FAME-DBMS: Tailor-made Data Management Solutions for Embedded Systems

Marko Rosenmüller[1], Norbert Siegmund[1], Horst Schirmeier[2], Julio Sincero[3], Sven Apel[4], Thomas Leich[5], Olaf Spinczyk[2], Gunter Saake[1]

[1]University of Magdeburg, {rosenmue,nsiegmun,saake}@ovgu.de

[2]Dortmund University of Technology, {horst.schirmeier,olaf.spinczyk}@tu-dortmund.de

[3]University of Erlangen-Nuremberg, sincero@cs.fau.de

[4]University of Passau, apel@uni-passau.de

[5]METOP Research Institute, thomas.leich@metop.de

ABSTRACT

Data management functionality is not only needed in large-scale server systems, but also in embedded systems. Resource restrictions and heterogeneity of hardware, however, complicate the development of data management solutions for those systems. In current practice, this typically leads to the redevelopment of data management because existing solutions cannot be reused and adapted appropriately. In this paper, we present our ongoing work on FAME-DBMS, a research project that explores techniques to implement highly customizable data management solutions, and illustrate how such systems can be created with a software product line approach. With this approach a concrete instance of a DBMS is derived by composing features of the DBMS product line that are needed for a specific application scenario. This product derivation process is getting complex if a large number of features is available. Furthermore, in embedded systems also non-functional properties, e.g., memory consumption, have to be considered when creating a DBMS instance. To simplify the derivation process we present approaches for its automation.

1. INTRODUCTION

Traditionally, research on data management software is discussed in the context of large-scale *database management systems (DBMS)* like Oracle, IBM DB2 or Microsoft SQL Server. In recent years, data management has also been shown increasingly important for embedded systems [17]. Embedded systems are used as control units in cars, cell phones, washing machines, TV sets, and many other devices of daily use. Visions of pervasive and ubiquitous computing [26] and smart dust [25] emphasize the importance of embedded systems for the future. Two factors make these systems special and challenging for data management: First, embedded devices have usually restricted computing power and memory in order to minimize production costs and energy consumption. Second, embedded systems are strongly heterogeneous, meaning that most systems differ in software and hardware. Software for these systems is usually implemented specifically for a single system and a special application scenario.

For new application scenarios data management is often reinvented to satisfy resource restrictions, new requirements, and rapidly changing hardware [7]. This practice leads to an increased time to market, high development costs, and poor quality of software [9, 15]. A general data management infrastructure could avoid this by separating data management and application logic [13]. Considering the limited resources and special requirements on data management, traditional DBMS are not suited for embedded environments [23, 6].

In this paper, we present our ongoing work on the FAME-DBMS research project[1]. Our goal is to develop, extend, and evaluate techniques and tools to implement and customize DBMS. Such techniques have to account for the special requirements of embedded systems. For that, we employ the *software product line (SPL)* approach based on static composition of features. With an SPL approach and techniques that enable to modularize also crosscutting features we can attain high variability which is needed for embedded systems. However, variability also increases the configuration space (i.e., the number of possible variants of a DBMS) and requires assistance to derive and optimize a concrete DBMS. We present two techniques to partially automate this *product derivation* process. First, to identify features of a DBMS SPL, needed for a particular client application, we use static program analysis. Second, we propose to partially automate the configuration process by using *non-functional constraints*, i.e., constraints that are used to restrict the non-functional properties of a DBMS like performance or memory consumption. We present first results and show what problems arise and what challenges are still ahead.

[1]The project is funded by German Research Foundation (DFG), Projects SA 465/32-1 and SP 968/2-1. http://www.fame-dbms.org/

2. TAILOR-MADE DATA MANAGEMENT

Resource constraints and a diversity in hardware of embedded systems forces developers to tailor software to a large number of application scenarios. Data management is needed in embedded systems but often reimplemented because existing solutions lack customizability. SPLs enable to develop software that can be tailored to different use cases with minimized development effort. This technology should also be applicable to tailor data management solutions for embedded systems.

