Nr.: FIN-012-2009

# Cellular DBMS: An Attempt Towards Biologically-Inspired Data Management

Syed Saif ur Rahman, Gunter Saake

*Arbeitsgruppe Datenbanken*

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Technical Report

Nr.: FIN-012-2009

# Cellular DBMS: An Attempt Towards Biologically-Inspired Data Management

Syed Saif ur Rahman, Gunter Saake

*Arbeitsgruppe Datenbanken*

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

**Impressum** (§ 5 TMG):

*Herausgeber:*
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Der Dekan

*Verantwortlich für diese Ausgabe:*
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Syed Saif ur Rahman
Postfach 4120
39016 Magdeburg
E-Mail: srahman@ovgu.de

http://www.cs.uni-magdeburg.de/Preprints.html

*Auflage:* 52

*Redaktionsschluss*: 04.08.2009

*Herstellung:* Dezernat Allgemeine Angelegenheiten,
Sachgebiet Reproduktion

*Bezug:* Universitätsbibliothek/Hochschulschriften- und
Tauschstelle

# Cellular DBMS: An Attempt Towards Biologically-Inspired Data Management

Syed Saif ur Rahman, Gunter Saake
{srahman,saake}@ovgu.de

Department of Technical and Business Information Systems,
Faculty of Computer Science,
Otto-von-Guericke University,
Magdeburg, Germany

August 3, 2009

**Abstract**

Existing database management systems (DBMS) are complex and less predictable (i.e., the consistency of performance with the increase of functionality and the data growth is not certain). Database researchers acknowledge the need for revisiting DBMS architectures to fulfill the needs of new hardware and application trends. We propose a biologically inspired DBMS architecture called *"Cellular DBMS"*. The Cellular DBMS architecture promises development of highly customizable and autonomous DBMS. This report explains in detail the design principles for Cellular DBMS architecture. It also explains an aspect-oriented programming based model to equip Cellular DBMS architecture with autonomy. Finally, it presents an extension to decomposed storage model (DSM) for use in Cellular DBMS.

# Contents

# List of Figures

# List of Tables

# 1 Introduction and Motivation

In the past, database research got motivations from arrival of new hardware, software, and applications for further progress. These motivations are still there and will persist in the future. Desire for the improvement kept researchers busy for finding the breakthrough in prevailing requirements. Sometimes we get many breakthroughs in a short period and sometimes we wait for decade to get few. In the prevailing era, we have explosion in the data growth and usage scenario, because of wide spread usage of internet and advent of new applications (e.g., social networking, virtual worlds, etc.). Hardware trends are changing and the processing and storage unit cost has reduced. Many assumptions about secondary storage and main-memory, etc., made in past are no longer valid and many bottlenecks, such as network communication cost have changed. Leading database researchers found a consensus on the need of revisiting database engines, accommodating architectural shifts in computing hardware and platforms, and finding solutions for new usage scenarios [3]. *Cellular DBMS*[1] is an effort to contribute to database research in above-mentioned directions.

Existing data management solutions are complex. These solutions have evolved over time and now they provide a multitude of functionalities. These functionalities are tightly coupled within their monolithic architecture [70]. Due to complexity, their performance is less predictable, i.e., the consistency of performance with the increase of functionality and the data growth is not certain and it is difficult to assess, how performance will vary for different hardware, workload, and, operating system, etc. Continuous administration and maintenance is needed to keep them performing at an optimal level, which results in high administrative and maintenance cost. Existing database management systems (DBMS) have dozens of tuning knobs. Internal sub-systems are tightly coupled. Effect of tuning a knob on other knobs and their performance is less predictable [14, 70]. Furthermore, existing DBMS architectures and solutions were designed decades ago considering legacy hardware and their bottlenecks. Now many opportunities exist to redesign existing data management architectures for exploiting features of new hardware.

Database researchers have suggested transition of DBMS from monolithic to diversified architecture with small, simple, and reusable components of limited functionality with clean inter-component interaction [3, 70]. The Cellular DBMS architecture is designed by considering these suggestions. The Cellular DBMS architecture takes inspiration from biological systems. We want to utilize the mechanisms that exist in biological systems for data management. Using these mechanisms, we want to develop highly customizable and autonomous DBMS with more predictable performance. The vision for

[1] "Cellular DBMS", `http://wwwiti.cs.uni-magdeburg.de/~srahman/CellularDBMS/index.html`

Cellular DBMS predictability is shown in Figure 1, i.e., a DBMS should be consistently predictability with the data growth and addition of functionalities. To achieve these goals in Cellular DBMS, we envision integration of techniques from different relevant fields, such as software engineering, distributed data management, computer networks, and parallel processing.

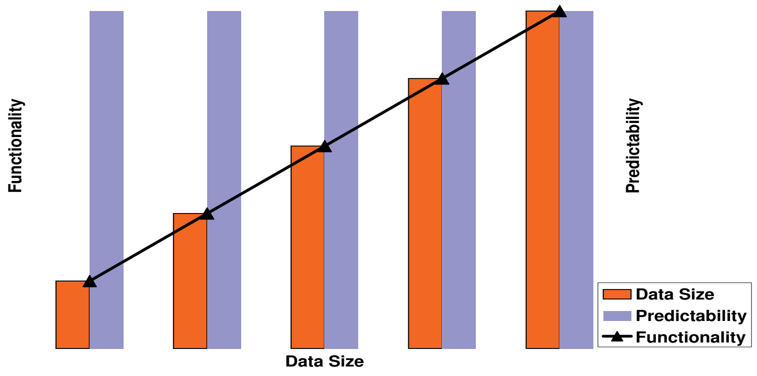

Figure 1: Cellular DBMS goal for predictability.

This report is organized as follows: Section 2 introduces the related concepts required for background information and technical discussion. A detailed related work is provided in Section 3. Cellular DBMS architecture and its design principles are explained in Section 4. Section 5 presents the implementation details. Sample implementation scenarios are discussed in Section 6. Section 7 concludes the report with some directions to future work.

## 2 Related Concepts

### 2.1 DBMS Aspect

#### 2.1.1 Storage Models

Storage model selection is an important design decision for DBMS architecture. In this sub-section, we will explain the two most commonly used storage models, i.e., N-Ary Storage Model and Decomposed Storage Model followed by discussion on design decision of selecting decomposed storage model for Cellular DBMS architecture.

**N-Ary Storage Model (NSM)** *N-Ary Storage Model (NSM)* stores data as seen in the relational conceptual schema, i.e., all attributes of a conceptual schema record are stored together [17]. Most of the existing DBMS are NSM based.

**Decomposed Storage Model (DSM)** *Decomposed Storage Model (DSM)* is a transposed storage model [9] that stores all values of the same attribute of the relational conceptual schema relation together [17]. Svensson et al. mentioned the Cantor project [28, 29] as the pioneer for this approach [59]. Copeland and Khoshafian in [17] concluded many advantages of DSM. We listed few of them as follows:

- Simplicity (Copeland and Khoshafian related it to RISC [44])
- Less user involvement
- Less performance tuning requirement
- Reliability
- Increased physical data independence and availability

In literature column-oriented [58], vertical fragmentation [19], vertical partitioning [2], etc., are terms also used to refer to DSM.

**Discussion** Copeland and Khoshafian in [17] analyzed both approaches and concluded that neither of the two approaches could be an ideal solution for all domains. DSM relatively required more storage space, however, the required storage can be reduced by using compression techniques [25]. Update and retrieval performance of both models depend on the nature of data and implementation of models. DSM is known for fast retrieval whereas NSM is efficient in fast updates [25]. Copeland and Khoshafian suggest that many disadvantages of DSM can be avoided by using hardware and software techniques, such as differential files, multiple disks, large main-memory, etc [17]. DSM allows using the CPU cache efficiently [73]. Zukowski et al. in [76] compared the two approached on most recent hardware for CPU performance trade-offs in block-oriented query processing. Zukowski et al. concluded that it depends on query to identify, which data layout is better, furthermore, they recommended on-the-fly conversion between these formats for better performance and stressed on research on hybrid data layout using best of both approaches. Example of hybrid data layout can be found in PAX [4] and MonetDB/X100 [73].

