A New Approach to Modular Database Systems

Florian Irmert
Friedrich-Alexander University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
florian.irmert@cs.fau.de

Michael Daum
Friedrich-Alexander University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
michael.daum@cs.fau.de

Klaus Meyer-Wegener
Friedrich-Alexander University of Erlangen-Nuremberg
Department of Computer Science
Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
kmw@cs.fau.de

ABSTRACT

In this paper we present our approach towards a modularized database management system (DBMS) whose components can be adapted at runtime and show the modularization of a DBMS beneath the record-oriented interface as a first step. Cross-cutting concerns like transactions pose thereby a challenge that we answer with aspect-oriented programming (AOP). Finally we show the implementation techniques that enable the exchange of database modules dynamically. Particularly with regard to stateful components we define a service adaptation process that preserves and transmits the component's state.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.11 [**Software Engineering**]: Software Architectures; D.2.13 [**Software Engineering**]: Reusable Software; H.2.4 [**Database Management**]: Systems

General Terms

Design, Management

Keywords

Adaptation, availability, service-oriented architecture, component replacement, modularity, migration

1. INTRODUCTION

For about 30 years no major company is able to manage the large volume of information without a database management system (DBMS). Well known "generic" database systems like Oracle or DB2 extend their functionality with every release. Such commercial database systems are often developed over many years, and to stand out on the highly competitive market, a lot of developers are needed to manage the extensive development. Although the "componentization" [18] of database management systems (DBMS) is a topic of research since the 90s, today's DBMSs still consist of a monolithic database kernel. Such a monolithic architecture increases maintenance and development costs additionally. In this paper we present our ideas towards a modular generic database. We describe the necessary properties and present work in progress. Current software engineering practices like service orientation and loose coupling are used to structure the design of the DBMS and therewith simplify the development and maintenance. As a result DBMSs for specific purpose can be developed much faster and cheaper compared to a traditional development process.

We aim to design a modular DBMS that can be adapted to different environments by assembling prefabricated modules. Therefore we need a kind of DBMS "construction kid". For each "building block" dependencies to other blocks have to be defined, and basic modules have to be identified that are necessary in every DBMS. Then a DBMS can be assembled by choosing the right modules for the specific task of the system.

To provide high availability, another challenge is the modification of modular DBMSs at runtime. We want to add, exchange, and remove modules while the database system is running. A possible scenario is the installation of a small DBMS with only a few features and its extension at runtime if new features are required. E.g. at the time of installation of an application a B-tree index is sufficient and therefore only the B-tree module is installed to save disk space. After a while a bitmap index would be very helpful. In a runtime adaptable DBMS the module for the bitmap index can be installed without stopping the DBMS. Towards this requirement a framework is needed to manage the individual modules.

2. RELATED WORK

The drawbacks of a monolithic DBMS are presented in [7]:

- DBMS become too complex. It is not possible to maintain them with reasonable costs.
- Applications have to pay performance and cost penalty

for using unneeded functionality.

- System evolution is more complex, because a DBMS vendor might not have the resources or expertise to perform such extensions in a reasonable period of time.

To solve such problems Dittrich and Geppert [7] propose componentization of DBMSs and identify four different categories of CDBMSs (component DBMS):

- Plug-in Components. The DBMS implements all standard functionality, and non-standard features can be plugged into the system. Examples for this category are Oracle Database [2], Informix [13] and IBM's DB2 [6].
- Database Middleware. DBMSs falling in this category integrate existing data stores, leaving data items under the control of their original management system, e.g. Garlic [16], Harmony [15] and OLE DB [4].
- DBMS Services. DBMSs of this type provide functionality in standardized form unbundled into services, e.g. CORBAservices [3].
- Configurable DBMS. DBMSs falling in this category are similar to DBMS services, but it is possible to adapt service implementations to new requirements or to define new services. An example is the KIDS project [8].

A good overview of these categories can be found in [20].

