



## **Object-Oriented Databases and Mission-Critical Directories**

*A Technical White Paper  
June 1998*

In an increasingly competitive world, it is important for organisations to consider how new technology can be used to solve business problems. NEXOR has been at the forefront of innovation and development in directory systems through its Messageware Directory Product. The latest development of this product uses new technology to provide a more flexible and robust solution to directory systems. This paper discusses how this was achieved and the benefits for NEXOR's customers.

*For more information, please contact:*

**NEXOR**  
Rutherford House  
Highfields Science Park  
Nottingham, NG7 2PZ

Phone: +44 115 952 0500  
Internet: [info@nexor.com](mailto:info@nexor.com)

## Table of Contents

1. Introduction.....	3
2. Directory Systems.....	3
2.1 Background.....	3
2.2 Queries.....	5
2.3 Summary.....	6
3. Implementing a Directory System – The Choices.....	6
3.1 Introduction.....	6
3.2 Main-Memory Systems.....	6
3.3 B-Tree Systems.....	7
3.4 Relational DBMSs.....	7
3.5 Object-Relational DBMSs.....	8
3.6 Object DBMSs.....	8
3.7 Summary.....	9
4. Performance.....	9
4.1 Introduction.....	9
4.2 The British Telecom Benchmark.....	10
4.3 The OO1 Benchmark.....	10
4.4 Summary.....	13
5. Conclusions.....	13
References.....	13
About the Author.....	14

## 1. Introduction.

Today, organisations face many competitive challenges. Rapid technological advances, increasingly deregulated markets and tighter timescales and deadlines have caused these. Using appropriate software development tools to solve business problems to meet these challenges is an important issue for many organisations. At present, object-oriented technology is seen by many as providing significant benefits, including increased programmer productivity, better quality software and better time to market.

NEXOR has taken a major step forward in the development of directory systems, by adopting object-oriented tools and techniques. This paper describes some of the issues and choices it has faced and outlines various reasons why it has decided to follow this path.

The remainder of this paper is organised as follows. Section 2 provides a short introduction to directory systems and discusses some of their requirements. Section 3 describes the major implementation choices available for directory systems and their relative strengths and weaknesses. Section 4 discusses performance issues and summarises the results of several independent performance benchmarks. Finally, Section 5 presents the major conclusions, from which it will become obvious that only object-oriented technology is the most appropriate for next-generation mission-critical directory systems.

## 2. Directory Systems.

### *2.1 Background. .*

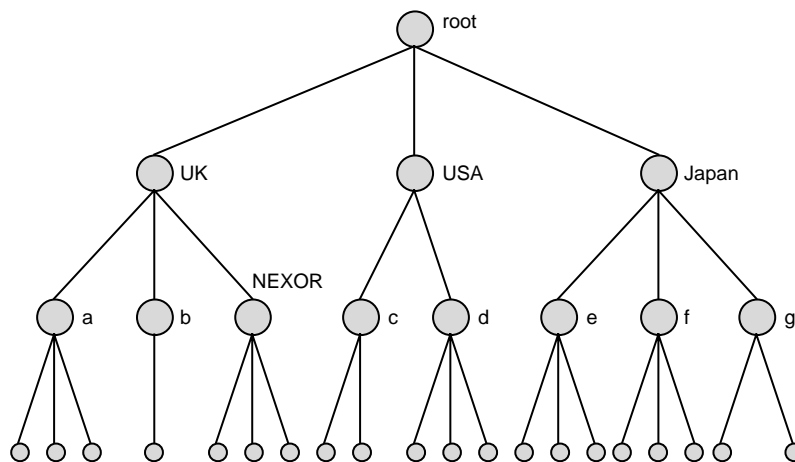
This section provides a brief introduction to X.500 directory systems. This introduction will provide the reader with a basic understanding of some of the concepts and requirements. The reader is also directed to [Coulo94; SURFn95] for further information.

