**Poseidon House**
**Castle Park**
**Cambridge CB3 0RD**
**United Kingdom**

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

**APM Business Unit**

# A Case Study of an Object Database Implementation

**Billy Gibson**

**Abstract**

The major of commercial database systems are based around the relational data model. However, the use of objects is now common in all aspects of computing. Utilising an object database in an project can reap benefits in both the design and implementation of the application.

This paper presents an overview of the construction of an object database engine. The system was developed as a key facility in a financial trading environment. The paper discusses the project requirements and design. In particular, it outlines the use of RPC communication. This greatly simplified the design of the database server and enabled fault tolerance to be achieved.

Using an object based database is radically different from a traditional relational tool. This paper demonstrates a typically database schema. It also explores the application interface and how information can be queried and extracted.

APM.1608.00.02 **Draft** 10th October 1995
Technical Report

**Distribution:**
**Supersedes**:
**Superseded by**:

Copyright © 1995 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

# A Case Study of an Object Database Implementation

# A Case Study of an Object Database Implementation

Billy Gibson

APM.1608.00.02

10th October 1995

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

| | |
|---|---|
| TELEPHONE UK | (01223) 515010 |
| INTERNATIONAL | +44 1223 515010 |
| FAX | +44 1223 359779 |
| E-MAIL | apm@ansa.co.uk |

**Copyright © 1995 Architecture Projects Management Limited**
**The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.**

Architecture Projects Management Limited takes no responsibility for the consequences of errors or omissions in this Report, nor for any damages resulting from the application of the ideas expressed herein.

# Contents


| | | |
|---|---|---|
| **1** | **1** | **A Brief History of an Object Database** |
| 1 | 1.1 | Introduction |
| 1 | 1.2 | Project Requirements |
| 2 | 1.3 | Third Party Systems |
| 2 | 1.4 | Design Overview |
| 2 | 1.4.1 | Server Process |
| 3 | 1.4.2 | Client API |
| 3 | 1.5 | Database Model |
| 4 | 1.6 | Working With Objects |
| 4 | 1.6.1 | Accessing Data Fields |
| 4 | 1.6.2 | Object Creation and Deletion |
| 5 | 1.6.3 | Queries |
| 5 | 1.7 | Fault Tolerance and Process State |
| 6 | 1.8 | Client / Server Communication |
| 6 | 1.9 | Summary |

# 1 A Brief History of an Object Database

## 1.1 Introduction

The benefits of employing object technology in application design and development are now well known. Viewing real life problems in terms of objects can often lead to a more natural design and thus a better implementation.

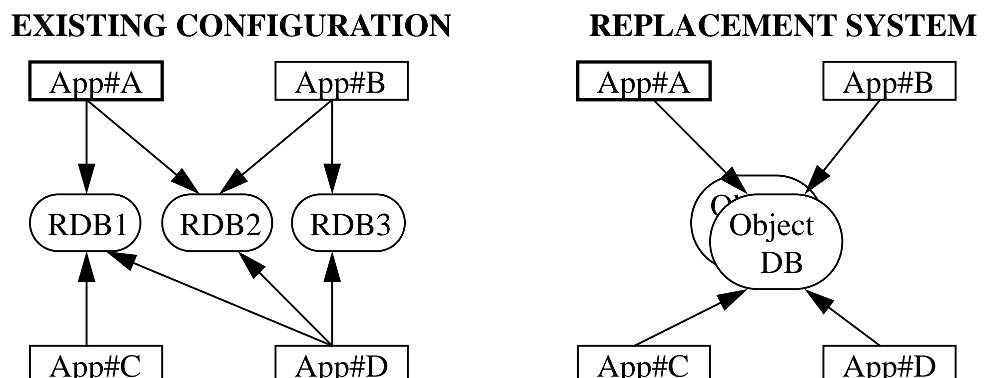
What has been missing from this object view of the world is the mechanism to store persistent data in a fashion that is consistent with the rest of the implementation. If an application is using objects internally to represent data, attempting to use a relational database to maintain the external copy of that information can be extremely cumbersome and unwieldy.

In recent years, a number of "object database" systems have reached the market place. There can be a number of advantages of utilising such a persistent storage mechanism. In the design phase, object oriented techniques can now be used throughout the project to include the database schemas. During the implementation, there is a much more natural interface between the internal representation of data and it's permanent storage.

## 1.2 Project Requirements

One such database was developed for a financial company. The initial use was to hold information on financial instruments (equities, bonds, etc.). The aim of the project was to replace a number of existing relational databases with a central depository. This rationalisation is represented in figure 1.

