

Cellular DBMS - Architecture for Biologically-Inspired Customizable Autonomous DBMS

Syed Saif ur Rahman, Azeem Lodhi, Gunter Saake
School of Computer Science, University of Magdeburg, Germany.
{srahman, azeem, saake}@ovgu.de

Abstract

Data management is one of the fundamental requirements of ubiquitous computing. Existing data management systems are complex and provide a multitude of functionalities. Due to complexity and their monolithic architecture, it is difficult to tune these data management systems for consistent performance. In this paper, we extend our existing work of Cellular DBMS with the concept of autonomy. We present an aspect-oriented programming based model that enables us to monitor and evolve cells during data management operations for consistent performance.

1. Introduction

Ubiquitous computing [9] have converged into our daily life. An increasing number of devices are interacting with us to fulfill our basic needs in an invisible way. All these interactions produce data that needs to be managed and processed in an efficient manner.

Classical DBMSs are complex and less predictable. They need continuous administration and maintenance to keep performing at an optimal level, which results in high administrative and maintenance costs. DBMS has dozens of tuning knobs, but their coupling is so tight that understanding the effect of tuning one knob on another is difficult [5, 26]. Administration and maintenance of DBMS in the ubiquitous computing environment is extremely difficult, because of hard to reach and invisible deployment for long duration. Autonomous administration and maintenance of such systems is critical for their success. Along these lines, we want to reduce human resources by replacing them with autonomous processes that perform monitoring, configuration, and maintenance activities for data management systems.

In classical DBMS, as we increase the functionalities the performance becomes highly unpredictable. Even the most fine-grained components of these DBMS are quite large. Administrator always gets larger picture of DBMS perfor-

mance and it is difficult to scale down performance prediction to small data management units, e.g., a database column.

In [24], we proposed Cellular DBMS architecture that is inspired by biological systems, such as tissues of cells or swarms of animals. One of the major goals of Cellular DBMS is to achieve consistent performance and highly predictable behavior of complete DBMS as number of functionalities and amount of data in DBMS grows. In this paper, we extend our architecture of Cellular DBMS with the concept of autonomy. From autonomy in data management, we mean the capability of DBMS to monitor, diagnose, and tune itself for consistent performance. We present how large autonomous DBMS can be developed by composing and instantiating multiple customizable autonomous embedded databases. In Cellular DBMS, each cell is an instance of customizable embedded database [24]. The behavior of the complete DBMS depends on the collective behaviors of all atomic cells. We bring autonomy to each cell by using Aspect-oriented programming (AOP) [14] based model. We argue that Cellular DBMS architecture reduces DBMS complexity and when blended with autonomy, it can be used to develop highly predictable autonomous DBMS.

To demonstrate the model we extended Cellular DBMS, which is based on FAME-DBMS¹ [18], a customizable DBMS for embedded systems that we use to generate the cells. However, the general concept is independent of the used customizable embedded database. The customization of FAME-DBMS is based on feature-oriented programming (FOP) [3]. Feature-oriented programming is a paradigm for developing software product lines where programs are synthesized by composing features [3]. A feature can be defined as “A distinguishable characteristic of a concept that is relevant to some stakeholder” [11]. FAME-DBMS untangles and modularize DBMS functionalities as features. A decomposition of DBMS into features, i.e., the functionalities individual DBMS differ in, allows a developer to generate a tailor-made DBMS based on the selection of required features [17].

¹<http://www.fame-dbms.org/>

For discussion, we will use In-Memory Data Management, Persistent, SortedList, and B+Tree features of Cellular DBMS. In-Memory Data Management feature contains the functionality of an in-memory embedded database and can alone be used to construct a simple DBMS cell. Persistent feature means an embedded database contains the functionality of storing data on persistent storage. SortedList manages data using multiple instances of In-Memory Data Management feature. More detailed information on Cellular DBMS features can be found in [24].

In this paper, Section 2 and 3 gives background knowledge of related concepts. Section 4 explains Cellular DBMS architecture and new extensions introduced in this paper. Section 5 gives the details of design principles for autonomy in Cellular DBMS. Section 6 gives a glance at implementation technologies. A detailed related work is provided in Section 7. Section 8 concludes the paper with some hints for the future work.