Looking at one specific application, X.500 can be described as a **White Pages Directory Service** used to locate information based upon a name. It will contain information about network users, organisations and system resources [Coulo94]. This type of service is required, since the number of networks and distributed systems is continually growing

and a name service is needed to serve a similar purpose to telephone directories [Coulo94].

Data stored in X.500 servers are organised in a tree structure called a **Directory Information Tree (DIT)**, with named nodes. These nodes hold a range of attributes. The entire tree with all the attributes is referred to as the **Directory Information Base (DIB)**. The tree can be searched not only on name, but also a combination of attributes. Using distributed systems principles, only one DIB would exist worldwide, but with parts of it being located in individual X.500 servers [Coulo94].

A typical interaction sequence would be for a user (client) to establish a connection with a particular X.500 server and issue requests to it. If the particular server cannot fulfil the user request, because the data are not held in the local segment of the DIB, the contacted server will call other servers or redirect the query to another server. Clients and servers are termed as **Directory User Agents (DUAs)** and **Directory Service Agents (DSAs)**, respectively. Communication between DSAs is through a **Directory System Protocol (DSP)**, part of the X.500 recommendations.



**Figure 1** – Directory Information Tree.

Figure 1 illustrates a simplified DIT, with the **root** of the tree at the top, followed by **countries** (UK, USA, Japan), **organisations** (a, b, NEXOR, etc.) and finally **people** at the leaves. There may be another level as well, called **organisational units**, but this has not been shown.

Attribute type	Attribute value
Object Class:	person
Common Name:	Colin Robbins
	C.J. Robbins
	Colin J. Robbins

Surname	Robbins
Postal Address:	NEXOR P.O. Box 132 Nottingham NG7 2UU
Phone Number:	+44 (0) 115 952 0500
Fax Number:	+44 (0) 115 952 0519
E-mail:	<a href="mailto:info@nexor.com">info@nexor.com</a>
Hobbies:	Eating, Poetry, Crocodile-Wrestling

**Figure 2** – Example Directory Entry.

For white pages applications each node, with the exception of root, belongs to one object class (i.e. country, organisation, organisational unit, person) as defined by the X.500 standard and, as mentioned earlier, each node contains attributes that depend upon the object class. For example, for the Internet person object class, attribute types such as **common name**, **telephone number** and **e-mail** would be valid, whilst for the organisation object class, attribute types such as **organisation name** and **business category** would be used [SURFn95]. Furthermore, attributes may have multiple values, as illustrated above for common name (Figure 2), which has three values.

For each object class, one of the attribute types is used to specify a name for an entry. For the person object class, for example, this is usually common name [SURFn95]. From the example shown above, the value **Colin Robbins** would be the name of this node as it occurs in the DIT, referred to as the Relative Distinguished Name (RDN).

## **2.2 Queries.**

Users of a directory system could pose several types of queries:

- Simple telephone directory queries to obtain a person's e-mail address.
- Yellow pages queries to obtain information about all organisations that provide particular products or services.
- Queries to obtain personal details about an individual.
- Security information (e.g., X.509 public key certificates)

The types of queries can, therefore, be quite varied and imprecise. There are three main methods in which a directory can be accessed:

1. **Read** – this involves providing a domain name (absolute or relative name) for an entry along with the attributes that must be read. For example, finding a particular person’s e-mail address.
2. **Modify** – from time-to-time, information may need to be updated or added. For example, a new organisation’s name and details are added or an existing organisation’s details are modified.
3. **Search** – this is equivalent to a yellow pages search and involves providing a base name and filter expression. The base name is the node in the tree from which the search is to begin and the filter expression is evaluated for every node below the base node. For example, finding the names of all employees at NEXOR.