In recent time service-oriented architecture (SOA) has become a popular buzzword. The principles of SOA are also suitable for building modular DBMSs. [19] present an approach towards service-based DBMS. They borrowed the architectural levels from Härder [9] and want to include the advantages introduced by SOA like loosely coupled services. They have defined four different layers, which are implemented as services. It is argued that DBMS build upon this architecture is easily extendable because service can be invoked when they are needed and in case of failure of services, alternative services can answer the request. Tok and Bressan [21] also introduce a DBMS architecture based on service orientation, called SODA (Service-Oriented Database Architecture). In their DBNet prototype web services are used as the basic building blocks for a query engine.

These approaches do not address the handling of cross cutting concerns (section 3.2) in a modular DBMS and in contrast to our approach do not provide runtime adaptation.

3. OUR APPROACH

The major goals of our CoBRA-DB project (Component Based Runtime Adaptable DataBase) are:

- Modularization of a relational database management system
- Exchanging modules at runtime

In this section we present work in progress. At first we introduce an educational DBMS called i6db, which is developed at our department. The i6db lays the foundations for our recent work towards a modular database system. Then we present our activities regarding the handling of cross cutting concerns inside a DBMS and finally we show our approach concerning runtime adaptation.

3.1 Modularization of database systems

Modularization and the definition of abstract interfaces are the first steps in order to get modules that can be exchanged at runtime. The challenge is to identify the appropriate modules. Härder [9] proposed a multi layer architecture of DBMSs that is very useful to understand the functionality of DBMS and to structure the important parts. The proposed layers are not fine grained enough to map them exactly to DBMS components; this would limit the dynamic adaptation (section 3.3), because the exchange of a whole layer's realization would lead to overhead if only small changes would be required.

In this section we present the architecture of i6db, a DBMS, which was designed and implemented at the department of computer science 6 (database systems) at the university of Erlangen-Nuremberg over the last years. The i6db is written in C++ and concentrates on layer abstraction, design patterns, and loose coupling.

i6db - a database for educational purposes

The i6db was originally designed for educational purposes, e.g. we set transactions and multi-user handling aside. Regarding to Härder's five level architecture (figure 1) [9] we have implemented the layer L1 to L4. The query engine can execute queries based on relational operators. Higher order query languages like SQL are taught on the basis of existing databases for educational purposes.

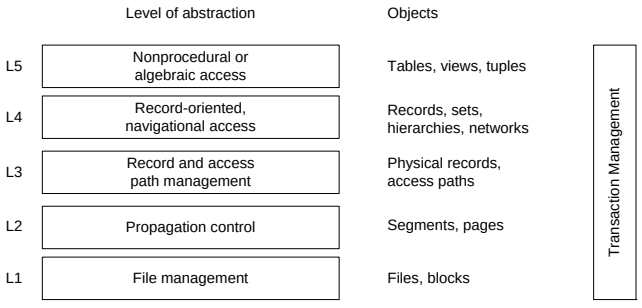

Figure 1: Five layer architecture

The core of i6db consists of seven modules. The module `file` is the L1 abstraction (file management) [9], `segment` and `systembuffer` are the L2 abstraction (propagation control). We assumed page/block-oriented data organization. All modules of L3 use accordingly the system buffer module. At the moment we have four different alternatives for the implementation of the record manager. There are two different algorithms (Tuple IDentifier, DataBase key Translation Table) both with the extension of support of records that are larger than database pages. The exchange of those record manager's realizations is quite simple.