2 Autonomy

“Organic computing, which is a new term covering the bio-inspired mechanisms in engineering and computer science related fields, is attempting to build highly-scalable architectures, which are self-organizing, self-maintaining, and self-healing” [8]. A key motivation of Cellular DBMS architecture is to bring autonomy for self-tuning data management [5, 26]. Autonomy is essential to reduce the human effort in DBMS administration. “The embedded vendors all acknowledge the need for automatic administration, but fail to identify precisely how their products actually accomplish this” [20]. Autonomous DBMSs monitor themselves and perform tuning operations based on pre-defined policies. Monitoring is the most fundamental functionality in an autonomous DBMS, but it also possesses overheads, and we need to reduce these overheads. In general, self-* [7] capabilities in autonomous DBMS systems should not be programmed explicitly. Instead, a DBMS should be programmed to learn its behaviors based on goals, e.g., such as performance, security, reliability, and power consumption.

3 Aspect-oriented programming

Aspect-oriented programming [14] is methodology that emerged with the aim to separate cross-cutting concerns. AOP ensures code scalability and maintenance by preventing code tangling and scattering [14]. Using AOP, cross-cutting code is separated from the program logic using aspects. These aspects, such as data persistence, transaction management, and data security, etc., can either be provided by a software component or may be required by it [14]. Using join points, pointcuts, and advice an aspect weaver brings the program code and aspect code together [13]. Join

points are points in the execution of a program and are events of interest for aspect weaving [13]. Pointcuts is the collection of join points and is used for selection of related method execution points [13]. An advice is the intended behavior to be weaved [13].

4 Cellular DBMS

A Cellular DBMS consists of many customized atomic embedded databases, i.e., Cells [24]. Each cell is based on RISC-style architecture with simple and limited functionality [26]. A simplest cell handles a key/value pair and has definite (optimal) data handling capacity. With the data growth, more cells should be induced into DBMS. Virtually each cell uses *Binary Fission*² mechanism to grow. In binary fission, biological cell grows to twice its starting size and then split into two cells, each cell having a complete copy of its essential genetic material. Not exactly, but similarly each DBMS cell splits into two equal halves. One-half is left in the parent cell where as the other half is moved to a newly induced cell. The main goal of Cellular DBMS architecture is to reduce the DBMS complexity and to ensure highly predictable data management with consistent performance. As an end-product, we envision a biologically-inspired highly customizable autonomous DBMS. A cell

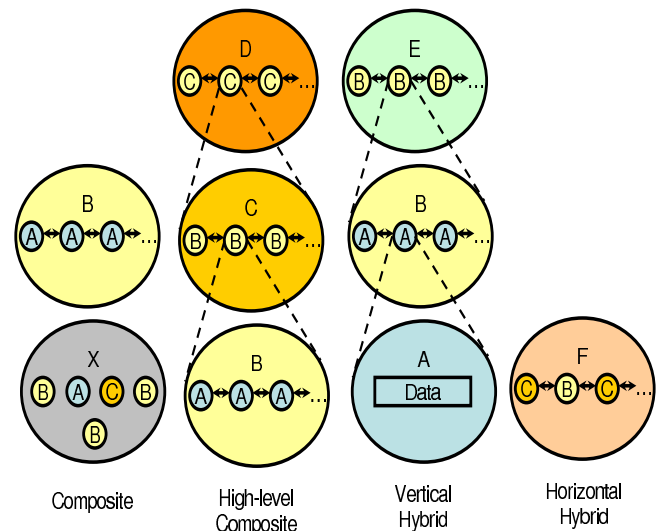


Figure 1. Different types of Cells.

can be composed of multiple similar or different cells related to each other as shown in Figure 1. Such composition of cells is termed as *Composite Cell*. Composite cell can be used to implement a table in Cellular DBMS where each column is implemented by a cell that could be of different

²<http://www.cals.cornell.edu/cals/micro/research/labs/angert-lab/binary.cfm>, Accessed: 23-06-2009

type, e.g., one cell contains in-memory data management functionality whereas another cell can also store persistent data. It can also be used to handle large amount of data that simple cells cannot handle. In Cellular DBMS, composite cells can be built from simple cells, as well as from composite cells, which results in high-level composite cells as shown in Figure 1. Each composite cell should have limited (optimal) data handling capacity. With the data growth, more composite cells could be induced into the DBMS.