### 2.1.2 Embedded Database

An embedded database is a data management solution that is embedded into its user-application. However, the same term is also used for a database that resides in an embedded system [41]. An embedded database is transparent to application end-user. An embedded database possesses many special characteristics as mentioned in literature [40, 42, 41, 50, 53]. We list some important embedded database characteristics as follows:

- Small footprint
- Small set of tasks
- Little maintenance
- Multiple-platforms support
- API based access

## 2.2 Software Engineering Aspect

For designing data management architectures, knowledge of software engineering aspects plays an important role. In the end, we have to implement data management using software engineering techniques. Many researchers already have considered software engineering aspect while designing data management architectures and found its impact on design decision as too high [36, 43, 50, 61]. In Cellular DBMS, we also consider the software engineering aspects that can benefit us to achieve the targeted goals.

**Software Product Line** *Software Product Line (SPL)* engineering is an efficient and cost-effective approach to produce a family of related programs for a domain [45]. A product line shares a common set of features developed from a common set of software artifacts [16]. It has been shown that a high degree of customizability makes an SPL a suitable candidate for the development of data management systems [36]. Rosenmüller et al. in [48] and Saake et al. in [51] demonstrated how SPL overcomes the limitation of customizability and performance for data management in embedded systems that exist in other approaches.

**Feature-oriented Programming** *Feature-oriented programming (FOP)* is a mechanism for developing software product lines where programs are synthesized by composing features [8]. A feature can be defined as “A distinguishable characteristic of a concept that is relevant to some stakeholder” [27]. When an SPL is designed in terms of features, creating a program is simply the selection of the required features and composition of the according feature modules [8].

**Aspect-oriented Programming** *Aspect-oriented programming (AOP)* [33] is a methodology that emerged with the aim to separate cross-cutting concerns. AOP ensures code scalability and maintenance by preventing code tangling and scattering [33]. 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 could be required by it [33]. Using join-points, pointcuts, and advice; an aspect weaver brings the program code and aspect code together [32]. Join-points are points in the execution of a program and are events of interest for aspect weaving [32]. Pointcuts is the collection of join-points and is used for selection of related method-execution points [32]. An advice is the intended behavior to be weaved [32].

### 2.3 Autonomy

Autonomy in data management means the capability of DBMS to monitor, diagnose, and tune itself for consistent performance. Autonomy is an essential feature to reduce the human effort in DBMS administration. Automatic administration can reduce the administration cost for data management of large enterprises as well as for embedded systems. "The embedded vendors all acknowledge the need for automatic administration, but fail to identify precisely how their products actually accomplish this" [53]. Similarly, Chang et al. based on their experiences of Bigtable [12] implementation stressed the importance of proper system-level monitoring of the system itself and its users to detect and fix problems. Autonomous DBMSs monitor themselves and performs tuning operations automatically based on pre-defined policies. A key motivation of Cellular DBMS architecture is to achieve autonomy for self-tuning data management [14, 70].

# 3 Related Work

Cellular DBMS is an innovation in its own, but it did not appear from nowhere. All concepts and technologies that are joined together in Cellular DBMS have their counterpart in literature and industry. We believe that few concepts are new, but to make such a claim is unrealistic. For decades, many researchers have worked on similar topics and always found the possibility to have similar findings with different names in different domains. Cellular DBMS inherit SPL-based approach from FAME-DBMS[2] [50]. Different aspects of Cellular DBMS have to be covered to convince the reader for the originality of Cellular DBMS, such as inspiration from biological systems, use of AOP, column-oriented storage, and embedded database, etc., all have to be covered to convince the reader for the originality of Cellular DBMS.

[2]"FAME-DBMS", `http://www.fame-dbms.org/`

## 3.1 Term ”Cellular DBMS” in Literature

The Infobionics Knowledge Server[3] 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 DataSets[4]. However, in patent [52] 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 current and related publications [64, 65] 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 available information in form of patent [52] and press releases[4], we found our work quite different in terms of both concept and implementation.

Kersten et al. in [31] proposed an 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. It assumes the internet as the underlying communication network. 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 customizable embedded database as cell.

In 2003, Kersten et al. along with other researchers in [30] again tried to draw the focus of database community towards Organic databases at VLDB. In 2006, Kersten and Siebes took step forward with the concept of an Organic Database System in [66]. They provided the vision of new database architectures as ”an Organic Database System where a large collection of connected, autonomous data cells implement a semantic meaningful store/recall information system” [66].

Verroca et al. in [67] used the term Cellular Database for a solution for cellular network data management. Kodama et al. in [34, 35] 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 [34].

[3] “The Infobionics Knowledge Server”, `http://www.infobionics.com/`

[4] “Cellular DBMS Seeks Business Intelligence Beta Sites”, PRESS RELEASE, infobionics, `http://www.infobionics.com/news/news_2/file_item.pdf`, Accessed: 21-07-2009

## 3.2 Embedded Database

**COMET** Tesanovic et al. in [61] 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). COMET DBMS was developed for resource-constrained embedded vehicle control-systems. COMET DBMS is highly tailorable for different requirements and was developed using component-based and aspect-oriented programming approaches. Cellular DBMS also target real-time embedded domain for its variants. It has similarity with COMET DBMS in terms of use of AOP in this domain.

**Berkeley DB** Berkeley DB [42] is a customizable embedded database system. Cellular DBMS takes many inspirations from Berkeley DB. Key/Value pairs, API-based access, main-memory database, and small footprint all these concepts have their counterpart in Berkeley DB.

**FAME-DBMS** FAME-DBMS [50] is developed based on an SPL approach. SPL approach promises benefits for the embedded domain as proposed by Leich et al. [36]. Our Cellular DBMS implementation is an extension of FAME-DBMS, however, concept of Cellular DBMS can be implemented using any customizable embedded database. Since we extend FAME-DBMS, all features of it can become part of Cellular DBMS, but we have many unique features of Cellular DBMS that are not part of FAME-DBMS, such as column-based storage, different cell type implementations, autonomy, evolution, etc. It is not an exhaustive list of features for Cellular DBMS. We have many new features in development phase and many are planned as future work. Data management of embedded system is the focus of FAME-DBMS, in contrast, Cellular DBMS is not confined to it. FAME-DBMS focus derivation of concrete instance of a DBMS by composing features of DBMS product line whereas Cellular DBMS derive one or more instances of any DBMS and exploits them in concert for data management.

## 3.3 In-Network Query Processor

Cellular DBMS also target sensor networks domain for its variants. Well-known databases in this domain are in-network query processors.

**TinyDB** TinyDB[5] [38] is an in-network acquisitional distributed query processor for sensor networks. In acquisitional processing, records in table are only materialized (i.e., acquired) as needed to satisfy the query, and are usually stored for a short period of time or delivered directly out of the network [38]. It runs on Berkeley mote platform on top of TinyOS operating system. It is different from traditional databases as it is not designed for large data storage instead it performs acquisition of data generated from sensor nodes. It uses declarative query based approach for querying sensor networks. Queries in TinyDB are parsed at the base station and disseminated in a simple binary format into the sensor network, where they are instantiated and executed [38]. On each node, it maintains a catalog of meta-data and periodically copies it to the root of the network for use by the query optimizer. It uses materialization points to store streaming view of recent data on local (i.e., single) node. It performs lifetime estimation based on the available energy on the node.

**COUGAR** COUGAR[6] [21, 23, 71] is an in-network query processor for sensor networks. Like TinyDB, it is also different from traditional databases as it is not designed for large data storage instead it performs acquisition of data generated from sensor nodes. It uses declarative queries for tasking sensor networks. It proposed a query layer consisting of query proxy on each sensor node to enable declarative querying of sensor networks. Query proxy also performs the task of in-network processing. It uses sensor networks as a processing platform.

**Discussion** Both TinyDB and COUGAR present an appropriate solution for sensor networks data management. There is no solution in the world that can be ideal for all circumstances. Similarly, in Cellular DBMS we want to adapt the good concepts of in-network acquistional query processing. Furthermore, we want to apply other valuable concepts like tailor-made data management for sensor networks as proposed by Leich et al. [36] and approaches for robust data storage in wireless sensor networks as proposed by Siegmund et al. [54]. Aggregation is the most common and important operation in the sensor networks domain [26, 71]. We argue that column-oriented storage of Cellular DBMS can benefit in sensor networks as it makes aggregation efficient. Column-oriented storage greatly reduces I/O demand using compression techniques, which is important in sensor networks domain [25]. Holloway et al. in [25] showed that performance of column-oriented storage is higher when number of columns is less and data distribution is uniform. Both characteristics of data exist in sensor networks domain making column-oriented storage a good solution.

