METU Object-Oriented DBMS Kernel

Asuman Dogac, Mehmet Altinel, Cetin Ozkan, Budak Arpinar, Ilker Durusoy and Ilker Altintas

Software Research and Development Center of TUBITAK
Middle East Technical University (METU)
06531 Ankara Turkiye
email: asuman@srdc.metu.edu.tr

Abstract. This paper describes the design and implementation of a kernel for an OODBMS, namely the METU Object-Oriented DBMS (MOOD). MOOD is developed on the Exodus Storage Manager (ESM). MOOD kernel provides the optimization and interpretation of SQL statements, dynamic linking of functions, and catalog management. SQL statements are interpreted whereas functions (which have been previously compiled with C++) within SQL statements are dynamically linked and executed. Thus the interpretation of functions are avoided increasing the efficiency of the system. A query optimizer is implemented by using the Volcano Query Optimizer Generator. A graphical user interface, namely MoodView, is developed using Motif. MoodView displays both the schema information and the query results graphically. Additionally it is possible to update the database schema and to traverse the references in query results graphically.

Keywords: OODBMS kernel implementation, query optimization in OODBMSs, dynamic function linker, Graphical User Interface

1 Introduction

In this paper we describe our experience in the design and implementation of the METU Object-Oriented DBMS (MOOD) kernel. The system is coded in GNU C++ on Sun Sparc 2 workstations. MOOD has a SQL-like object-oriented query language, namely MOODSQL [15, 8], and a graphical user interface, called MoodView [2, 3] developed using Motif. MOOD has a type system derived from C++, eliminating the impedance mismatch between MOOD and C++. The users can also access the MOOD Kernel from their application programs written in C++. For this purpose MOOD Kernel defines a class named UserRequest that contains a method for the execution of MOODSQL statements. MOOD source code is available both for anonymous ftp users from ftp.cs.wisc.edu and for the WWW users from the site http://www.srdc.metu.edu.tr along with its related documents.

MOOD is developed on top of the Exodus Storage Manager (ESM) which has a client-server architecture [10, 5] and each MOOD process is a client application in ESM. ESM provides the MOOD some of the kernel functions like storage management, concurrency control, backup and recovery of data.

Additional kernel functions provided by the MOOD are the optimization and interpretation of SQL statements, the dynamic linking of functions and catalog management. MOOD query optimizer is implemented by using the Volcano Query Optimizer Generator [12]. During the interpretation of SQL statements functions (which have been previously compiled with C++) are dynamically linked and executed. This late binding facility is essential since database environments enforce runtime modification of schema and objects. With our approach, the interpretation of functions are avoided increasing the efficiency of the system.

An alternative way to handle dynamic linking is to extend a C++ interpreter with DBMS functionality. In this alternative there is a problem of performance decrease due to interpretation. The advantage is to be able to use the full power of C++.

The implementation of a query optimizer for a SQL-like object-oriented query language, namely MOODSQL, is presented. The Volcano Query Optimizer Generator is used in developing the MOOD optimizer.

The paper is organized as follows: An overview of the MOOD Kernel is described in Section 2. In Section 3, the implementation of the Catalog Manager is given. The Query Manager and Dynamic Function Linker are discussed in Sections 4 and 5 respectively. Section 6 describes the MOOD Query optimizer. In Section 7, the implementation of the Database Engine is summarized. Kernel interaction with MoodView is given in Section 8. Finally conclusions are presented in Section 9.

2 MOOD Kernel Design Considerations and an Overview

There are some design choices that we have made and some decisions are enforced by the implementation language, namely C++. These are discussed in the following within the framework presented in [11].

Object Specification in MOOD: Each object is given a unique Object Identifier (OID) at object creation time by the ESM which is the disk start address of the object returned by the ESM. Object encapsulation is considered in two parts, method encapsulation and attribute encapsulation. These encapsulation properties are similar to the public and private declarations of C++.

Declarative properties of the MOOD : In the MOOD, the aspect concept given in [11] is limited to type specifications, in other words, class definitions are analogous to type definitions in a programming language. Hence each instance in the system is defined as a member of a class.

Procedural properties of the MOOD: Methods can be defined in C++ by users to manipulate user defined classes. Method binding is performed dynamically at run time. Dynamic linking primitives are implemented by the use of the shared object facility of SunOS [17]. Overloading is realized by making use of the signature concept of C++.

Object Abstractions in the MOOD: Objects are grouped in the abstraction level of a class, in other words, classes have extensions. Class extensions are implemented as ESM files. A class in the system has an unique type identifier which is inherited from a meta class named MoodsRoot. This type identifier is used in accessing the catalog to obtain the type information to be used in interpreting the ESM storage objects which are untyped arrays of bytes. The relation between classes and instances is a 1:n relation, i.e., under a class there could be any number of instances associated with it, but an instance can not be associated with more than one class. Class inheritance mechanism of the MOOD Kernel is multiple inheritance. The name resolution is handled as in standard C++. Additionally in our system in case of name conflicts, if the scope resolution operator is not used, the first class in the inheritance order having that attribute is assumed as default.

Aggregates in the MOOD: Aggregate definitions are handled in the MOOD system by introducing type constructors (Set, List, Ref and Tuple). Aggregate classes can be constructed by recursive use of these type constructors.

2.1 MOOD Overview

The general flow of execution in the MOOD system is shown in the Figure 1. MoodView [2, 3] is the graphical user interface of the system. MoodView displays both the schema information and the query results graphically. Additionally it is possible to update the database schema and to traverse the references in query results graphically. In displaying the schema, MoodView either makes direct calls to the Catalog Manager (1) or issues the necessary commands to the Query Manager (3). For primitive operations involving a single function call, MoodView directly communicates with the catalog. Complex operations are passed to the Query Manager. Query Manager parses and executes these commands by obtaining the necessary information from the Catalog Manager (5,6). Results are returned from the Catalog Manager or from the Query Manager depending on which subsystem received the request (2,4). Query Manager handles the method creation through Dynamic Function Linker (8, 9).

MoodView passes the MOODSQL [8, 14, 15] queries to the Query Manager without any modification (3). Query Manager obtains the necessary information from the Catalog Manager (5,6) and then makes syntax and semantic checks on the queries. If no error is detected, a query tree is generated and passed to the query optimizer (7). After the optimization phase, resulting query tree is ready to be executed by the MOOD engine (10). During execution, class extents are read from the Exodus Storage Manager (ESM) and temporary results are stored in the ESM (16, 17). If the class methods are used in the query, they are activated through the Dynamic Function Linker subsystem (14,15). At the end of the execution, results are returned directly to MoodView (13). In Moodview, a user can traverse the links in the database or execute class methods with void return type by using the cursor mechanism provided.

MOOD also has a textual interface to the database. For this interface, MOODSQL provides schema definition and modification commands.