Advanced Database Systems: From Monoliths to Unbundled Components

Jürgen Zimmermann* Thomas Kudraß

Abstract

The unbundling of components from a (monolithic) database management system is an upcoming research topic. In this paper we describe the origin of unbundling techniques: unbundling in operating systems like SunOS 5.x and construction of a component-oriented database system like Open OODB from Texas Instruments, Inc. by using the EXODUS toolkit. Afterwards we discuss the commercial OODBMS ObjectStore to identify unbundling issues. In the core part we present the active database systems REACH and SAMOS from an unbundling point of view. Finally we introduce an object-oriented mediator for relational legacy systems and show that such a component also can be seen as unbundled.

Keywords: *unbundling, architecture, components.*

1 Introduction

Unbundling is an upcoming buzzword in the database community. However, it is not clear what does it mean when a component is declared as unbundled, how can components be identified for unbundling, and how should components be designed to make them unbundled components. In this paper we look back and illustrate the original meaning of unbundling in the area of operating systems. Using SunOS 5.5 as an example we show how unbundled products like a C compiler can be grouped around a full-functioning core component.

Is such a technology and design principle adaptable to DBMSs? What would be the benefits? This paper is the first approach to answer these questions. We show that the techniques from component-oriented DBMSs like EXODUS [CDRS86], GENESIS [Bato85], and Open OODB [WBT92] from Texas Instruments, Inc. provide a good starting point to understand how components should be designed for unbundling.

The practical benefit of unbundling is illustrated for the active DBMSs REACH [BZBW95] and SAMOS [GFV96], and for a mediator which is developed on top of Persistence [KLB96], [Pers94]. The whole systems are becoming more modular and lean if the unbundled components are not required. Additionally, in the case of active DBMSs we show the limitations of unbundling when the state changes of objects should be detected as relevant events.

The remainder of this paper gives an overview of unbundling for SunOS 5.5 and for component-oriented DBMSs like EXODUS, GENESIS, and Open OODB in section 2. In section 3 we discuss how the active component of an active DBMS and how a mediator between relational legacy systems can be unbundled. The conclusions are addressed in 4.

2 Background

To understand the term *unbundling* we take a look at products and prototypes which are designed with respect to unbundling. The main goal of unbundling is to provide *several* components instead of one large monolith so that an independent and functioning software package can be the core of an (extensible) system. Therefore, we give an overview of Solaris 2.5 from Sun Microsystems, Inc. to show how unbundled products like a compiler can be grouped around the core, namely the operating system SunOS 5.5. Having this well-established commercial area in mind the component-oriented DBMSs

*Authors' address: Jürgen Zimmermann, Thomas Kudraß; University of Darmstadt, Julius-Reiber-Strasse 17, 64293 Darmstadt, Germany. Email: {zim,kudrass}@dvs1.informatik.th-darmstadt.de

GENESIS [Bato85], EXODUS[CDRS86], and Open OODB [WBT92] are presented to introduce the base technology for unbundling in the area of DBMSs. Finally, we discuss a coarse principle of unbundling for ObjectStore [LLOW91] which provides different libraries to build the clients resp. applications.

2.1 The Environment Solaris 2.5

When Sun Microsystems Inc. started to ship the SVR4 based operating system SunOS 5.x they introduced the name Solaris 2.x to indicate that this environment consists of several components which are grouped around the core, namely the operating system SunOS 5.x. In the beginning there were a lot of discussions because the C compiler was declared as an unbundled product which does not belong to the operating system itself. Meanwhile there are several unbundled products (some are free of charge) like OpenWindows 3.x or the new Common Desktop Environment (CDE) which is Sun's implementation of the COSE specification made by Sun, HP, IBM, and Novell. Other useful unbundled components as shown in fig. 1 are Online Disksuite for disk striping, Networker to backup disks, Wabi to emulate MS-Windows 3.1, and last but not least ODBC driver.