Figure 1.1: Database Rationalisation



The existing databases had been incrementally developed over a number of years. With the rapid expansion of distribute computing, the data had become fragmented and the systems over used. It was crucial that any new system would scale well - not only in terms in the number of users but in the amount and complexity of data.

Another key requirement was fault tolerance. The current databases did not provide reliability in a single coherent manner. The aim of the new system was to enable applications, where possible, to remain unaware of a database failure.

It was also essential that the new system satisfy a very high rate of requests. The interface should not only be small and flexible, but must also integrate well into current object based applications.

Clients of the database would be spread over a number of different hardware platforms, often with different data representations (word order and alignment).

It was estimated that in the initial project, the database would hold a maximum of 50,000 objects. During the day, the vast majority of requests would be read only. However, the nightly batch cycles would update almost all of the objects stored in the system.

## 1.3 Third Party Systems

Although there were a number of commercially available object databases. At that time, none fitted the requirements. Some were too slow, others had fundamental bugs. A further drawback was the interface language. Certain products could only be used from C++. The ability to access the data from C, Objective C and C++ was essential.

It was therefore decided that a home grown product was the only viable option.

## 1.4 Design Overview

Although the system was designed with a particular project in mind, it was essential that it would not be limited to this one use. This was partly due to the continually evolving scope of the project and the desire to reuse the database in other applications.

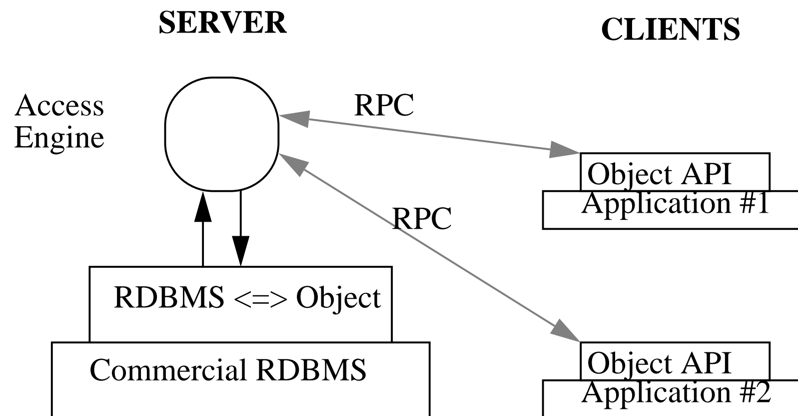
### 1.4.1 Server Process

Rather than creating a complete database system, the initial project set out to build an access engine which used a commercially available RDBMS for its permanent storage. Apart from reducing development time, this approach also provided a safety net for the project. If the access engine proved unsuccessful, there would still be the data and the schemas in the RDBMS.

A major aim of the access engine was to provide high speed access to data. To achieve this, the engine extracted all information from the RDBMS into its own storage space. During this process, data is converted from a table format to an object representation.

Although the access engine did not (initially) maintain its own permanent storage it did provide all other aspects of data serving. All queries and data retrievals were satisfied by the engine directly. Modifications to the data (object creation, deletion and updates) needed only to be sent to the engine. The access engine was responsible for changing its own copy of the data and also the copy under the RDBMS. This flow of information can be seen in figure 2.

**Figure 1.2: Communication Paths**




### 1.4.2 Client API

Applications accessed the database through a small but flexible API. This interface enabled queries to be created and data to be retrieved and updated. In order to speed data access the API employed a number of caching techniques. The application programs were typically unaware that these were being used.

The client API was object based. Each object in the database is referenced by its unique ID. In order to retrieve the value in a particular field of an object, the application needed to supply the object’s ID and the field name in question. Object IDs were typically found by performing a query. A more detailed explanation of the API can be found below.

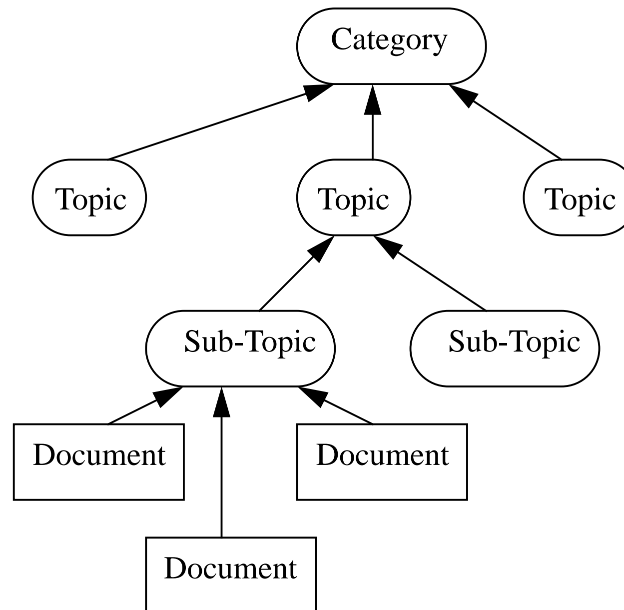
## 1.5 Database Model

As discussed above, the database provides a storage mechanism for data objects. When designing a database schema, the user is at liberty to create a variety of different object types. This is analogous to the concept of a “record” or a “structure” in many programming languages.