2.1 Software Product Lines

Developing software that contains only and exactly the functionality required can be achieved using an SPL. Existing implementation techniques like *components* are inappropriate to support fine-grained customizability also of crosscutting concerns if they are used in isolation [17]. Using static composition, e.g., based on C/C++ preprocessor statements, achieves high customizability while not effecting performance. However, C/C++ preprocessor statements degrade readability and maintainability of source code [22, 5].

To avoid such problems new techniques – that are applicable to embedded systems – have to be explored and developed. In the FAME-DBMS project, we study *aspect-oriented programming (AOP)* [14] and *feature-oriented programming (FOP)* [3, 18] for implementing SPLs. In contrast to components, AOP and FOP also support modularization of crosscutting concerns. By using *AspectC++*[2] and *FeatureC++*[3], both language extensions of C++, we are able to use these paradigms for embedded systems.

Here we concentrate on FOP and FeatureC++, however, most of the presented concepts also apply to AOP. Using FOP, features are implemented as increments in functionality of a base program [3]. For *product derivation* (i.e., creating a concrete instance of a product line) a base program has to be composed with a set of features. This results in a number of different variants of an application.

2.2 Downsizing Data Management

There are solutions to apply the SPL approach to data management software. One possibility is to design a DBMS product line from scratch, starting with domain analysis and implementing and testing its features. Alternatively, instead of beginning from scratch, one can refactor existing data management systems, e.g., using FOP. When starting with existing, tested, and optimized data management systems and incrementally detaching features to introduce variability this results in a stripped-down version that contains only the core functionality. Additional features can be added when required for a certain use case. This approach, also known as *extractive adoption* [8], reduces the required effort and risk which makes it especially attractive for companies that want to adopt SPL technology for their products. In the FAME-DBMS project, we chose this approach to compare downsized versions with the original application which makes it useful as a research benchmark.

In a non-trivial case study we refactored the C version of the embedded database engine Berkeley DB into features. The footprint of Berkeley DB is fairly small (484 KB) and there are already a few static configuration options available. Even though, it is still too large for deeply embedded devices and contains several features like TRANSACTIONS that might not be required in all use cases. Therefore, we transformed the Berkeley DB code from C to C++ and then refactored it into features using FeatureC++. We used behavior preserving refactorings to maintain performance and to avoid errors.

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 (24 optional features) so that we are able to create far more variants that are specifically tailored to a use case, and (3) successfully decreases binary size by removing unneeded functionality to satisfy the tight memory limitations of small embedded systems.

The results are summarized in Figure 1. Before 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 (Figure 1a) while maintaining the original performance (Figure 1b)[4]. By extracting additional features that were not already customizable with preprocessor statements we are able to derive further configurations. These are even smaller and faster if those additional features are not required in a certain use case (Configurations 7 and 8 in Figure 1). This illustrates the practical relevance of downsizing data management for embedded systems.

2.3 FAME-DBMS

Decomposing Berkeley DB showed us that FOP and FeatureC++ are appropriate for implementing DBMS for embedded systems. We could furthermore show that a fine granularity, also of core functionality like the storage management, can be achieved using FOP [16]. However, when decomposing Berkeley DB we recognized that it is a complex task to decompose an existing DBMS that is not designed for a fine-grained decomposition. Thus, further decomposition of Berkeley DB was not possible in reasonable time because remaining functionality was heavily entangled. For example, decomposition of the B-tree index functionality is hardly possible without reimplementing large parts of it.

We argue that a DBMS SPL for embedded systems has to be designed with an adequate granularity. This means

[2]http://www.aspectc.org/

[3]http://wwwiti.cs.uni-magdeburg.de/iti_db/fcc/

[4]In Figure 1b Configuration 8 was omitted since it uses a different index structure and cannot be compared to Configurations 1–7.

