A Systematic Performance Study of Object Database Management Systems

Akmal Bashir Chaudhri

Submitted for part of the examination for Doctor of Philosophy

The City University, London
Computer Science Department

October 1997

“And my success (in my task) can only come from Allah.”
- The Holy Koran [Surah Hud 11:88]

“They ask thee concerning The Spirit (of inspiration). Say: ‘The Spirit (cometh) by command of my Lord: of knowledge it is only a little that is communicated to you, (O men!)’”
- The Holy Koran [Surah Bani Isra-il 17:85]

Table of Contents

List of Tables

List of Figures

Acknowledgements

My sincere thanks to my supervisors, Professor Peter Osmon and Dr. Julie McCann, for their guidance, advice and constructive comments during this research work. Thanks also to Professor Norman Revell for his support during the early stages of this work.

I am also very grateful to the five organisations that provided case studies. Alphabetically, individuals in particular that I would like to thank are Charlie Knight (Treasury Dealing Project at Nomura International), Julian Onions (Messageware Directory Project at NEXOR), Howard Rice-Smith (DCx Project at Reuters), Stephen Ross-Talbot (HOODINI Project at Nomura International), Dr. George Simpson (MMIS Project at Earth Observation Sciences) and Dr. Mark Thursz (Uncle Project at St. Mary's Hospital).

Without commercial database software, the benchmark experiments in this research would not have been possible. I am, therefore, very grateful to Object Design, Objectivity and UniSQL for providing copies of their software for testing and evaluation. Also, I greatly appreciate the free technical support provided by all three vendors.

My thanks also to the Technical Support Group (TSG) within the School of Informatics (SOI) at City University for their help. Chris Marshall, in particular, for installing all the database software used in this research, patiently renewing the C++ compiler licence every month and providing useful technical insights into the system configuration within SOI. I would also like to thank Andy Lack in Computer Services for useful technical advice and assistance during the early stages of this research.

The Inter Library Loans (ILL) section within the City University Library and Peter Williams, in particular, tracked-down many rare and obscure references to object database performance. I am grateful to them for this.

Finally, my thanks to the Engineering & Physical Sciences Research Council (EPSRC) for partially funding this work.

Declaration

Abstract

Many previous performance benchmarks for Object Database Management Systems (ODBMSs) have typically used arbitrary sets of tests based on what their designers felt were the characteristics of Engineering applications. Increasingly, however, ODBMSs are being used in non-engineering domains, such as Financial Trading, Clinical Healthcare, Telecommunications Network Management, etc. Part of the reason for this is that the technology has matured over the past few years and has become a less risky choice for organisations looking for better ways to manage complex data. However, the development of suitable application- or industry-specific benchmarks, based on actual performance studies, has not paralleled this growth.

The research reported here approaches performance evaluation of ODBMSs pragmatically. It uses a combination of case studies and benchmark experiments to investigate the performance characteristics of ODBMSs for particular applications, following the successful use of this approach by Youssef [Youss93] for studying the performance of On-Line Transaction Processing (OLTP) applications for Relational Database Management Systems (RDBMSs).

Six case studies at five organisations show that organisations consider a wide range of factors when undertaking their own performance studies or benchmarks. Furthermore, none of the studied organisations considered using any public benchmarks. Six current and derived benchmarks also highlight statistically significant performance differences between three major commercial products: Objectivity/DB, ObjectStore and UniSQL. These benchmarks indicate the suitability of the products tested for particular application domains.

The research could not find any evidence at this time to support the concept of a generic or canonical performance workload for ODBMSs. This is demonstrated by the case studies and supported by the benchmark experiments. However, the research shows that performance benchmarks serve a very useful role in ODBMS evaluations and can help identify architectural and quality problems with products that would not otherwise be observed until significant application or system development was already in progress.

Abbreviations

DBMS	Database Management System
ESQL	Embedded SQL
GB	Gigabyte
Hr	Hour
Hrs	Hours
KB	Kilobyte
LOR	Locality of Reference
MB	Megabyte
N/A	Not Available
NLOR	No Locality of Reference
ODBMS	Object Database Management System
OO	Object-Oriented/Object-Orientation
OQL	Object Query Language
ORDBMS	Object-Relational Database Management System
RDBMS	Relational Database Management System
RM	Relational Model
SD	Standard Deviation
Secs	Seconds
SQL	Structured Query Language

CHAPTER 1 - Introduction

1.1 Introduction

Today, the rate of technological change is rapidly accelerating. Organisations are faced with increasingly difficult decisions about their choices for hardware and software platforms. Furthermore, increasing deregulation in many countries has forced companies into more aggressive deadlines and timescales to compete and survive. At present, object-oriented technology is viewed as being able to deliver more reliable and better quality software, because it enables the building of more modular software by using well-defined interfaces and hiding implementation details. This can be seen in Table 1.1, which shows the results of a survey of visitors to Object World UK in 1995.

Flexibility to change	40%
Reduced time to market	24%
Distributed application requirements	23%
Programmer productivity	19%
Application complexity	18%
Ease of use	9%
Financial savings	7%
Don't know	5%
Other	17%

Table 1.1 - Main Reasons for Moving to Object Technology [Leach95].

According to Leach [Leach95], the top reason shown in Table 1.1 was also number one in 1993 and 1994.

The total worldwide sales of object-oriented development tools by US companies in 1995 were valued at US$1.3 Billion [OOS96]. Figure 1.1 shows a breakdown of the various strands. Object-Oriented Languages (OOLs) is the largest, followed by Application Development Environments (ADEs) and then Object-Oriented Databases (OODBs).

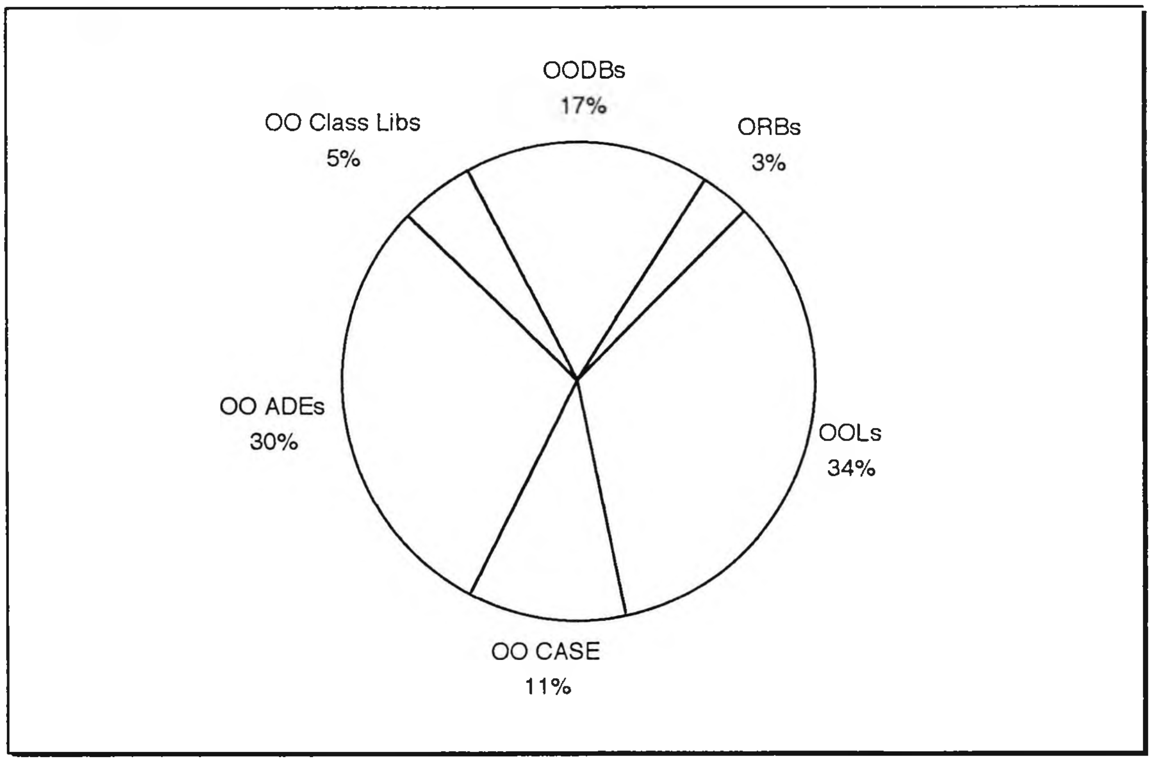

Figure 1.1 - OO Development Tools in 1995 [OOS96].

Object-oriented languages are the most mature of all the strands shown, since Simula (the precursor to languages such as C++) has been in existence since the late 1960s. Other languages, such as Smalltalk, have also been present for several decades. On the analysis and design side, current efforts are underway through the Object Management Group (OMG) to develop a standard technique to communicate object-oriented designs. This is analogous to Electrical Engineering, for example, where a standard notation can be used to communicate electrical circuit designs. Such a notation would be understood by engineers wherever they were in the world. Object databases, however, are still rarely used in production [Dick95b; Loomi96], although Loomis & Chaudhri [Loomi98] contains examples of 18 commercial implementations. The reason for this is that a database is often at the core of many business processes within a company and changing it would be difficult and expensive, as there may be many application dependencies. This view is supported by Brodie & Stonebraker [Brodi93], who discuss some of the problems in migrating legacy information systems. In contrast, applications can often be re-written in another language without affecting existing applications and system or application designs can also be produced using a different notation.

1.2 The Growth of Object Databases

Most of the major commercial object database products were developed towards the end of the 1980s and early 1990s. Many of them were built around object-oriented programming languages and attempted to provide object persistence for programming language objects. The engineering domain was an early adopter of this technology, since in the past, the approach may have been to store design objects in a relational database or to use some proprietary or flat file system. This often incurred performance and other overheads. For retrieval of design objects, for example, many database objects may have to be read and reconstructed together in memory. For storage, the design object would need to be flattened or decomposed. Some examples to illustrate these problems have been described by Loomis [Loomi95]. Recently, the technology has become increasingly popular within other industry sectors. This shift towards non-engineering applications has been supported by survey evidence reported in [Leach95] and shown in Table 1.2.

Commercial application	17%
Other application	14%
Multimedia application	12%
Document management	9%
Workflow and financial modelling	9%
Mapping and GIS	8%
CASE	7%
Network management	5%
Scientific applications	5%
Manufacturing	4%
CAD	1%
Transportation	1%
No applications will run on ODBMS	28%

Table 1.2 - Use of Object Databases [Leach95].

Additional evidence showing the shift away from engineering to more mainstream commercial usage is shown in Table 1.3, which contains a breakdown of some of the industry areas using and deploying ODBMS applications, based on a study of 24 organisations by Barry [Barry97].

The market for this technology has grown steadily over the last decade, although it still accounts for only a few per cent of total sales of database systems [Dick95b; Loomi96].

However, Table 1.2 also shows that there is a large minority that have indicated that they will not use object databases. Figure 1.2 shows the software sales of object and object-relational database vendors for 1992-93.

	1991	1992	1993	1994	1995	1996
Aerospace						1
Finance, Banking, Insurance					3	1
Manufacturing				2		2
Research & Development	1					
Transportation, Utilities, Communications			1	1	1	4
Publishing						1
Software Development		1	1		1	3

Table 1.3 - Year Deployed and Primary Business [Barry97].

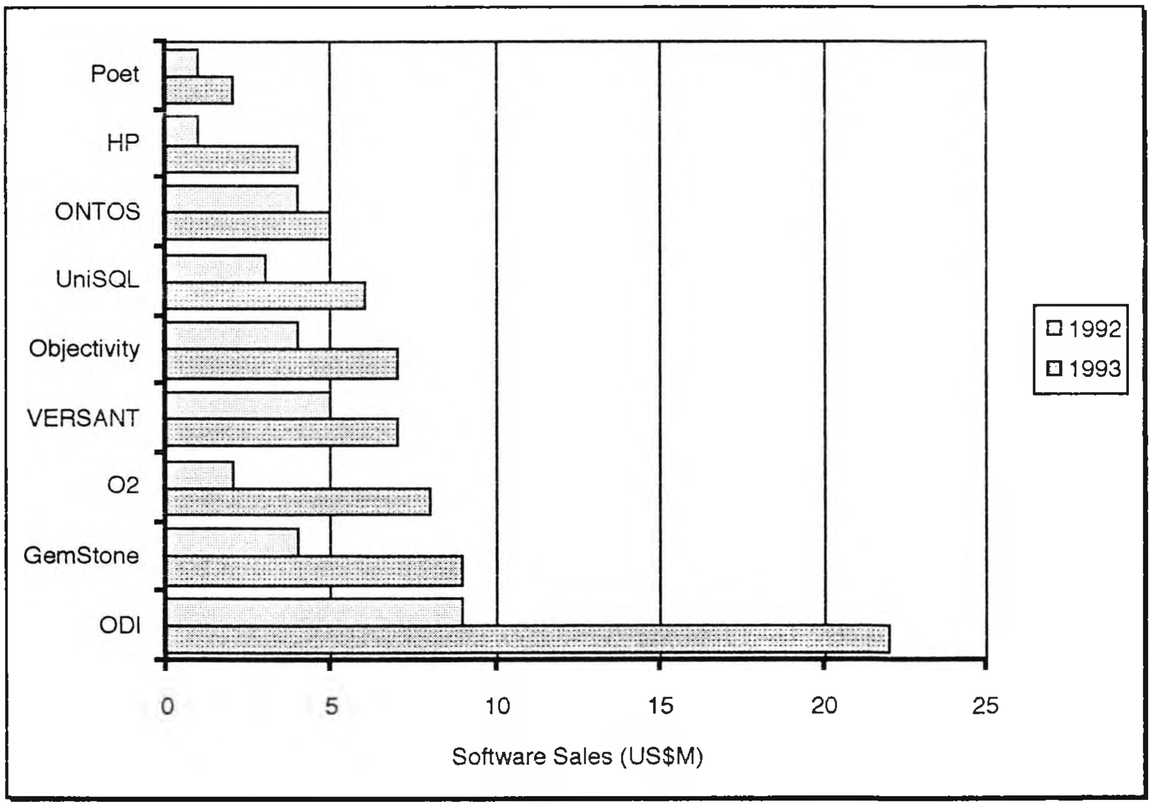

Figure 1.2 - Database Software Sales [IDC94].

Part of the reason for this slow growth is that the relational vendors have not totally ignored developments in object technology in the same way that the hierarchical and network vendors ignored developments in relational technology, several decades ago. Furthermore, many companies do not wish to move to a new data management strategy [Hodge89]. Many of the major relational vendors have announced that they will be providing support for object-oriented concepts and Informix has already released a Universal Server on the market in late 1996. This, therefore, provides a smoother upgrade path for those organisations considering object data management.

1.3 Benchmarks

Jain [Jain91] states that:

> "Performance analysis is a key step in the design and procurement of new computer systems including processors, languages, operating systems, networking architectures, or database systems."

He goes on to say that the techniques that may be used for performance evaluation include analytical modelling, simulation and measurement. Deciding which of these techniques to use is governed by a number of issues. These are illustrated in Table 1.4 and ordered from most to least important in the extreme left-hand column.

Criterion	Analytical Modelling	Simulation	Measurement
Life-Cycle Stage of System	Any	Any	Post-Prototype
Time Required for Evaluation	Small	Medium	Variable
Tools Required	Performance Analysts	Computer Languages	Instrumentation
Level of Accuracy	Low	Moderate	Variable
Varying Parameter Values	Easy	Moderate	Difficult
Cost	Small	Medium	High
Saleability of Results	Low	Medium	High

Table 1.4 - Criteria for Selecting an Evaluation Technique [Jain91].

The three techniques can also be used together to improve the accuracy of performance evaluations. For this research project, however, only measurement was used. The process of performance comparison of systems by measurements is called **benchmarking** and the workloads used are called **benchmarks** [Jain91].

The collection of papers presented in [Donga93] covers some benchmarks for measuring the performance of processors, programming languages and database systems. For example, some of the benchmarks for processors and programming languages include:

- Ackermann's Function.
- Dhrystone.
- Lawrence Livermore Loops.
- LINPACK.
- Sieve Kernel.
- SPEC Benchmark Suite.
- Whetstone Kernel.

Further details about these benchmarks can be found in [Jain91; Donga93]. The reader is also directed to [Bell84] for a discussion of database performance within the context of overall system performance. Survey's of relational database benchmarks are presented in [Inmon89; Gray91; McCan92; Youss93]. Some of the characteristics of good database benchmarks have been discussed in [Gray91; McCan92; Youss93].

Performance is often a major selection factor for Database Management Systems (DBMSs) [Chaud94a; Chaud94b]. A standard technique to evaluate database performance is to use a benchmark, since the cost of implementing a full application is often too expensive [Ander90]. Generally, a benchmark is designed to be representative of the characteristics and load of a real application or to test some specific components of a DBMS [Chaud95a]. However, performance evaluation and benchmarking can serve a number of different purposes [Jain91; Longb93], such as:

- Specifying performance requirements.
- Evaluating design alternatives.
- Comparing systems.
- System tuning.
- Bottleneck identification.
- Workload characterisation.
- Capacity planning.
- Forecasting.