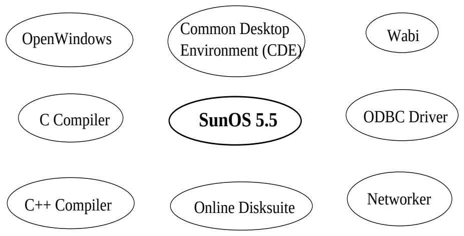

Figure 1: SunOS 5.5 with Unbundled Products

2.2 Component-Oriented Database Systems

As shown in fig. 1 unbundling of components is only useful if a full-functioning core package like SunOS remains. The "only" problem is: how to determine components which can be unbundled so that a full-functioning and lean kernel remains? Therefore, we look at DBMSs which were designed from a component-oriented point of view. First of all there are toolkits like EXODUS [CDRS86] and GENESIS [Bato85] providing the relevant components (e.g. lock or recovery manager) to build a DBMS consisting of these building blocks. In the next paragraph we introduce Open OODB [WBT92] from Texas Instruments, Inc. which is build on top of EXODUS and aims to be extensible and changeable in the sense that so-called policy managers (resp. components) like the "Query PM" can be substituted or omitted.

2.2.1 The Toolkits EXODUS and GENESIS

The EXODUS storage manager [CDRS86] can be used as a platform to develop OODBMSs like the first prototype of O2 [BDK92], the GOM prototype [KeMo94], and Open OODB (see 2.2.2). EXODUS has a client/server architecture. The server is a separate and possibly remote process which provides a variety of services to multiple clients. These services include *storage objects*, *versions* of objects, *transactions*, *concurrency control*, *recovery*, and *indexes*. A storage object is an uninterpreted container of bytes which can be referenced by an object identifier (OID) and may be versioned. Several objects can be physically clustered by storing them in the same EXODUS file, and additionally indexes can be created which may be either a B+ tree or a linear hash index. The provided transactions are flat ACID transactions which are isolated by using a hierarchical two phase locking (2PL) protocol. Recovery from software, operating system, or CPU failure is provided by the ARIES recovery algorithm [Moha89].

The design principle of the GENESIS toolkit is a layered architecture. The higher the level is the more complex is the service provided by a layer. For instance the basic layer contains a component for blocks which can be anchored or not, and have fixed or variable length. If there is an overflow in a block these data will be stored in other blocks which can be shared between other blocks or unshared, and which can be ordered or not. In a higher layer there are data structures for B+ trees, heaps, hash tables, and even ISAM files. The components in the highest layer are designed for relational and CODASYL systems, namely merge join, nested loop join, pointer array, and ring list.

2.2.2 Open OODB

Although EXODUS is called a DBMS in [CDRS86] it is hard to implement applications on top of EXODUS. But if EXODUS and its client library is used as a platform for developing OODBMSs like Open OODB [WBT92] it is a good toolkit. Open OODB provides database functionality (such as transactions, persistence, queries, etc.) via an extensible set of seamless behavioral extensions of the objects and operations from application programming languages like C++ and Lisp. Open OODB is designed to have orthogonal policy managers (i.e. components) as shown in fig. 2.

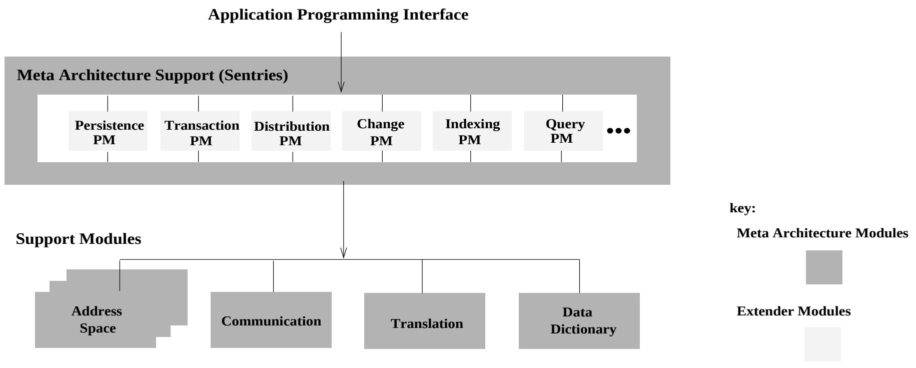

Figure 2: Open OODB Architecture.

