RobbyDBMS – A Case Study on Hardware/Software Product Line Engineering

Jörg Liebig and Sven Apel and Christian Lengauer
Department of Informatics and Mathematics
University of Passau, Germany
{joliebig,apel,lengauer}@fim.uni-passau.de

Thomas Leich
Metop Research Center
Magdeburg, Germany
thomas.leich@metop.de

ABSTRACT

The development of a highly configurable data management system is a challenging task, especially if it is to be implemented on an embedded system that provides limited resources. We present a case study of such a data management system, called RobbyDBMS, and give it a feature-oriented design. In our case study, we evaluate the system's efficiency and variability. We pay particular attention to the interaction between the features of the data management system and the components of the underlying embedded platform. We also propose an integrated development process covering both hardware and software.

Categories and Subject Descriptors

D.2.10 [**Software**]: Design—*Methodologies*;
D.2.11 [**Software**]: Software Engineering—*Domain-specific architectures*

General Terms

Design

Keywords

Hardware Product Lines, Software Product Lines, Domain Engineering, Feature Oriented Software Development, FeatureC++

1. INTRODUCTION

Current statistics reveal that 98 % of all microprocessors sold worldwide are part of embedded systems [18, 12]. Embedded systems play an important role in domains such as automotive, industrial automation, and control systems. Carrying out tasks in these domains involves the gathering, processing, and storage of data. Embedded systems handle data collected from several sources such as sensor data, technical service specifications, configuration, or protocol data. Although the amount of data is usually small, an efficient data management plays a crucial role, since most embedded systems come with very low computational power and only a small amount of memory.

The need for an efficient data management system in the embedded systems domain has been observed before. Researchers and engineers proposed various systems, such as IBM DB2 Everyplace [10], LGeDBMS [11], Smart Card DBMS [1], TinyDB [14], and Stones DB [8]. Each system has been developed from scratch to serve a specific application that the developers had in mind. To this end, a fixed set of data management functionalities is sufficient and, therefore, each system contains no or few variabilities. This results in a specialized software and is the reason why so many different systems have been proposed. Moreover, each of these systems is designed for a specific embedded platform, and, to this end, existing hardware variabilities are not taken into account.

A way to overcome such limited hardware and software variability is *product line engineering* (*PLE*). PLE has already been applied successfully in hardware engineering such as of cars or mobile phones, and is gaining increasing attention in software engineering [6, 15]. Since more and more products consist of hardware and software variants, the combined PLE application for hardware and software engineering is promising. However, very little is known about the impact that PLE has on products that consist of hardware and software.

To investigate the impact of PLE, we use *feature-oriented programming* (*FOP*), one implementation approach for PLE, and develop the case study RobbyDBMS: an efficient data management system for various embedded systems. We evaluate RobbyDBMS regarding efficiency and variability. Based on the results obtained, we discuss the benefits of the PLE approach. PLE has been applied successfully earlier for the development of efficient data management systems [17].[1] Although Fame-DBMS aimed at a database family for embedded systems, it did not meet our hardware requirements in terms of computational power and size of available memory.

During the implementation of RobbyDBMS, we observed that the data management system and the underlying embedded platform exhibit functional interactions. For example, for permanent data storage, a non-volatile memory or storage facility is necessary. We explain these interactions and propose an integrated development process for hardware/software PLE. Additionally, we highlight the benefits of our approach.

Specifically, we make the following contributions:

- We present our case study RobbyDBMS: an efficient data management system for embedded systems.
- We evaluate and discuss FOP as one implementation

[1] Project Fame-DBMS – `http://fame-dbms.org/`

FOSD'09, October 6, 2009, Denver, Colorado, USA.

technique for PLE. The evaluation covers the efficiency and the variability of the feature-oriented design.
- We highlight interactions between RobbyDBMS and the underlying embedded platform.
- We propose and discuss the feasibility of an integrated development process for systems like RobbyDBMS based on domain/application engineering exhibiting both hardware and software variability.

