The Soprano Extensible Object Storage System

JUNG-HO AHN, Seoul National University, Korea
HYOUNG-JOO KIM, Seoul National University, Korea

An efficient object manager, a middle layer on top of a storage system, is essential to ensure acceptable performance of object-oriented database systems, since a traditional record-based storage system is too simple to provide object abstraction. In addition, an object storage system–object managers in combination with storage systems - should be extensible to meet the various requirements of emerging applications. In this research, we design and implement an extensible object storage system, called Soprano, in an object-oriented fashion which has shown great potential in extensibility and code reusability. Soprano provides a uniform object abstraction and gives us the convenience of persistent programming through many useful persistent classes. Also, Soprano supports efficient object management and pointer swizzling for fast object access. This paper investigates several aspects of the design and implementation of the extensible object storage system. Our experience shows the feasibility of using an object-oriented design and implementation in building an object storage system that should have both extensibility and high performance.

INTRODUCTION

In recent years, many object-oriented database systems have been developed and have become widely accepted in the next generation of telecommunications, Internet and financial applications around the globe. Due to the complexity of data management in such applications, key issues are performance and the requirement for a flexible and transparent object management environment. Thus, the commercial success of the object-oriented database systems largely depends on how well they meet these stringent requirements.

Contemporary relational database systems consist of two main modules: a query processor and a storage system. A query processor returns the result of a given query by translating it into a series of internal storage system calls. The low-level storage system provides data persistency and transaction management with full control of physical devices. In object-oriented database management systems (OODBMSs), however, it is no longer adequate for upper layers, such as a query processor, to call a low-level storage system directly. This is because the upper layers of an OODBMS should be adapted to the rich and extensible nature of the object-oriented data model directly, while a traditional relational storage system supports only record-oriented data abstraction. That is, upper layers (if built directly on top of the relational storage system) would have to implement object abstraction, resulting in poorer performance due to increased complexity (Bancilhon et al., 1992).

To overcome this problem, most OODBMSs employ a middle layer, which is called an object manager, on top of the storage system. The objective of an object manager is to reduce the impedance mismatch between upper layers (e.g., object query processor) and lower layers (e.g., storage system) by implementing object abstraction using the facilities of the underlying storage system. We summarize the basic functionalities of an object manager as follows (Bancilhon et al., 1992):

1. To generate object identifiers
2. To create and delete persistent objects
3. To support object access method
4. To support object naming service

Besides the above features, an object manager is also involved in method binding, object versioning and object clustering.

Along this line, an efficient object storage system, the object manager together with the storage system, is essential to ensure a reasonable performance of OODBMSs. In addition, object storage systems need to simplify the addition and modification of application-specific functions, since new database applications differ from traditional ones in their requirements of operations and storage structures. That is, the extensibility of the object storage system is the key to flexible object management.
The extensibility of an object storage system heavily depends on the system architecture. First, the system must be based on the architecture that can easily support the addition of new operations and storage structures. Second, the system should provide uniform interfaces for operations and its facilities. In addition, many of the basic architectural and performance tradeoffs involved in its design should be well understood. Considering the great potential of object-oriented paradigm in extensibility and reusability, it is evident that these prerequisites for an extensible object storage system could be achieved by an object-oriented design and implementation.

In this paper, we develop a high-performance extensible object storage system for next-generation database applications, called Soprano (SNU Object Persistent Repository with Advance Novel Operations). One of the key design features is to provide an object abstraction of all facilities (like B+-tree index) as well as persistent data for various advanced database applications which demand high performance. By treating everything as an object, the system as well as application programs are simplified. Soprano also helps persistent programming through many useful persistent classes and full support of C++ features like virtual functions and virtual base classes'. Soprano supports efficient object management and pointer swizzling for fast object access.

A number of research prototype object managers have been developed. Examples include Mneme at Massachusetts University (Moss, 1990), E Persistent Virtual Machine (EPVM) (Schuh et al., 1990) implemented on top of Exodus storage system (EXODUS Project Group, 1991), and ObServer (Hornick & Zdonik, 1987) at Brown University. There are also many object managers of commercial OODBMS products, including O2 (Bancilhon et al., 1992). However, these object managers do not consider the extensibility of itself. They also use a storage system of a relational DBMS for object persistency and thus, they cannot provide object concept uniformly over all around of the system and may cause unnecessary overhead.

The remainder of this paper is organized as follows. The next two main sections present the overall architecture of Soprano and describe the features and details of implementation techniques for major components. Then we evaluate the performance of our system based on the OO1 benchmark (Cattell & Skeen, 1992) and offer conclusions.

### Figure 1: Soprano Architecture

![Diagram of Soprano Architecture](image)

**Architecture Overview**

Figure 1 illustrates the overall (single-site) architecture of Soprano in terms of the major sub-modules that constitute the system. The lowest level is the objectbase that controls physical storage devices. The page cache and the object cache manage main memory buffers for pages and objects, respectively. The transaction manager, the lock manager, and the log manager coordinate concurrent object accesses and provide recovery capabilities.

Most existing object managers including that of O2 OODBMS implemented an object abstraction using the facilities of the relational storage systems. Although these systems made the most of existing storage systems, the two-layered architecture may suffer from performance degradation due to the complexity from unnecessary concept like a file. Instead, Soprano is one system, not two separate subsystems with arbitrary boundaries between them. Such tight integrated architecture can consistently control the flow of persistent objects from a physical device to an object cache. This also provides the extensibility such that a new type of system objects can be easily added, and leads to high performance by avoiding unneeded overhead. In addition, the uniformly designed system has good advantages in software engineering aspects like maintainability.

For example, Soprano, unlike the existing systems, does not support file concept directly. This is because a user should be able to access persistent objects without any intervention by file operations like 'open' and 'close'. That is, a