Surveys of benchmarks for relational databases, such as [Inmon89; McCan92; Youss93], show that the focus of many benchmarks is towards measuring the performance of relational operations, such as Selection-Join query processing or towards application domains where relational database technology is widely used, such as On-Line Transaction Processing (OLTP). These benchmarks are generally unsuitable for measuring the performance of ODBMSs, where pointer navigation is a common feature and the technology is generally not used for OLTP [Dick95b; Loomi96]. As previously mentioned, an early adopter of object database technology was the engineering community and perhaps, not surprisingly, initial benchmarks for object databases were designed to be suggestive or representative of this type of use. However, as the technology has matured, it has gained wider acceptance and examples of its use now exist in many application domains. This was highlighted earlier in this chapter and supported by evidence from several surveys. Public benchmarks for ODBMSs that model the data manipulation characteristics of these domains, however, have not been available. Reasons for this include a perceived lack of maturity, the lack of standard interfaces among products and the general reluctance of vendors to participate in competitive benchmark studies. There are a number of reasons, therefore, why new benchmarks for object databases are needed:

1. Performance is typically among the top three selection criteria for users when deciding which object database to purchase [Rotze91].
2. The cost of implementing a complete application to test performance is expensive [Ander90].
3. A benchmark is only a valid yardstick for applications that are similar to the benchmark [Dietr92; Hallo93a].
4. Treating performance and its measurement generically is wrong and can lead to incorrect conclusions [Inmon89].

Points 3. and 4. are particularly relevant to many existing object database benchmarks, which focus on engineering applications. Trying to extrapolate the results from such benchmarks to other application domains can be highly misleading and dangerous [Dick95b; Loomi96].

1.4 Research Objectives

The previous discussion has exposed a shortfall in suitable application- or industry-specific benchmarks for object databases. It is the objective of this research project to address this deficiency since, by undertaking suitable studies, it should be possible to derive the data manipulation characteristics and requirements of particular applications and possibly generalise these to industry levels. In this respect, this research project follows the approach used successfully by Youssef [Youss93]. However, as correctly observed by Lakey et al. [Lakey87], many of the applications being developed with object databases and the object database products themselves show greater variety than traditional applications and DBMSs and the problem in constructing benchmarks is finding a small number of example applications that cover the many possible kinds of variability, in both data and operation. The question of what operations to measure with object databases has also been raised in [Josep89] and [Zorn95].

The following research objectives will be described in more detail in Chapter 4 and are presented here only briefly. These objectives are:

1. Study the performance of commercial object database applications.
2. Attempt to identify which classes of applications are more suitable for particular object database architectures.
3. Determine if a generic, simple and accurate performance model for object databases can be derived.

To achieve these objectives, this research will:

1. Take a pragmatic approach to the performance evaluation of object databases.
2. Use empirical techniques, such as case studies and laboratory experiments (benchmarks) to support 1.
3. Use appropriate statistical techniques for verification of results to support 2.

Using recognised empirical and statistical techniques will provide rigour, since many previous object database performance studies have selected an arbitrary set of operations to measure and have provided no evidence to show whether the results reported are significant. This author is not aware of any previous efforts to use this combination of techniques for evaluating the performance of object databases on the scale attempted in this research project.

The results will show that commercial organisations generally undertake performance studies or benchmarks for a variety of reasons, such as product selection or evaluating design alternatives, before implementing complete applications. Furthermore, public benchmarks are never considered, but companies always undertake their own performance tests (an observation also reported elsewhere by Chaudhri [Chaud98a; Chaud98b]). Additionally, results from a suite of benchmarks described later in this research project show that there are statistically significant performance differences between a number of major commercial products and that one benchmark alone would not have shown some of the observed differences.

1.5 Thesis Structure

This thesis is organised into seven further chapters. Each of these will now be briefly described to show the structure of this report.

Chapter 2 will begin with a discussion of the shortcomings of relational technology for managing complex and inter-related data. This will be followed by a detailed description of the major commercial object databases (GemStone, O_2, Objectivity/DB, ObjectStore and VERSANT) and object-relational databases (Illustra and UniSQL). This will aid the reader in the subsequent chapters on the case studies (Chapter 6) and benchmarks (Chapter 7), where these products (and their features and architectures) are referenced.

Chapter 3 presents a survey and critique of the most popular and well-known benchmarks related to object data management. This includes benchmarks such as OO1 [Catte92], HyperModel [Ander90] and OO7 [Carey93]. The chapter also discusses the suitability of two well-known relational benchmarks for measuring the performance of ODBMSs: the Wisconsin Benchmark and the TP1 Benchmark. The chapter concludes with a discussion of the limitations and shortcomings of current object database benchmarks. Some of the limitations have been directly observed in the case studies and benchmarks described in subsequent chapters.

Chapter 4 describes the **Research Design**. The problem statement is presented, based on the limitations of previous benchmarks discussed in Chapter 3 and the growth in the use of object data management for non-engineering applications, as discussed earlier in this chapter. The three research objectives are described in detail and alternative research techniques are discussed. It is argued that the empirical approach is relevant to this research project and that a combination of case studies and laboratory experiments (benchmarks) provide a suitable route to theory extension. Possible limitations of the research approach are also highlighted. Finally, some statistical techniques are introduced, based on their use by previous database performance researchers.

Chapter 5 presents a domain analysis to determine some of the characteristics of applications that are often cited as being appropriate for use with object databases. Whilst the framework is based on one that was used for engineering applications, it is shown that other application domains have many similar requirements. The application domains are:

Computer Integrated Manufacturing (CIM), Engineering, Financial, Geographical Information Systems (GISs), Healthcare, Scientific Data and Telecommunications.

Chapter 6 describes the results of six case studies at five organisations in the UK. The organisations include some of the most well-known companies from their respective industry-sectors and were selected based on their availability. These companies are Nomura International and Reuters (Financial), Earth Observation Sciences (GIS), St. Mary's Hospital (Healthcare) and NEXOR (X.500 Directory Systems). The case studies show a variety of factors that can influence performance and confirm some of the problems with existing benchmarks, suggested at the end of Chapter 3.

Chapter 7 presents the results of six current and derived performance benchmarks undertaken on three commercial products: Objectivity/DB, ObjectStore and UniSQL. The benchmarks include the OO1 Benchmark [Catte92], the AFIT Wargame Simulation Benchmark [Hallo93a] and the CITY Benchmark [Youss93]. The three other benchmarks include one based on data supplied by NEXOR in one of the case studies described in Chapter 6, a benchmark based on GIS data provided by one of the object-relational database vendors and, finally, a benchmark based on portfolio management systems from the financial domain. Analyses of the performance results show statistically significant differences in the three products tested.

Chapter 8 summarises the research findings, contributions of this work and presents the major conclusions. It shows that the case research approach has helped to provide insights into how object data management is being used in commercial applications. Furthermore, the benchmark experiments have shown performance differences between several major products, that were not previously reported. The research could not find any evidence at this time to support the idea of a generic or canonical performance workload for ODBMSs. It discovered, however, that it is important to undertake performance studies for particular applications. Some areas for future research in ODBMS performance are also identified.

1.6 Chapter Summary

This chapter has introduced some of the reasons for the development of object databases, briefly discussed why new object database performance benchmarks are needed and has set the scene for the remainder of this report by describing the content and structure of subsequent chapters.

CHAPTER 2 - Object Databases

2.1 Introduction

The previous chapter briefly discussed some of the reasons for the development of object databases. This chapter expands upon that discussion and looks at some of the reasons that led to the emergence of ODBMSs, such as the deficiencies in relational database technology. A number of major commercial object and object-relational database products are also described. From the descriptions, it will be clear that current products differ considerably in many areas, such as architectures, programming language interfaces, query capabilities, etc. This variability among products makes the task of developing performance benchmarks more difficult, since the number of dimensions that a benchmark developer has to consider is far greater than with previous generations of database technology. The product descriptions will also aid the reader in subsequent chapters, where some of the features discussed are again referenced.

It is assumed that the reader is already familiar with object-oriented concepts (e.g. inheritance, encapsulation, etc.) and data management concepts (e.g. concurrency, recovery, etc.). Appendix A also describes these concepts.

The remainder of this chapter is organised as follows. Section 2.2 provides a working definition of an ODBMS. Section 2.3 discusses some of the limitations of relational database technology. Section 2.4 presents a number of commercial object and object-relational database products in detail. Finally, section 2.5 contains the chapter summary.

2.2 What is an ODBMS?

ODBMSs can be considered as the convergence of a number of technologies, most notably Object-Oriented Programming Languages (OOPLs), semantic models and complex object models [Khosh90]. Additional influences have come from artificial intelligence [Jeffc89; Paton91]. Despite this convergence, there is currently no single object data model for ODBMSs (in contrast to the Relational Model for RDBMSs). The reason for this has been noted by Cattell [Catte94a]:

"There is no single object-oriented paradigm, and therefore there are a variety of object-oriented data models."

Research and commercial efforts have, therefore, resulted in various interpretations and implementations of the paradigm. As a result of this, there is considerable confusion as to exactly what constitutes an ODBMS, e.g. [Lagun89], although Kim [Kim90a] has noted that many ODBMSs share a set of core concepts.

Today, there are at least a dozen or more good definitions of ODBMSs:

- Atkinson et al. [Atkin89].
- Cattell [Catte94a].
- Committee [Commi90].
- Dittrich [Dittr86].
- Joseph et al. [Josep91].
- Khoshafian & Abnous [Khosh90].
- Kim [Kim90b].
- Lockemann [Locke92].
- ODMG [ODMG93].
- OODBTG [OODBT91].
- Ullman [Ullma87].
- Unland & Schlageter [Unlan90].
- Zdonik & Maier [Zdoni90].

Strictly speaking, the Object Data Management Reference Model [OODBT91] does not define an ODBMS, but proposes a design space for a family of models. In effect this provides an umbrella over all the current definitions, since most of the concepts suggested by the leading researchers and academics have been included in the proposed reference model. Arguably, the most appropriate definition is from the Object Database Management Group (ODMG) which has defined an ODBMS as one that transparently extends one or more existing OOPLs with DBMS capabilities [ODMG93]. An example is GemStone which is based on Smalltalk but also provides support for C++. Other products, such as Objectivity/DB and VERSANT, are based on C++ but also provide support for Smalltalk. Another group of products, such as ITASCA and O_2, store objects in a language-neutral format and provide several OOPL bindings. Throughout the remainder of this report, the ODMG definition will be used.

2.3 Why do we need ODBMSs?

"Everyone agrees that traditional relational database systems do great on business data processing, and lay an egg if you ask them to do anything else ... If you want them to store documents or CAD-related information, they just don't do very well." Michael Stonebraker, cited in [Hazza90].

Some of the reasons for the growth of ODBMSs were the inadequacies of existing database technology for managing increasingly complex data and the growth in the often-cited next-generation applications, i.e. the ubiquitous CASE, CAD, CAM, etc. These shortcomings are discussed below.

2.3.1 Lack of Expressive Power

The only data structure (aggregate or object) in the Relational Model (RM) is the table. This reduces all data to flat two-dimensional form, with relationships among tables being dynamically re-imposed at run-time. The lack of a facility for establishing relationships between tables at design time is probably the main backwards step from the structured database [White89]. Khoshafian & Abnous [Khosh90] have also noted that the "semantic eloquence of complex object composition is lost" when real-world objects are mapped to tables, resulting in the so-called "semantic gap."

Lack of expressive power could also be extended to the Data Definition Language (DDL) and Data Manipulation Language (DML). For example, SQL provides DDL and DML capabilities, but needs to be embedded in a host language for general computation. In contrast, OOPLs are seamlessly extended with DDL and DML capabilities, besides being computationally complete.

2.3.2 Simple Data Types

To support next-generation applications, a richer set of data types is required, e.g. arrays, lists, multimedia data types, etc. Some popular RDBMS vendors have already enhanced their products to support new data types. For example, OpenIngres allows the incorporation of user-defined Abstract Data Types (ADTs) into database tables. However, these enhancements are still not comparable to the facilities provided by Object-Relational DBMSs (ORDBMSs), such as Illustra and UniSQL, which provide support for inheritance, functions/methods and better Application Programming Interfaces (APIs) for OOPLs.

2.3.3 Loss of Data Protection

Most RDBMSs support simple data types, e.g. integer, char. Many programming languages, however, enable new enumerated types to be defined. When these enumerated types are mapped to the database types, data protection is lost [Hughe91].

2.3.4 "Impedance Mismatch"

> "Impedance mismatch ... the metaphor comes from the field of electrical engineering, and refers to the fact that an impedance mismatch in an electrical circuit will prevent the maximum power transfer from being achieved." [Gray92].

This is often cited as a problem with existing (relational) technology by strong advocates of ODBMSs. The issue stems from the fact that two languages are needed when using RDBMSs - one database language and one programming language. With ODBMSs, this problem disappears, since the same language is used for accessing the database as well as programming. However, it should be remembered that in RM, a database *sub-language* was originally defined to provide support for relational operations only and was never intended to be a complete programming language. There is evidence to suggest that mapping data structures between programs and the database can account for up to 30% of the code in an application [Atkin83]. ODBMSs fare little better, since in one respect, the programmer is brought down to the level of non-declarative programming again. However, this is being redressed by the efforts of ODMG, who are working towards an Object Query Language (OQL).

2.3.5 Performance

Within the literature, it is widely accepted that certain classes of applications are particularly well suited to ODBMSs. These are primarily applications involving the storage and manipulation of complex and heavily inter-related data. Traditionally, RDBMSs have been unable to provide the necessary performance for applications with complex relationships and multi-way joins, as discussed by Maier [Maier89].

By its very nature, RM "flattens" entities. Putting these entities together in a meaningful way requires joining (described as "unnatural" by Khoshafian & Abnous [Khosh90]) and sorting. These operations compete for being the slowest in relational systems [Loomi92]. In addition, commercial language optimisers are often unable to cope efficiently with queries involving large numbers of joins [Gray92], which has resulted in static binding strategies being developed by some relational database vendors.

In many ODBMSs, the equivalents of relational joins are pre-computed by explicit pointers. Additionally, since ODBMSs can store methods, this incremental knowledge about the operations being performed provides the ability to better control the concurrent execution of transactions and, hence, provide improved performance. This is possible since, as [Dawso89] has noted, the operations are not simply reads or writes, but have more semantics, as illustrated by the following example.

For a queue data type, it is possible to have operators such as **enque** and **deque**. From one point of view these are write and read operations respectively. However, when considering the special semantics of these operators, a higher degree of concurrency can be achieved. Consider a queue object Q and two transactions, T1 and T2. If T1 performs an enque on Q, then T2 is prevented from performing a deque until T1 has committed by common read/write semantics. However, for non-empty queues, it is possible to execute both operations without conflict, since they do not affect each other's results.

Atwood [Atwoo92] has also noted that in an ODBMS, the operations that objects of a given type are capable of performing are known to the DBMS. The DBMS can therefore exploit this knowledge to optimise performance. Performance tools could be built to monitor reference trace patterns for frequently performed operations. Anticipatory pre-fetching is then possible for objects that an application is about to reference. Currently, however, no commercial ODBMS product uses any of the special semantics just discussed.

In contrast, according to Ketabchi et al. [Ketab90], in an RDBMS, the set of operations is limited and fixed (operations on the sets of tuples and the two operations **select** and **project**). Therefore, any operation that an application wishes to perform must be mapped onto this limited set. As applications become complex, so does this mapping, leading to large application programs. To retrieve an application-specific object, several DBMS operations are required. This makes applications slow.

2.4 Review of Commercial Products

In this section, a number of major ODBMS and ORDBMS products will be described in detail. The term *major* is used based on published sales figures and market share from IDC [IDC94].

Five pure ODBMSs are described first. A number of reports, e.g. [McFar93; Potte94; Haber95], were used as the basis for the evaluations. Many of the pure ODBMS products follow the definition of an object database, as defined by ODMG, i.e. they tend to extend one or more OOPL with DBMS services. This is fundamentally different from the object-relational database products, which are discussed later.

The pure ODBMS products described are GemStone, O_2, Objectivity/DB, ObjectStore and VERSANT. All of these products appear in the case studies described in Chapter 6 and both Objectivity/DB and ObjectStore are also benchmarked in Chapter 7.