[5] ”TinyDB”, `http://telegraph.cs.berkeley.edu/tinydb/index.htm`

[6] “COUGAR”, `http://www.cs.cornell.edu/bigreddata/cougar/index.php`

## 3.4 Column-oriented DBMS

There exist many column-oriented DBMS in industry as shown in Table 1[7]. Only few we found important for further discussion based on similarity with Cellular DBMS.

| DBMS | Web Reference |
|---|---|
| **MonetDB** | http://monetdb.cwi.nl |
| **Vertica (Formerly: C-Store)** | http://www.vertica.com http://db.csail.mit.edu/projects/cstore/ |
| **Infobright (Formerly: Brighthouse)** | http://www.infobright.com |
| **HBase** | http://hadoop.apache.org/hbase/ |
| **Kdb+** | http://kx.com/Products/kdb+.php |
| **TokuDB for MySQL** | http://www.tokutek.com |
| **Calpont** | http://www.calpont.com |
| **The ParAccel Analytic Database** | http://www.paraccel.com |
| **EXASolution** | http://www.exasol.com |
| **Sybase IQ** | http://www.sybase.com/products/datawarehousing/sybaseiq/ |
| **LucidDB** | http://www.luciddb.org |

Table 1: Column-oriented DBMS.

**MonetDB** MonetDB[8] [10] is an open-source database system for high-performance applications (e.g., data mining, OLAP, etc.). It is a column-oriented database. MonetDB supports multiple data models simultaneously. MonetDB architecture is based on RISC-approach for database systems. MonetDB uses MonetDB Interpreter Language (MIL) to abstract internal implementation from higher-level models. To support extensibility, it supports MonetDB Extension Language (MEL), which can be used to extend the MonetDB functionality, e.g., datatypes, commands, etc. "MonetDB is designed as a main-memory system, and achieves high performance for problems of a limited size" [72].

**MonetDB/X100** Zukowski et al. in [73] presented X100. A new execution engine for the MonetDB system. X100 uses in-cache vectorized processing that improves execution speed of MonetDB and overcomes its main-memory limitation. It further introduced the ColumnBM storage layer to handle large disk-based datasets using techniques of ultra lightweight compression [75] and cooperative scans [74]. We found Cellular DBMS evolutionary column-oriented storage quite close to MonetDB/X100. Cellular

[7] List of column-oriented DBMS is not exhaustive.

[8] "MonetDB", `http://monetdb.cwi.nl/`

DBMS gets inspiration from MonetDB/X100 and intend to adapt and integrate the best of MonetDB/X100 concepts with its unique cellular architecture.

**C-Store** C-Store [1, 58] is an open-source read-optimized relational DBMS. It is a column-oriented DBMS. Its architecture is designed to reduce the number of disk accesses per query.

**Brighthouse** Brighthouse [55] is a column-oriented data warehouse with the concept of a meta-data layer called Knowledge Grid. Knowledge Grid is used as an alternative to classical indexes. In use of meta-data, Cellular DBMS evolutionary column-oriented storage finds some similarity with the concept of Brighthouse; however, they are different. Cellular DBMS allows the use of common indexes. Meta-data in Cellular DBMS is not used as an alternative to classical indexes. For database functionality, Brighthouse uses MySQL's pluggable storage engine platform[9], whereas Cellular DBMS can be developed using any customizable embedded database. Cellular DBMS also gets inspiration from Brighthouse and intend to adapt and integrate the best of Brighthouse concepts within its unique cellular architecture. An important feature of Brighthouse is the selection of different compression algorithms for different Data Packs, based on the data types and regularities automatically observed over data.

## 3.5 AOP for Autonomy

Use of AOP to implement autonomic behavior is not a new concept. Many researchers in past have used it successfully to develop autonomic systems. Greenwood et al. in [24] outlined the case of the use of dynamic AOP for autonomic systems. Truyen et al. in [62] demonstrated the applicability of AOP for implementing self-adaptive frameworks. Tesanovic et al. in [61] 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). In Cellular DBMS, we use an AOP based model to implement autonomic behavior at cell as well as at DBMS level.

[9] "MySQL 6.0 Reference Manual: Storage Engines", `http://dev.mysql.com/doc/refman/6.0/en/storage-engines.html`, Accessed: 13-07-2009

## 3.6 Biological Inspiration

To take inspiration from biological systems in computer science is not a new approach. There have been many attempts by many researchers to take benefits from the concepts in biological systems. An important step in this direction was taken by John von Neumann in his work on self-reproduction and cellular automata [68,69]. We found a major contribution from Gheorghe Păun and Cristian Calude in the area of membrane computing [11,15,46,47]. Kersten and Siebes proposed an organic database System in [66]. It is similar to our approach, but with many differences as we have already discussed in section 3.1. We want to use the best of it regarding how to take the inspiration from biological systems.

# 4 Cellular DBMS

A *Cellular DBMS* is composed of multiple atomic and autonomic customizable embedded database instances, called *Cells* [64, 65]. The motivation behind this approach is to ensure that a DBMS can be reduced to a fine-grained atomic unit (i.e., a cell) with predictable behavior and reduced complexity [70]. This approach enables us to assess the behavior of a complete DBMS by accumulating the behavior of all atomic cells.

## 4.1 DBMS Cell

A *Cell* is an atomic and autonomic instance of a customized *embedded database* [64,65]. Each cell is based on RISC-style architecture with simple and limited functionality. A cell can be customized based on different criteria, such as hardware, software, application scenario, nature of data, etc. Decisions about cell composition require a detailed analysis of all these criteria. Cellular DBMS architecture restricts cell functionality to a manageable complexity. It ensures that each cell is optimized for its task and is predictable for its performance. Each cell is customized to handle a single kind of data (i.e., data with unique characteristics, e.g., aggregated data; for details refer to section 6.1). If a cell supports handling multiple tables than the same kind of data should be stored in these tables. It ensures customization of each cell according to the kind of data. Multiple cells should be used to handle different kinds of data.

The most fine-grained variant of the cell can handle key/value pairs of data. Variants that are more complex can handle tables and maintains data dictionary, however, complex variants should be composed by using multiple fine-grained variants of cell. As mentioned above, the simplest cell handles a key/value pair and has definite (optimal) data-handling capacity, however, with the data growth, more cells could be induced into DBMS to extend its data-handling capacity. Virtually each cell uses *Binary Fission* [5] mechanism to grow. In binary fission, biological cell grows to twice of its starting size and then splits-up 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.

Deployment of cells depends on many criteria, such as the kind of data, the distribution of computing resources[10], as well as the hardware of computing resources, etc. For example, in the simplest sensor network scenario, a single cell can be deployed on an individual node. However, for more complex scenario, multiple cells can be deployed on a single node or can be distributed over multiple nodes in a network.

## 4.2 Types of DBMS Cells

Cellular DBMS defines many different types of cells. Each type differs from the other based on its composition and characteristics. These types enhance the diversity of Cellular DBMS for many data management scenarios. Currently, implementation of Cellular DBMS is a work in progress. More cell types are expected to appear in the future. In this report, we explain the types that we have defined based on our existing architecture and implementation.

**Composite Cells** A cell can be composed of multiple similar or dissimilar cells related to each other as shown in Figure 2. Such composition of cells is termed as *Composite Cell*. Each composite cell should have limited (optimal) data-handling capacity to ensure it has manageable complexity and predictable performance. With the data growth, more composite cells could be induced into the DBMS to extend its data management capacity. Each composite cell maintains a meta-data of cell composition. 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 type, e.g., one column cell contains in-memory data management functionality whereas another column cell can also store persistent data. It can also be used to handle large amount of data that simple cells cannot handle. From software engineering perspective, when using multiple cells, composite cell avoid code replication and allow us to reuse the program code between different cells on a single computing resource.

[10] From computing resource, we mean any device with processing capability. It may range from small-embedded device to high-end enterprise server machines.

**High-level Composite Cells** 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 2. The reason for such architecture is to provide hierarchical data management functionalities to manage complexity. According to Cellular DBMS architecture, data-handling capacity of a cell is optimally limited for highly predictable performance and reduced complexity. Cellular DBMS uses high-level composite cell for handling large amount of data. In high-level composite cell, cell composition becomes deeper with the increase in the size of data. Each high-level composite cell maintains a meta-data cell (i.e., stores meta-data) that helps in fast retrieval and updation of records. To retrieve data from high-level composite cell, only meta-data cells are used to trace the data cell. Once the targeted data cell is traced, only this cell or its related cells are used for data management.