The database allows objects to be grouped into hierarchies. An object can have a single parent, buy may have any number of children! When an object is deleted from the system, all descendants are automatically killed.

An example of how different object types can be employed in a hierarchy is presented in figure 3. This example comes from a typical document management system.

**Figure 1.3: Object Hierarchy in a Document Management System**




### 1.6 Working With Objects

Each object in the database is uniquely referenced by an ID. To an application program, data in an object appears "flat". That is, an object simply consists of one or more data fields. An object always contains a "Parent-ID" field, which is the ID of its parent.

#### 1.6.1 Accessing Data Fields

To retrieve that value stored in a particular field of an object, the user would call a function similar to the following:

OK = odb_get_value(*Object-ID, Field-Name, &Value*);

The database supports the usual data types (integer, real and strings). The API provides various utility functions that enable more than one value to be fetched in a single call.

#### 1.6.2 Object Creation and Deletion

When an object is created, the user must specify the type of object desired and the ID of its parent.

New_ID = odb_create_object(*Object-Type, Parent-ID*);

Deleting an object can have a drastic effect on the database. Not only is the specified object remove, but the same fate meets all of its descendants.

```
OK = odb_delete_object(Object-ID);
```

### 1.6.3 Queries

Unlike a relational database, a query simply returns the set of matching objects. For example, if we created a database of books in the library we might perform a query which stated: "*Give me all books published in 1964*". In the object database, this query will return the IDs of all matching books. Once the query has been performed, we are then at liberty to retrieve information about each book in turn. This might be author, title, etc.

Performing a query is a three stage process. First the query is created by supplying a typical "where" clause. The query is then executed. Only at this stage is the server contacted and the database interrogated. The third stage is to find out which objects actually matched the query (reaping).

These stages are presented in pseudo code:

```
Query = odb_create_query(Where-Clause);

OK = odb_do_query(Query);

while ((Object-ID = odb_next_objectid(Query)) > 0)
{
        OK = odb_get_value(Object-ID, Field-Name, &Value);
        ...
}

odb_dispose_query(Query);
```

## 1.7 Fault Tolerance and Process State

As the system would be used in a critical environment, it was crucial that any failure on the server machine should not effect the client applications. To achieve hardware fault tolerance it would be necessary to have more than one database server machine. To achieve software fault tolerance each request from a client to the database engine was designed to be stateless.

This stateless model extended to the lowest level of client server communication. In the above example, each request to retrieve a data value (odb_get_value) may be fired off to a different database server.

## 1.8 Client / Server Communication

Given that communication between the client and server should be stateless, and supporting different client hardware was essential, a natural choice of protocol was RPC.

The use of RPC greatly simplified the database engine process. Each client request was design to be satisfied very quickly. This occasionally meant that one call to the client API from an application, resulted in a number of RPC requests being sent to a server.

This technique, combined with their stateless nature, allowed client requests to interweave. As a result, a very large number of clients could successfully access the same *single threaded* database engine.

## 1.9 Summary

The object database developed for storing financial information proved to be a success. The system surpassed the response time requirements. Developers found that the object database model integrated very easily into their applications. They also reported a marked decrease in the amount of code needed to access the database. In a number of cases thousands of lines of code were replace by a few hundred.

The database engine was later extended to maintain its own permanent backing store. This removed the requirement to use any RDBMS.

As a further enhancement, the server / client relationship was improved to provide the client with “real-time” updates to queries. This was achieved by applications marking a query as “active”. Clients would then be automatically informed of any database modification that affected the results of their active queries.

The object database system has been used in a production environment since February 1994. The original project for which it was designed has grown substantially. When last heard, that database consisted of over 80,000 objects. Many of these contained hundreds of data fields. Although the system was designed for a financial application, it is now employed in a variety of applications. One such project is a large international document management system.

# References

[LINDEN 93]

van der Linden R. J., *An Overview of ANSA*; **AR.000.00**, APM Ltd., Cambridge U.K., May 1993.