2.4.1 GemStone

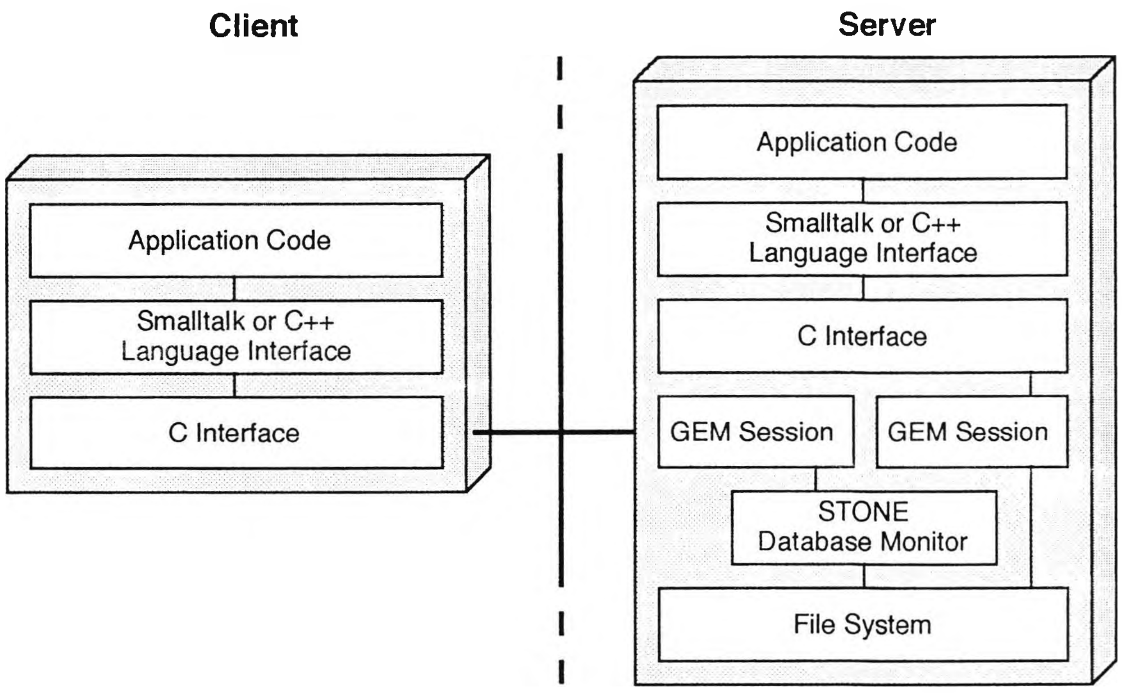

Figure 2.1 - GemStone Architecture.

GemStone [Bretl89; Maier90], along with Vbase [Andre90], was one of the first ODBMSs commercially available. GemStone is based on the Smalltalk object-oriented language with extensions to support DBMS services. It has a client-server architecture (Figure 2.1) and can be described as using a page-server [DeWit90]. It can also be categorised as an active[1] ODBMS with the ability to execute methods on the server. This feature can be very useful in querying large collections of objects on the server and transferring only the results back to the client, rather than shipping large numbers of objects from the server to the client to perform the query processing.

[1] The term "active" when applied to databases can mean different things [Talbo98], so in this research project its meaning is the ability of an ODBMS to execute methods or queries on the server.

The two major components of GemStone are the GEM and STONE processes. GEM handles object access and execution, database file I/O and caching and may execute either on the client or the server. Deciding whether to link a GEM with an application to run on the client or whether to run the GEM on the server and use a Remote Procedure Call (RPC) mechanism to communicate with an application running on the client has performance implications, as discussed by Skiadelli [Skiad94]. There is one GEM per logged-on user (but a user can have more than one GEM). STONE is the database monitor and manages concurrency control, transactions, recovery, etc. There is one STONE per database.

Database definition and manipulation are through a language called Smalltalk DB, which is used to define the active behaviour that can be executed on the server. Objects defined in this way can also be invoked from other object-oriented languages, such as C++. Because Smalltalk only supports single-inheritance, however, the C++ language binding is also restricted to this. The GemStone Object Development Environment (GeODE), was also developed, enabling users to design applications using drag-and-drop techniques with blocks of pre-defined code that could be linked to produce applications quickly (Figure 2.2). However, GemStone Systems have now ceased further development of this.

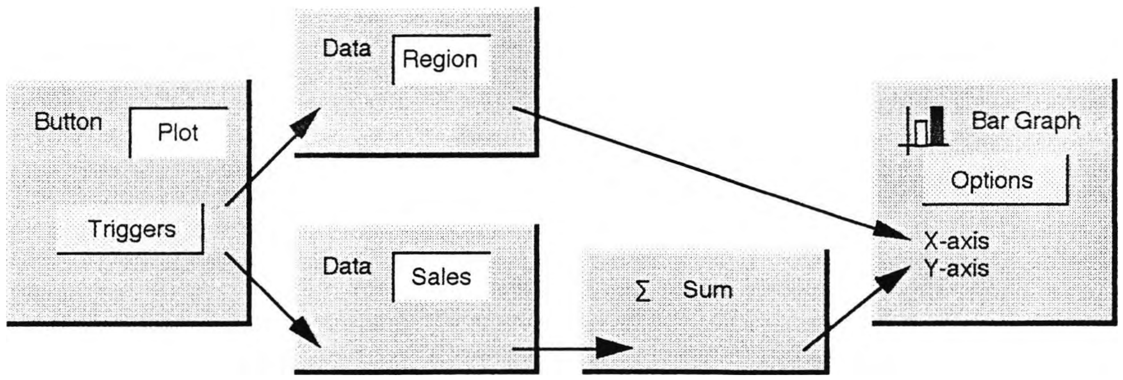

Figure 2.2 - Example GeODE Program.

Besides using the message sending paradigm, associative access queries are also possible by using what are termed "selection blocks", delimited by { and }, for fast path traversal and comparison using the '.' notation. This approach, however, violates encapsulation. For example, the following query would find 40-year old employees:

```
MyEmployee select: {:anEmpVar | anEmpVar.age = 40}
```

GemStone uses the *persistence independent of type* model - to make a new object persistent in an application, a message is sent to a class to create a new object, which is then automatically made persistent when a transaction commits. It also uses garbage collection and will only delete an object when all references to it are deleted. This is in contrast to some other ODBMSs, where explicit memory management is required. Objects can also be clustered to improve performance. Clustering can be assigned at anytime in an object's life (not only when it is created). Furthermore, objects can be clustered with other connected objects (e.g. an attribute of an object may itself be an object) and even heterogeneous collections of objects can be clustered together.

GemStone supports both optimistic and pessimistic concurrency control. In the latter, however, explicit locks must be acquired by an application. The work reported in [Emmer93], demonstrates that the optimistic approach can have a greater overhead, since more time is required to determine conflict resolution, etc. However, this approach may be more appropriate for design and engineering environments, whilst the pessimistic approach would be more suitable for traditional business applications. Otis [Otis96] adds the following about optimistic versus pessimistic locking. Firstly, the optimistic approach, whilst using more CPU than pessimistic approaches, is a practical solution even for traditional applications given the ratios between CPU performance and disk speeds on modern machines. Secondly, in the optimistic approach, read-only queries do not have to acquire any locks and this leads to less contention between updating and read-only transactions. Also, since there is no need to wait for a lock to be released by another transaction that may be holding it, no deadlock detection is necessary. Currently, GemStone supports Read, Write and Exclusive locks. Locks are held across transaction boundaries and must be explicitly released. Locking is possible for individual objects or collections of heterogeneous objects. However, GemStone does not support locking, copying or manipulating a collection of *interrelated* (composite) objects.

Authorisation is supported in GemStone by segments. An object is assigned to a segment, which holds information about the operations that users are permitted to perform on that segment. This authorisation is orthogonal to both the location of an object in the database and concurrency control.

In terms of relationships, GemStone can support 1:1 uni-directional. Other cardinalities, such as 1:N and N:M are also possible by using sets and arrays. However, a bi-directional relationship must be modelled as two uni-directional ones and must be explicitly maintained.

Location transparency is also supported, so that objects can be moved freely without users having to make any code changes. Furthermore, database replicates (a GemStone database consists of a number of files, called extents and a replicate is an on-line copy of an extent) are also supported - these can be used to improve query performance or for recovery purposes.

Object versioning and work-group support (useful for engineering environments) are very limited in GemStone. However, schema evolution and schema versioning are provided, with support varying from one language interface to another. Runtime schema access, definition and modification are also possible.

In conclusion, Dick & Chaudhri [Dick95b] propose that GemStone has the following major strengths:

- **Smalltalk Engine and Library** - the tight integration of GemStone and Smalltalk is, without doubt, one of its major features.
- **Application Partitioning** - the flexibility to configure where a GEM process runs enables better utilisation of available processing power.
- **Large-Scale, Business-Critical Support** - GemStone was designed to support MIS and business applications with large quantities of data [Butte91b].

2.4.2 O_2

The O_2 ODBMS was originally developed as a research prototype and was subsequently released as a commercial product. Originally, it was based on an object-server architecture, but because cache inconsistencies were discovered and reported in [DeWit90], the architecture was subsequently changed to a page-server. O_2 can be classed as an active ODBMS, since indexed access for large collections is possible on the server [Manol94]. Furthermore, the OQL also executes on the server. The major architectural components, illustrated in Figure 2.3, are:

- **Type Manager** - creates, stores and maintains type descriptions and applications.
- **Method Manager** - creates, stores and maintains methods.
- **Object Manager** - handles message passing, garbage collection, indexes, clustering.
- **Transaction Manager** - provides concurrency control, recovery, etc.
- **I/O Manager** - handles database file I/O.

Object persistence is achieved by *persistence through reachability* - a persistent root is defined and objects that are directly or indirectly connected to this root also become persistent. This approach has the advantage that objects can be created without needing to specify at creation time whether they are to be persistent. As with GemStone, garbage collection is used to clean-up objects that are no longer referenced.

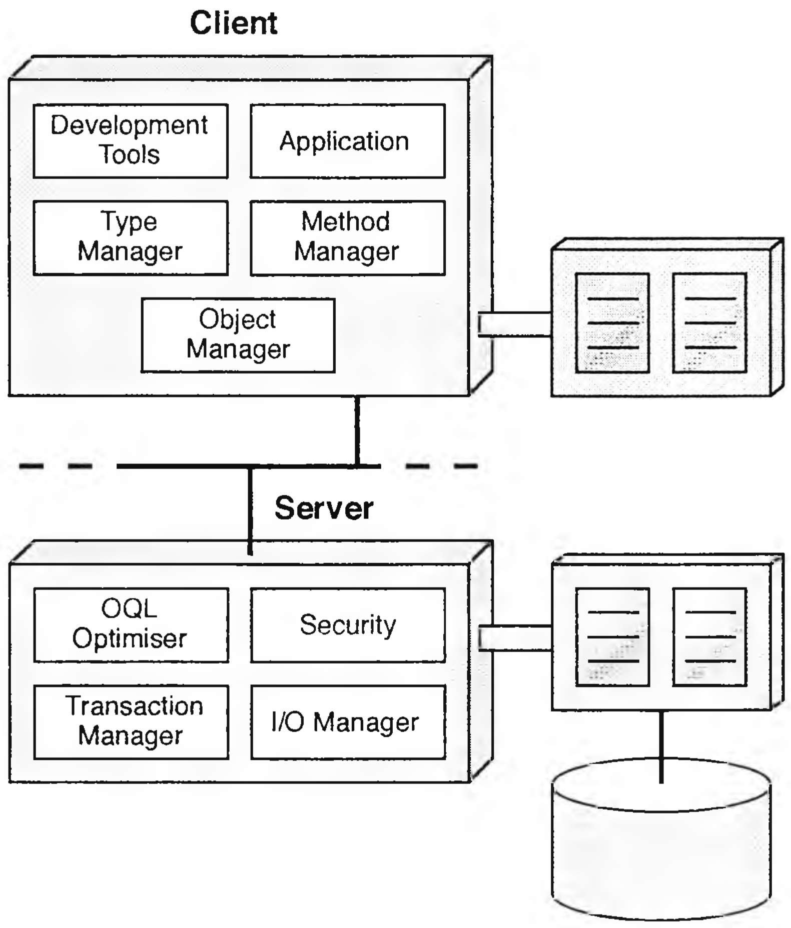

Figure 2.3 - O_2 Architecture.

Language bindings include C++ and Smalltalk, both of which are ODMG-compliant [Coope96]. Furthermore, O_2 is language-neutral and C++, O_2C and Smalltalk applications can all access the same objects and execute methods, etc. [Coope96]. Access to RDBMSs across LANs or WANs is provided by O_2DBAccess.

Nested transactions are not supported by O_2, since it is based on a storage system that implements only conventional transactions. Pessimistic concurrency control is provided and transaction rollbacks/recovery are supported by a write-ahead (redo) log. In addition to inheritance, 1:N and M:N relationships, both uni- and bi-directional, are supported [Coope96].

Additional features include object versioning and schema evolution. There is also a meta-schema that allows applications to interrogate the content and form of a database [Coope96]. Another recent development has introduced a capability to integrate O_2 with an

Object Request Broker (ORB) which enables O_2C schema definitions to be translated automatically to the Interface Definition Language (IDL).

To conclude, three major strengths of O_2 are:

- **Proven Storage Technology** - O_2 uses the Wisconsin Storage System (WiSS) to manage persistent objects.
- **Tools Support** - various graphical tools allow applications to be quickly prototyped.
- **Theoretical Basis** - considerable research has been undertaken to develop a more rigorous and formal object model for O_2, as reported in [Banci91].

2.4.3 Objectivity/DB

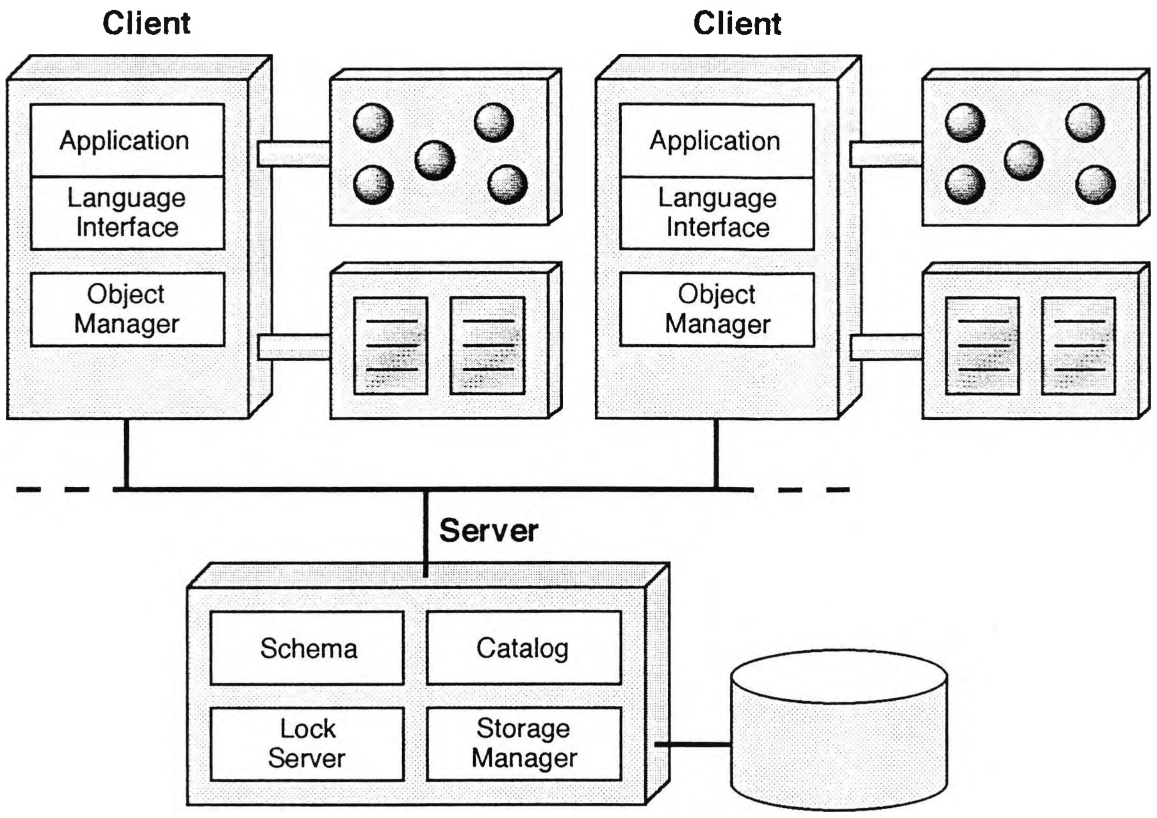

Figure 2.4 - Objectivity/DB Architecture.

Objectivity/DB is based on a file-server architecture [DeWit90] and some of the database components are illustrated in Figure 2.4. However, [Wade96] comments that the server can use either NFS or Objectivity's Advanced Multi-threaded Server (AMS). The original product was designed as a multi-language, multi-operating system, multi-network, distributed ODBMS [Wade96]. Although the first release of the product in April 1990 supported only C and C++, support for Smalltalk has also been provided recently, in-line with the ODMG standards effort for multiple-language bindings. ANSI SQL with extensions is also supported, with queries against any objects possible. ODBC support allows the wealth of ODBC-compliant tools to be used directly and immediately. All language interfaces share the same objects and the same databases. Furthermore, objects may be created or modified through any of these interfaces simultaneously [Wade96]. Persistence-capability is achieved by inheritance - a pre-supplied class (ooObj) provides persistence behaviour and objects that need to be made persistent must be members of classes that inherit from this class.

Each object has a structured Object Identifier (OID), since Objectivity/DB uses a hierarchical storage model which at the top level is viewed as a single logical database. This logical database is composed of multiple distributed databases, which are themselves composed of a number of containers. Containers contain pages and objects reside on pages. The unit of locking is the container, which may hold individual "hot-spot" objects or thousands of objects and this granularity can be changed dynamically [Wade96]. Moving an object, however, requires the allocation of a new OID and updating of all references, although this is transparent to the user and is a trade-off between purely logical OIDs that are slow, since hashing is required on access and relationships become essentially the same as an RDBMS indexed join [Wade96].