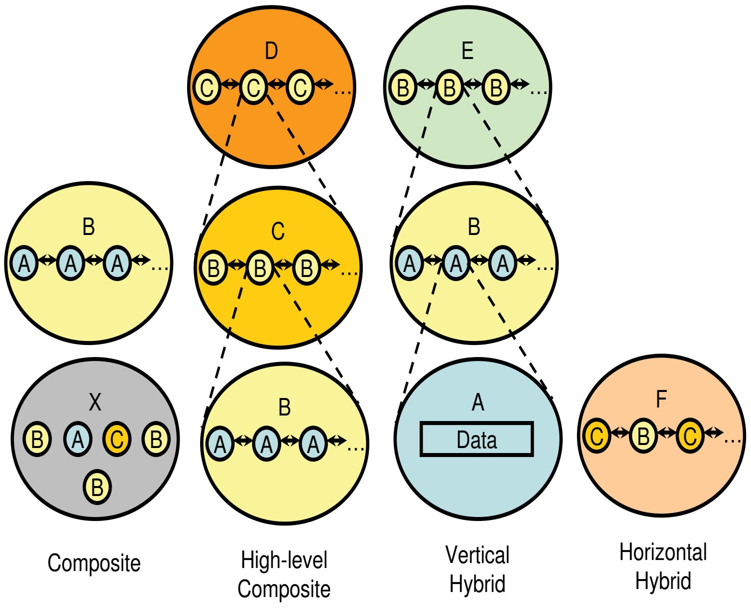

Figure 2: Different types of cells.

**Hybrid Cell** For diversified data management, Cellular DBMS introduce the concept of *Hybrid Cell*. We could have horizontal as well as vertical hybrid cells as shown in Figure 2. 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 as 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 report; however, implementation of hybrid cells is future work.

**Evolving Cell** Evolution in Cellular DBMS means run-time transformation of cells. We term a cell that supports evolution as an *"Evolving Cell"*. 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 3. 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.

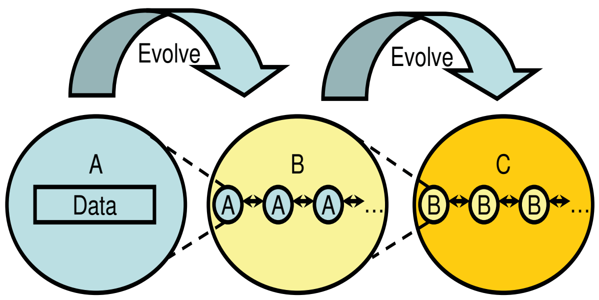

Figure 3: Evolving cell.

## 4.3 Clean API and Interaction

From software engineering aspect, providing a consistent API for simple as well as composite cells is an important design criterion, which is required for communication between cells. We argue that, two communicating cells should not care about the concrete type of one-another. On the other hand, simple cells provide limited data management functionality and should exhibit a simple API that reflects the limited functionality, which is in contrast to a consistent API and has to be considered when generating cells.

For solution, we use two different mechanisms. First, we allow only interface extensions, but not modifications of an interface [65]. For example, a DBMS feature might add a method to the interface of the DBMS, but is not allowed to modify the signature of an existing method. This ensures upward compatibility, i.e., we can use cells with a more complex API when cells with a simple API are expected.

The second approach is to generate wrappers for simple cells when complex cells are expected [65]. For example, if a method for creating an index is expected from an in-memory cell without index support, an empty wrapper method can be generated to provide this method. Wrappers are used to achieve only downward compatibility and wrappers that are more complex might be required. Furthermore, it has to be analyzed for which scenarios it is not possible to generate such wrappers.

**Distributed Cells** In Cellular DBMS, cells are not confined to a single computing resource. Cells can be distributed across network, or more ambitiously speaking across internet. Important distribution criteria could be size and locality of data. For example, in a complex distributed sensor network scenario, cells are deployed on multiple nodes and collaborate for data management. On each node of such a distributed scenario, a single cell might be used or a composite cell might provide complex data management. Distributed cells interact with each other through API calls over the network. For distributed deployment, we envision a Cellular DBMS using a global data dictionary and statistics as well as distributed monitoring functionality to implement distributed autonomy. However, it has to be further analyzed how distributed deployment of interacting cells can be achieved in Cellular DBMS.

## 4.4 Resource Balancing

In distributed environment, we envisioned the possibilities of resource balancing using distributed cells. We have listed down our vision, but complete implementation is future work.

**Cell Mobility** In Cellular DBMS, we propose the concept of cell mobility. Cell mobility means the capability of Cellular DBMS to move a cell from one processing environment to another. Mobility of cells could be across processes on single system or across systems connected via network. The motivation behind mobility is to achieve load balancing and to use resources efficiently. Cell mobility can be used in many different ways. For example, one scenario in distributed network of interconnected embedded devices is that the embedded device on which a cell is deployed, is heavily loaded with processing. We envision moving that loaded cell to another relatively idle device. If all devices are over-consumed, then a new device can be brought into the network and then cells can be moved to that new device for load balancing.

**Virtual Resources** Embedded systems are different from high-end systems by means of resources. In embedded system, we normally have resource constraints on a single device, but in the network of interacting embedded systems there are many resources that are available across network and are idle. We envision in Cellular DBMS to virtually-combine these scattered resources as Virtual Resource, i.e., it gives a virtual view of scattered small resource across embedded devices as one single large resource. For example, on three embedded devices we have 10 kB, 6 kB, and 13 kB of free memory. Now if we have to store data that is 18 kB large, none of these devices has enough capacities on its own. In this case, Cellular DBMS approach is capable of storing data distributed across devices using cells and transparently provides a view of a single large resource capable of accommodating 18 kB of data to an application. This concept also gives us a clue that how Cellular DBMS can use cells for fragmenting data on multiple embedded devices, sensor nodes, or high-end enterprise servers.

## 4.5 Cell Classification

Based on our current architecture and implementation, we can also classify cells in two types based on the data they store, i.e., data cell and meta-data cell. Data cell manages data. Meta-data cell is also a data cell, but it stores meta-data.

## 4.6 Design Principles for Autonomy in Cellular DBMS

Autonomy of each cell is an important design principle for Cellular DBMS. Cellular DBMS envision the development of complete autonomous DBMS by accumulating autonomic behavior of all participating cells. For autonomy, the most fundamental functionalities are Monitoring, Diagnostics, and Tuning [37, 13]. According to the proposed architecture, monitoring, diagnostic, and tuning components should also be customizable according to the cell functionalities to ensure reduced monitoring overhead. We present an AOP based model for autonomy at cell-level. We argue based on provided related work in section 3.5 that AOP join-point model can be used to implement efficient monitoring functionality for data management.

According to Cellular DBMS architecture, each cell contains an optional lightweight monitoring functionality. The purpose of monitoring functionality is to monitor the cell for specific parameters. These parameters are defined as a policy for DBMS cell goals. Each cell should be able to adapt to changes based on events identified by the monitoring component. 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 to DBMS changes and fixing of DBMS problems according to defined DBMS policy. A symbolic monitoring functionality distribution is depicted in Figure 4. For diagnostics, we use the state of the cell, and results of data management operations to identify the definite tuning points. For tuning we use the evolution and evolving cell approach presented in section 4.2.

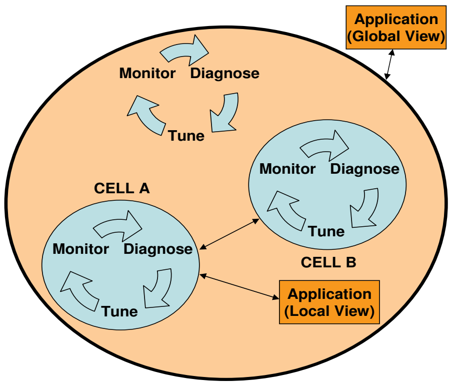

Figure 4: Monitoring functionality distribution.

According to the model, tracing is an important functionality during monitoring. By tracing, we mean collection of cell state information that is needed to diagnose and tune the individual cell as well as complete Cellular DBMS. For each join-point, before advice should be used for tracing whereas after advice should be used to diagnose the abnormality. If any abnormality is detected during diagnostics, tuning should be executed to counter the abnormality.

To explain the concepts 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

| 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 2: Average execution time for stress test in millisecond for different Cellular DBMS cells.

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 3. Both cells store definite/limited 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 2[11] and Figure 5. 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 execution 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 soon 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

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

approach, we ensures that complexity of data management implementation should only be increased as the amount of data is increased.