### **2.3 Summary.**

To summarise, both read and search could be costly operations, since large parts of the tree may need to be traversed and may reside on other servers. Requests to these servers would need to be made as discussed earlier. Two mechanisms that could be used to reduce the cost of such operations would be to use **replication** and **caching**. The following sections will demonstrate that both of these mechanisms, the model of navigation described above and the tree-like structure (DIT) fit very well with Object Database Management Systems (ODBMSs), which is the technology underlying NEXOR’s next-generation directory systems.

## **3. Implementing a Directory System – The Choices.**

### **3.1 Introduction.**

Current directory products tend to fall into two main categories. First, those that are based on main memory systems and second, those that use a B-Tree file package or Relational Database Management System (RDBMS). These could be termed as **first-** and **second-generation** directory systems, respectively.

### **3.2 Main-Memory Systems.**

Memory-resident systems have the advantage that they are very flexible, since they are not constrained by a particular data model. A disadvantage, however, is that memory size

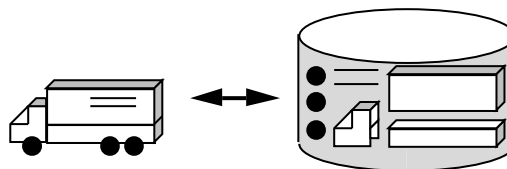
will limit the quantity of data that can be resident. Furthermore, node attributes would typically be stored in standard operating system files. These files could be shared between applications, but will prove difficult to share between more than one user at a time. For example, between a person undertaking a search operation, whilst another performs an update. In such cases, it would be necessary to “roll-your-own” Database Management System (DBMS) – a non-trivial task.

### **3.3 B-Tree Systems.**

Binary-Tree (B-Tree) systems are a natural choice for modelling tree structures and would appear well suited for directory systems. Binary search trees also provide very good performance for operations such as search, insert and delete. For example, a binary search tree of height  $h$  can be searched in  $O(h)$  time – better than most other data structures. However, they cannot guarantee reliability, since they do not provide any DBMS services and, therefore, suffer from the same problems as the memory-resident approach.

### **3.4 Relational DBMSs.**

RDBMSs are a mature technology and provide good database services and reliability, but impose a relational data model consisting of tables of rows and columns. An object class could be mapped to one or more relational tables, but the tree structure would need to be flattened to be stored in rows. To reconstruct the tree, many database operations may be required. An analogy that is commonly used to illustrate the problems with using RDBMSs to store complex or “bill-of-materials” structures is that of a vehicle and garage (Figure 3).



**Figure 3** – Relational Garage.

A relational garage requires that the vehicle be disassembled into smaller parts and these are stored onto various shelves (tables). The vehicle needs to be reassembled when it is required. Obviously this may be an expensive operation, particularly if many tables are used. Certain queries may also be costly, since the RDBMS needs to perform “search-

and-match” operations. Further examples to demonstrate performance and other well-known problems with using RDBMSs in this way have been discussed by Loomis [Loomi95].

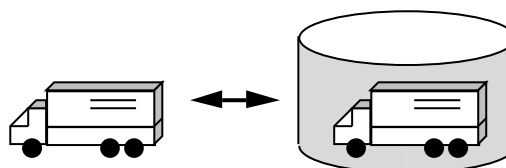
To provide the next-generation of directory systems, clearly new technology needs to be considered. This technology must provide the advantages of previous systems, without incurring any of their disadvantages. At present, there are two major alternatives that could form the basis of next-generation directory systems: Object-Relational DBMSs and pure ODBMS.

### **3.5 Object-Relational DBMSs.**

Object-Relational DBMSs generally refer to those products where there is an attempt to integrate both object-oriented and relational concepts. The underlying philosophy is to try and marry the best of both worlds – keeping all the benefits of relational technology that have accrued over the past twenty or more years (such as good optimisers, declarative query languages, etc.), but at the same time trying to provide some of the benefits of object-orientation (such as better abstraction). At present, however, this is still a young and mostly unproven technology for mission-critical applications. Furthermore, performance benchmarks described in [Asgar97] demonstrate that considerable work is needed to improve this technology to at least the standard of RDBMSs.

