration (1–6). After transformation from C to FeatureC++ we could slightly decrease the binary size (Fig. 4) while maintaining the original performance (Fig. 5). By decomposing additional features that were not already customizable with preprocessors we even allow to derive configurations that are smaller and faster if those additional features are not required in a certain use case (Configurations 7 and 8 in Fig. 4 and 5).

This shows the practical relevance of downsizing data management for embedded systems. Already the decomposition of Berkeley DB, an existing application, has shown the benefits. By modeling and implementing a data management product line for embedded systems from scratch we could provide an even finer-grained and more customizable solution that can be used even in deeply embedded systems, e.g., on a small and cheap sensor inside a car.

6 Perspective

We have seen how SPLs and FOP can be applied to create highly customizable data management solutions. But there are still several open issues regarding the application of FOP to DBMS development.

6.1 Granularity of Features

Features are the building blocks of software when using FOP. At the moment there is less known about an appropriate granularity for decomposing DBMS when applying

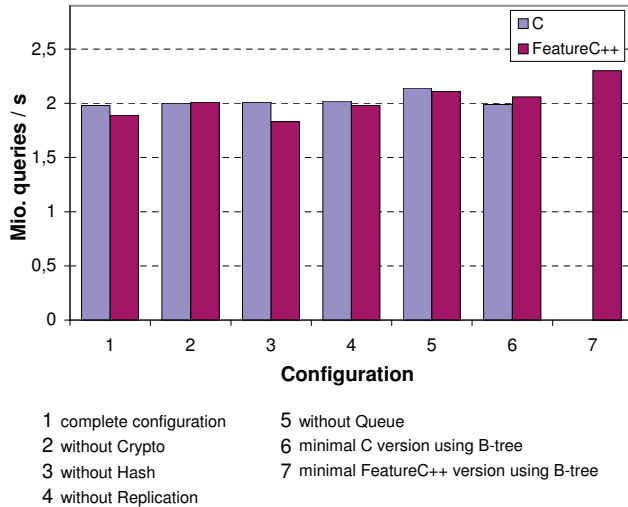


Figure 5. Performance comparison (Oracle benchmark) of C and FeatureC++ variants for different feature selections. Configuration 8 was omitted since it uses a different index structure.

FOP. While FOP supports an arbitrary fine-grained decomposition this might be counterproductive for DBMS development.

If considering DBMS for embedded systems there are many possible features. A finer grained decomposition of a DBMS, e.g., by using data types as features or further decomposition of index structures, is possible. This is essential to derive very small DBMS that are to be used in deeply embedded systems. However, there is also a disadvantage: By increasing the number of features it becomes more difficult to derive a specific program as more decisions are required. Already with 33 independent optional features, it is possible to create a program variant for every person on the planet. Additionally, the development overhead (not performance-relevant runtime overhead) for managing features increases and as features are not always independent there will be an increasing amount of feature interactions that need to be handled.

We propose that the granularity should depend on the field of application that defines requirements on performance, resource consumption, and maintenance. Thus in embedded systems also small sized features are of interest and even smaller features if deeply embedded systems are the destined environment. But more research is needed to identify the appropriate granularity for different application scenarios.

6.2 DBMS Architecture

DBMS architectures have to anticipate constantly changing requirements and are thus still under consideration [21]. Using feature-oriented concepts can help to achieve customizability of DBMS by introducing variability into the underlying architecture. Based on this variability different variants of architectures can be generated.

The analysis of relevant features in the DBMS domain and their interactions is a basic requirement for the development of tailor-made DBMS for embedded systems. But few attempts have been made to identify relevant features in the DBMS domain [27]. Based on a detailed domain analysis and the selection of a granularity of decomposition an appropriate architecture of DBMS for use in embedded systems can be developed. Furthermore, it has to be determined if a single architecture can cover the whole domain of embedded systems or if different architectures are needed. In order to derive such an appropriate architecture expert knowledge in the DBMS domain and further case studies are needed.

6.3 Application to SQL

The *Structured Query Language (SQL)* grows with every new standard and supports a lot of features while only a small subset is used by applications. With the release of SQL3 even traditional DBMS do not support the complete functionality of the standard. With *Structured Card Query Language (SCQL)* [22] a standard was released that supports functionality specialized for use in smart cards. This shows that SQL is not the appropriate solution for every use case.