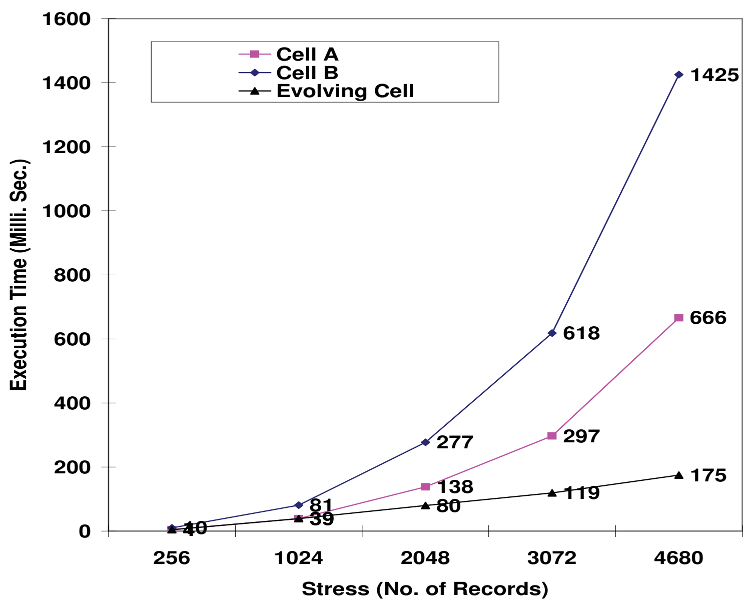

Figure 5: Average execution time graph for stress test in millisecond for different Cellular DBMS cells.

## 4.7 Cellular DBMS Storage Model

Customization capability of Cellular DBMS gives us provision to use any of the available storage models. Cellular DBMS does not restrict the usage of any specific storage model; however, we recommend one in this section based on the Cellular DBMS goals. Cellular DBMS stores data using Decomposed Storage Model (DSM) [9] also know as Column-oriented Storage (COS) [58]. Based on the discussion in the related concepts section, we found COS most appropriate for implementing atomic and autonomous cells. Use of COS enables simple cell design and gives more control over data. We envision achieving all benefits from COS as discussed in related concepts section. In Cellular DBMS, each column is a separate cell. A column data can be stored using a simple as well as composite cell. COS in Cellular DBMS is shown in Figure 6.

Cellular DBMS usage of COS is different from its traditional usage. Cellular DBMS combines the concept of evolution, high-level composite, and meta-data cells with COS to enhance the storage model to overcome the deficiencies that we identified in the related concept and related work sections. We term this enhanced model as “Evolutionary

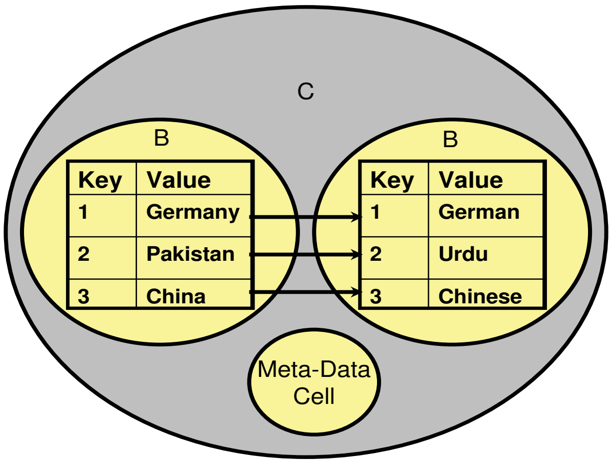

Figure 6: Column-oriented storage using composite cell.

Column-Oriented Storage".

### 4.7.1 Evolutionary Column-Oriented Storage

*Evolutionary Column-Oriented Storage (ECOS)* is an extension of column-oriented storage with the concepts of customization, autonomy, evolution, high-level composition, and meta-data. We outline the principles that govern the ECOS as follows:

- ECOS stores data using DSM.
- Each column is customized based on the kind of data.
- Each column evolves from simple cell to high-level composite cell with the data growth.
- Each high-level composite cell for column maintains the meta-data about cell composition.
- Each high-level composite cell for column maintains the user defined real-time aggregation for its stored data.

Using ECOS, Cellular DBMS manages data management complexity and resource consumption based on the size and kind of data. Using meta-data approach, it ensures fast data management operations. Use of high-level composite cell approach enables highly predictable behavior of each cell by managing complexity. As shown in Figure 7, in ECOS each column is customized based on kind of data it handles. As we have already discussed, in Cellular DBMS, each column is handled by a separate cell and each cell data storage capability is limited to optimal limit with manageable complexity. In the sample ECOS implementation shown in Figure 7, among all three columns, column 2 is managed by the simplest cell. The cell type for column is defined by the end-user. However, if the selected type is found to be insufficient for handling the data storage, cell evolve to a higher level in form of high-level composite cell.

For large-scale data storage, ECOS uses separate persistent storage for each cell. This mechanism in conjunction with meta-data reduces the I/O demand for ECOS. For example, to fetch a particular record, Cellular DBMS searches for relevant cell using the meta-data information. As it finds the relevant cell, it gets data from it. As call to get data is received by cell, it searches for data in internal in-memory cells (similar to pages). If data is not found in the already loaded in-memory cells only than I/O is performed and relevant page with data is loaded into main-memory as in-memory cell.

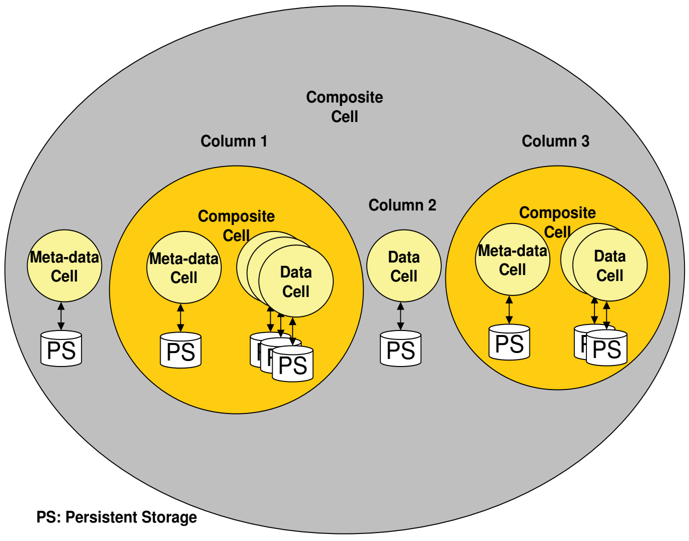

Figure 7: Evolutionary column-oriented storage.

Meta-data cells play an important role in reducing the search space for data retrieval and updates. We envision meta-data to store real-time data aggregation as well. It is on user discretion to define the data aggregates. We have implemented basic ECOS in Cellular DBMS, however, complete implementation and its performance comparison with other approaches is a work in progress.

### 4.7.2 Compression

Compression is an important technique for optimizing column-oriented storage for space, here we describe the few compression techniques that we intend to use in Cellular DBMS; however, it is currently part of future work.

**Delta Encoding** In our current implementation, each cell in Cellular DBMS stores data in sorted order. If data is in sequential order, delta encoding [39] can be used to compress data at a cell level. Using delta encoding we store the differences of the two values in sequence. The decision of either to use delta encoding or not depends on the average size difference of encoded and real value. It could only be used when the average size difference of encoded values is small.

**Run-Length Encoding (RLE)** For data sequence with high repetition, Run-Length Encoding (RLE) [63] can be used to transform a sequence into vector form, i.e., key/-value pairs. Key/value pair is the basic storage mechanism for data in currently implemented cells. This approach promises high compression for sequences with high repetition.

# 5 Cellular DBMS Implementation

Existing Cellular DBMS implementation is an extension of FAME-DBMS, a highly customizable embedded database management software product line developed for deeply embedded systems [50]. We used FAME-DBMS to generate the cells, however, any customizable embedded database can be used to generate cells. FAME-DBMS is implemented using feature-oriented programming. It untangles and modularizes 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 variants based on the selection of required features [36]. These different variants are built from the same code base as depicted in Figure 8 [65]. Based on such an SPL, multiple heterogeneous DBMS cells can be generated [50].

The *feature model* of Cellular-DBMS, as shown in Figure 9 [12], is based on the FAME-DBMS feature model. Feature model describes the features of an SPL and their relationships [27]. As depicted, the implementation of Cellular DBMS consists of five main features, i.e., In-Memory Data Management, Buffer Manager, Access Path, Autonomy, and OS Abstraction. Each functionality can be implemented differently to achieve benefits, e.g., better performance, and can be described as alternative features. For example, feature Index provides two variants for effective data access, i.e., B+Tree and Hash. It enables us to generate specialized cells by selecting one feature or the other.