2. BACKGROUND

Product Line Engineering

Product line engineering is a concept comprising methods, tools, and techniques for the development of product lines [2]. A *product line* is a set of mutually related products that are tailored to a specific domain or market segment and that share a common set of features [9]. A *feature* represents a commonality or a variability among the set of products [19]. By selecting varying sets of features, different products (a.k.a. *variants*), fulfilling the requirements of a specific application, can be generated. Product lines are being developed in both hardware and software engineering, and we refer to them as *hardware product lines* (*HPL*) and *software product lines* (*SPL*), resp. Furthermore, we refer to features of the HPL and the SPL as *hardware features* resp. *software features*.

The process behind PLE is domain and application engineering (Figure 4) [7]. It comprises the development of reusable features in domain engineering that are used in application engineering to derive a specific product. Domain and application engineering both consist of three phases (domain engineering: domain analysis, domain modeling, and domain implementation; application engineering: requirements analysis, design analysis, and integration/test).

Feature-Oriented Programming

One approach to the implementation of SPLs is *feature-oriented programming*, which aims at the modularization of software systems using features [16, 5]. The idea is to modularize features in *feature modules* and, consequently, to obtain a 1-to-1 mapping between features as a domain abstraction and feature modules as an implementation abstraction.

In particular, we use FeatureC++ [4, 3], an extension of the programming language C++, which enables programmers to encapsulate the functionality of a feature in a feature module. A feature module represents program functionality and is composed with the base system (*feature composition*) via declarative expressions. With FeatureC++, one can add variables and methods to existing classes or one can even add new classes. Furthermore, FeatureC++ provides capabilities for extending existing methods using method refinement [4].

3. CASE STUDY ROBBYDBMS

In this section, we describe our case study of RobbyDBMS. Furthermore, we evaluate RobbyDBMS regarding efficiency and variability, and discuss the benefits of using FOP for the implementation of RobbyDBMS.

3.1 Overview

AVR HPL

The AVR HPL is a product line of 8-bit microprocessors used in the embedded systems domain. Using the AVR HPL, programmers face a number of resource constraints, such as the low performance of the CPU (1 - 32 MHz), the small supply of program storage (1 - 256 KB), and the small main memory (32 B - 32 KB). Furthermore, different variants of the AVR series have up to 4 KB of EEPROM storage, which can be used as a permanent storage for data. These resource constraints result in cost and energy savings. AVR microprocessors are integrated in embedded systems, which have additional hardware features like sensors or actuators. Figure 1 contains a *feature diagram*, which is an excerpt of the AVR HPL *feature model* [7], representing hierarchical relationships of features in the AVR HPL. For example, feature RAM is optional and, in case this feature is available, different variants can have memory from 32 B to 32 KB.

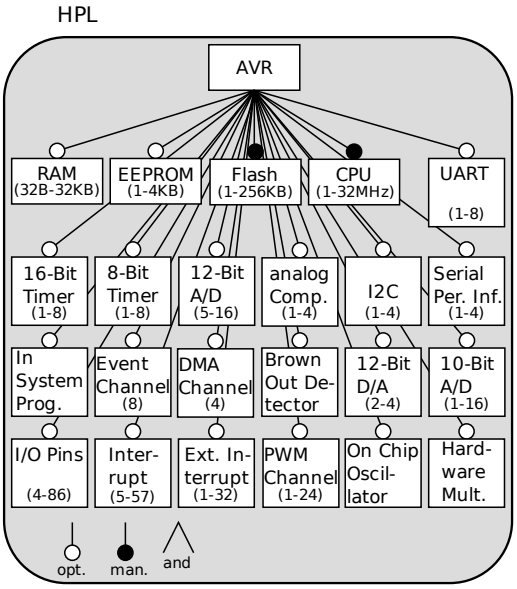

Figure 1: AVR HPL; feature diagram

The resource constraints mentioned before oblige to create a data management systems that is both: (1) highly configurable in terms of the features that the data management provides and (2) tailorable to a specific variant of the HPL that is being selected.