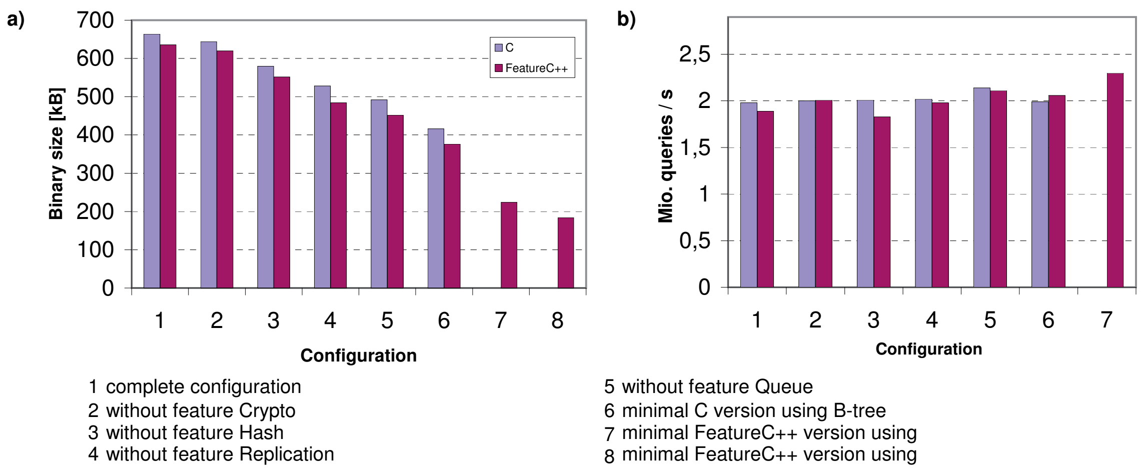

Figure 1: Binary size and performance of different C and FeatureC++ variants of Berkeley DB.

that different levels of granularities have to be supported: Functionality used in highly resource constrained environments should be decomposed with a fine granularity to ensure high variability. In contrast, functionality that is only used in larger systems, where resources are not highly limited, may not be further decomposed or should only be decomposed with a coarse granularity to avoid increasing complexity. Thus, the granularity is the key for a trade-off between complexity and variability.

Another reason for decomposing a DBMS into features is to impose a clean structure (e.g., for source code, documentation, etc.). However, since a decomposition can increase the complexity the benefit of such a decomposition has to be estimated. This is not the case for features that are only included to aggregate already decomposed features and thus do not further increase the complexity.

To analyze the granularity of an appropriate DBMS decomposition in more detail, we are currently implementing a DBMS product line from scratch. The decomposition of this FAME-DBMS prototype is depicted in Figure 2. While we are using a fine-grained decomposition for features that are also used in small systems (e.g., the B-tree index) we use a coarse granularity for features like TRANSACTION which is decomposed into a small number of features (e.g., alternative commit protocols). To further structure the DBMS product line aggregating features are used. For example, feature STORAGE aggregates different features but does not provide own functionality. Using FOP this structuring can be applied to all software artifacts including the source code.

3. AUTOMATED PRODUCT DERIVATION

Fine granularity of a decomposition of a DBMS is important to achieve customizability but it also increases the development effort. Furthermore, the product derivation process is getting more complex if there is a large number of features. The developer of an application that uses a DBMS has to tailor the DBMS for the specific needs of his application which imposes significant configuration effort. Furthermore, the application developer needs detailed knowledge of the DBMS domain for this configuration process. Thus an automated product derivation is desirable.

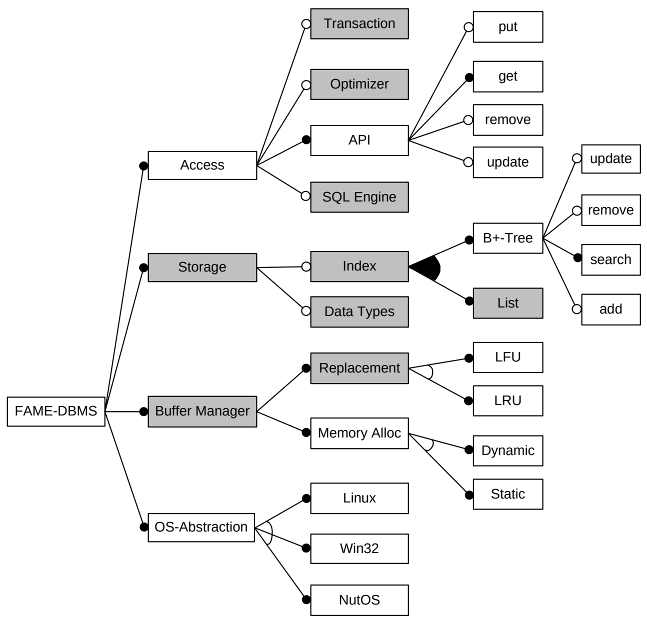

Figure 2: Feature diagram of the FAME-DBMS prototype. Gray features have further subfeatures that are not displayed.