Functionality for storing data is provided by feature In-Memory Data Management.

[12]Shown is an excerpt of the feature model with only modified features/concepts required for discussion.

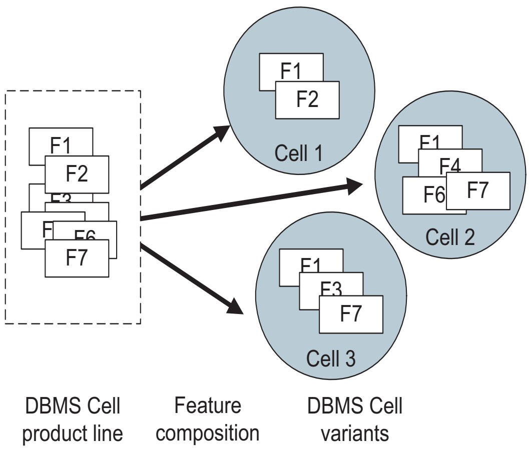

Figure 8: Generating different DBMS cells by composing features (F1–F7) of a DBMS cell product line.

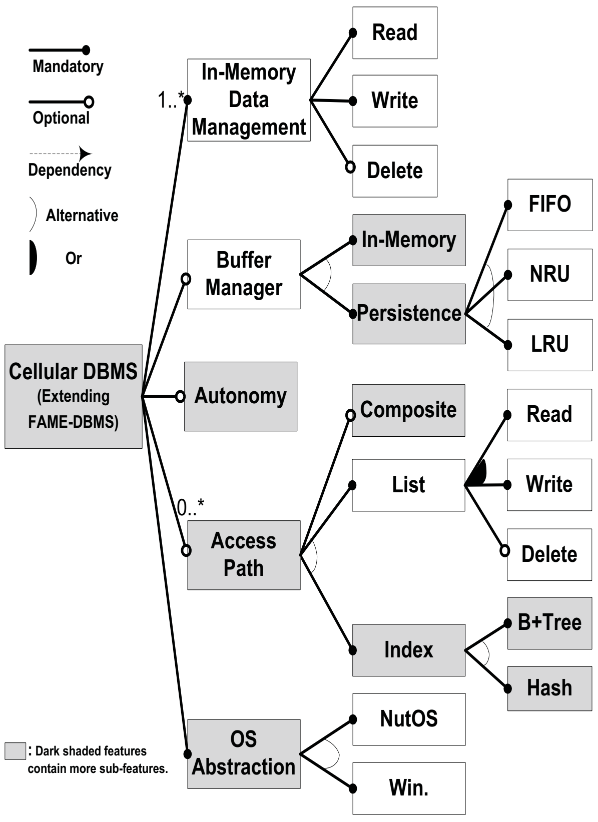

Figure 9: Cellular DBMS feature model.

This feature contains the functionality of an in-memory embedded database. It can alone be used to construct a simple DBMS cell. It performs data management operations in an in-memory environment and does not have any unneeded persistence functionality, resulting in good performance in terms of fast operations [22]. Most sensor nodes are equipped with storage memory that can be used to store data persistently. To scale a Cellular DBMS cell for such nodes feature Persistence can be used.

The simplest cell, consisting only of feature In-Memory Data Management, supports exactly one column or a fragment of column (if used as a part of high-level composite cell). For multiple columns, we can clone the In-Memory Data Management feature. Cloning a feature means to create multiple instances of it [18]. For example, to support two columns we have to create two instances of the In-Memory Data Management feature and each instance handles one of the columns. Whether a feature can be cloned is depicted with cardinalities in Figure 9. For example, there has to be at least one instance of the In-Memory Data Management feature, but an arbitrary number of instances are allowed.

We bring autonomy to each cell by using AOP based model. We utilized AspectC++[13] [56] for using AOP constructs. FeatureC++ also supports AOP extensions as discussed in [6, 7], 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 10.

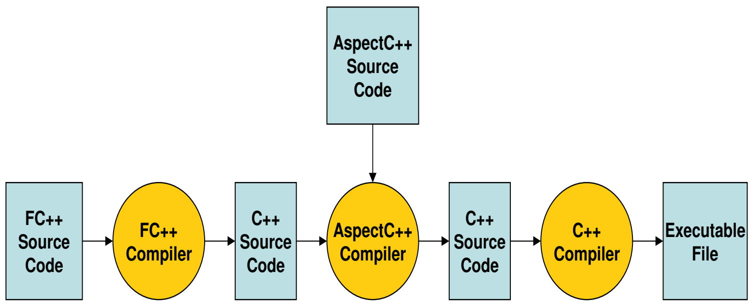

Figure 10: Source code transformation.

## 6 Discussion

We envision that Cellular DBMS architecture is scalable for use in embedded systems to enterprise systems. For explanation, we discuss two assumed sample scenarios for

[13] “AspectC++”, `http://www.aspectc.org/`

Cellular DBMS use in sensor networks and enterprise data management.

## 6.1 Sensor Networks

Sensor networks are important data-centric systems with hardware and software heterogeneity as depicted in Figure 11. Hardware in sensor networks may vary from 8 bit motes to 32 bit microservers with the program memory that can vary from 48 kB to 512 kB, whereas the data memory may vary from 4 kB to 64 kB [20]. Each node varies in terms of the processing power and the memory configuration. Considering extreme resource scarcity and high hardware and software heterogeneity as discussed above, one of the requirements of sensor networks is to make the best use of available resources and exploit the hardware heterogeneity for efficient data management.

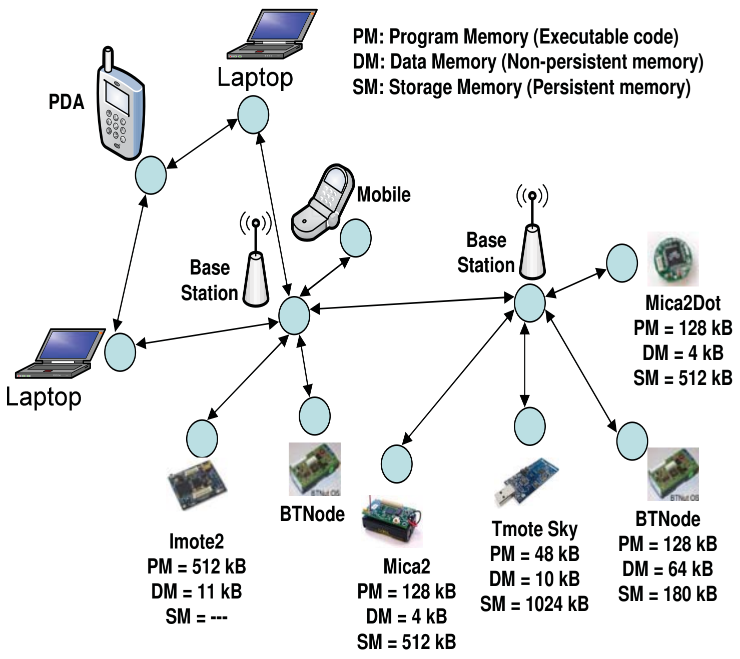

Figure 11: Sensor network scenario.

For deployment on sensor networks, each cell should be customized based on the resources and kind of data that cell handles on the deployment node. Our data classification consists of four kinds: *standing data*, *setup data*, *transactional data*, and *aggregated data*. *Standing data* is generated during the deployment and is never changed during its lifetime. It is read quite often, e.g., fixed time intervals for sensing the environment. *Setup data* is also initialized during the deployment, but may be subject to

| **Kind of Data** | **Standing** | **Setup** | **Transactional** | **Aggregated** |
|---|---|---|---|---|
| **Data Size** | Small | Small to Medium | Medium to Large | Medium to Large |
| **Read Frequency** | High | High | Medium to High | Low to High |
| **Write Frequency** | No Write | Low | Medium to High | Low to Medium |

Table 3: Data categorization

| **Features** | **In-Memory Data Management** | | | **Persistence** | **List** | | | **Index** | **Binary Size (.ELF) Compiler: avr-g++** |
|---|---|---|---|---|---|---|---|---|---|
| | Read | Write | Delete | LRU | Read | Write | Delete | B+Tree | |
| **Cell A** | X | X | X | | | | | | 32 KB |
| **Cell B** | X | X | X | X | X | | | | 42 KB |
| **Cell C** | X | X | X | X | X | X | X | | 45 KB |
| **Cell D** | X | X | X | X | | | | X | 48 KB |

Table 4: Binary size for different Cellular DBMS cells.

change during its lifetime. For example, it is needed for routing information in a wireless sensor network. If nodes fail due to limited power, new neighbor nodes have to be registered for communication. *Transactional data* is generated during data operations (e.g., add, update, remove) and is often changed during its lifetime, e.g., sensed data in a sensor network. *Aggregated data* is the result of some aggregation operations. This kind of data can be found on aggregation nodes in a sensor network. Each kind of data has its own characteristics in terms of usage frequency and size, as shown in Table 3. Since the nature of data may vary for different sensors on single node as well as across different nodes. We argue that customized data management is needed to manage each kind of data, i.e., using different types of DBMS cells. In described scenario, we propose to build data management for nodes using a Cellular DBMS that consist of multiple individual cells, each customized for optimal data management based on available resources and kind of data. Since data could be distributed over multiple nodes in a sensor network. DBMS cells should also be distributed over nodes and should collaborate for complete sensor network data management.