RobbyDBMS SPL

RobbyDBMS is an embedded data management system aiming at high configurability and the support of low-performance embedded systems. To increase configurability, we model most features, which are usually an integral part of data management systems, as optional (e.g., INDEXING, CHECKSUMS, BUFFERING, and TRANSACTION). This reduces the amount of program storage and main memory so as to leave as much of the system resources as possible to the programmer who makes use of the data management system. Figure 2 shows an excerpt of a feature diagram representing the feature model for RobbyDBMS.

Overall, RobbyDBMS consists of 33 feature modules with 46 classes and 37 class refinements. These refinements involve 33 method refinements, 39 additions of a function, and 15 additions of a field. The number of feature modules exceeds the number of features because of functional dependencies between different features that necessitated the split of a

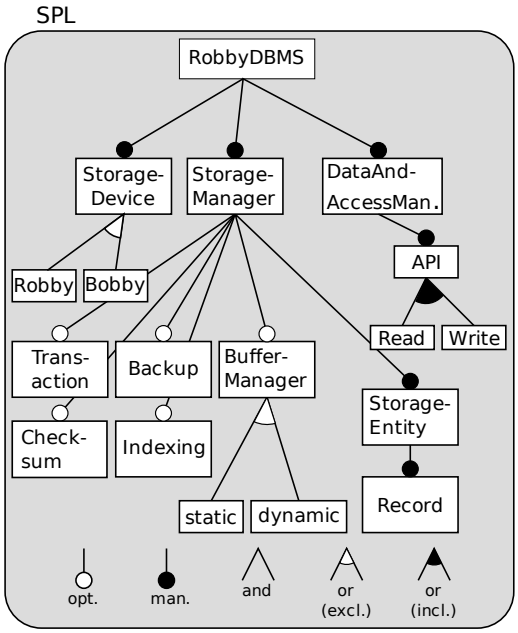

Figure 2: RobbyDBMS SPL; feature diagram

module. One example is feature CHECKSUMS that relies on READ and WRITE. Since feature WRITE is optional, we had to split feature CHECKSUMS for both variants (with and without write support).

3.2 Evaluation

The usage of FeatureC++ for developing SPLs has been evaluated before [13, 17]. Our results coincide with these evaluations and exhibit further insights on the efficiency and variability.

Efficiency

To evaluate the efficiency of FeatureC++, we developed several test programs that record the overhead in program storage (program size) and main memory caused by the use of FeatureC++. Table 1 displays the results of our analysis. The numbers reveal that, starting from a minimal variant of RobbyDBMS (only feature READ is selected), different variants with very little consumption of program storage and main memory can be generated. Furthermore, the table shows that the consumption of program storage and main memory increases almost linearly with the number of features selected. The overall deviation between the increase in program size caused by each feature separately and by the composition of all features is less than 2 %. To this end, the composition mechanisms of FeatureC++ do not introduce an overhead. Beside the generation of RobbyDBMS variants and the measurement of their memory footprints, we also investigated two optimization strategies for the composition.

First, we investigated whether the order of features has an impact on the program size. We expected that the order influences the program size, since a different order may activate different optimizations of the compiler during program compilation. To this end, we selected two features (CHECKSUMS and INDEX), that are independent but that address partially the same classes, and measured the memory footprint. We observed that the order of the two features has a small,

variant	Read	Write	Buffering	Transaction	Checksums	Index	program size	main memory
1	✓	o	o	o	o	o	658	0
2	✓	✓	o	o	o	o	996	0
3	✓	✓	o	o	✓	o	1278	0
4	✓	✓	o	o	o	✓	1686	0
5	✓	✓	o	o	✓	✓	2002	0
6	✓	✓	✓	o	o	o	2294	10
7	✓	✓	✓	o	✓	✓	3330	10
8	✓	✓	✓	✓	✓	✓	3734	12

✓feature selected; o feature not selected

Table 1: Different RobbyDBMS variants with the used program storage and main memory (in bytes)

but measurable impact (∼ 0,53 %), on the program size. A deeper analysis of the savings revealed that the resorting affected multiple places in the binary and a clear relation to the resorting was not possible. Thus, we were not able to draw any conclusions; the savings might have occurred only by chance. However, we suspect that the order of features has only a very little effect on the program size. Usually, a C++ compiler applies several optimizations during the program compilation. These optimizations also involve the resorting of source code pieces, which exceeds FeatureC++'s resorting capabilities.