Evolution In this paper, we extend the Cellular DBMS architecture with the concept of *Evolution*. Evolution in Cellular DBMS means run-time transformation of cells. Evolution can be constructive as well as destructive. From constructive evolution, we mean the transformation of a cell from one form to another in such a way that the previous form becomes an atomic integral unit of new form as shown in Figure 2. New form of such an evolved cell should have larger data handling capacity. Evolution is a mandatory concept to bring autonomy in Cellular DBMS. For example, consider a cell X that is initially an in-memory data management cell. We also support a SortedList that stores data using multiple in-memory data management cells. SortedList is the simplest composite cell. From evolution, we mean the transformation of cell X to SortedList so that cell X becomes an atomic integral unit of SortedList. We elaborate the evolution in detail in the design principles section.

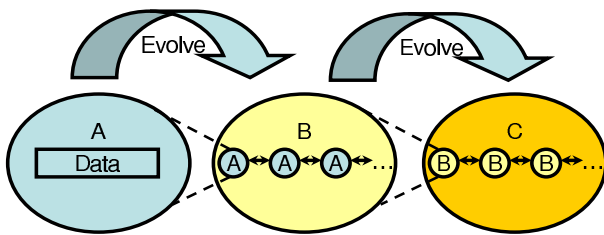


Figure 2. Evolving cell.

Hybrid Cell In this paper, we also introduce the concept of *Hybrid Cell*. We could have horizontal as well as vertical hybrid cells as shown in Figure 1. From horizontal hybrid cell, we mean a composite cell that is composed of different type of cells such that each type is handling a definite data range. For example, we want to store city codes to be used in the contact book of a mobile phone product. If mobile is to be used in European Union (EU), frequency to access city codes of EU countries is much higher compared to city codes of Australia. Using horizontal hybrid cell, we can store data in a composite cell in such a way that EU city codes should be stored in cell with a type that is suitable for faster access time whereas we store remaining city codes in a cell, which requires less storage space. We can exploit this

feature in conjunction with autonomy to move data among different cells based on their usage scenario and available resources.

From vertical hybrid cell, we mean a high-level composite cell that is composed of different type of cells at different levels. For example, we have In-Memory Data Management type cell at the fine-grained level, i.e., level 0. At one level above, i.e., level 1, we have B+Tree composite cell using multiple In-Memory Data Management cells, and finally one more level above, i.e., level 2, we have SortedList using multiple B+Tree composite cell. Vertical hybrid cell can be generated using the evolution approach discussed in this paper; however, implementation of hybrid cells is future work.

5. Design Principles for Autonomy in Cellular DBMS

Once a DBMS is customized and implemented according to the application need, deployed DBMS may need to possess self-* autonomous behavior. According to our proposed architecture, each cell can contain optional lightweight monitoring functionality. The purpose of monitoring functionality is to monitor the cell for specific parameters according to defined policy for DBMS cell goals. As monitor identifies some event of interest, each cell should be able to adapt to changes. Additional to a cell-level monitoring there should be a monitoring component at composite cell as well. It should get feedback from an individual cell-monitoring component and should by itself monitor certain parameters at composite cell level. It enables global monitoring of cells for adaptation and fixing according to defined policy.

For autonomy, the most fundamental functionalities are Monitoring, Diagnostics, and Tuning. We present an AOP based model for autonomy at cell-level. We argue based on literature survey that AOP join-point model can be used to implement efficient monitoring functionality for data management. For diagnostics, we use the state of the cell, and result of data management operations to identify the definite tuning points. For tuning we present an approach that we named "*Evolving Cell*". From an evolving cell, we mean a DBMS cell that can change itself at run-time. For example, an evolving cell can change from an in-memory data management cell to a persistent data management cell. In the proposed architecture, monitoring, diagnostic, and tuning components should also be customizable according to the cell functionalities to ensure reduced monitoring overhead.

In the presented model, we propose the use of AOP join-point model for implementing monitoring functionality. During monitoring, the most important functionality is tracing. From tracing, we mean collection of cell state information that is needed to diagnose and tune the DBMS.

Stress (No. of Records)	256	1024	2048	3072	4680
Cell A	4	39	138	297	666
Cell B	10	81	277	618	1425
Evolving Cell	4	39	80	119	175

Table 1. Average execution time for stress test in millisecond for different Cellular DBMS cells.