Another important issue in embedded systems are *non-functional properties (NFPs)* of a generated DBMS instance. Often these are of interest to the stakeholder but cannot be configured directly. For example, a developer wants to tailor the functionality of a DBMS product line for his or her application, but also has to stay within the resource constraints of a given embedded device with fixed RAM and ROM size. Other NFPs like performance or response time are also very

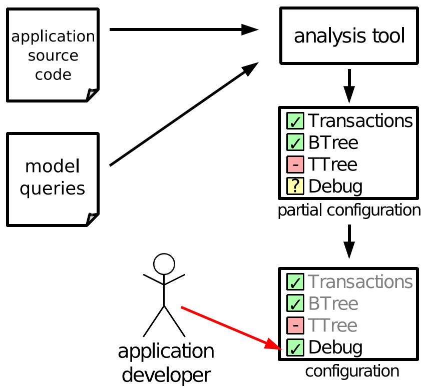

Figure 3: Automated detection of needed features with the analysis tool.

important in the embedded domain. Thus NFPs should also be considered in the product derivation.

In the FAME project we aim at improving tool support for product derivation. We address the configuration complexity by an approach that partially automates the configuration process using the functional requirements of a client application on a DBMS and furthermore integrate non-functional constraints.

3.1 Functional Requirements

When developing client applications that use a DBMS SPL the inherent *uses* relationship between application (e.g., a personal calendar application) and DBMS suggests to derive the need for DBMS features from the application itself.

We developed an analysis tool (see Figure 3) which automatically detects an application's need for infrastructure features by analyzing the C++ sources of the application [19]. For example, it would be beneficial to detect the applications's need for the feature JOIN of a DBMS to remove this functionality if it is not used. First, we statically analyze the application's sources which results in an application model (a control flow graph with additional data flow and type information), abstracting from syntactic details in the source code. Infrastructure features that are suitable for automatic detection can be associated with queries on the application model (*model queries* in Figure 3). These queries answer the question whether the application needs a particular feature. For example, in an application that uses Berkeley DB as a database infrastructure, a certain flag combination used to open a database environment indicates the need for the feature TRANSACTION, which can be formulated as one of the abovementioned queries.

The result of this process is a list of features that need to be included in the DBMS that the application is using. This list can be further refined by analyzing constraints between features of an SPL that are part of the feature model of that application. Ideally, large parts of a feature diagram can be configured automatically. The developer has to manually select only features that cannot be derived from the application's sources.

An evaluation of the approach with the refactored Berkeley DB (cf. Section 2.2) confirmed the assumption that the need for infrastructure features can be derived from the application sources in most cases. Our experiments with a benchmark application (that uses Berkeley DB) showed that 15 of 18 examined Berkeley DB features can be derived automatically from the application's source code; only 3 of 18 features were generally not derivable, because they are not involved in any infrastructure API usage within any application.

3.2 Non-functional Properties

As already stated also NFPs of an SPL are important for embedded systems. For example, binary size and memory consumption are critical for resource constrained environments and should also be considered in the product derivation process. To allow control over those properties our objective is to further automate product derivation. Using *non-functional constraints* we can exclude variants of an SPL that do not fulfill these constraints.

We support this by (1) measuring information about NFPs of concrete DBMS and (2) assisting or automating the selection of features based on measured NFPs and user defined constraints. To achieve these goals, our idea is to store as much information as possible about generated products in the model describing the SPL. This data is used to assist the derivation of further products.