### **3.6 Object DBMSs.**

Pure ODBMSs generally integrate closely with object-oriented programming languages, such as C++, Smalltalk and Java<sup>a</sup>. They do this by extending these languages with DBMS services in a seamless manner, so that the programmer does not have to contend with two different languages, as found in RDBMSs and Object-Relational DBMSs. Furthermore, since object-oriented programming languages provide rich modelling constructs, an ODBMSs data model is richer and more flexible/extensible than other DBMSs.



**Figure 4 – Object Garage.**

Using the vehicle analogy again, an object garage would allow the vehicle to be stored “as is” without requiring any decomposition (Figure 4). This is because object databases support direct connections between objects. This can result in very good performance, since the network of connections can be traversed very quickly. Finally, ODBMSs are also mature and are being used for many mission-critical applications, as demonstrated by 18 case studies documented in [Loomi98]. Several of these case studies also show that ODBMSs are being used for projects with data in the terabyte and petabyte range with many users, demonstrating the scalability of this technology.

### **3.7 Summary.**

To summarise, first- and second-generation directory systems have a number of limitations, based on the underlying storage technology. New technology, such as ODBMSs, overcomes these limitations without sacrificing any of the benefits of previous systems. A **third-generation** directory system, based on an ODBMS, provides a robust, reliable and scalable solution to support the requirements of mission-critical systems.

## **4. Performance.**

### **4.1 Introduction.**

Performance has been briefly discussed in several previous sections. This section discusses this issue in more detail and provides evidence from several independent performance studies that demonstrate the advantages offered by ODBMSs against other technologies, such as B-Tree file packages and RDBMSs.

An important requirement for directory systems is good performance. This is generally difficult to achieve, since users pose queries that are both varied and imprecise, as mentioned earlier. Additionally, the read, modify and search operations impose competing requirements upon the storage manager, be it a file system, a B-Tree package or a DBMS.

NEXOR’s experience with RDBMSs has demonstrated that good performance for all three operations is difficult to achieve. For example, an RDBMS can be configured to provide good search performance, but at the expense of read and modify. Internal

benchmarks at NEXOR have demonstrated that only an ODBMS can provide good performance for all three operations. The results of other (independent) benchmarks to compare directory systems are difficult to obtain, since organisations rarely allow such information to be published. However, there are several applications from the Telecommunications and Engineering domains that exhibit similar characteristics to directory systems. These involve the modelling of hierarchical and network structures with considerable object navigation. Two examples are discussed below.

#### ***4.2 The British Telecom Benchmark.***

Baker & Salman [Baker91] report on the results of some work at British Telecom to compare an RDBMS, an RDBMS with object extensions and a pure ODBMS. The database schema was designed to represent a hypothetical network consisting of **Multiplexors, Repeaters** and **PBXs**. Six operations were defined (one select, three traversals, two structural modifications) and the test databases were populated with approximately 100,000 objects.

The results showed that the pure ODBMS provided superior performance than either of the other two database systems. In fact, the results from the hybrid system were so disappointing that the benchmark developers discounted them altogether and only discussed the results of the RDBMS and ODBMS systems. Not surprisingly, the performance gap between relational and object became wider as queries became more complex. The relational database was also two and a half times the size of the object database, partly due to index tables.

To conclude, Baker & Salman attributed the superior performance of the ODBMS to two main reasons. First, the application itself was object-oriented and was therefore easier to implement on the object database, whilst the relational database incurred a cost for trying to support a paradigm it wasn't designed for. Second, since the ODBMS was suited to navigational queries, the required objects could be retrieved directly rather than the expensive "search-and-match" approach used in the relational database.

#### ***4.3 The OO1 Benchmark.***

This benchmark was developed during the late 1980s and was designed to measure interactive performance for engineering applications in object and relational database systems.