For each join-point, before advice should be used to trace the execution. For each join-point, after advice should be used to diagnose the execution for abnormality. If any abnormality is detected during diagnostics, tuning should be executed to counter the abnormality.

To explain the concept in detail, we describe a scenario. We compose a Cellular DBMS that supports an in-memory data management cell and an in-memory data management composite cell, i.e., a SortedList. We term in-memory data management cell as Cell A and in-memory data management composite cell as Cell B. Cell A stores data in a single memory chunk where as Cell B is composed from multiple Cell A. It is also shown in Figure 2. Both cells store definite amount of data, however, capacity of data storage in Cell B is larger. In contrast, the complexity and main memory requirement of Cell A is relatively low. To differentiate the behavior of two cells we presented the average execution time in millisecond of stress test on both cells in Table 1³ and Figure 3. We executed test with different stress values, i.e., number of records that are inserted, retrieved, and deleted. For Cell A, we kept memory allocation large enough to accommodate all test data into a single cell. For Cell B, we kept memory allocation of each Cell A small enough so that multiple cells can be used to demonstrate the change in behavior. It can be observed that Cell A performs much faster than Cell B, because of reduced execution complexity. Cell A also consumes less main memory, because of simple data management structure. Based on the results, we argue that cell complexity should only be increased with the data growth. For example, we should use the Cell A as long as the data is small enough for it to handle. As data grows to exceed the limit of Cell A capacity, we bring the concept of evolving cell to evolve cell from type A to type B, i.e., Cell A becomes part of Cell B and evolved cell has relatively larger data management capability. In Cellular DBMS we can evolve cells to higher level, e.g., compose Cell C based on multiple B cells and so on. Autonomy should be kept at the fine-grained level of Cell A to ensure highly predictable and tunable behavior at the smallest data management unit.

To generate better results, we first analyzed the optimal memory allocation of Cell A that resulted in the fastest ex-

ecution time for stress test using Cell B. We observed that for our sample stress data, both, i.e., too small as well as too large memory allocation was found to be inefficient. Once we identified the optimal memory allocation for Cell A, our evolving cell implementation uses Cell A until its data management limit is reached. Monitoring component keeps monitoring the Cell A based on join-point specification and keeps trace of the required information. As our diagnostic implementation detects that Cell A is out of memory. It executes the tuning implementation, which evolves Cell from type A to B by injecting Cell A in Cell B. From end-user and application point of view, it is kept transparent when evolution occurs. By using this approach, we ensure that complexity of data management implementation should only be increased as the amount of data is increased.

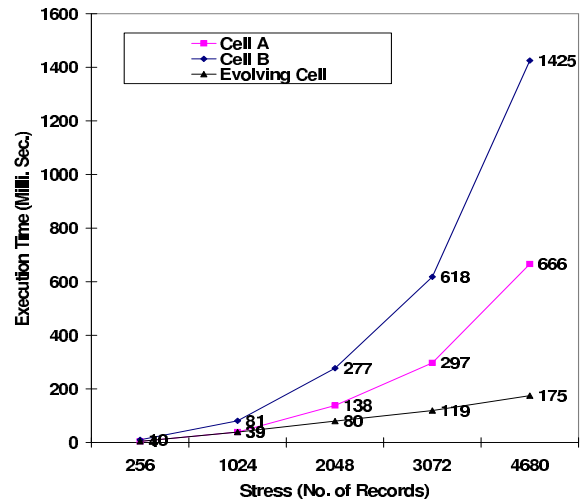


Figure 3. Average execution time graph for stress test in millisecond for different Cellular DBMS cells.

6. Implementation Technologies

To implement the model we extended Cellular DBMS, which is based on FAME-DBMS. FAME-DBMS is developed using FeatureC++ [18]. FeatureC++⁴ supports feature-oriented programming for C++. We utilized AspectC++⁵ [21] for using AOP constructs. FeatureC++ also supports AOP extensions as discussed in [1, 2], however, we used AspectC++ independently to have greater control on AOP constructs. Code transformation model for our implementation using FeatureC++, AspectC++, and C++ compiler is shown in Figure 4. Monitoring implementation is optional in Cellular DBMS and can be removed for ubiqui-

³Average execution time is used to demonstrate the concept and may vary in future work.