The ideas are part of the *Feedback Approach* [21], which enables the application engineer to perform analysis (both static and dynamic) on generated products so that knowledge about specific NFPs can be obtained. This information can be assigned to a complete product specification (*configuration properties*), to a specific product feature (*feature properties*), or to implementation units, e.g., classes, aspects, or components (*component properties*).

The result of the analysis is stored to be used during product derivation to obtain the NFPs of a new product that is to be derived. This can be based on a calculation of an optimal configuration using the properties assigned to features or by estimating the properties based on heuristics (e.g., by using similarities between the product to be derived and earlier created instances). Calculating optimal solutions based on constraints is known as the *constraint satisfaction problem (CSP)* that belongs to the complexity class *NP-complete*. Currently we are using a greedy algorithm to calculate optimal solutions to cope with the complexity of the problem. Furthermore, we can give hints to the user what the properties of a configured instance will be by using information about already instantiated products.

We think that calculating an optimal solution has to be based on both: Properties assigned to features and properties of concrete instances. First, a greedy algorithm can be used to derive promising product configurations using feature properties. In the second step, more accurate values for non-functional properties can be obtained by including heuristics and information about already instantiated products and components. An optimal solution is selected by comparing theses corrected values.

Our work on NFPs is at an early stage, however, our preliminary results are promising. We have shown the feasibility of the idea for simple NFPs like *code size* [21] and are developing heuristics and analysis components to address *performance* of SPL instances.

4. RELATED WORK

Development of customizable data management software has been in the focus of research for several years. Batory and Thomas used code generation techniques to customize DBMS [4]. They aimed at creating special language extensions, e.g., to ease the use of cursors. As one of the origins of FOP, Batory et al. focused on customizing DBMS with Genesis [2]. In contrast to FOP as it is known today, it was not based on OOP and its complexity decreased usability. There have been many other developments to support extensibility of DBMS in the last 20 years. These approaches found their way into current DBMS but cannot provide appropriate customizability to support embedded systems (e.g., Kernel Systems). Additionally, detailed knowledge is often needed to implement a concrete DBMS or extend existing ones [11]. Component-based approaches are getting popular for traditional DBMS [11, 7, 17]. These, however, introduce a communication overhead that degrades performance and increases memory consumption. Furthermore, limited customizability because of crosscutting concerns does not allow for fine-grained customization. To overcome this limitation Nyström et al. developed a component-based approach named COMET that uses AOP to tailor the components [17]. In another approach, Tešanović et al. examined AOP for DBMS customization [24]. They evaluated their approach using Berkeley DB and showed customizability for small parts of the system. Both approaches (and there is no other approach that we are aware of) could not show concrete implementations of a complete customizable DBMS nor detailed evaluations. Other approaches like PicoDBMS [6] or DELite [20] concentrate on special requirements on data management for resource constrained environments and not on customizable solutions.

There is less research on automated tailoring of DBMS, infrastructure SPLs, with their special relationship to applications built on top of them. Fröhlich takes this relationship into account and aims at automatic configuration [12]. In this approach the set of infrastructure symbols referenced by the application determines which product line variant is needed. Apart from the comparably simple static analysis the main difference to our approach is the lack of logical isolation between analysis and configuration, established by a feature model.

NFPs of SPLs are getting more into the focus of current research. Cysneiros et al. propose the modeling of NFPs during application engineering [10]. This approach considers required non-functional behavior and adds it to design documentation in order to make the non-functional properties traceable. Bass et al. also address NFPs during software architecture design [1] to relate the NFPs to the system's architecture. Since these techniques tackle the same problem in a different stage of development, we see our work as a complementary approach and believe in the synergy between them.

5. CONCLUSION

We illustrated that FOP can be used to develop tailor-made data management solutions also for embedded systems. We applied an extractive approach to an existing DBMS and thereby could show that FOP has no negative impact on performance. We also presented our current work on the FAME-DBMS product line implemented using FOP and a mixed granularity for decomposition. The resulting high customizability is needed for embedded devices but increases the configuration space, rendering manual configuration complex and error-prone. Addressing this increased complexity, we presented two approaches that help simplifying variant selection by partially automating the configuration process and by providing non-functional properties for product derivation. Although we need to extend the approach and need further evaluation, first results are very promising.