Second, we investigated whether the program size could benefit from the application of optimization directives. For example, an optimization directive forces a compiler to apply methods for reducing the program size. FeatureC++'s compostion mechanisms rely on function inlining, which can be controlled by such directives.[2] We observed that, when several features refine a function, function inlining is not applied automatically by the C++ compiler. However, inlining can be enforced by the optimization directive `__attribute__((always_inline))`.[3] We measured that the application of this directive reduces the program size by 6.3 %. Although the reduction seems to be small, a different variant of the underlying HPL may be applicable. Furthermore, the C++ compiler used provides additional optimization directives that may further reduce the program size.

Our analysis reveals that FeatureC++ is appropriate in our case study. However, further research is necessary to show that our observations also hold in different domains and case studies.

Variability

Using PLE, we were able to model and implement a data management system that can be configured to a large extent based on features, such as BUFFERING, INDEXING, or TRANSACTION. The overall number of features is 19: 7 mandatory features that subdivide the base system and 12 optional

[2]Function inlining instructs the compiler to replace a function call with the body of the called function.

[3]This optimization directive is limited to the GCC compiler (`http://gcc.gnu.org`). However, other compilers have similar method modifiers, such as `__forceinline` in Visual C++.

features, which can be added via feature selection. However, a software feature like TRANSACTION represents only the variability of the SPL in terms of data management functionality.

The tailoring of the data management to a specific embedded systems platform is handled in the RobbyDBMS SPL, too. This includes the development and use of device drivers. Since hardware features are also variable (Figure 1; the EEPROM storage ranges from 1 KB to 4 KB) we used PLE for the development of these device drivers.

While implementing RobbyDBMS, we observed that a software feature requires one ore more hardware features. This requirement arises from functional interactions between hardware and software features. In the next section, we address this issue and discuss the influence of interactions between hardware and software features on the PLE process.

4. INTERACTIONS BETWEEN THE HPL AND THE SPL

In Figure 3, we give an example of possible interactions between the HPL and the SPL. Hardware feature EEPROM interacts with software features READ and WRITE, which can be traced back to the activity of the EEPROM as a data storage. Furthermore, feature BACKUP, which triggers write-backs to the EEPROM in case the data is held partially in main memory, either relies on feature 16-BIT TIMER or feature 8-BIT TIMER. Each hardware feature, such as EEPROM or 8-BIT TIMER, can be used by software features. We denote interactions between features on either side of the figure with bidirectional, dashed arrows. Although the necessity for a hardware feature arises on the SPL side, we show next that the interaction actually goes both ways and both sides influence each other.

Interactions between hardware and software features are bidirectional. As stated before, a software feature interacts functionally with hardware features. For example, software feature BACKUP relies on two different hardware features. From the product developer's point of view, the opposite direction, i.e., the dependence from the HPL to the SPL, becomes useful. This way, a hardware feature determines all possible variants of the SPL, which can be deployed on the HPL variant chosen. For example, hardware feature 8-BIT implies that 8 different variants of the SPL can be generated. The interactions observed allow to create a restricted feature model for the HPL and the SPL.

The treatment of interactions between HPLs and SPLs complicates the development of product lines. We believe that an integrated analysis and development process is necessary to face this complexity. In the following, we propose and discuss such an integrated process and highlight its benefits.

HPL and SPL Codesign.