A decomposition of SQL (the standard itself, the parser, etc.) with a feature-oriented approach might be a solution to solve the complexity problems. This results in a family of SQL dialects which could satisfy the requirements of different application scenarios. In [36] we have shown that such a decomposition is possible and a family of customizable SQL parsers can be automatically generated from a decomposed grammar. Using SQL in embedded systems could be based on such a dialect of the SQL standard and different dialects might be used for different applications. In order to derive a complete tailor-made DBMS that processes only a concrete SQL dialect, a customizable query optimizer and a customizable underlying DBMS have to be developed.

7 Conclusion

In this paper, we have shown how to customize and downsize DBMS to provide tailor-made data management for embedded systems. We propose to use an SPL approach and FOP to generate specialized DBMS based on a common

architecture. The fine-grained customizability supported by FOP is the basis for tailoring DBMS to satisfy the resource constraints of embedded systems. We have also shown that there are many open issues and also problems with FOP. These should be in the focus of future research to optimize the implementation techniques for SPLs in general and for tailor-made data management for embedded systems as special application.

Acknowledgments

Marko Rosenmüller and Norbert Siegmund are funded by German Ministry of Education and Research (BMBF), project number 01IM08003C. The presented work is part of the projects FAME-DBMS⁴ and ViERforES⁵.

References

- [1] S. Apel and D. Batory. When to Use Features and Aspects? A Case Study. In *Proceedings of the International Conference on Generative Programming and Component Engineering*, pages 59–68. ACM Press, 2006.
- [2] S. Apel, T. Leich, M. Rosenmüller, and G. Saake. FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming. In *Proceedings of the International Conference on Generative Programming and Component Engineering*, pages 125–140. Springer, 2005.
- [3] S. Apel, T. Leich, and G. Saake. Aspectual Mixin Layers: Aspects and Features in Concert. In *Proceedings of the International Conference on Software Engineering*, pages 122–131. ACM Press, 2006.
- [4] S. Apel, C. Lengauer, D. Batory, B. Möller, and C. Kästner. An Algebra for Feature-Oriented Software Development. Technical Report MIP-0706, Department of Informatics and Mathematics, University of Passau, 2007.
- [5] D. Batory, L. Coglianese, M. Goodwin, and S. Shafer. Creating Reference Architectures: An Example from Avionics. In *Proceedings of the Symposium on Software Reusability*, pages 27–37. ACM Press, 1995.
- [6] D. Batory, C. Johnson, B. MacDonald, and D. v. Heeder. Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study. *ACM Transactions on Software Engineering and Methodology*, 11(2):191–214, 2002.
- [7] 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.
- [8] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, 2004.
- [9] I. D. Baxter and M. Mehlich. Preprocessor Conditional Removal by Simple Partial Evaluation. In *Proceedings of the Working Conference on Reverse Engineering*, pages 281–290. IEEE Computer Society, 2001.
- [10] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDMBS: Scaling Down Database Techniques for the Smartcard. In *Proceedings of the International Conference on Very Large Data Bases*, pages 11–20. Morgan Kaufmann, 2000.
- [11] S. Chaudhuri and G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *Proceedings of the International Conference on Very Large Data Bases*, pages 1–10. Morgan Kaufmann, 2000.
- [12] P. Clements and C. Krueger. Point/counterpoint: Being proactive pays off/eliminating the adoption barrier. *IEEE Software*, 19(4):28–31, 2002.
- [13] Y. Coady and G. Kiczales. 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*, pages 50–59. ACM Press, 2003.
- [14] Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn. Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code. In *Proceedings of the European Software Engineering Conference*, pages 88–98. ACM Press, 2001.
- [15] A. Colyer and A. Clement. Large-Scale AOSD for Middleware. In *Proceedings of the International Conference on Aspect-Oriented Software Development*, pages 56–65. ACM Press, 2004.
- [16] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
- [17] K. R. Dittrich and A. Geppert. Component Database Systems: Introduction, Foundations, and Overview. In *Component Database Systems*, pages 1–28. dpunkt.Verlag, 2001.