In future work, we plan to create complete tool support that allows to develop SPLs optimized for embedded systems. As already outlined, we will continue working on tailor-made DBMS and plan to extend SPL composition and optimization to cover multiple SPLs (e.g., including the operating system and client applications) to optimize the software of an embedded system as a whole. Furthermore, we think that knowledge about the application domain has to be included in the product derivation process to automatically tailor the DBMS with respect to a concrete application scenario. For example, the data that is to be stored could be considered to statically select the optimal index.

Acknowledgments

Marko Rosenmüller and Norbert Siegmund are funded by German Research Foundation (DFG), Project SA 465/32-1, Horst Schirmeier and Julio Sincero by Project SP 968/2-1. The presented work is part of the FAME-DBMS project[5], a cooperation of Universities of Dortmund, Erlangen-Nuremberg, Magdeburg, and Passau, funded by DFG.

6. REFERENCES

[1] L. J. Bass, M. Klein, and F. Bachmann. Quality Attribute Design Primitives and the Attribute Driven Design Method. In *Revised Papers from the*

[5]http://www.fame-dbms.org/

International Workshop on Software Product-Family Engineering, pages 169–186. Springer-Verlag, 2002.

[2] D. Batory, J. R. Barnett, J. F. Garza, K. P. Smith, K. Tsukuda, B. C. Twichell, and T. E. Wise. GENESIS: An Extensible Database Management System. *IEEE Transactions on Software Engineering*, 14(11):1711–1730, 1988.

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

[4] D. Batory and J. Thomas. P2: A Lightweight DBMS Generator. *Journal of Intelligent Information Systems*, 9(2):107–123, 1997.

[5] 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 Press, 2001.

[6] 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.

[7] 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.

[8] P. Clements and C. Krueger. Point/Counterpoint: Being Proactive Pays Off/Eliminating the Adoption Barrier. *IEEE Software*, 19(4):28–31, 2002.

[9] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[10] L. M. Cysneiros and J. C. S. do Prado Leite. Nonfunctional Requirements: From Elicitation to Conceptual Models. *IEEE Transactions on Software Engineering*, 30(5):328–350, 2004.

[11] K. R. Dittrich and A. Geppert. Component Database Systems: Introduction, Foundations, and Overview. In *Component Database Systems*, pages 1–28. dpunkt.Verlag, 2001.

[12] A. Fröhlich. *Application-Oriented Operating Systems*. Number 17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik, Sankt Augustin, 2001.

[13] T. Härder. DBMS Architecture – Still an Open Problem. In *Datenbanksysteme in Business, Technologie und Web*, pages 2–28. Gesellschaft für Informatik (GI), 2005.

[14] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 220–242. Springer-Verlag, 1997.

[15] C. W. Krueger. New Methods in Software Product Line Practice. *Communications of the ACM*, 49(12):37–40, 2006.

[16] 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-Verlag, 2005.

[17] 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 Press, 2004.

[18] 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-Verlag, 1997.

[19] H. Schirmeier and O. Spinczyk. Tailoring Infrastructure Software Product Lines by Static Application Analysis. In *Proceedings of the International Software Product Line Conference*, pages 255–260. IEEE Computer Society Press, 2007.

[20] R. Sen and K. Ramamritham. Efficient Data Management on Lightweight Computing Devices. In *Proceedings of the International Conference on Data Engineering*, pages 419–420. IEEE Computer Society Press, 2005.

[21] J. Sincero, O. Spinczyk, and W. Schröder-Preikschat. On the Configuration of Non-Functional Properties in Software Product Lines. In *Proceedings of the Software Product Line Conference, Doctoral Symposium*. Kindai Kagaku Sha Co. Ltd., 2007.

[22] 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.

[23] 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. IEEE Computer Society Press, 2005.

[24] A. Tešanović, K. Sheng, and J. Hansson. Application-Tailored Database Systems: A Case of Aspects in an Embedded Database. In *Proceedings of International Database Engineering and Applications Symposium*, pages 291–301. IEEE Computer Society Press, 2004.

[25] 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.

[26] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, 1993.