Objectivity/DB uses non-persistent *handles* (ooHandle) or *references* (ooRef) to automatically manage pointers to persistent objects and map them to virtual memory addresses, providing a level of indirection and helping to improve protection from wild pointers for database pages that have been mapped to an application's address space. Handles also pin objects in the client cache which makes access to multiple fields in the same object instance more efficient [Dick95a]. Although adding a level of indirection adds some slight overhead, this is likely to be offset by other database and application operations, such as disk accesses [McFar93]. Furthermore, the indirection enables the object manager to swap objects, which is necessary for scalability [Wade96]. Products that give direct pointers to an application give up the ability to swap objects during a transaction and are reduced to the virtual memory-mapped approach with the associated problems of thrashing, concurrency control and hard limit to swap space size [Wade96].

Support for relationships include 1:1, 1:N and M:N. Both uni- and bi-directional associations are supported, although in the former, referential integrity will not be automatically guaranteed and must be explicitly programmed in an application. Various performance-enhancing options are also available when using associations. For example, a *short association* provides the ability to store objects close to the object containing the association, e.g. a department object may contain references to all its employee objects, but this is at the cost of location transparency. Another option is to use *inlines*, which enables connections to other objects to be stored as part of an object, providing better performance and resulting in less storage overhead.

The transaction model supported is a pessimistic (locking) one, using Read, Write and Update locks and additionally, Multiple Readers, One Writer (MROW) is also provided with readers being notified if data have changed since last read. Normally, locks are implicitly provided in read mode when objects are accessed from within a transaction, but may also be explicitly acquired. Nested transactions are not currently supported and an application may only have a single transaction active at a time, although multiple top-level transactions (one per thread) are supported in Smalltalk and in C++ [Wade96]. Deadlock detection is provided and various options can be programmed to specify what should occur when there is a lock conflict. Objectivity/DB uses the standard two-phase commit protocol and provides database commit (saving objects to disk and freeing resources) or commit-with-hold (acting as a checkpoint and holding resources) as well as transaction abort.

Composite object support is possible through two mechanisms. Firstly, by making a class as the type of an attribute of another class and secondly, through associations. As mentioned earlier, associations of various cardinalities are supported and operations are available that can treat a composite object as a single unit, e.g. for deletion or locking. Versioning is also available and some basic operations are provided to support application-defined version control. Both linear- and branch-versioning are available and the object handle is used to keep track of the version status of an object. Rather than keeping the delta (differences between one versioned object and another), Objectivity/DB stores each version as a separate object and users can simultaneously access multiple versions and configurations as desired [Wade96]. When versioning a composite object, only the changed objects within the composite are versioned and any associations are automatically managed in this process [Wade96]. Work-group support is also provided through check-out and check-in and persistent (long duration) locks.

Automatic schema evolution is supported and most changes are handled without user code [Wade96]. More complicated changes may require users to add some methods. Dynamic schema changes and instance upgrades are supported without requiring application stoppage or database shutdown [Wade96].

To conclude, Objectivity/DB has the following major strengths [Dick95b]:

- **High Availability** - the product is used in many industries where this is a major requirement, e.g. Telecommunications, Manufacturing, Process Control.
- **Distribution** - the architecture of Objectivity/DB was originally designed with this very much in mind.
- **Heterogeneity** - the product is available on all major UNIX platforms, VMS and Windows NT [Dick95a]. Objects created on one platform are automatically mapped to another.

2.4.4 ObjectStore

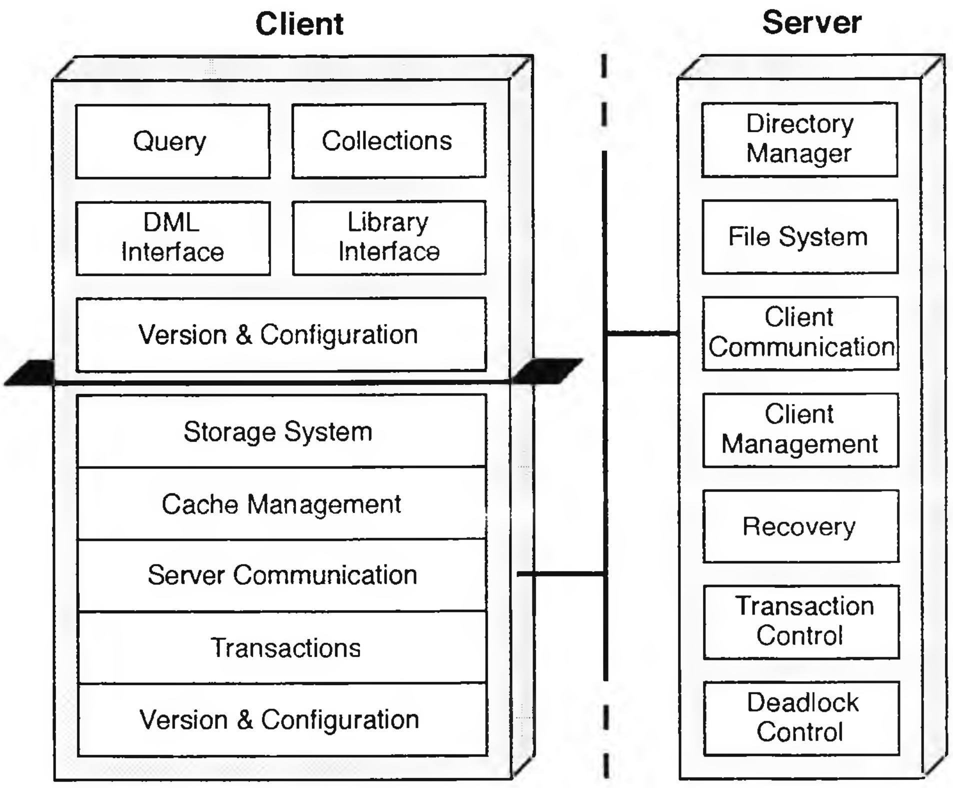

Figure 2.5 - ObjectStore Architecture.

ObjectStore, like GemStone, is based on a page-server architecture [DeWit90] and major elements of the client-server partitioning are illustrated in Figure 2.5. On the client-side, only elements above the black line are user-visible. Unlike GemStone, however, it is not classed as an active object database, although the client and server processes could be configured so that it effectively behaves as one.

ObjectStore uses the *persistence orthogonal to type* approach which allows any object to be made persistent by overloading the **new** operator in C++. SQL and ODBC support are also provided through the OpenAccess product. A relational gateway, called DBConnect, is also available.

ObjectStore is unique among commercial ODBMSs in using a Virtual Memory Mapping Architecture (VMMA) based on virtual memory support provided by the underlying operating system. All objects are referenced by virtual memory pointers and, therefore, both transient and persistent objects are treated in the same way. One drawback with the direct memory reference model, however, is that physical database design is more tightly coupled to overall system design [Alfre94]. Furthermore, ObjectStore supports extensive tuning options, resulting in many different versions of even simple benchmarks [Hallo93a].

Performance improvements can be achieved though the use of clustering and indexing. Indexes are supported on paths, which can be viewed as pre-computed joins [Lamb91]. Objects that are likely to be referenced together can be placed in clusters (fixed-size containers) which reside in segments (expandable storage containers). Clustering hints can be defined when an object is created. By default, if a virtual memory page fault is generated when an application accesses a persistent object, the client traps this and reserves address space for the entire segment that contains the object. The client then sends a request to the server for this segment. However, this could be too large for the client cache and so users can specify that only a certain number of bytes are retrieved. Alternative data fetch policies include page-mode or stream-mode. The latter being useful for applications that do not need to access all the data immediately.

Access to object attributes is possible through the normal message-sending paradigm, but a proprietary DML is also provided that supports associative retrieval, e.g.

```
set<Person*> teenagers = people[: age >= 13 && age <= 19 :];
```

which creates a collection of teenagers (people aged between 13 and 19). This type of query is guaranteed to be type-safe and is optimised at compile-time.

Relationships of 1:1, 1:N and M:N are supported, both uni- and bi-directional. Three mechanisms are available to support these associations. Firstly, by using pointers for uni-directional without referential integrity - these are similar to pointers in C++. Secondly, by macros, which enforce referential integrity for bi-directional associations. Thirdly, by a class (os_reference) which has a storage overhead when compared to the other two techniques, although it reduces the overhead of virtual memory [McFar93].

For non-workgroup applications, multiple concurrent users are supported by the traditional lock-based approach. Locking is on a Read, Write or Update basis, with support for Multi-Version Concurrency Control (MVCC) with multiple readers and one writer also available (which eliminates lock waiting). According to [Wade96], MVCC is at database- and session-level granularity. The MVCC reader may never update objects in the database they've opened through MVCC. Each MVCC user sees a different snapshot, each of which is stored on the client, causing large portions of the database to be moved to each such client and managed separately. The result is that MVCC can be useful for a process that wants to read and doesn't want to interfere with writers, but is not suitable for environments with many readers and writers performing multiple transactions. This restriction is not just a performance one - MVCC cannot support multiple transactions, i.e. some read, some write, etc., from the same users [Wade96]. The access mode must be explicitly specified when commencing a transaction, but locks (at the page-level) are implicitly acquired. Locks are held on the client, so that if another client requests an object, the server must do a "call back" to the holding client - this extra network call may be expensive in certain circumstances, depending on database access patterns [Dick96]. Nested transactions are also supported. For workgroup applications, an optimistic approach is available since readers can be allowed to access one version of a configuration (described below), whilst allowing a writer to modify another version. This applies to any ODBMS that supports versioning [Wade96]. In ObjectStore, however, an application may access only one configuration and versioning is at the physical level of pages and segments [Wade96].

Composite object support is provided by making a class the type of an attribute of another class. Composite objects can be treated as one unit for the purposes of locking or deletion. Versioning and work-group support are also provided through configurations and workspaces and are directly related to composite objects. A configuration can be nested and is a unit of locking that is composed of objects from one or more classes. Workspaces are used for access control to configurations. Typically, a configuration will be checked-out of a global workspace into a local one and then modified. When it is checked-in, a new version of the configuration will be created. Both linear- and branch-versioning are supported.

ObjectStore provides good support for schema evolution. An eager evolution strategy is used by invoking an evolve function from an application. Changes that can be made include adding and deleting attributes and classes, changing the type of an attribute and changing inheritance relationships. There is also extensive support for runtime access to schema information.

To conclude, ObjectStore has the following major strengths [Dick95b; Dick96]:

- **Compatibility with Other Tools** - the unique approach to persistence used by ObjectStore enables libraries and third-party tools to be used without modifications.
- **Retrieval Performance** - read performance is rated very highly when the data being accessed fit entirely within memory (otherwise thrashing occurs and performance degrades [Wade96]).
- **Scalability** - using virtual memory pointers, large databases can be easily created. Wade [Wade96] argues, however, that this is inappropriate, since once the data set gets larger than real memory, thrashing slows it down exponentially and there is a hard limit (swap space size) in data accessible per transaction.

2.4.5 VERSANT

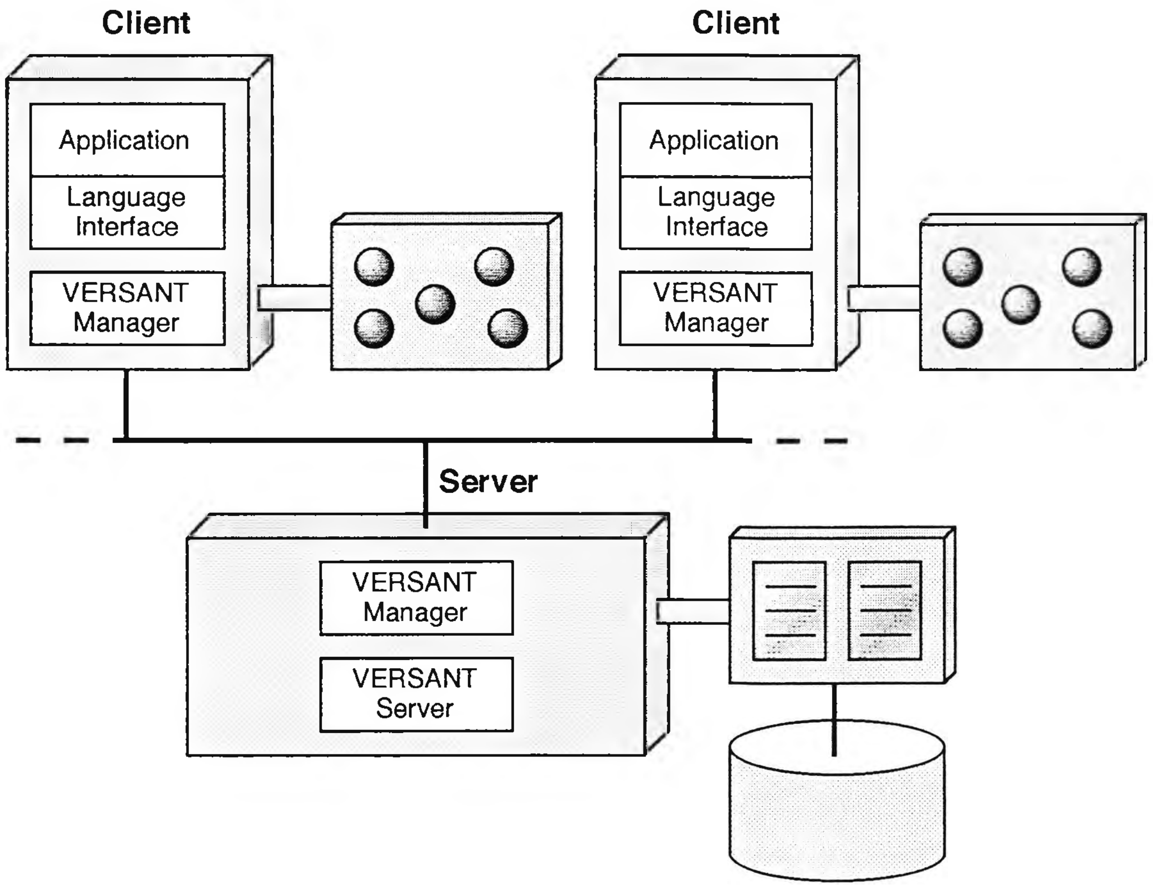

Figure 2.6 - VERSANT Architecture.

VERSANT ODBMS is an example of an object-server architecture [DeWit90], but is passive in that method execution takes place on the client and only some system-defined methods can be executed on the server. However, VERSANT does support query execution on the server, which has the benefit that large collections of objects can be queried on the server and only the results transmitted over the network to the client (as per GemStone).

The two main components illustrated in Figure 2.6 are the VERSANT Manager and VERSANT Server. VERSANT Manager executes partly on the client and partly on the server and handles object caching and validation, object query support, long transactions, schema management, object links, versions and check-out/check-in. VERSANT Server handles object retrieval, object updates, page caching, storage class management, index support, query support, short transactions and logging/recovery. Databases can be stored as files or as raw disk partitions. Roll-forward recovery is supported.

VERSANT supports language bindings for a number of languages including C++ and Smalltalk. In C++, persistent objects are created by sending a **new persistent** message to a persistent-capable class. Clustering of objects is also on a class-basis, allowing objects of different classes to be clustered together. For example, all employees of a particular department could be clustered with the department. This type of clustering is an approach similar to that provided by RDBMSs.

Traditional pessimistic locking is available as well as optimistic concurrency control with timestamping. Lock modes include Read, Write, Update and Null. Nested transactions are also supported. Locking is at the level of an object, in contrast to other ODBMSs, such as ObjectStore which locks at the page-level and Objectivity/DB at the container-level. Locking at the object-level is more suitable for applications that require fine-grained control of objects, such as CAD applications, as reported by Kempe et al. [Kempe95a; Kempe95b]. Wade [Wade96] argues, however, that CAD applications would never use object-locking, since they tend to use groups of related objects together and object-level locking would cause performance problems. He further adds that object locks could be useful for holding a lock for a long time on an object, but that other ODBMSs could provide the equivalent functionality with versioning of objects. Two types of locks are supported - short locks and long (persistent) locks. The latter being used for check-out/check-in of objects into personal databases and survive short transactions, application executions and system crashes [McFar93]. Locks are implicitly acquired at the start of a transaction and commit-and-hold is also supported.

Uni-directional relationships (1:1, 1:N) and bi-directional relationships (1:1, N:M) are supported with referential integrity being guaranteed for the latter. Composite objects are also supported in a manner similar to ObjectStore and Objectivity/DB. Deletion and locking of entire composite objects is also possible.

Location transparency is guaranteed, since VERSANT uses logical OIDs - objects can be easily moved without the need to update any relationships. Versioning (both linear- and branch-) is also supported with extensive methods available to manipulate and modify versioned objects. VERSANT uses a generic object to maintain information about versioned objects including the derivation graph, latest version, etc.

Query capability in VERSANT is supported by an Object SQL, based on work undertaken at Texas Instruments. Path queries are supported as well as regular expression support for string queries. SQL support is also available through a middleware product.