⁴http://www.witi.cs.uni-magdeburg.de/iti_db/fcc/

⁵<http://www.aspectc.org/>

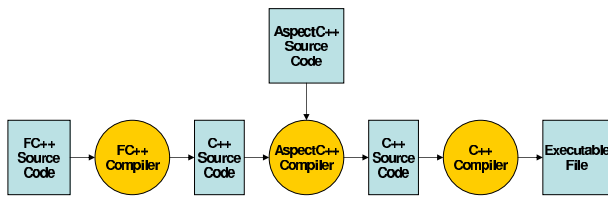


Figure 4. Source code transformation.

tous embedded systems with extremely high resource constraints.

7. Related Work

In 60's research on self-organization started and Eigen [6] first introduced the concept of combination of nature and self-organizing technical systems. Current research reviews on biological self-organization can be found in [4, 8].

The Infobionics Knowledge Server⁶ also know as Infobionics Cellular DBMS claims to be first fluid dynamic solution for managing, navigating, and querying data. The Infobionics Cellular Database Management System places information in individual Data Cells, which can be flexibly compiled via Link Cells into an infinite number of DataSets⁷. However, in patent [19] it is stated as "A system for acquiring knowledge from cellular information. The system has a database comprising a database management module ("DBMS")." The concepts presented for Cellular DBMS in this paper and related publications are different from the ones used by Infobionics Cellular DBMS. We are inspired from human cellular organization whereas in contrast Infobionics Cellular DBMS is inspired from human brain. For each cell in Cellular DBMS high customizability, limited functionality, and highly predictable behavior is backbone of the concept. Internal architectural details of Infobionics Cellular DBMS are not publicly available, however, based on the little available information we found our work quite different in terms of both concept and implementation.

Kersten et. al in [12] proposed architecture for Cellular database system. According to the proposal, each cell is a bounded container, i.e., a workstation or a mobile unit linked into a communication infrastructure. This work also envision a cell as an autonomous DBMS as we do, however, realization of autonomy is different in our approach. We utilized an AOP based model for implementing autonomy. Furthermore, we suggested freedom of using any embed-

ded database as cell, which supports customization. Verroca et. al in [25] used the term Cellular Database for a solution for cellular network data management. Kodama et. al in [15, 16] proposed a cellular DBMS that is based on the layer model. It is based on incremental modular abstraction hierarchy. Mechanisms are gradually added in it as a global model. They have applied the cellular model to model web-based information spaces for designing the cellular DBMS [15].

Greenwood et. al in [10] outlined the case of the use of dynamic AOP for autonomic systems. Truyen et. al in [23] demonstrated the applicability of AOP for implementing self-adaptive frameworks. Tesanovic et. al in [22] proposed the concept of aspectual component-based real-time system development (ACCORD) and applied it successfully in the design and development of a component-based embedded real-time database system (COMET). Rosenmüller et. al in [18] used AOP and FOP for implementing SPLs. FAME-DBMS [18] is based on an SPL approach and promises benefits for the embedded domain as proposed by Leich et. al [17]. Cellular DBMS is developed by extending FAME-DBMS and this work focus on using AOP based model to achieve autonomy in Cellular DBMS. By incorporating autonomy, Cellular DBMS should be able to self-tune itself for required performance with less human intervention.

8. Conclusion and Future Work

Developing self-* autonomous DBMS based on processes in biological systems has good potential for ubiquitous computing environment. In this work, we presented an AOP based model for implementing autonomy at cell level in Cellular DBMS. We also give the idea how evolving cells can be used to self-tune data management with data growth. Our presented implementation ensures that initially for small amount of data, simpler data management functionality is used. We evolve the functionality with the data growth maintaining consistent performance. In our proposed model, we argue that we will be able to develop highly customizable autonomous DBMS that can scale from requirement of small embedded systems to large-scale enterprise systems. As future work, we intend to implement the hybrid cell. Cell mobility is key future research area. In Cellular DBMS architecture, each cell is based on specialized customized implementation of embedded database. We need specialized mechanisms for monitoring heterogeneous data management components and it is an important future work.

Acknowledgment

Syed Saif ur Rahman is funded by Higher Education Commission of Pakistan and NESCOM, Pakistan.