For discussion of our proposed architecture, we consider storage memory, and program memory as parameters of interest. To explain the idea, how specialized DBMS cells can be beneficial for data-centric embedded systems, four types of DBMS cells are generated based on different feature selections using FAME-DBMS prototype as shown in Table 4 [14]. For each cell, the binary size is different and depends on the selected features of FAME-DBMS prototype. Each cell is a candidate for a different type of node based on the available program and storage memory as well as type of data it handles. Cell A is suitable for nodes without any storage memory, e.g., Imote node. Cell D is suitable for nodes with relatively large data and storage memory, e.g., BTnode rev3. A sample deployment of these customized cells based on a node's resources is shown in Figure 12. In the sample deployment, Tmote sky contains the smallest pro-

[14] Binary size contains additional overhead of dependencies and may vary in future work.

gram memory and can only handle small cells like Cell A. However, it also contains largest storage memory making Cell B, C, and D a good candidate for storing relatively large data on storage memory. In contrast, Imote contains the largest program memory, but lacks storage memory, again making Cell A best candidate for deployment. BTnode rev3, Mica2, and Mica2Dot all contain moderate program and storage memory. We argue that, in the demonstrated sample deployment, the Cellular DBMS is a promising solution. The Cell implementation is lightweight and allows for deployment of multiple heterogeneous cells on a single node, enabling specialized handling of data based on available resources and the nature of data.

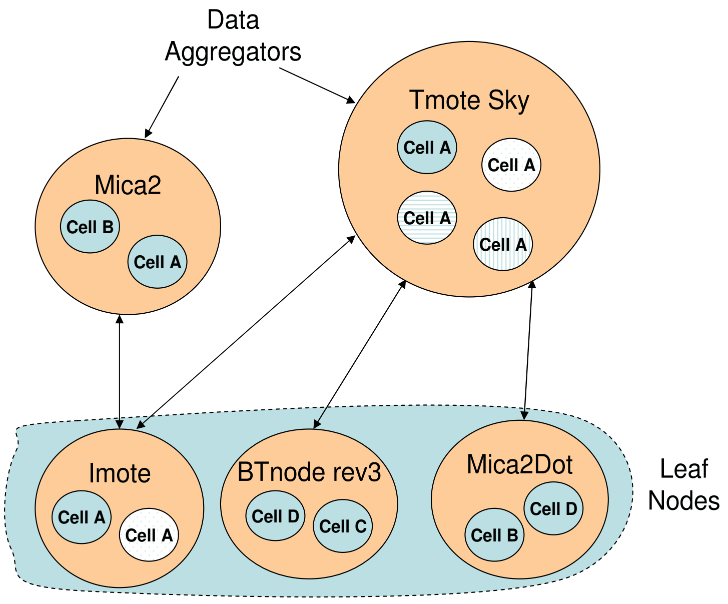

Figure 12: Sample deployment of different Cellular DBMS cells.

## 6.2 Enterprise Data Management

Industries with enterprise data management needs are quite satisfied with existing DBMSs in term of their performance. They have many solutions to choose from based on different criteria, e.g., cost, performance, etc. However, maintenance cost of most of the existing DBMSs is high. We argue that Cellular DBMS architecture with its goals to achieve highly predictable, customizable, autonomous DBMS will be able to reduce the maintenance cost.

The data classification we provided in Table 3 is also applicable for enterprise data management. Cellular DBMS gives an end-user the provision for data management customization at many different levels. An end-user can specify, what functionalities DBMS should have, how these functionalities should be used, and how DBMS should be tailored based on application and data [36, 49, 50, 60]. For example, consider the case for setup data. The frequency of update in setup data is low whereas frequency of retrieval is high. Furthermore, setup data (except some exceptions) is not too large. In an enterprise application, we normally have many setup tables. Existing DBMS handles all tables similarly. Either we have large data or not, we cannot customize the internal implementation to optimize it for handling small data. The same column implementation is used for handling a column with only five values as well as for a column with many MB's of data. Cellular DBMS approach is different. It provides the provision for customization at the fine-grained level of cell. It is possible to compose different cells based on the kind of data, i.e., we can compose four types of cells for all four kinds of data we presented in Table 3. This approach is similar to using four small databases customized for their task instead of using a single large DBMS customized for nothing. Another important aspect is that existing DBMS uses complex internal structures irrespective of existing data size. In contrast, in Cellular DBMS data management complexity only increases with the data growth, i.e., utilizing the resources only when they are needed.

## 7 Conclusion and Future Work

We proposed a novel DBMS architecture based on composition of multiple cells that are atomic and autonomic customized embedded databases. As explained, these cells provide restricted data management functionality and collaborate to constitute one large *Cellular DBMS*. This cell based approach ensures predictable behavior and efficient utilization of resources by keeping the cells simple.

We argue that Cellular DBMS architecture reduces DBMS complexity and when blended with autonomy, it can be used to develop highly predictable autonomous DBMS. In this work, we also presented an AOP based model for implementing autonomy at cell level in Cellular DBMS. We also explained 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. Furthermore, we also introduced an extension of column-oriented storage with the concepts of customization, autonomy, evolution, high-level composition, and meta-data to overcome the deficiencies of classical column-oriented storage. In our proposed architecture, we argue that we can develop highly customizable autonomous DBMS that can scale from requirement of small embedded systems to large-scale enterprise systems.

As a future work in Cellular DBMS, we found many opportunities that are listed below:

- Many concepts, such as hybrid cell, cell mobility, resource balancing, self-* [57] (e.g., self-tuning, self-managing, self-adaptation, etc.) capabilities, etc., that we presented in this report need implementation and performance comparison with existing approaches.
- In the era of multi-core processors, we want to enable Cellular DBMS to exploit parallelism.
- Using differently composed cells simultaneously while minimizing code replication is an important open issue. A software engineering based solution is needed to solve this issue.
- Monitoring is an overhead for high-end embedded system. For implementation of Cellular DBMS in such systems, we want to investigate mechanisms to reduce this overhead.
- Current implementation of cell evolution is explicitly programmed. An important future direction is to enable implicit learning in Cellular DBMS for self-* capabilities.
- Query processing is a mandatory feature for all existing DBMS. For Cellular DBMS architecture, we need specialized mechanism for efficient query processing.

## Acknowledgment

We thank Marko Rosenmüller, Sven Apel, Norbert Siegmund, Christian Kästner, Azeem Lodhi, Qaisar Hayat, Ateeq Lodhi, and Sagar Sunkle for their feedback and contribution. Syed Saif ur Rahman is funded by Higher Education Commission of Pakistan and NESCOM, Pakistan.

## References

[1] D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980, New York, NY, USA, 2008. ACM.

[2] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB '07: Proceedings of the*

*33rd international conference on Very large data bases*, pages 411–422. VLDB Endowment, 2007.

[3] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, and G. Weikum. The Claremont report on database research. *Commun. ACM*, 52(6):56–65, 2009.

[4] A. Ailamaki, D. J. DeWitt, and M. D. Hill. Data page layouts for relational databases on deep memory hierarchies. *The VLDB Journal*, 11(3):198–215, 2002.

[5] E. R. Angert. Alternatives to binary fission in bacteria. *Nature Reviews Microbiology*, 3(3):214–224, 2005.

[6] S. Apel, T. Leich, M. Rosenmüller, and G. Saake. FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming. In *GPCE '05: Proceedings of the International Conference on Generative Programming and Component Engineering*, pages 125–140. Springer, 2005.

[7] S. Apel, T. Leich, and G. Saake. Aspectual Feature Modules. *IEEE Transactions on Software Engineering*, 34(2):162–180, 2008.