Experiments by the benchmark developers were initially conducted with three different systems, which they called:

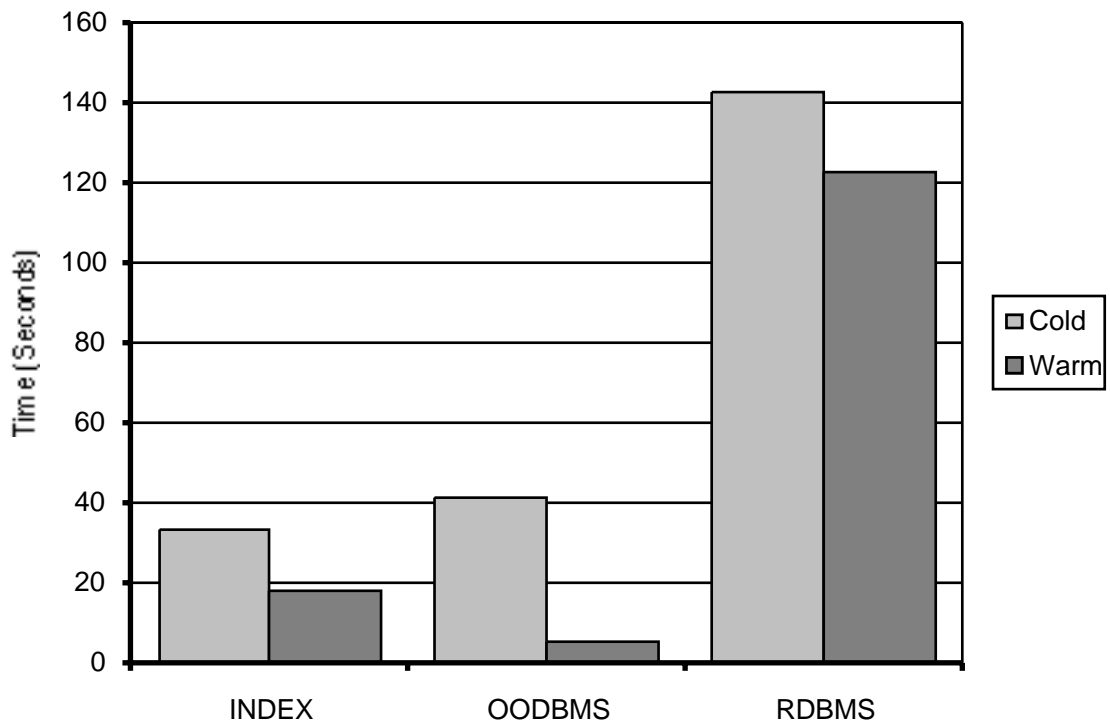
- **INDEX** – a B-Tree file package,
- **OODBMS** – a beta version of a commercial object database system,
- **RDBMS** – a commercial relational database system.

The systems were tested in a variety of client-server configurations, either running the application and database processes on the same machine or across a network and two database sizes were used.

The results showed that the ODBMS provided the best overall performance. Cattell & Skeen [Catte92] proposed that this was due to efficient access methods (e.g. parent-child links), minimised concurrency control, use of a local cache and no interprocess communication for database calls. All these were achieved without any overhead being incurred for the higher-level semantics provided by the object database system.

The RDBMS was the slowest, primarily due to its client-server architecture. For example, approximately 100 seconds were just due to the overhead of database calls across the network. Even when the DBMS process and benchmark application were running on the same machine, it was found that there was still a significant overhead, again primarily due to communications between the application and the DBMS and copying of data back and forth between database buffers and program variables [Catte91].

Figure 5 shows the results of the small database with the application and database processes running across a network (analogous to a DSA and Database backend).