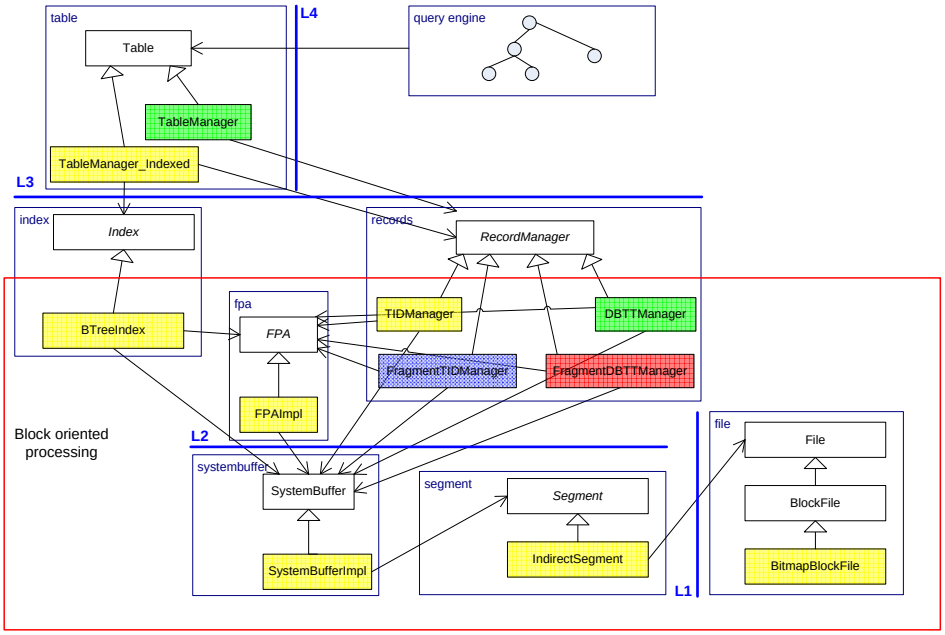

Figure 2: Modules of i6db

In figure 2 the large box enframes all realizations whose algorithms are based upon the use of pages. If i6db should be used as a main memory database, the block organization is obsolete. Then, all block based algorithms and structures must be abolished. As indexes and record manager don't need block orientation necessarily, the interfaces (Index, RecordManager) of the L3 (record and access path management) modules could be used further, but the modules must be exchanged. The *block*-tree index, the *free place administration* and all modules that organize records in blocks must be replaced by structures that are organized in main memory.

The L4 module (record-oriented, navigational access) `table` has two alternatives. The one holds a use-dependency to the `index`-module, the other doesn't. The `table`-module can be accessed by the methods that `table` provides. It depends solely on the table's implementation if indexes can be used. If an index-access method is called and there is no index available a full table scan has to be performed.

Our query engine can be used for the definition of tables and indexes. Records can be stored, removed and altered, too. Queries are defined by a query graph that consists of implemented relational operators. Due to the formal definition of relational operators each operator gets a set of input operators and predicates. Each operator can iterate the result. There are two table iterators implemented that access the tables of the `table`-module either by full-table-scan or by index-table-scan.

The i6db project lays a solid basis for creating a modular and adaptable database at least. In future we will integrate transaction mechanisms. As discussed below transactions are cross-cutting concerns. With the integration by using aspect-oriented programming techniques we can provide an interface that support transaction handling. Then we can abolish the selfmade query engine and use MySQL 5.1 and integrate our i6db with its sound architecture as storage engine of MySQL [1].

3.2 Uncoupling Transactions

Existing approaches propose the incorporation of SOA to develop a modular DBMS [19, 21]. Modules like the query engine or layers [9] are realized as services. But cross cutting concerns like logging or transaction management (figure 1) make modularization in a full-fledged DBMS difficult, because e.g. to realize transactions, nearly every module is involved in the transaction management [14]. A "transaction object" is created at the beginning of a transaction and can not be destroyed until the transaction commits. Such an object could not be realized as a stateless service and all procedure calls which belong to the statements that are executed within a transaction must be associated with this object.

We are currently working on an extraction of the transactional aspect. The DBMS modules should not be aware of the fact that they are part of a transaction. For a prototype we have removed the implementation of transaction management in SimpleDB [17] and we are currently "re-integrating" the transaction support with the help of aspect-oriented programming (AOP) [12]. With AOP it is possible to intercept the methods of the modules and gain all necessary information to support transactions without the drawback of direct coupling. While we are presenting work in progress, there are still problems to be solved in our prototype, like the tracing of method invocations during the execution of a SQL statement.