[8] 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.

[9] D. S. Batory. On searching transposed files. *ACM Trans. Database Syst.*, 4(4):531–544, 1979.

[10] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in MonetDB. *Commun. ACM*, 51(12):77–85, 2008.

[11] C. S. Calude and G. Păun. Computing with cells and atoms in a nutshells. *Complex.*, 6(1):38–48, 2000.

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[13] S. Chaudhuri and V. Narasayya. Self-tuning database systems: a decade of progress. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 3–14. VLDB Endowment, 2007.

[14] 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.

[15] G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.

[16] P. Clements, L. Northrop, and L. M. Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional, August 2001.

[17] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. In *SIGMOD '85: Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, pages 268–279, New York, NY, USA, 1985. ACM.

[18] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration using feature models. In *SPLC '04: Proceedings of the 3rd software product line conference*, volume 3154 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2004.

[19] A. P. de Vries, N. Mamoulis, N. Nes, and M. L. Kersten. Efficient image retrieval by exploiting vertical fragmentation. Technical Report INS-R0109, CWI, Amsterdam, The Netherlands, November 2001.

[20] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. EmStar: An Environment for Developing Wireless Embedded Systems Software. Technical report, Center of Embedded Networked Systems (CENS), University of California, March 25 2003.

[21] W. F. Fung, D. Sun, and J. Gehrke. Cougar: the network is the database. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 621–621, New York, NY, USA, 2002. ACM.

[22] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Trans. on Knowl. and Data Eng.*, 4(6):509–516, 1992.

[23] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.

[24] P. Greenwood and L. Blair. Using Dynamic Aspect-Oriented Programming to Implement an Autonomic System. In *DAW '04: Proceedings of the 2004 Dynamic Aspects Workshop*, pages 76–88, 2004.

[25] A. L. Holloway and D. J. DeWitt. Read-optimized databases, in depth. *Proc. VLDB Endow.*, 1(1):502–513, 2008.

[26] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM.

[27] 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.

[28] I. Karasalo and P. Svensson. An overview of cantor: a new system for data analysis. In *SSDBM'83: Proceedings of the 2nd international workshop on Proceedings of the Second International Workshop on Statistical Database Management*, pages 315–324, Berkeley, CA, US, 1983. Lawrence Berkeley Laboratory.

[29] I. Karasalo and P. Svensson. The design of cantor: a new system for data analysis. In *SSDBM'86: Proceedings of the 3rd international workshop on Statistical and scientific database management*, pages 224–244, Berkeley, CA, US, 1986. Lawrence Berkeley Laboratory.

[30] M. Kersten, G. Weikum, M. Franklin, D. Keim, A. Buchmann, and S. Chaudhuri. A database striptease or how to manage your personal databases. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 1043–1044. VLDB Endowment, 2003.

[31] 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.

[32] 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.

[33] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *ECOOP '97: 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.

[34] 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.

[35] 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.

[36] T. Leich, S. Apel, and G. Saake. Using Step-Wise Refinement to Build a Flexible Lightweight Storage Manager. In *ADBIS '05: Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems*, volume 3631 of *Lecture Notes in Computer Science*, pages 324–337. Springer Verlag, 2005.

[37] S. S. Lightstone, G. Lohman, and D. Zilio. Toward autonomic computing with DB2 universal database. *SIGMOD Rec.*, 31(3):55–61, 2002.

[38] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[39] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. *SIGCOMM Comput. Commun. Rev.*, 27(4):181–194, 1997.

[40] A. Nori. Mobile and embedded databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1175–1177, New York, NY, USA, 2007. ACM.

[41] M. A. Olson. Selecting and implementing an embedded database system. *Computer*, 33(9):27–34, 2000.

[42] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *ATEC '99: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 43–43, Berkeley, CA, USA, 1999. USENIX Association.

[43] M. T. Özsu and B. Yao. *Building component database systems using CORBA*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[44] D. A. Patterson and D. R. Ditzel. The case for the reduced instruction set computer. *SIGARCH Comput. Archit. News*, 8(6):25–33, 1980.

[45] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[46] G. Păun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[47] G. Păun, G. Rozenberg, and A. Salomaa. Membrane computing with external output. *Fundam. Inf.*, 41(3):313–340, 2000.

[48] 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)*, Oct. 2009. accepted for publication.

[49] M. Rosenmüller, C. Kästner, N. Siegmund, S. Sunkle, S. Apel, T. Leich, and G. Saake. Sql à la carte – toward tailor-made data management. In *13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, Mar. 2009. to appear.

[50] 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.

[51] G. Saake, M. Rosenmüller, N. Siegmund, C. Kästner, and T. Leich. Downsizing Data Management for Embedded Systems. *Egyptian Computer Science Journal (ECS)*, 31(1):1–13, Jan. 2009.

[52] J. H. Sabry, C. L. Adams, E. A. Vaisberg, and A. Crompton. Database system for predictive cellular bioinformatics, United States Patent 6631331, October 2003.

[53] 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.

[54] N. Siegmund, M. Rosenmüller, G. Moritz, G. Saake, and D. Timmermann. Towards Robust Data Storage in Wireless Sensor Networks. In *DAIT '09: Proceedings of Workshop on Database Architectures for the Internet of Things*, volume 5588 of *Lecture Notes in Computer Science*. Springer, July 2009.

[55] D. Ślęzak, J. Wróblewski, V. Eastwood, and P. Synak. Brighthouse: an analytic data warehouse for ad-hoc queries. *Proc. VLDB Endow.*, 1(2):1337–1345, 2008.

[56] O. Spinczyk, D. Lohmann, and M. Urban. AspectC++: an AOP Extension for C++. *Software Developers Journal*, pages 68–74, 2005.

[57] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A concise introduction to autonomic computing. *Advanced Engineering Informatics*, 19(3):181–187, July 2005.

[58] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented dbms. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[59] P. Svensson. The Evolution of Vertical Database Architectures — A Historical Review (Keynote Talk). In *SSDBM '08: Proceedings of the 20th international conference on Scientific and Statistical Database Management*, pages 3–5, Berlin, Heidelberg, 2008. Springer-Verlag.

[60] A. Tesanovic, K. Sheng, and J. Hansson. Application-Tailored Database Systems: A Case of Aspects in an Embedded Database. In *IDEAS '04: Proceedings of the International Database Engineering and Applications Symposium*, pages 291–301, Washington, DC, USA, 2004. IEEE Computer Society.

[61] 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.

[62] 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.

[63] T. Tsukiyama, Y. Kondo, K. Kakuse, S. Saba, S. Ozaki, and K. Itoh. Method and system for data compression and restoration, April 1986.

[64] S. S. ur Rahman, A. Lodhi, and G. Saake. Cellular DBMS - Architecture for Biologically-Inspired Customizable Autonomous DBMS. In *NDT '09: Proceedings of the First International Conference on the Networked Digital Technologies*, pages 310–315, Washington, DC, USA, July 2009. IEEE Computer Society.

[65] 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 *EDACS*

*'09: 20th International Workshop on Database and Expert Systems Application (DEXA 2009)*. IEEE Computer Society, 2009. To appear.

[66] W. Verhaegh, E. Aarts, and J. Korst. *Intelligent Algorithms in Ambient and Biomedical Computing (Philips Research Book Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[67] 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.

[68] J. von Neumann. The computer and the brain. *New Haven*, 1958.

[69] J. von Neumann. The General and Logical Theory of Automata. John von Neumann–Collected Works, 5. AH Taub, 1963.

[70] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 20–31. VLDB Endowment, 2002.

[71] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[72] M. Zukowski. Hardware-Conscious DBMS Architecture for Data-Intensive Applications. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Trondheim, Norway, August 2005. PhD Workshop.

[73] M. Zukowski, P. A. Boncz, N. Nes, and S. Heman. MonetDB/X100 - A DBMS In The CPU Cache. *IEEE Data Engineering Bulletin*, 28(2):17–22, June 2005.

[74] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative scans: dynamic bandwidth sharing in a DBMS. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 723–734. VLDB Endowment, 2007.

[75] M. Zukowski, S. Heman, N. Nes, and P. A. Boncz. Super-Scalar RAM-CPU Cache Compression. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, Atlanta, GA, USA, April 2006.

[76] M. Zukowski, N. Nes, and P. Boncz. DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing. In *DaMoN '08: Proceedings of the 4th international workshop on Data management on new hardware*, pages 47–54, New York, NY, USA, 2008. ACM.