⁶<http://www.infobionics.com/>

⁷"Cellular DBMS Seeks Business Intelligence Beta Sites", PRESS RELEASE, infobionics, http://www.infobionics.com/news/news_2/file_item.pdf, Accessed: 23-06-2009

References

- [1] 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, Apr. 2005.
- [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] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.
- [4] S. Camazine, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulez, and E. Bonabeau. *Self-organization in biological systems*. Princeton University Press, 2001.
- [5] S. Chaudhuri and G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 1–10, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [6] M. Eigen and P. Schuster. *The Hypercycle: A Principle of Natural Self-Organization*. Springer-Verlag, Berlin, 1979.
- [7] G. R. Ganger, J. D. Strunk, and A. J. Klosterman. Self-* storage: Brick-based storage with automated administration. Technical report, Carnegie Mellon University, 2003.
- [8] C. Gershenson and F. Heylighen. When Can we Call a System Self-organizing? *CoRR*, nlin.AO/0303020, 2003. informal publication.
- [9] A. Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, Indianapolis, Indiana, March 2006.
- [10] P. Greenwood and L. Blair. Using Dynamic Aspect-Oriented Programming to Implement an Autonomic System. In *Proceedings of the 2004 Dynamic Aspects Workshop (DAW04)*, pages 76–88, 2004.
- [11] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.
- [12] M. L. Kersten. A Cellular Database System for the 21st Century. In *ARTDB '97: Proceedings of the Second International Workshop on Active, Real-Time, and Temporal Database Systems*, pages 39–50, London, UK, 1998. Springer-Verlag.
- [13] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.
- [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 11th European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä, Finland, June 1997. Springer-Verlag.
- [15] T. KODAMA and T. KUNII. Development of new DBMS based on the cellular model-from the viewpoint of a data input. *IEIC Technical Report (Institute of Electronics, Information and Communication Engineers)*, 102(208):97–102, 2002.
- [16] T. KODAMA, T. KUNII, and Y. SEKI. A Development of a Cellular DBMS Based on an Incrementally Modular Abstraction Hierarchy. *Joho Shori Gakkai Kenkyu Hokoku*, 2004(45):43–50, 2004.
- [17] T. Leich, S. Apel, and G. Saake. Using Step-Wise Refinement to Build a Flexible Lightweight Storage Manager. In *Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems (ADBIS'05)*, volume 3631 of *Lecture Notes in Computer Science*, pages 324–337. Springer Verlag, 2005.
- [18] 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 *SETMDM '08: Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*, pages 1–6, New York, NY, USA, 2008. ACM.
- [19] J. H. Sabry, C. L. Adams, E. A. Vaisberg, and A. M. Crompton. Database system for predictive cellular bioinformatics, United States Patent 6631331, October 2003.
- [20] M. I. Seltzer and M. A. Olson. Challenges in embedded database system administration. In *WOES'99: Proceedings of the Workshop on Embedded Systems on Workshop on Embedded Systems*, pages 11–11, Berkeley, CA, USA, 1999. USENIX Association.
- [21] O. Spinczyk, D. Lohmann, and M. Urban. AspectC++: an AOP Extension for C++. *Software Developers Journal*, pages 68–74, 2005.
- [22] A. Tesanovic, R. Teanovic, D. Nyström, J. Hansson, and C. Norström. Towards Aspectual Component-Based Development of Real-Time Systems. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 278–298. Springer-Verlag, 2003.
- [23] E. Truyen and W. Joosen. Towards an aspect-oriented architecture for self-adaptive frameworks. In *ACP4IS '08: Proceedings of the 2008 AOSD workshop on Aspects, components, and patterns for infrastructure software*, pages 1–8, New York, NY, USA, 2008. ACM.
- [24] S. S. ur Rahman, M. Rosenmüller, N. Siegmund, S. Sunkle, G. Saake, and S. Apel. Data Management for Embedded Systems: A Cell-based Approach. In *20th International Workshop on Database and Expert Systems Application (DEXA 2009)*. IEEE Computer Society, 2009. To appear.
- [25] F. Verroca, C. Eynard, G. Ghinamo, G. Gentile, R. Arizio, and M. D'Andria. A Centralised Cellular Database to Support Network Management Process. In *ER '98: Proceedings of the Workshops on Data Warehousing and Data Mining*, pages 311–322, London, UK, 1999. Springer-Verlag.
- [26] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *Proceedings of the 28th VLDB Conference*, Hongkong, China, 2002.