3.3 Adaptation at runtime

Beside modularization the second major goal of the CoBRA-DB project is runtime adaptation. There are different sce-

narios where runtime adaptation is useful:

- Change a module that contains a bug
- Upgrade the DBMS with new functionality
- Remove unused features to increase performance and save space
- Change of a set of modules by cross cutting concerns or multiple changes of different modules as a result in order to change bigger parts or strategies of a DBMS

An example for a far-reaching upgrade would be adding transactions management to a running DBMS, which has not the need for transactions at time of its installation.

We pick up the idea of Cervantes and Hall, who have introduced a service-oriented component model: "A service-oriented component model introduces concepts from service orientation into a component model. The motivation for such a combination emerges from the need to introduce explicit support for dynamic availability into a component model" [5]. This concept can be used as basis for our work towards runtime adaptation. DBMS "modules" are realized as components that provide their functionality as services. Some of these modules are mandatory and provide the DBMS's core functionality like inserting and querying data. Other services are optional and can be added or removed depending on the application and the environment, e.g., if transaction support is not required, the "transaction module" is removed.

We do not propose the distribution of services. Remote procedure calls are much to slow to be used inside a DBMS. The required components should be able to invoke one another locally. To accomplish loosely binding a service searches in a global registry to locate services which are required to fulfill its role and than these service can be bounded and invoked.

One characteristic of service orientation is the fact that service can arrive and disappear at any point in time. Business applications which rely on specific service are not available if a mandatory service disappears and can not be replaced with another adequate service. This behavior is not acceptable if we are building a DBMS, because if one service of the DBMS disappears, all depending applications would not be able to work correctly. With this requirement in mind it is absolutely necessary for a DBMS which relies on service-oriented components that these components are available and their provided services are accessible. To swap services at runtime, the adaptation has to be done transparently for all consumers of that service. Obviously some time is needed to replace a running component with a new implementation. Therefore the method calls have to be interrupted and redirected to the new component. This "switch over act" has to be done in an atomic operation for all services which rely on the adapting service.

To handle the whole adaptation process we introduce an Adaptation Manager which coordinates the individual steps. The process is divided in 3 phases (figure 3). During the

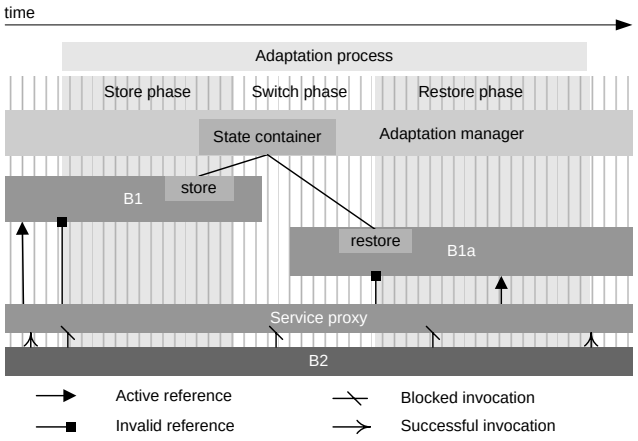

Figure 3: Service adaptation process

store phase the references of all depending service are invalidated and the state of the "old" component is saved in a special data structure. In the switch phase the state is injected in the new component. Last the references are set to the new component in the restore phase, and the references are set to the new component. We describe the adaptation process in [10] in detail.

To enable the adaptation of aspects at runtime we integrate a dynamic AOP (d-AOP) framework in our prototype because d-AOP supports the modification of aspects at runtime. Therefore we can use the techniques we have presented in [11] to integrate dynamic AOP into a service-oriented component model.