Schema changes include support for renaming classes and attributes, adding or deleting attributes, adding or deleting leaf classes and adding or deleting ancestor classes of a class. A lazy schema migration strategy is used. Extensive support is also provided for dynamic (runtime) schema changes and modifications.

Dick & Chaudhri [Dick95b] conclude that VERSANT has the following major strengths:

- **Object-Level Granularity** - for some applications that require fine-grained access to objects, this would be more suitable than page- or container-based approaches.
- **Balanced Client-Server** - both the client and server can undertake some processing, although VERSANT does not have an active object-server.
- **Transparent Distribution and Administration** - the use of logical OIDs enables objects to be easily moved across a distributed database (for databases and servers that were known at compile-time [Wade96]).

This concludes the descriptions of the pure ODBMS products. The next two products are object-relational databases. Currently, there is no standard definition for an ORDBMS, although the design philosophy is to build upon the tried and tested techniques used in relational database technology, such as good DBMS services, whilst providing support for richer data types and more complex objects.

The products described are Illustra and UniSQL. One of the benchmarks described in Chapter 7 is based on a customer benchmark used by Illustra. UniSQL is one of the three products tested with a suite of benchmarks in Chapter 7.

2.4.6 Illustra

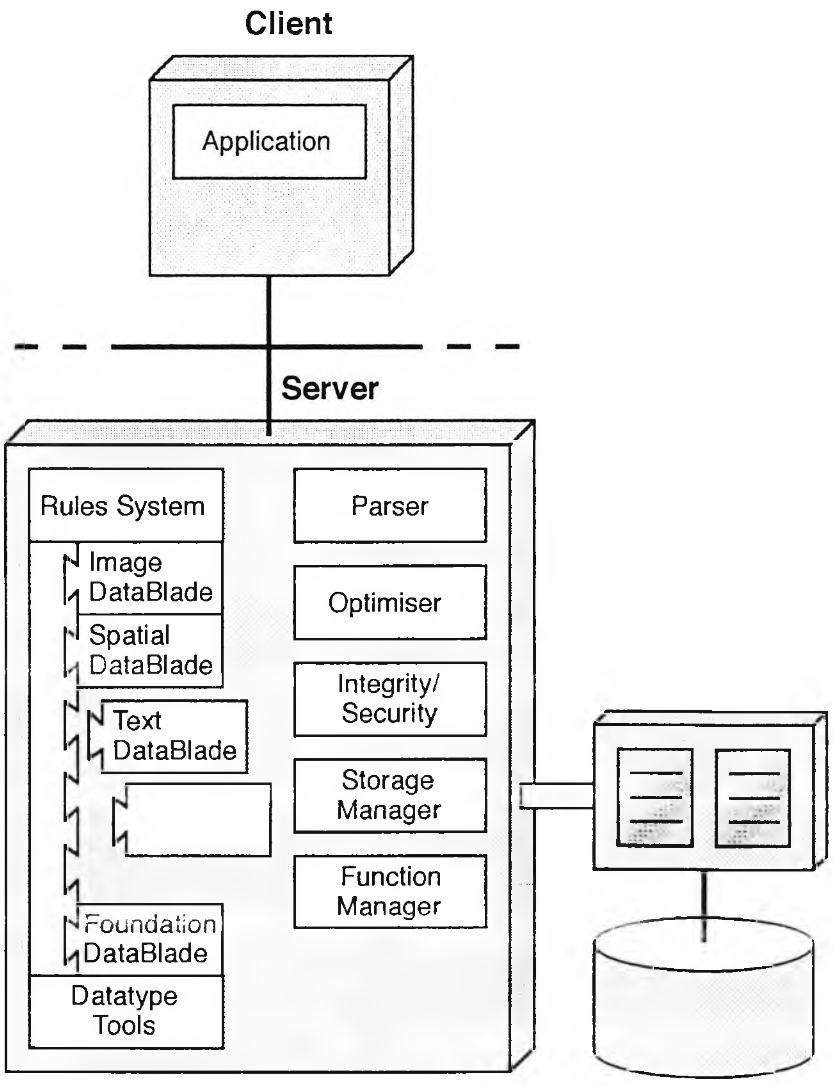

Figure 2.7 - Illustra Architecture.

One of the longest-running research prototypes, investigating the feasibility of extending relational databases with object concepts, is POSTGRES [Stone90b; Stone91]. Specifically, the aims of POSTGRES were, in order of priority, [Hales89]:

- Provide better support for complex objects.
- Provide user extensibility for data types, operators and access methods.
- Provide facilities for active databases (i.e. alerters and triggers) and inferencing including forward- and backward-chaining.
- Simplify the DBMS code for crash recovery.

- Produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors and custom-designed VLSI chips.
- Make as few changes as possible (preferably none) to RM.

The aim being to keep all the features already familiar to users of commercial DBMS systems, such as transactions, data loading, tracing, tuning, etc. and to make them work well with ideas from OO [Brown96].

The commercial realisation of this was (after several name changes) called Illustra [Stone93a; Stone96] and the architecture is illustrated in Figure 2.7. It is very much server-oriented, although functions can be executed on either the client or the server. SQL is optimised and executed on the server. According to Brown [Brown96], while the queries are optimised centrally, the execution may be distributed between the client and server processes. However, he adds, there are performance penalties to be paid in this approach, since the data need to be moved from the server to the client for client-side evaluation and the results of these functions may need to be shipped back to the server to participate in other nodes of the query. Currently, the decisions of where to execute a particular function is left entirely to the discretion of the application programmer [Brown96]. Ideally, this decision should just be another aspect of the system's overall optimisation strategy [Brown96].

Illustra uses the familiar rows and columns paradigm as with RDBMSs, but OO features include multiple data types, extensibility, multiple inheritance, polymorphism and encapsulation. Each record in the database also has an OID, which enables navigational access to be used but, undermines the mathematical foundation of relational technology [Taylo92]. Brown [Brown96] adds, however, that the OID used by Illustra is value-based, not reference-based and is simply another candidate key. Extensibility is provided by plug-in modules that extend database capabilities with new data types, functions and access methods. The storage manager can also be extended. For example, [Brown96] describes the time series module that allows the creation of "virtual relations" that break the tuple→page→buffer→disk paradigm. Using the time series module, tuples of a relation with constrained cardinality, i.e. every tuple is the same length, are loaded into a contiguous bit stream. This enables code to access the data very quickly by calculating the seek() offsets and jumping around on disk [Brown96]. Although this may not have wide utility, it seems useful for some data types.

The Illustra general purpose rules system can be used to build and modify policies and procedures. DBMS crash recovery has been simplified by using no-overwrite storage. In other words, no data are ever deleted, which means that *time-travel* queries are possible on past data and recovery will be instantaneous. A process, called the vacuum cleaner, periodically moves old data from primary to secondary storage. However, the no-overwrite storage is unlikely to appear in the Universal Server, which is the merged Informix and Illustra product [Brown96].

Language interfaces include SQL (with some SQL3 extensions), C++ and C. Functions written in an external language, such as C, can be registered with Illustra. Such functions have input and output parameters defined, are equivalent to methods in ODBMSs and can be called from SQL queries. For example [Brown96]:

```
CREATE TABLE Farms OF NEW TYPE Farm_Type (
     Farmer_Name      Name        NOT NULL PRIMARY KEY,
     Boundary         Poly        NOT NULL,
     Farm_House       Pnt );

CREATE TABLE Rivers OF NEW TYPE River_Type (
     River_Name       text        NOT NULL PRIMARY KEY,
     Watershed        Poly        NOT NULL )
UNDER Geo_Features;
```

These two extended SQL statements show the use of spatial data types (Poly, Pnt) and inheritance (UNDER).

```
SELECT F.Farmer_Name
FROM Farms F, Rivers R
WHERE Contains(R.Watershed,F.Boundary);

SELECT R.River_Name, COUNT(*)
FROM Farms F, Rivers R
WHERE Contains(R.Watershed,F.Boundary)
GROUP BY R.River_Name;
```

These two SQL queries demonstrate the use of a function (Contains) which takes two parameters of type Poly and returns a Boolean result.

In conclusion, Illustra has the following three major strengths:

- **Based on Relational Technology** - the familiar rows and columns found in RDBMSs are still used.
- **Extensibility** - various modules can be added to provide additional functionality for specific domains or vertical markets.

- **Flexibility** - both declarative and navigational interaction styles are supported and additionally, Illustra is not tightly coupled to any particular programming language.

2.4.7 UniSQL

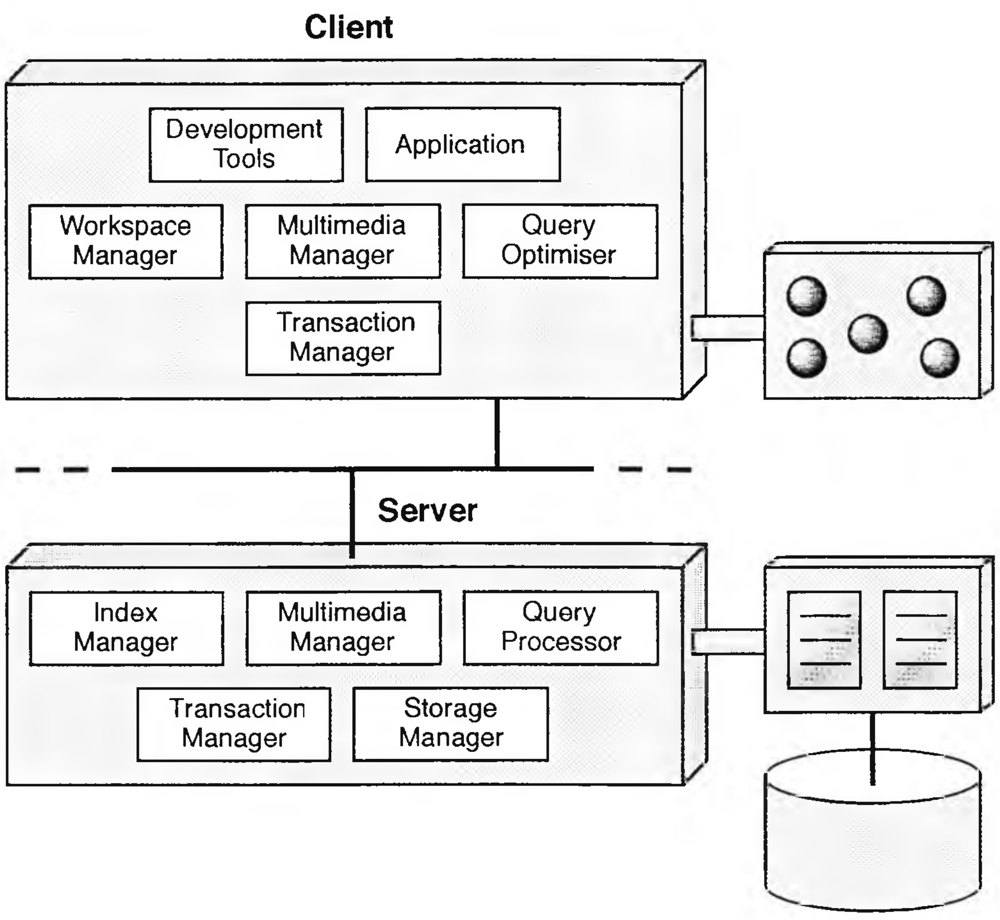

Figure 2.8 - UniSQL Architecture.

The UniSQL architecture is illustrated in Figure 2.8. It can be classed as both an object-server and a query-server. The major components are:

- **Index Manager** - to create and delete B-Tree indexes on attributes.
- **Workspace Manager** - to handle objects loaded from database into memory.
- **Multimedia Manager** - to manage user-defined multimedia classes and methods.
- **Query Optimiser/Processor** - devises optimal plan for Object SQL queries based on cost estimation.
- **Storage Manager** - handles disk storage, indexes and system catalogs.

- **Transaction Manager** - provides serialisable transactions, two-phase commit, locking and logging (before- and after-updates).

UniSQL integrates relational concepts with object-oriented ones [Kim92] and provides extended support for [Finkl93]:

- Nested tables.
- Registered procedures.
- Class (table) inheritance.
- Generalised framework for multimedia data.

The approach is to generalise rows and tables to object instances and classes, respectively. The rows and columns paradigm can be used directly with SQL and UniSQL can behave as a pure relational database in this configuration. However, support for object-oriented concepts such as object identity and inheritance enable UniSQL to also behave as an ODBMS. Registered procedures (methods) can be defined in any complied language, attached to tables and can be inherited and overridden. Furthermore, since object identifiers are used, it is possible to directly connect a row in one table to a "nested" row in another by using hashed index pointers. This approach is faster than using index searches (since these are I/O intensive as reported in [Baker91]) and allows path navigation using the '.' notation to be used, as demonstrated by the following example derived from [Finkl93; Manol94]:

```
CREATE TABLE Employee (
     name       char (20),
     position   char (15),
     location   CityState,
     skills     Set-of Skills,
     employer   Company );

CREATE TABLE CityState (
     city       char (20)
     state      char (2) );
```

A query to find the names of all employees in California would then be:

```
SELECT name
FROM Employee
WHERE Employee.location.state = "CA";
```

Relationships of 1:N cardinality are modelled by using the *Set-of* construct above. A cursor would then be used to iterate over all the instances in this set.

UniSQL supports on-line backups and dynamic schema changes. Object migration is possible through lazy evaluation techniques. Standard shared and exclusive locking techniques are available, with five isolation levels, to maintain database consistency. The locking granularity can be either at the object-, class- or page-level and is automatically adjusted. Clustering of related classes is also supported, but there is no versioning capability. Apart from the ANSI SQL interface, C, C++ and Smalltalk are also available. Various graphical tools for database inspection and application development are also provided.

In conclusion, UniSQL has the following major strengths:

- **Based on Relational Technology** - the familiar rows and columns found in RDBMSs are still used.
- **Support for OO Concepts** - subsumes and generalises relational concepts.
- **Client-Server Architecture** - makes better use of modern hardware technology than existing RDBMSs.

2.5 Chapter Summary

This chapter has presented a working definition of an ODBMS as one that transparently extends one or more object-oriented programming language with DBMS services, such as concurrency, recovery, etc. This definition is based on that described in [ODMG93]. This approach is in contrast to relational databases, where there are separate languages for database manipulation and application programs. The work reported by Lakey [Lakey89], presented in Appendix B, demonstrates that using a single language for database manipulation and application program development can provide benefits over using two separate languages.

Some of the reasons for the development of ODBMSs, based on the deficiencies of relational database technology for managing complex and inter-related data, were also discussed. Briefly, these deficiencies can be summarised as:

- A simple data model that "hammers" the world flat.
- A limited set of atomic data types, e.g. integer, char, floating point, etc.
- No support for useful modelling constructs, such as specialisation/ generalisation, aggregation, etc.
- The often-cited "impedance mismatch" problem.
- Poor performance for some applications, e.g. CAD, simulations, etc.

The features of a number of major commercial object and object-relational database products were also presented and showed wide variations in these products. Major differences occur in a number of areas, such as architectures, whether queries can be performed on the server, support for relationships, tuning options, etc.

From a performance benchmarking perspective, several important issues have emerged in this chapter.

Firstly, since the object model is very rich and extensible, the choice of data structures and application design is less obvious. This is because the design space is much larger than current implementations of RM and there are no widely published techniques equivalent to relational normalisation. The increased flexibility that object-orientation brings, therefore, is not without certain costs.

Secondly, the discussion of the commercial products showed that implementation choices, architectures, etc. vary considerably, making the task of developing a standard generic workload very difficult. Some products are also very flexible in where processes can execute. For example, GemStone allows some processes to execute on either the client or the server and provides several mechanisms for linking applications to these processes, which have performance implications, as reported in [Skiad94].

Thirdly, the tuning options on most products are very extensive. For example, Halloran [Hallo93a], commented that the number of tuning options for ObjectStore was staggering. Most commercial ODBMS product evaluations or application developments would use available tuning features to obtain a more accurate picture of product capabilities. This is confirmed by several of the case studies presented in Chapter 6.

CHAPTER 3 - Survey of Benchmarks

3.1 Introduction

Previous work reported in [Chaud94a; Chaud94b], explained that leading DBMS researchers and authors regarded performance as one of the major issues in the selection of a particular ODBMS product. One approach to measuring the performance of any DBMS is to use a benchmark, since the cost of implementing a complete application to test performance is often too expensive [Ander90]. However, a benchmark can be misused, as this cynical definition of a benchmark shows [Jain91]:

> **benchmark** *v. trans.* To subject (a system) to a series of tests in order to obtain prearranged results not available on competitive systems.
>
> \- The Devil's DP Dictionary.

Users, therefore, need to be wary of "Benchmarketing" (vendors publishing selected performance numbers) and "Benchmarking Wars" (vendors publishing new performance numbers in response to another vendors' performance numbers) [Gray91]. These problems were particularly experienced in recent times with the OO1 Benchmark [Catte92], something freely admitted by vendors themselves, e.g. [ODI93a].

Performance evaluation and benchmarking can, as mentioned in Chapter 1, serve a number of useful purposes [Jain91; Longb93], such as:

- Specifying performance requirements.
- Evaluating design alternatives.
- Comparing systems.
- System tuning.
- Bottleneck identification.
- Workload characterisation.
- Capacity planning.
- Forecasting.