In the last shipped release 0.2.1 alpha the policy managers (PM) for persistence, transactions, data dictionary, query language, and extensibility were implemented - distribution, versioning, and indexing were missing. Assuming the policy managers are really orthogonal several questions have to be answered: How can the persistence PM be substituted without affecting the transaction PM, e.g. the current persistence PM supports persistence by reachability which has to be checked at commit time and a new persistence PM shall support persistence by inheritance? How can the transaction policy manager be exchanged, i.e. how can a new manager for optimistic concurrency control be integrated without considering the underlying EXODUS which provides flat transactions based on page locks?

2.3 Libraries in ObjectStore

In the previous section we presented the fine-granular components of Open OODB. Now we briefly sketch the coarse components of ObjectStore [LLOW91] from a user's point of view. ObjectStore consists of two libraries: `libos` and `liboscol`. In `libos` all necessary classes are implemented for the handling of databases, transactions, objects, and root objects. The library `liboscol` provides templates for collection classes: sets, bags, lists, arrays, their base class `os_Collection`, and the associated cursors to traverse the collections. Obviously it is possible to work with ObjectStore without using the templates[1]. However, as soon as the query language has to be used collections are required for query purposes.

3 Unbundling in Advanced Database Systems

After presenting the background for unbundling techniques in database systems we focus on unbundling in advanced database systems. First, we look at the active DBMSs REACH [BZBW95] and SAMOS [GFV96]. From an architectural point of view we elaborate how an active component can be unbundled from the remaining DBMS. In the second paragraph we discuss a mediator system based on Persistence [Pers94] which can be used to integrate RDBMSs like Oracle or Sybase.

3.1 Active Database Systems

An active DBMS provides the facility to define event-condition-action (ECA) rules within the database schema. In an active OODBMS like REACH or SAMOS (fig. 3) the typical events that can be detected are method invocations, transaction events (BOT, COMMIT, ABORT), time events and compositions

[1]The precompiler of Open OODB 0.2.1 alpha does not understand templates which are conform to ANSI C++ 3.0.

of them. When an event is detected the rule's condition is evaluated and if it holds the action will be executed. Thus, the whole rule firing process consists of three phases: a) event detection, b) rule retrieval, and c) rule execution. This principle can be directly implemented for an active DBMS with a centralized rule manager like SAMOS. Since REACH [BZBW95] consists of specialized and intelligent event detectors rule retrieval is not required because each detector knows exactly which rules have to be triggered by the detected event. Anyway, the rule retrieval and execution can be implemented in components that are independent from the underlying OODBMS[2].

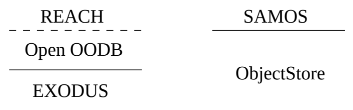

Figure 3: Components of REACH and SAMOS.

To unbundle the active component from the underlying OODBMS the event detection mechanism has to be isolated. Both REACH and SAMOS detect method invocations, but not state changes of the objects. Therefore, the detection of the method events can be done by wrapping *each* method automatically by a (new or modified) precompiler [Deut94]. However, if state changes should be detected as in the active DBMS NAOS [CCS94] a strict layered architecture is impossible and the OODBMS needs some major modifications.

3.2 A Mediator Based on Persistence

A mediator which uses Persistence [Pers94] is described in [KLB96] to enforce global consistency between relational legacy systems. Persistence itself provides the facilities to access RDBMSs like Oracle or Sybase in an object-oriented application.