4. FUTURE WORK AND CONCLUSION

In the paper we introduced the CoBRA-DB project. The goal of this project is a modularized runtime adaptable DBMS. We argued the problems of "slicing" a database into loosely coupled components and the challenge regarding cross cutting concerns. We are currently implementing a prototype framework to adapt components at runtime. Thereby the state of a component is transferred to the replacing component in an atomic step. With the lessons learned in the i6db project, where we have implemented a DBMS in C++, we are now going to develop a prototype in Java. In parallel we remove the transaction management from a sample DBMS (we use SimpleDB) and reintegrate it with the help of AOP to provide a foundation for further modularization. This is a major difference in contrast to other projects which use SOA to modularize DBMSs. Another distinction is the ability to swap modules at runtime and thereby adapt a DBMS to a changing environment without the need to shutdown the database.

5. REFERENCES

[1] D. Axmark, M. Widenius, P. DuBois, S. Hinz, M. Hillyer, and J. Stephens. *MySQL 5.1 Referenzhandbuch*. MySQL, 2007.

[2] S. Banerjee, V. Krishnamurthy, and R. Murthy. All your data: the oracle extensibility architecture. *Component database systems*, pages 71–104, 2001.

[3] R. Bastide and O. Sy. Formal specification of CORBA

services: experience and lessons learned. *Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 105–117, 2000.

[4] J. A. Blakeley. Data access for the masses through ole db. *SIGMOD Rec.*, 25(2):161–172, 1996.

[5] H. Cervantes and R. S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 614–623, Washington, DC, USA, 2004. IEEE Computer Society.

[6] J. Cheng, J.; Xu. Xml and db2. *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 569–573, 2000.

[7] K. R. Dittrich and A. Geppert, editors. *Component database systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[8] A. Geppert, S. Scherrer, and K. R. Dittrich. KIDS: Construction of Database Management Systems based on Reuse. Technical Report ifi-97.01, University of Zurich, 1997.

[9] T. Härder. DBMS Architecture - the Layer Model and its Evolution (Part I). *Datenbank-Spektrum*, 5(13):45–56, 2005.

[10] F. Irmert, T. Fischer, and K. Meyer-Wegener. Improving availability in a service-oriented component model using runtime adaptation. University of Erlangen and Nuremberg, to be published, 2007.

[11] F. Irmert, M. Meyerhöfer, and M. Weiten. Towards Runtime Adaptation in a SOA Environment. RAM-SE'07 - 4th ECOOP Workshop on Reflection, AOP and Meta-Data for Software Evolution, co-located at the 21th European Conference on Object-Oriented Programming - ECOOP (Berlin, Germany), July 2007.

[12] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[13] M. Olson. DataBlade extensions for INFORMIX-Universal Server. *Proceedings IEEE COMPCON*, 97:143–8, 1997.

[14] A. Rashid. *Aspect-Oriented Database Systems*. Springer, 2004.

[15] U. Röhm and K. Böhm. Working Together in Harmony - An Implementation of the CORBA Object Query Service and Its Evaluation. In *ICDE'99: Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia*, pages 238–247, 1999.

[16] M. T. Roth and P. M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 266–275. Morgan Kaufmann, 1997.

[17] E. Sciore. SimpleDB: a simple java-based multiuser system for teaching database internals. *ACM SIGCSE Bulletin*, 39(1):561–565, 2007.

[18] A. Silberschatz and S. Zdonik. Strategic directions in database systems - breaking out of the box. *ACM Comput. Surv.*, 28(4):764–778, 1996.

[19] I. E. Subasu, P. Ziegler, and K. R. Dittrich. Towards service-based database management systems. In *Datenbanksysteme in Business, Technologie und Web (BTW 2007), Workshop Proceedings, 5.-6. März 2007, Aachen, Germany*, pages 296–306, 2007.

[20] A. Tesanovic, D. Nystrom, J. Hansson, and C. Norstrom. Embedded databases for embedded real-time systems: A component-based approach. Technical report, Dept. of Computer Science, Linkoping University, and Dept. of Computer Engineering, Malardalen University, 2002.

[21] W. H. Tok and S. Bressan. DBNet: A Service-Oriented Database Architecture. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 727–731, Washington, DC, USA, 2006. IEEE Computer Society.