Of these, *comparing systems* (to select one product from a number of alternatives) may be the major reason why users would wish to use benchmarks. Furthermore, in some performance work undertaken by researchers comparing various commercial ODBMSs, e.g. [Carey93; Hallo93a; Kempe95b], a number of significant observations were reported which would not have been made without using performance benchmarks.

From a users' perspective, there are three choices when considering undertaking a benchmarking exercise [Chaud96b].

Firstly, users could use existing benchmarks. The major advantages here are that published results will generally be available and essentially someone else has done the hard work. However, not all vendors may wish to publish results (something evident in recent times following disagreements between the University of Wisconsin and Object Design, Inc. to publish OO7 results for ObjectStore). Another serious problem is that available benchmarks may not model a user's application - a benchmark is only a valid yardstick for applications that are similar to the benchmark [Dietr92; Hallo93a].

Secondly, users could adapt or modify an existing benchmark to meet their requirements. This has the same advantages as the previous approach. However, if the benchmark is particularly complex, this requires a detailed understanding of its workings and the knowledge of what to modify. Furthermore, useful results may still not be available. For example, Tiwary et al. [Tiwar95] described the difficulties they experienced in attempting to modify the OO7 Benchmark to meet their CAD application requirements.

Finally, users can opt to develop their own benchmarks. The advantage here is that the benchmarks should meet their exact requirements, but the disadvantage is that it can be a costly exercise in terms of time and resources. Barry [Barry94], for example, has cited a figure of US$100,000 to benchmark several object databases.

The remainder of this chapter is organised as follows. Section 3.2 reviews several of the major object database performance benchmarks. Due to space constraints, other benchmarks are presented in Appendix B. The format will be to describe each benchmark, discuss any published results available and describe what contribution each benchmark has made to the understanding of database performance. The appropriateness of using RDBMS and ORDBMS performance benchmarks for measuring ODBMS performance is discussed in sections 3.3 and 3.4, respectively. Section 3.5 discusses areas where current benchmarks are deficient. Finally, section 3.6 contains the chapter summary.

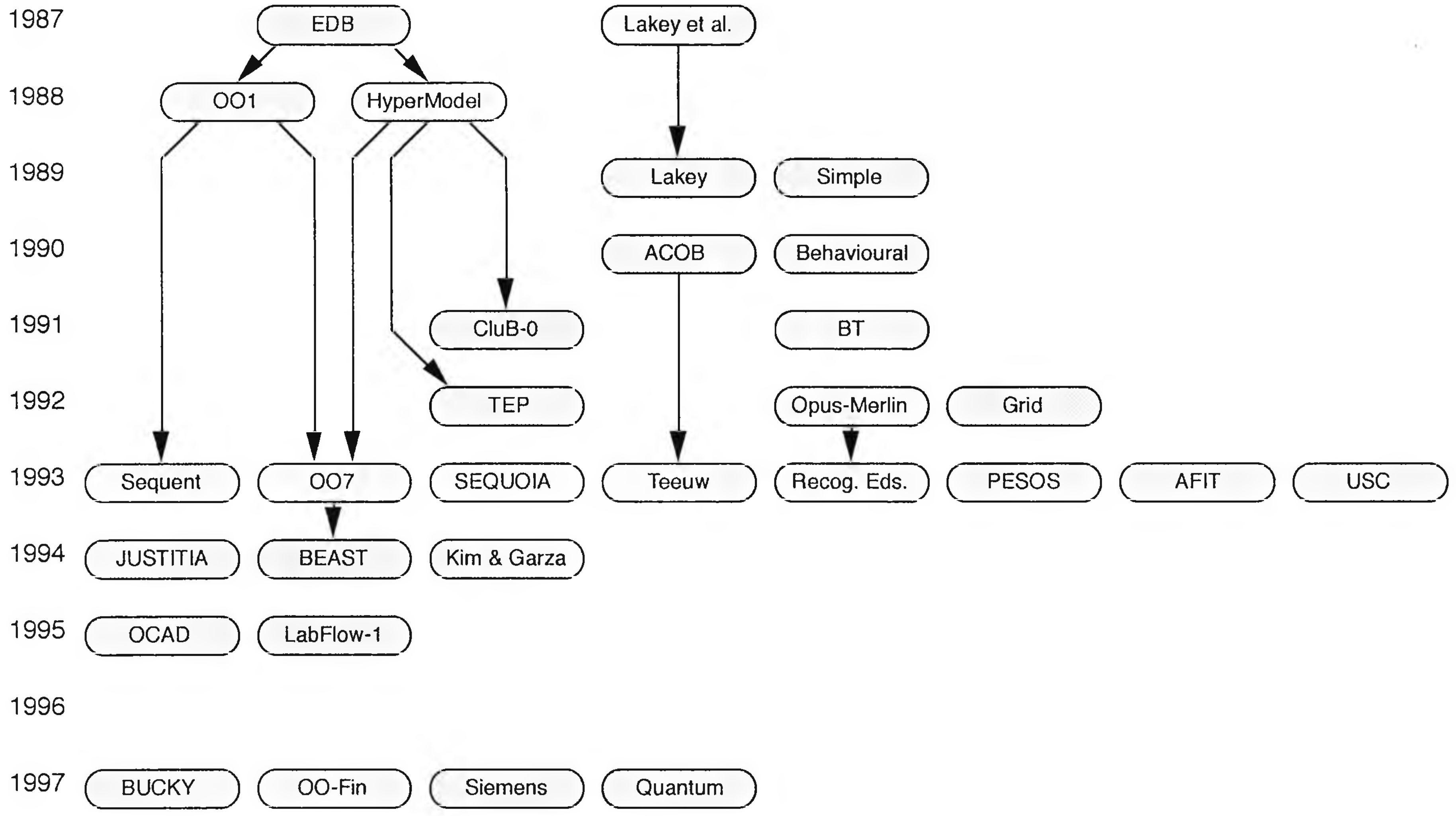

Figure 3.1 - Genealogy of Object and Object-Relational Database Benchmarks.

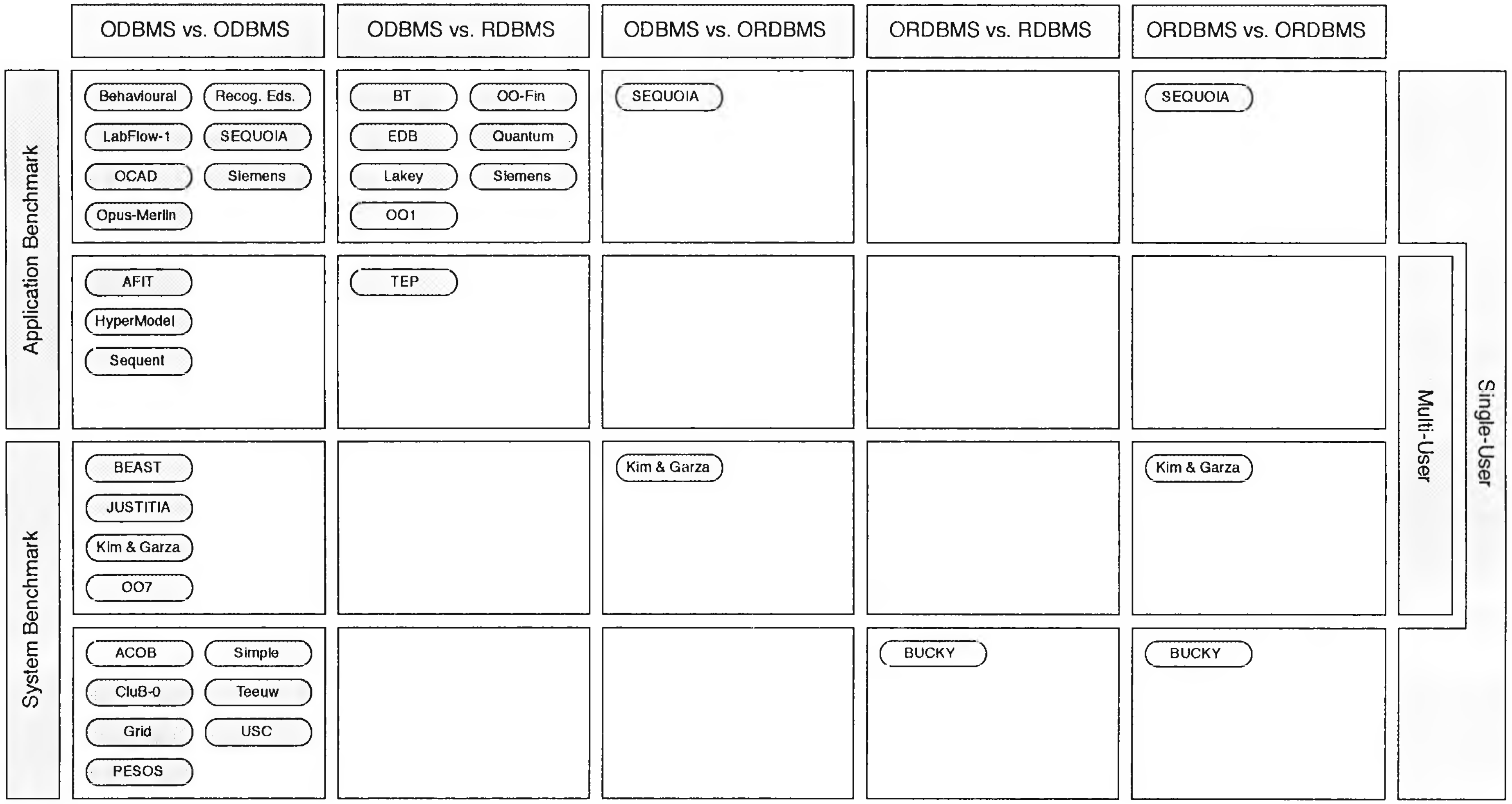

Figure 3.2 - Classification of Object and Object-Relational Database Benchmarks.

3.2 ODBMS Performance Benchmarks

Figure 3.1 presents a genealogy of many object database benchmarks and, additionally, Figure 3.2 provides several major classifications. Future research should investigate rigorous taxonomies, as discussed by Worlton [Worlt93].

As mentioned in Chapter 1, a benchmark is generally designed to be representative of the characteristics and load of a real application or to test some specific components of a DBMS [Chaud95a]. In Figure 3.2, these are represented as **Application Benchmark** and **System Benchmark**, respectively. For a critique of the benchmarks, the reader is directed to [Hallo93c; Chaud94b; Chaud95a; Chaud95b].

3.2.1 The Original Engineering Database Benchmark (EDB)

The Engineering Database Benchmark (EDB) was developed to measure *response time* for simple queries, typical of engineering applications found on network, hierarchical, relational, object-oriented or other custom application-specific database systems [Ruben87]. Response time being defined by the benchmark developers in their paper as:

> "... the real (wall clock) time elapsed from the point where a program calls the database system with a particular query, until the results of the query, if any, have been placed into the program's variables."

Other factors that motivated the development of the benchmark were:

- **Caching the Entire Database in Main Memory**
  This alone can provide an order of magnitude improvement in performance for some types of database applications, e.g. [Salem90].
- **Avoiding the Overhead of Query Optimisation**
  Efficient optimisation coupled with main memory residency for object databases have also been explored by Litwin & Risch [Litwi92].
- **Using Pre-Computed Links**
  Some RDBMSs can provide direct connectivity by using parent-child links. ODBMSs already support this capability.
- **Alternative Database Server Architectures**
  New architectures that take better advantage of client-server capabilities have emerged in recent years.

Rubenstein et al. [Ruben87] noted that whilst relational systems could respond to queries in tenths of a second, equivalent operations on in-memory structures could be performed in microseconds. DBMS architectures supporting some or all the above factors would provide a third alternative, with performance possibly two or three orders of magnitude better than that of relational systems, making them more suitable for engineering applications. One of the major aims of EDB was to try and identify such systems.

One of the reasons why existing DBMSs are unable to provide the necessary performance for simple operations is that queries parsed, interpreted or compiled at run-time swamp the time for any simple operation [Catte88]. The time to parse and optimise an SQL query, for example, was reported at nearly a second on a Sun-3 processor [Ruben87]. Additionally, traversal of a relationship between two records in a DBMS can take 10,000 instructions and generate significant disk I/O [Atwoo91].

The database schema illustrated in Figure 3.3 and Table 3.1 shows that **author** is required to handle the M:N relationship between **person** and **document**. Each person can be author of zero or more documents. For the benchmark, each document was associated with three randomly selected persons.

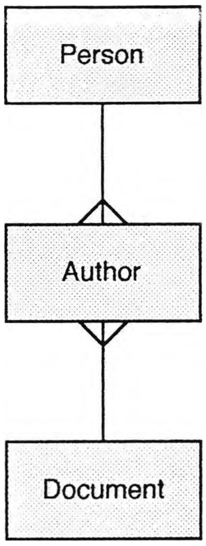

Figure 3.3 - The EDB Database Schema.

The schema was implemented on INGRES, UNIFY and RAD-UNIFY. UNIFY is an RDBMS that supports parent-child links [Catte94a]. A modified version, RAD-UNIFY, used a simplified locking scheme to more closely model the requirements of engineering applications (one database writer at a time), plus caching to keep as much of the database in memory as possible [Ruben87].

Person		**Author**		**Document**	
person id	4 byte int	**person id**	4 byte int	**doc id**	4 byte int
name	40 bytes	**doc id**	4 byte int	title	80 bytes
birthdate	4 byte int			page count	4 byte int
				doc type	4 byte int
				pub date	4 byte int
				publisher	80 bytes
				description	80 bytes

Table 3.1 - EDB Attribute Sizes and Types.

For the **small** database, 20,000 persons, 5,000 documents and 15,000 authors are specified, giving a database (just data with no storage overhead) of approximately 2.3 MB - sufficient to fit entirely within the main memory of a typical workstation. A scale-up factor of 10 is used for the **large** database.

The benchmark measures the following database operations:

1. **Name Lookup**
   Retrieve the name of a person with a randomly generated id.
2. **Range Lookup**
   Retrieve the names of all people with birthdates within a particular randomly generated ten-day range.
3. **Group Lookup**
   Retrieve the author ids for a given random document id. In relational systems, a join would be required, whereas in other DBMSs, logical links (e.g. OIDs) or physical links (e.g. record pointers) would be used.
4. **Reference Lookup**
   Retrieve the name and birthdate of a person referenced by a randomly selected author. This is the reverse of 3. Results may be identical to 1. for relational systems [Ruben87].
5. **Record Insert**
   A new author record is added. The time to update any physical structures is also measured. In general, relational systems are not very fast for updates [Ruben87].

6. **Sequential Scan**
Retrieve the title of each document from the document table. Included for completeness.
7. **Database Open**
Performed once, at the start and includes a number of initialisation operations needed to prepare the database system for the benchmarks.

The benchmark does not measure the effects of clustering author records with either person or document records. The reason for this is that person and document records in an actual database may be connected to many other record types and cannot be clustered with more than one of them at a time [Ruben87].

The results for the small database showed that, excluding DB Open, INGRES and UNIFY produced comparable results. UNIFY performed better for Reference Lookup, perhaps due to the pre-computed links, whilst INGRES performed better for Record Insert, possibly because fewer physical structures needed to be updated. RAD-UNIFY gave the best results, in some cases an order of magnitude better than UNIFY, due to the modifications.

The large database results showed a fairly constant scale-up between INGRES and UNIFY, when compared to the small database results. RAD-UNIFY performed no better than UNIFY in many cases - the advantages of main memory residency and the other modifications were lost.

Duhl & Damon [Duhl88] reported upon their experiences and results whilst porting the benchmark to the Vbase object database system. For example, they observed that since object databases could provide direct connectivity between a single object and many related objects (which they referred to as a *distributed property*), when implementing the schema on an object database, author could be eliminated as an intersection entity and be treated either as a subtype of person or as an optional property of person. Furthermore, two versions of the schema were implemented. Firstly, using person, author and document to produce an object-relational version (Vbase-OR) and secondly, using person and document to produce an object-oriented version (Vbase-OO). The latter making use of direct connectivity between person and document.

From the results, Duhl & Damon drew the following conclusions:

1. An object system can meet and in many cases exceed the performance of a fast relational system even in a problem clearly from the relational domain (this is probably a reference to the schema, which uses an intersection entity and the set of benchmark operations which would use joins as a result).
2. An object system can model a relational implementation (Vbase-OR) and achieve response times comparable to the relational system.
3. Improvements in performance can be achieved by using an alternative schema definition (Vbase-OO).

In general, however, it is difficult to define and measure benchmarks that compare object and relational database systems. Manola [Manol89] has suggested several reasons for this. Firstly, the responsibilities of the DBMS and application may be partitioned differently. In most object databases, for example, the DBMS and application are more closely integrated. This makes it difficult to compare on a "like-for-like" basis. Secondly, many developers believe that the two classes of database systems are intended for different purposes. A contrived benchmark could, therefore, be used to show the superior performance of one class of DBMS for certain applications, rather than how much faster one class of DBMS is against another.