⁴<http://fame-dbms.org>

⁵<http://vierfores.de>

- [18] R. P. Feynman. There's Plenty of Room at the Bottom. In *Feynman and Computation: Exploring the Limits of Computers*, pages 63–76. Perseus Books, 1998.
- [19] A. F. Garcia, C. Sant'Anna, C. Chavez, V. T. da Silva, C. J. P. de Lucena, and A. von Staa. Separation of Concerns in Multi-Agent Systems: An Empirical Study. In *Software Engineering for Large-Scale Multi-Agent Systems*, pages 49–72. Springer, 2003.
- [20] A. Geppert, S. Scherrer, and K. R. Dittrich. KIDS: Construction of Database Management Systems based on Reuse. Technical Report ifi-97.01, Department of Computer Science. University of Zurich, 1997.
- [21] T. Härder. DBMS Architecture – Still an Open Problem. In *Datenbanksysteme in Business, Technologie und Web*, pages 2–28, 2005.
- [22] International Organization for Standardization (ISO). Part 7: Interindustry Commands for Structured Card Query Language (SCQL). In *Identification Cards – Integrated Circuit(s) Cards with Contacts*, ISO/IEC 7816-7, 1999.
- [23] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.
- [24] C. Kästner, S. Apel, and D. Batory. A Case Study Implementing Features Using AspectJ. In *Proceedings of the International Software Product Line Conference*, pages 223–232, 2007.
- [25] C. H. P. Kim and K. Czarnecki. Synchronizing Cardinality-Based Feature Models and Their Specializations. In *European Conference on Model Driven Architecture Foundations and Applications*, pages 331–348, 2005.
- [26] M. Kuhlemann, M. Rosenmüller, S. Apel, and T. Leich. On the Duality of Aspect-Oriented and Feature-Oriented Design Patterns. In *AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, 2007.
- [27] T. Leich, S. Apel, and G. Saake. Using Step-Wise Refinement to Build a Flexible Lightweight Storage Manager. In *Proceedings of the East-European Conference on Advances in Databases and Information Systems*, pages 324–337. Springer, 2005.
- [28] D. Lohmann, F. Scheler, R. Tartler, O. Spinczyk, and W. Schröder-Preikschat. A Quantitative Analysis of Aspects in the eCos Kernel. In *Proceedings of the International EuroSys Conference*, pages 191–204. ACM Press, 2006.
- [29] R. Lopez-Herrejon and D. Batory. From Crosscutting Concerns to Product Lines: A Function Composition Approach. Technical Report TR-06-24, University of Texas at Austin, 2006.
- [30] D. Nyström, A. Tešanović, M. Nolin, C. Norström, and J. Hansson. COMET: A Component-Based Real-Time Database for Automotive Systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems*, pages 1–8. IEEE Computer Society, 2004.
- [31] C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of the European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 419–443. Springer, 1997.
- [32] M. Rosenmüller, N. Siegmund, H. Schirmeier, J. Sincero, S. Apel, T. Leich, O. Spinczyk, and G. Saake. FAME-DBMS: Tailor-made Data Management Solutions for Embedded Systems. In *EDBT'08 Workshop on Software Engineering for Tailor-made Data Management (SETMDM)*, pages 1–6, Mar. 2008.
- [33] H. Spencer and G. Collyer. Ifdef Considered Harmful, or Portability Experience With C News. In *Proceedings of the USENIX Summer 1992 Technical Conference*, pages 185–197, 1992.
- [34] M. Stonebraker and U. Cetintemel. One Size Fits All: An Idea Whose Time Has Come and Gone. In *Proceedings of the International Conference on Data Engineering*, pages 2–11, 2005.
- [35] B. Stroustrup. C and C++: Siblings. *The C/C++ Users Journal*, 20(7):28–36, 2002.
- [36] S. Sunkle, M. Kuhlemann, N. Siegmund, M. Rosenmüller, and G. Saake. Generating Highly Customizable SQL Parsers. In *Workshop on Software Engineering for Tailor-made Data Management (SETMDM)*, pages 29–33, 2008.
- [37] D. Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5):43–50, 2000.
- [38] S. Trujillo, D. Batory, and O. Diaz. Feature Refactoring a Multi-Representation Program into a Product Line. In *Proceedings of the International Conference on Generative Programming and Component Engineering*, pages 191–200. ACM Press, 2006.

- [39] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. *Computer*, 34(1):44–51, 2001.
- [40] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, 1993.
- [41] B. Xin, S. McDirmid, E. Eide, and W. C. Hsieh. A Comparison of Jiazi and AspectJ for Feature-Wise Decomposition. Technical Report UUCS-04-001, School of Computing, The University of Utah, 2004.
- [42] C. Zhang and H.-A. Jacobsen. Quantifying Aspects in Middleware Platforms. In *Proceedings of the International Conference on Aspect-Oriented Software Development*, pages 130–139. ACM Press, 2003.