In Figure 4, we depict our proposal of an integrated development process covering both hardware and software. It is based on domain engineering, where reusable features are being developed, and application engineering, where the reusable features developed are being used to generate a specific product. The figure illustrates that both domain and application engineering constitute chains of processes, such as analysis, design, and implementation, and that both chains are linked. We propose to extend this process with another ingredient covering the development of the HPL. While the

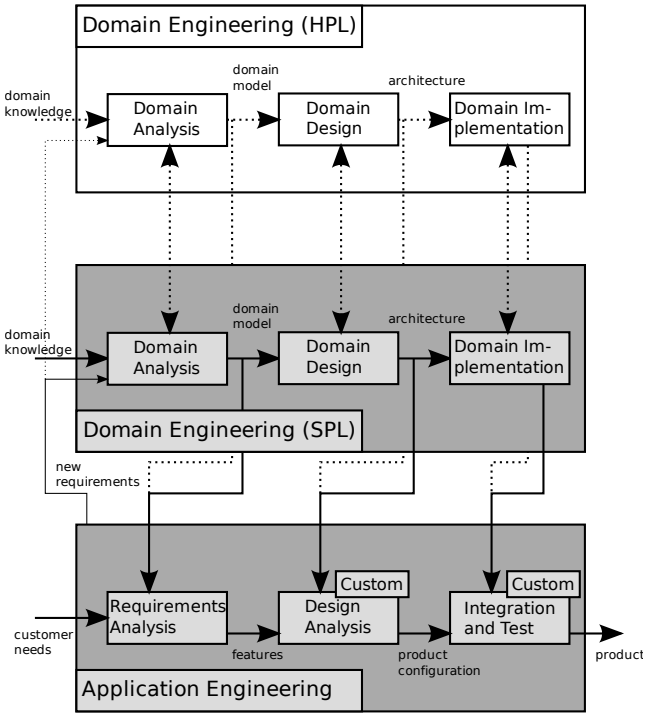

Figure 4: Integrated HPL and SPL development

output of each phase of HPL domain engineering is linked to its corresponding phase in application engineering, all phases of both domain engineering processes are linked mutually as well.

The links between the domain engineering phases cover the results of each phase (domain model, architecture, and implementation). We have already highlighted the interaction between domain models, looking at feature models, which constitute one form of representation of domain models (Figure 3). An interaction between both domain design phases covers architectural design decisions based on patterns used to describe a generic structure to achieve a highly configurable system. Finally, an interaction between domain implementations can be traced back to the software development. We have highlighted this interaction, too, as hardware features and their corresponding device drivers interact with software features.

We are aware that a full hardware/software codesign is not possible when dealing with a fixed HPL that a manufacturer supplies. However, we highlighted that PLE can benefit from taking the HPL in the process of domain engineering into account. The benefit arises from choosing among variants of an HPL.

5. PERSPECTIVE

In future work, we will investigate whether the observations we made and our proposed concept of an integrated development process also apply to other domains like operating systems for embedded system platforms. To this end, we plan to conduct further case studies. We will also study domains other than embedded systems. An interesting case study might be the Linux kernel, which consists of a huge number of features, which also relies on many capabilities

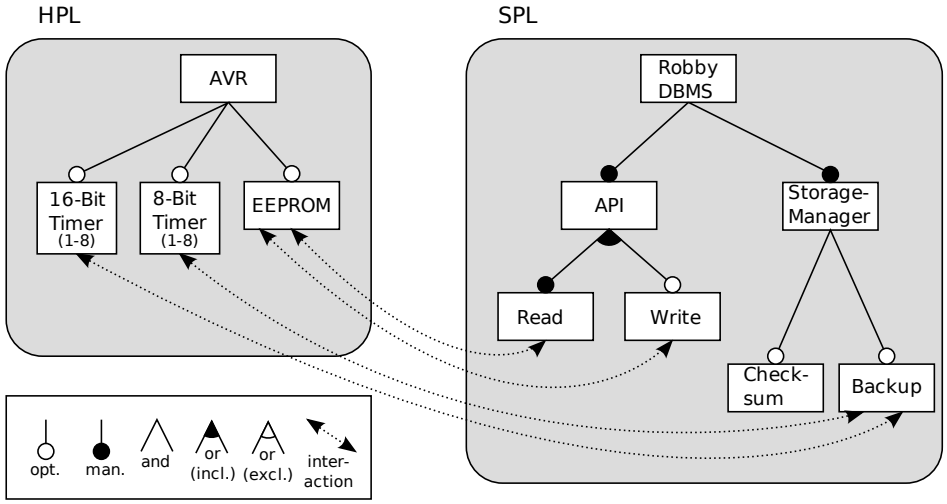

Figure 3: Example interactions between the HPL and the SPL

that different supported hardware platforms provide.