**Figure 5** – Small Remote Database.

The distinction between cold and warm numbers is as follows. The first benchmark run produces the cold cache numbers, where the application cache is initially loaded with objects from the database. Subsequent iterations may access objects that are already in the cache and are, therefore, referred to as warm numbers. As mentioned earlier, caching provides a very useful mechanism to improve the performance of directory systems.

Results from other experiments showed that cost of remote file access was about one third more than local access. Additionally, using the file package, an implementation using parent-child links provided far superior performance than B-Trees on traversal operations. This is an important result, since B-Trees are used extensively in relational database systems.

To conclude, the OO1 benchmark has demonstrated that ODBMSs can provide substantial performance benefits over B-Tree and RDBMS systems. Although it was designed to represent engineering applications, directory systems also exhibit some of the same characteristics and requirements.

#### **4.4 Summary.**

The examples discussed above show the performance advantages that ODBMSs provide over other technologies. Internal benchmarks implemented by NEXOR agree with these results. It is clear, therefore, that only ODBMSs provide the necessary performance for third-generation directory systems.

### **5. Conclusions.**

A number of major conclusions follow from the discussions in previous sections. These can be summarised as follows.

First, ODBMSs provide the flexibility to model complex structures without imposing a data model. These structures can be directly stored in the database without requiring any decomposition. This also has performance implications, discussed next.

Second, ODBMSs provide the necessary performance for mission-critical directories. This is achieved by eliminating the mismatch between application and database objects and using intelligent caching.

Third, ODBMSs support all the data management and multi-user services found in existing relational technology, adding reliability and data integrity.

It is clear, therefore, that only ODBMSs provide the advantages of existing technologies, without incurring any of their limitations. This is the technology that underlies NEXOR's third-generation mission-critical directory systems.

### **References**

- [Asgar97] M. Asgarian, M.J. Carey, D.J. DeWitt, J. Gehrke, J.F. Naughton and D.N. Shah (1997) The BUCKY object-relational benchmark. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997, pp. 135-146.
- [Baker91] S. Baker and M.A. Salman (1991) Performance comparison between an object-oriented, a relational and an object management database. *Proceedings of Advanced Information Systems*, London, UK, 1991, pp. 61-67.

- [Catte91] R.G.G. Cattell (1991) An engineering database benchmark. In: *The Benchmark Handbook for Database and Transaction Processing Systems*, J. Gray (ed.) (San Mateo, California: Morgan-Kaufmann)
- [Catte92] R.G.G. Cattell and J. Skeen (1992) Object operations benchmark. *ACM Transactions on Database Systems*. 17 (1):1-31.
- [Coulo94] G. Coulouris, J. Dollimore and T. Kindberg (1994) Distributed systems: concepts and design (Wokingham, England: Addison-Wesley)
- [Loomi95] M.E.S. Loomis (1995) Object databases: the essentials (Reading, Massachusetts: Addison-Wesley)
- [Loomi98] M.E.S. Loomis and A.B. Chaudhri (eds.) (1998) Object databases in practice (Upper Saddle River, New Jersey: Prentice-Hall)  
<http://www.soi.city.ac.uk/~akmal/html.dir/book.html>
- [SURFn95] SURFnet (1995) Introducing a directory service. Final report introduction phase SURFnet X.500 pilot project. SURFnet, The Netherlands, 1995.

## About the Author

Akmal B. Chaudhri has been investigating ODBMS performance for a number of years. He has been a regular tutorial presenter at Object World, Object Technology and OOPSLA. He has contributed papers to journals, seminars and conferences and has co-edited the book “Object Databases in Practice” published by Prentice-Hall. He has also provided ODBMS consulting to several organisations, including NEXOR and Union Bank of Switzerland. He is a member of the British Computer Society Data Management and OOPS Specialist Interest Groups and holds a BSc in Computing & Information Systems, a MSc in Business Systems Analysis & Design and a PhD in Computer Science. He may be contacted by sending email to: `akmal(at)bigfoot(dot)com`.