More recently, Skiadelli [Skiad94] also described some performance work undertaken using EDB. This work was used to evaluate the suitability of GemStone and ITASCA for use in a data acquisition system (called DAQ) used at CERN in Switzerland for High Energy Physics (HEP) experiments. An existing database system developed in-house, called QUID, was used to hold parameters for the hardware and software required for a particular configuration of the DAQ system. Skiadelli attempted to determine if ODBMSs might be more suitable for this task and eventually replace the QUID system. She was able to find correspondences for person, author and document in the QUID system and proceeded to model these on the two ODBMSs. The results showed that both systems provided good performance in comparison to QUID as they were able to use local caching, indexing, etc. Furthermore, benefits were also foreseen in other areas such maintenance, uniformity and ease of implementation.

In summary, EDB was the first published attempt to develop a benchmark that could be used to measure the performance of ODBMSs. It was designed to measure simple operations on individual objects or records. The benchmark developers noted, though, that engineering applications might also require more complex operations, such as manipulating nested objects, so using their performance metrics alone did not make a system acceptable.

3.2.2 The Modified Engineering Database Benchmark

Following experience with the original EDB, Cattell [Catte88] suggested a number of modifications.

Firstly, the schema was changed to reflect a database with a better engineering "flavour", consisting of **parts** and **connections**, e.g. typical of a circuit board, as illustrated in Figure 3.4.

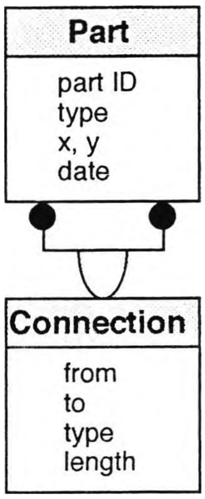

Figure 3.4 - Modified EDB Database Schema.

Secondly, the Range Lookup operation was dropped, as it was found that measurements for this differed from the Name Lookup by a constant time. However, an extension was proposed to measure complex operations, e.g. transitive closure, by combining the Reference and Group Lookup measurements.

The database itself was now populated with 20,000 parts and 60,000 connections for the small database. Each part being connected to three other randomly selected parts, giving a total database size (just data with no storage overhead) of approximately 1.8 MB.

A new set of operations were proposed to coincide with the changes to the schema:

1. **Lookup**
   Retrieve the x and y co-ordinates of 500 random parts.

2. **Traversal**
   Retrieve the x and y co-ordinates of a randomly selected part and of all other parts connected to that part, to 5 "hops" (total of 364 parts traversed, including possible duplicates).
3. **Insert**
   Generate 500 connection records between randomly selected parts.

After a number of further minor modifications, this benchmark became known as the OO1 Benchmark, discussed next.

3.2.3 The Object Operations 1 (OO1) Benchmark

The Object Operations version 1 (OO1) Benchmark [Catte91a; Catte92] was developed as a successor to EDB. OO1 was designed, like its predecessor, to measure interactive performance for engineering applications in object and relational database systems.

Cattell & Skeen [Catte92] attempted to identify those features of a database system that would enable significant improvements in performance (1,000 operations per second). They proposed that such performance would need to sacrifice concurrency control, since many engineering applications could be checked-out of a central database into a local workstation, where designers could work on them for extended periods of time. Additionally, large improvements in performance would not be achieved by minor improvements in the data model, physical representation or query languages. However, substantial improvements could be achieved by the following changes in DBMS architecture: efficient remote access to data, caching a large working-set of data in main memory, avoiding the often quoted "impedance mismatch" between programming and database query languages and using new access methods with fixed, rather than logarithmic access times.

The database schema was identical to that used by the modified EDB. Two logical records, part and connection, were defined. These could be stored as a single object type in an object database or as two relations in a relational database.

The parts are connected randomly, with the proviso that 90% of connections are made to 1% of the parts with the numerically closest part numbers.

Two database sizes, **small** (4 MB) and **large** (40 MB) are specified, with the same number of parts and connections according to the modified EDB.

The operations are identical to the modified EDB. However, there are some slight changes to the number of objects processed. For example, the Lookup operation generates 1,000 random part ids, the Traversal operation traverses 3,280 parts (7 "hops") and the Insert operation creates 100 new parts. Cattell & Skeen claim that these three operations are representative of the types of operations in actual engineering applications, as illustrated in Table 3.2.

	CASE	ECAD
Lookup (L)	Find programs for a particular system or range of dates.	Find components with particular types.
Traversal (T)	Determine a compilation plan for a system configuration.	Optimise a circuit layout.
Insert (I)	Enter new information after a module is compiled.	Add new components to a circuit board.

Table 3.2 - OO1 Database Operations.

Reads (Lookup, Traversal) are performed an order of magnitude more often than writes (Insert). Traversals are performed more often than Lookups.

Experiments were initially conducted with three different systems called **INDEX** (a B-Tree file package), **OODBMS** (a beta version of a commercial object database system) and **RDBMS** (a commercial relational database system) [Catte92].

The results showed that the object database provided the best overall performance. Cattell & Skeen 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 query-server architecture. For example, approximately 100 seconds were just due to the overhead of database calls across the network. This could be improved by using the benchmark in batch mode, but would be counter to its intent to measure response time for interactive operations. 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 [Catte91a]. There is no indication, however, whether some of the features described in the last paragraph, such as minimised concurrency control, also applied to the RDBMS.

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 and techniques such as direct physical links "under-the-covers" may be the way forward for relational systems. The UniSQL object-relational database system, for example, uses direct links to enhance nested table performance [Finkl93].

The OO1 Benchmark has also been more recently used on a number of commercially available ODBMS products including Objectivity/DB, ObjectStore, ONTOS and VERSANT. The results published in [Catte92] showed that a number of products provided comparable performance for the small remote database. However, more recent work reported by Halloran [Hallo93a] showed large performance differences between a number of products, as illustrated in Table 3.3.

	ITASCA	MATISSE	ObjectStore
L+T+I Warm	594.23	153.38	6.16

Table 3.3 - L+T+I for Small Remote Database (Secs).

Performance numbers reported by Halloran for the large remote database showed that the differences were less pronounced, especially for the cold cache numbers, clearly demonstrating that scalability should be an important factor in ODBMS evaluation. Additional interesting results emerged from the database load times, with some products taking considerable time to complete this process, whilst one system failed the large database load completely due to lack of disk space, as shown in Tables 3.4 and 3.5.

	ITASCA	MATISSE	ObjectStore
Load Time (Hrs)	17.15	2.77	0.03
DB Size (MB)	12.18	16.93	4.56

Table 3.4 - Small Database Load (Hrs).

	ITASCA	MATISSE	ObjectStore
Load Time (Hrs)	-	~1.5 Weeks	23.13
DB Size (MB)	Not enough space	167.38	44.23

Table 3.5 - Large Database Load (Hrs).

Clearly, only ObjectStore comes close to meeting the 4 MB and 40 MB storage requirements for the benchmark and it is also the fastest to load both database sizes. Load time could be an important factor for some application domains [Chaud94a].

Whilst OO1 and its predecessors have provided some insight into ODBMS performance, they have not escaped the following criticisms:

- **Simple Data Model**
  The data model is too simple to measure transitive closures and other traversal operations found in engineering applications [Ander90]. Also, there are no metrics to measure the effects of type hierarchies, inheritance or complex relationships [Duhl88].
- **Simple Operations**
  Measuring simple operations is insufficient, as many engineering applications require support for higher-level conceptual operations. For example, assessing the efficiency of transitive closures when clustering objects along some relationship, such as aggregation [Ander90].
- **Single-User**
  No measure for concurrency control and co-operation, to assess multiple users editing parts of the same data structure [Ander90]. Also, vendors exploit the benchmark by disabling concurrency and recovery features essential for commercial applications [Butte91a]. However, a multi-user version of OO1 is described in [Seque93], which also presents some results for GemStone running on Sequent SMP hardware.
- **Comparing Object Databases**
  The benchmark provides a useful comparison between relational and object systems, but is weak for comparisons *between* object systems [Duhl88]. This reason is not elaborated by Duhl & Damon.

- **Random Objects**
  The benchmark operates on randomly selected objects. Few applications have random distributions [Butte91a]. In many engineering applications, closely related objects are accessed successively, with greater frequency and to a much higher degree than are random, disjoint objects [Duhl88].
- **No Clustering**
  As a result of the last point, "semantic clustering" cannot be measured [Duhl88].
- **No Dynamic Behaviour**
  Because the benchmark is very simple, dynamic behaviour is not modelled. However, for certain applications such as event-simulation, this would be useful [Duhl88].
- **Low Level Operations**
  The benchmark allows comparisons at low level database operations, instead of more meaningful comparisons at the application level [Duhl88].
- **Database Size**
  Typically, databases are much larger than those used in the benchmark being quoted by ODBMS vendors [Butte91a].

To summarise, the OO1 Benchmark became a *de facto* standard in the late 1980s and early 1990s for evaluating ODBMS performance, due to its simplicity and timeliness. It measured raw performance and did not address higher-level operations. However, the results from the benchmark demonstrated that ODBMSs could provide the necessary performance requirements for engineering applications, when compared to RDBMSs. These requirements were identified and discussed earlier.

3.2.4 The HyperModel Benchmark

The HyperModel Benchmark was developed in response to the growing requirement for implementing engineering applications on database systems [Ander90].

Anderson et al. [Ander90] noted that although the best method to benchmark a specific application was to implement the whole application on a number of different DBMSs and then compare the results, this methodology was generally prohibitive due to the cost in time, resources, etc. A generic benchmark was, therefore, the next best alternative.

The design of the HyperModel Benchmark was based on a study undertaken at Tektronix, Inc. to identify the functionality and performance requirements of engineering applications. The DBMS requirements that were identified from this study were [Berre88; Ander90]:

- **Data Model Requirements**
  - Modelling of complex object structures.
  - Description of different data types.
  - Integration with application programming languages.
  - Dynamic modifications to the database schema.
  - Support for versions and variants.
- **System Architecture Requirements**
  - An architecture of workstations and servers.
  - Performance suitable for interactive design applications.
  - Concurrency control.
  - Co-operation between users.
  - Logging, backup and recovery.
  - Access control.
  - Ad-hoc query language.

Based on the requirements study, the schema illustrated in Figure 3.5 was proposed. The justification for this is discussed shortly. In the diagram, lines represent bi-directional relationships, with black circles at the end-points representing many and white circles one. Aggregation is represented by the arrow, which points to the composite object. The white circle on the line indicates an ordered relationship. Generalisation is represented by the triangle.

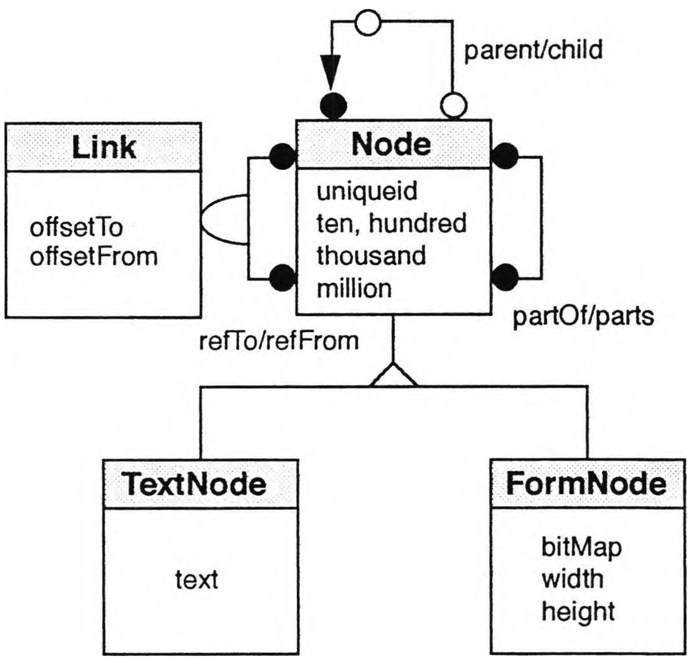

Figure 3.5 - The HyperModel Database Schema.

On the original object database systems benchmarked (Vbase and GemStone), the schema was implemented as four classes - **Node**, **TextNode**, **FormNode** and **Link**. Nodes were connected by three relationships - **parent/children**, **partOf/parts** and **refTo/ refFrom**.

The benchmark uses the node-and-link graph structure common in Hypertext applications, as shown in Figure 3.6. Two hierarchies are added over the nodes - **parent/children** (1:N cardinality) with each node at level **n** having links to five other nodes at level **n+1** (referred to as the "fan out") and **partOf/parts** (M:N cardinality). The justification for this is that at least one hierarchy is required, since recursive queries are common in many applications, but that two hierarchies are better, since multiple hierarchies often exist over the same data structure in reality. Additionally, the use of more than one hierarchy enables clustering strategies to be evaluated.

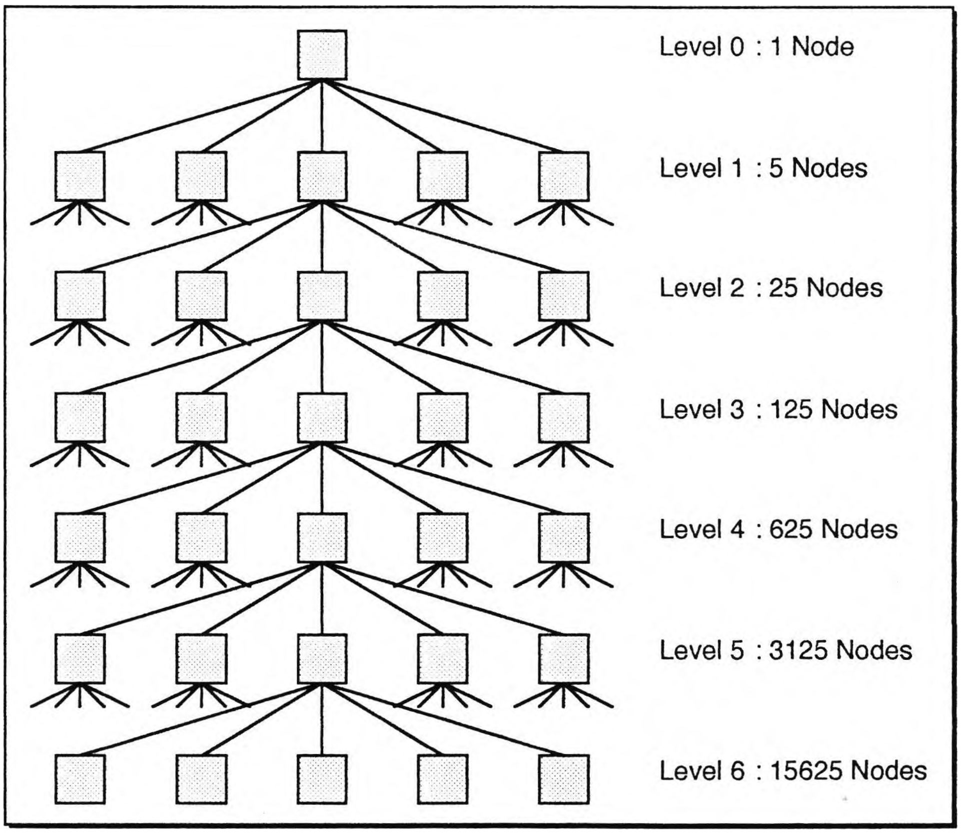

Figure 3.6 - Network of Nodes used for HyperModel Database.

The benchmark is run against level 4, 5 and 6 databases (781, 3,906 and 19,531 nodes respectively).

A schema based on Hypertext was chosen, since this was found to be representative of an entire class of engineering applications [Berre91].

The benchmark design was strongly influenced by the object database systems under test, since neither fully met the data model and system architecture requirements mentioned earlier and you "can't test what isn't there" [Ander90]. It includes the seven operations of the original EDB and adds additional operations to test transitive closure and other more complex computational functions [Ander90]. It measures cold and warm results and consists of twenty operations in total, grouped as follows [Ander90]:

1. **Name Lookup Operations**
   Two operations - return an object based on an attribute id or an object id. Use of indexing is permitted for attribute id.
2. **Range Lookup Operations**
   Two operations - return all objects whose attributes fall under a specified range, using 1% and 10% selectivity. Indexing is permitted.
3. **Group Lookup Operations**
   Three operations - retrieve objects by traversing the parent/children, partOf/parts and refTo/refFrom relationships.
4. **Reference Lookup Operations**
   Three operations - as 3., but retrieve objects by traversing relationships in reverse order.
5. **Sequential Scan**
   One operation - retrieve all objects in the database.
6. **Closure Traversal Operations**
   Seven operations - start with a randomly selected object, perform an operation on that object and on all other objects recursively reachable from that object to **n** levels using parent/children, partOf/parts or refTo/refFrom. The use of clustering is also tested.
7. **Editing Operations**
   Two operations - update objects already in the database and commit the changes to the database. These operations are designed to test the power of the database programming language.

The benchmark could be used to evaluate the effects of indexing, message sending versus procedure calls and clustering.

Hormann et al. [Horma90] also described efforts to implement the HyperModel Benchmark on several graph storage systems, e.g. GRAS, which according to [Dewal90], required two to four months per system. The results in [Horma90] were tabulated for comparison purposes with those for GemStone and Vbase from the original paper by Anderson et al., but there was no effort to analyse the results. The design of GRAS and its evaluation using the HyperModel Benchmark is also described in [Kiese92].