Besides conducting further case studies, we will also study each link between both domain engineering phases in more detail. The HPL used for this case study comprises a defined set of variants. An interaction between an HPL and an SPL might even be stronger than the ones we have observed so far. One reason for a stronger interaction is the existence of two technologies: (1) FPGAs[4] and (2) hardware description languages, such as VHDL[5] or Verilog. An FPGA is a computer chip, which contains programmable logic components that can be configured by customers. FPGAs are being programmed in hardware description languages. This way, changing requirements of customers regarding the hardware can also be handled. Furthermore, FPGAs add a further degree of freedom to PLE, since a feature, such as decoding a data stream, can be realized either in hardware with programmable logic or in software with a program running on a general purpose computer. Using both technologies together, the border between HPLs and SPLs becomes blurred and we expect that domain engineering for HPLs and SPLs consolidates.

6. CONCLUSION

We have reported on the development of an efficient data management system, RobbyDBMS, using product line engineering and employing the paradigm of feature orientation. We found that a feature-oriented design is suitable for modularizing variability in software like a data management system. We have achieved great variability in terms of data management functionalities and support for different embedded platforms. While implementing RobbyDBMS, we observed that the data management system and the underlying embedded platform interact and both the hardware and the software variability have to be taken into account. Consequently, we proposed a unified development process based on application and domain engineering, which combines hardware and software variabilities enabling an easier development of product lines that consist of hardware and software.

[4]field programmable gate arrays

[5]very high speed integrated circuit hardware description language

Acknowledgments

This work is being supported in part by the German Research Foundation (DFG), project number AP 206/2-1 and by the Metop Research Center.

7. REFERENCES

[1] N. Anciaux, L. Bouganim, and P. Pucheral. Smart Card DBMS: where are we now? Technical Report 80840, Institut National de Recherche en Informatique et Automatique (INRIA), Juni 2006.

[2] S. Apel and C. Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84, 2009.

[3] S. Apel, T. Leich, M. Rosenmüller, and G. Saake. FeatureC++: Feature-Oriented and Aspect-Oriented Programming in C++. Technical Report 3, Fakultät für Informatik, Universität Magdeburg, April 2005.

[4] 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 (GPCE)*, volume 3676 of *Lecture Notes in Computer Science*, pages 125–140. Springer-Verlag, 2005.

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

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

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

[8] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy. Rethinking Data Management for Storage-centric Sensor Networks. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 22–31, 2007.

[9] 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, Carnegie-Mellon University Software Engineering Institute, November 1990.

[10] J. Karlsson, A. Lal, C. Leung, and T. Pham. IBM DB2 Everyplace: A Small Footprint Relational Database System. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 230–232. IEEE Computer Society, 2001.

[11] G.-J. Kim, S.-C. Baek, H.-S. Lee, H.-D. Lee, and M. Joe. LGeDBMS: a Small DBMS for Embedded System with Flash Memory. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1255–1258. ACM Press, 2006.

[12] R. Krishnan. Future of Embedded Systems Technology. Technical Report GB-IFT016B, BCC Research, Juni 2005.

[13] M. Kuhlemann, S. Apel, and T. Leich. Streamlining Feature-Oriented Designs. In *Proceedings of International Symposium on Software Composition (SC)*, volume 4829 of *Lecture Notes in Computer Science*, pages 168–175. Springer-Verlag, 2007.

[14] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005.

[15] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[16] C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 1241 of *Lecture Notes in Computer Science*, pages 419–443. Springer-Verlag, 1997.

[17] M. Rosenmüller, S. Apel, T. Leich, and G. Saake. Tailor-Made Data Management for Embedded Systems: A Case Study on Berkeley DB. *Data and Knowledge Engineering (DKE)*, 2009.

[18] D. Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5):43–50, 2000.

[19] P. Zave. An Experiment in Feature Engineering. In *Programming Methodology*, pages 353–377. Springer-Verlag, 2003.