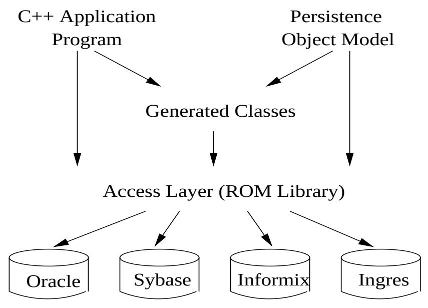

Figure 4: Persistence Architecture.

Thus, we have two kinds of unbundling: a) Persistence provides an independent access layer on top of RDBMSs, b) the mediator [KLB96] uses the hooks of Persistence to detect events concerning global consistency issues. The relational legacy systems remain full-functional; Persistence is sitting on top of them as an object-oriented access layer and provides the platform for the mediator.

4 Conclusion

In this paper we presented a first approach to identfy database components which can be unbundled from full-functional core DBMSs which can be relational or object-oriented. The overall architecture of such a system consists of several unbundled components and at least one core component. This kind of modularization not only improves the maintainability but also results in leaner systems because components can be omitted if they are not required for the applications. For instance if a company only uses Oracle then there is no need to use a mediator for heterogeneous DBMSs or if there are only a few rules for the applications a company may consider to dispense with the active component.

[2]REACH's rule engine only requires that the underlying DBMS can provide OIDs for the objects to be transferred to new top-level transactions which execute rules with detached coupling. For this purpose a modified Open OODB is used in REACH.

References

[Bato85] Batory, D. S.; *Modeling the Storage Architectures of Commercial Database Systems.* ACM TODS, Vol. 10, No. 4, Dec 1985, pp 463-528.

[BDK92] Bancilhon, F., Delobel, C., Kanellakis, P. (Ed.); *Building an Object-Oriented Database System - The Story of O2.* Morgan Kaufmann, 1992.

[BZBW95] Buchmann, A. P., Zimmermann, J., Blakeley, J. A., Wells, D. L.; *Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions.* Proc. 11th Intl. Conf. on Data Engineering, Taipeh, March 1995.

[CCS94] Collet, S., Coupaye, T., Svensen, T.; *NAOS - Efficient and Modular Reactive Capabilities in an Object-Oriented Database System.* Proc. 20th Intl. Conf. on Very Large Databases, Santiago, Chile, 1994

[CDRS86] Carey, M., DeWitt, D., Richardson, J., Shekita, E.; *Object and File Management in the EXODUS Extensible Database System.* Proc. 12th Intl. Conf. on VLDB, Kyoto, Japan, 1986.

[Deut94] Deutsch, A.; *Method and Composite Event Detection in the REACH Active Database System.* Master Thesis, University of Darmstadt, 1994.

[GFV96] Gatziu, S., Fritschi, H., Vaduva, A.; *SAMOS an Active Object-Oriented Database System: Manual.* University Zurich, Tech. Report 96.02, 1996.

[KeMo94] Kemper, A., Moerkotte, G.; *Object-Oriented Database Management.* Prentice Hall, 1994.

[KLB96] Kudraß, T., Loew, A., Buchmann, A. P.; *Active Object-Relational Mediators.* Proc. 1st Intl. Conf. on Cooperative Informations Systems (CoopIS), Brussels, 1996.

[LLOW91] Lamb, C., Landis, G., Orenstein, J., Weinreb, D.; *The ObjectStore Database System.* Communications of the ACM, Vol 34, No 10, Oct. 1991, pp 34-49.

[Moha89] Mohan, C. et al.; *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging.* IBM Research Report RJ6649, 1989.

[OODB94] Texas Instruments, Inc; *Open OODB C++ API User Manual.* Release 0.2.1 (Alpha), 1994.

[Pers94] Persistence Software Inc.; *Persistence User Manual for Release 2.3.* Oct 1994.

[WBT92] Wells, D. L., Blakeley, J. A., Thompson, C. W.; *Architecture of an Open Object-Oriented Database Management System.* IEEE Computer, Vol 25, No 10, Oct. 1992.