A derivative benchmark is described by Larsen [Larse92]. This work, discussed in Appendix B, describes a Test Evaluation Procedure (TEP) based on a simplified version of the HyperModel Benchmark. The TEP was implemented on one relational and two object databases and included both single- and multi-user tests with detailed performance results. The TEP was specifically designed with scalability, simplicity and portability in mind.

In a critique of the HyperModel Benchmark in [Catte91a; Catte92], the following points of concern were cited:

- **Benchmark Specification**
  Better implementation details were needed to ensure that the benchmark could be used to accurately compare systems in areas such as main memory size for DBMS and cache, processor and disk performance, control of initialisation timing, etc.
- **Versions**
  Applications such as CAD, CASE, etc. used versioning extensively which was not modelled.
- **Clustering**
  Complex data structures were used, but there were no guidelines on how the data should be clustered using these.
- **Client-Server**
  A remote database (resident on a server) was more representative of many engineering and office applications. In [Ander90] local-only behaviour was measured.
- **BLOBs**
  Read and write operations on Binary Large Objects (BLOBs) were really limited by disk speed. Therefore, using only small character and integer fields might be more useful. This criticism may be less appropriate today, since there are applications that require support for BLOBs, such as GIS, Multimedia, etc.
- **Limited Use**
  The HyperModel Benchmark has been used to measure the performance of early versions of only two commercial ODBMSs (Vbase and GemStone).

Carey et al. [Carey93] also commented that the experimental measurements were presented as an appendix to the original paper by Anderson et al., with no real analysis of the results.

In concluding, it is worth noting that the authors also used several other benchmarks on the object database systems mentioned earlier. Comparing the results of these benchmarks with their own, they found differences that required further study and explanation [Ander90]. With this in mind, they posed the following question:

> "What do the various benchmarks measure and how can they be used in concert to more accurately predict the performance of a particular application or mix of applications?"

Chapter 7 shows that using a series of benchmarks can provide greater insight into the suitability of object database architectures for particular applications. The benchmark developers also proposed that the HyperModel Benchmark could be used as a basis for evaluating non-engineering applications, since it tested two important features of object database systems - **complex object representation** and **complex operation implementation**. Additionally, they noted that functionality and performance were only a part of object database evaluation and that economic and market factors should also be considered - views also shared by others, e.g. [Atwoo91; Rotze91; Stein92].

3.2.5 The OO7 Benchmark

The OO7 Benchmark [Carey93; Carey94] was a recent benchmarking effort from the University of Wisconsin-Madison. According to the benchmark developers, since a number of different object database products were now quite well established in the commercial marketplace, better metrics were needed to evaluate the performance of these products. The OO7 Benchmark was specially designed to test the following characteristics of object databases [Carey93]:

- **Speed of Pointer Traversals**
  Traversals over cached data, disk-resident data, sparse traversals and dense traversals.
- **Update Efficiency**
  Updates to indexed, non-indexed attributes, repeated updates, sparse updates, updates of cached data and creating and deleting objects.
- **Query Processor or Query Programmer**
  Performance using different types of queries with 1%, 10% and 100% selectivity.

Many of these characteristics have not been measured by the two most well-known object database benchmarks (OO1 and HyperModel). The benchmark schema is also far richer than either OO1 or HyperModel and is based on the idea of a design library consisting of parts and assemblies and is, according to the benchmark developers, suggestive of a number of applications such as CAD, CAM, CASE, etc. However, OO7 should more correctly be termed a **System Benchmark** (rather than an **Application Benchmark**) from the characteristics mentioned above and that neither the schema nor operations have been based on any performance studies to identify the characteristics of engineering applications [Chaud95a]. Figure 3.7 illustrates the entities and relationships. The schema illustrated uses modelling notation similar to that described earlier for the HyperModel Benchmark. It can be seen that an atomic part and its recursive relationship correspond to the part and connection of the OO1 Benchmark.

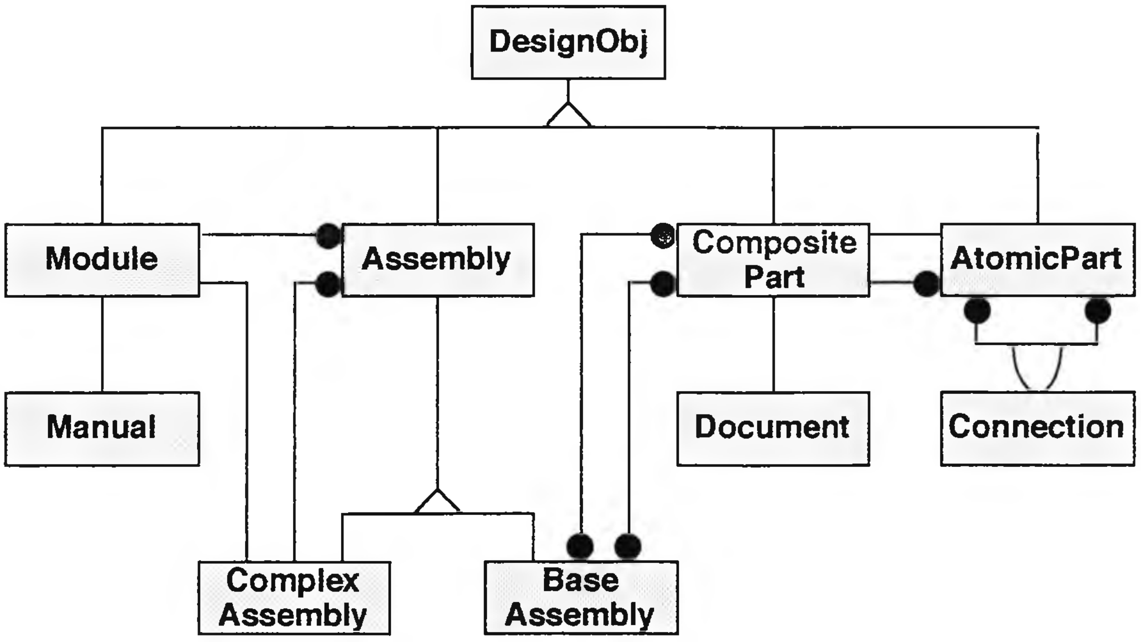

Figure 3.7 - The OO7 Database Schema.

A **Composite Part** could be representative of a procedure in a CASE application. Associated with this would be a **Document** with some descriptive text. Each composite part is an aggregate of a number of **Atomic Parts**, which could represent variables, statements or expressions in a CASE procedure. Over these composite parts, a **Base Assembly** hierarchy has been added to represent higher-level design objects, such as an ALU in a CAD application. A **Complex Assembly** is an aggregate of other assembly objects, as illustrated in Figure 3.8.

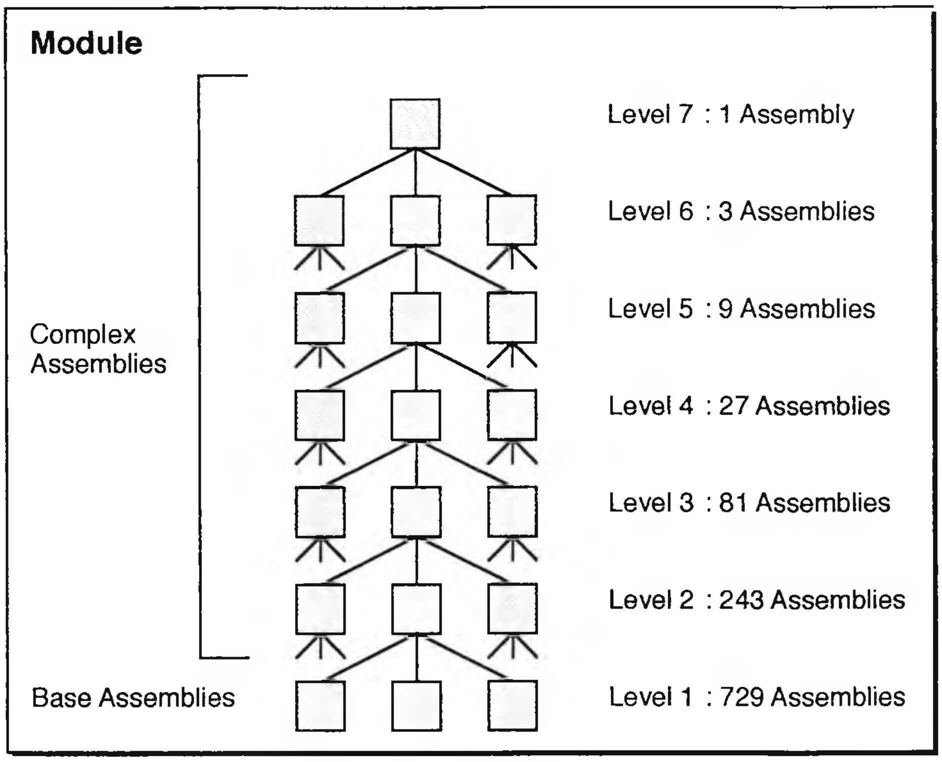

Figure 3.8 - OO7 Assembly Hierarchy.

Figure 3.9 illustrates the relationship between base assembly and composite parts.

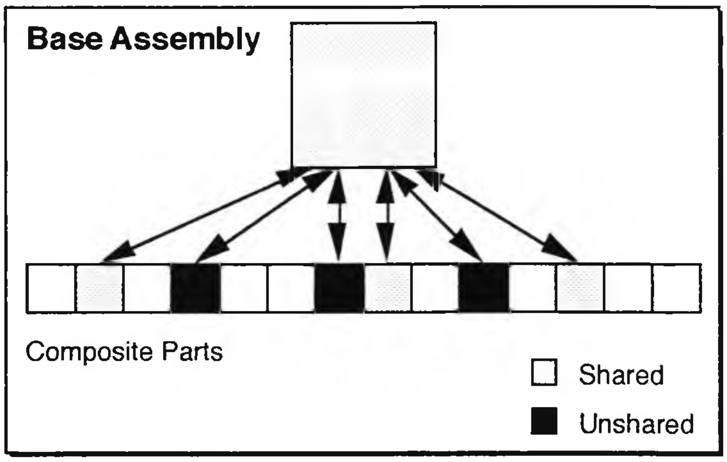

Figure 3.9 - OO7 Base Assembly and Composite Parts.

A **Module** describes a complete assembly hierarchy. For the small and medium database tests (5 and 50 MB respectively), a single module is used. For the large database (500 MB) and multi-user tests, multiple modules are proposed. Associated with a module is a **Manual** with some descriptive text.

Whilst three database sizes and single- and multi-user benchmarks were originally proposed, only results for the small and medium single-user database systems have been reported so far. There are twenty operations (eleven traversals, seven queries, one operation to insert objects, one operation to delete objects), three levels of complexity ("fan out") and 105 tests overall. Since the small database may fit entirely within workstation memory, empty and full cache measures are reported for this size only.

The major operations that were implemented and measured were:

1. **Traversals**

   T1 - raw pointer traversal speed. This traverses an assembly hierarchy and performs a depth-first search on its atomic parts. It returns the number of atomic parts visited. This is similar to the traversal operation used in OO1.

   T2 - traversal with updates. The update is to swap the x and y co-ordinates of atomic parts. Three types of updates are used: one atomic part per composite part, every atomic part, each atomic part in a composite part four times.

   T3 - traversal with indexed updates. Similar to T1, except the update is to a date field, which has been indexed.

   T8 - traverse the manual object, counting the number of occurrences of the character "I".

   T9 - check if the first and last character in the manual object are the same.

   TCU - cached update traversal. The aim is to estimate the cost of updating objects in the cache.
2. **Queries**

   Q1 - exact match lookup. Ten random atomic part ids are generated. These are then used to find the corresponding atomic parts. This is similar to the lookup query in OO1.

   Q2 - range query with 1% selectivity. Uses the date attribute to retrieve the appropriate atomic parts.

   Q3 - range query with 10% selectivity.

   Q7 - full scan of all atomic parts.
3. **Inserts**

   Insert five new composite parts.
4. **Deletes**

   Delete five composite parts.

The original benchmark results reported in [Carey93] were for the commercial systems Objectivity/DB and ONTOS and the University of Wisconsin research prototype EXODUS, with results for the commercial system VERSANT added in [Carey94]. The drawback with this is that fair comparisons cannot be made, since products that are tested later have the benefit of additional time for improvements and enhancements. The comparison between VERSANT and the other products is therefore misleading, since only new numbers for VERSANT are reported in [Carey94] and old numbers for the other systems remain. Furthermore, since the benchmark was designed for extensive testing and did not provide a single number, some confusion resulted when the researchers provided three alternative approaches to condense the results and a different commercial product came top in each approach. According to [Objec93], the differences in times for the hot results were small (milliseconds or seconds) and the cold results were large (minutes or hours). Another interesting observation is the database size (including storage overheads) with some published figures for the medium database in [Darni93] showing large variations between several products, as illustrated in Table 3.6.

	O_2	ONTOS	Objectivity/DB	ObjectStore
DB Size (MB)	80.9	122.3	74.9	55.4

Table 3.6 - OO7 Medium Database Sizes.

In Table 3.6, only ObjectStore comes close to the 50 MB specification (its results are from [ODI93b]). This illustrates that storage overhead could be an important factor in product selection - something also highlighted by Halloran [Hallo93a].

An implementation of the OO7 Benchmark has also been reported for Persistence in [Nag95]. Berg & Hoeven [Berg96] used OO7 to evaluate a prototype GIS, since they suggest that the same types of traversal operations found in OO7 are also common to GIS applications.

The benchmark suffers from some of the criticisms levelled at other benchmarks, such as OO1 and HyperModel [Chaud94b]:

- **Single-User**
  Whilst not specifically aimed at OO7, Barry [Barry94] has suggested that multi-user benchmarks for multiple uses are needed. A multi-user version of OO7 was reported as being under development with results due to be reported soon [Carey94].
- **Low Level Operations**
  The benchmark measures low-level primitives, but complete operations at the application level may be more useful to end-users.

- **Database Size**
  Larger database sizes are required. Barry [Barry94], for example, has suggested that 1 GB would be a useful starting point for object database benchmarks.
- **Lack of a Single Performance Metric**
  It is arguable whether this is really a major problem. However, reducing the measurements to a single number would provide an easier method of comparing database systems, as there is already some confusion, partly caused by vendors themselves.
- **Lack of Performance Studies**
  Since the benchmark has not been based on any performance studies, certain design decisions are questionable, such as the choice of the database schema [Chaud94b]. Furthermore, the OO7 developers used code to simulate what a query executor *would* do to evaluate a test query. However, as mentioned earlier, "you can't test what isn't there" [Ander90].

Several other researchers have also reported significant problems and limitations with the OO7 Benchmark:

- **Tiwary et al.**
  Tiwary et al. [Tiwar95] identified the following limitations of OO7 as an application benchmark in comparison with their CAD application requirements:
  - Difficult to increase object sizes without modifying class descriptions.
  - Difficult to specify distribution of object sizes for a class.
  - No control over database layout.
  - No use of any clustering mechanism provided by an ODBMS.
  - Impossible to map workloads against OO7 traversals.
  - Uniform workload unrealistic.

  Furthermore, they even criticised the benchmark's applicability for measuring system features:
  - Tight coupling between OO7 workloads and data structures.
  - Use of product-specific data structures makes code non-portable.
  - Primitive support for instrumentation.
- **Hohenstein et al.**
  The work by Hohenstein et al. [Hohen97a; Hohen97b] is presented in detail in Appendix B. Briefly, the criticisms they reported were:
  - Varying parameters in OO7, such as the fan-out or the database size, does not necessarily mean that it is more representative of specific applications.

  - Fairness in testing the query programmer is extremely doubtful, since query optimisation is not one of the strengths of ODBMSs and hand-coded queries are sometimes more efficient.
  - Tuning is neglected, but C++ or Smalltalk programmers can master an ODBMS after adequate training.

- **Jun & Gruenwald**
  Jun & Gruenwald [Jun97] criticised OO7 (as reported in [Chaud98b]) and proposed the following improvements to make it more useful for future object database concurrency control work:
  - Better class definition read and write transactions.
  - More instance access read and write methods per class.
  - A deeper class hierarchy.
  - A bigger fan-out for nested methods.

In summary, the OO7 Benchmark has provided the object database community with a very comprehensive set of metrics that can be used in evaluating performance. It is too early to tell whether it will become a widely used benchmark. There may be an opportunity now for a range of new metrics to be devised which do not focus on many of the low-level operations that OO7 measures. As mentioned earlier, higher-level conceptual operations need to be measured, since object databases provide better data abstraction than previous generations of database systems. Additionally, new metrics are needed to address issues such as versioning, clustering and many feature interactions, as suggested by Stein [Stein92].

3.3 RDBMS Performance Benchmarks

This section includes several RDBMS benchmarks. The aim is to discuss the appropriateness of using these for measuring ODBMS performance.

3.3.1 The Wisconsin Benchmark

The Wisconsin Benchmark [Bitto83] is the most frequently used single-user benchmark for relational database systems, according to Khoshafian et al. [Khosh92b]. The reason for its popularity has been attributed to its timeliness, simplicity and portability. Additionally, it attempted to measure the performance of relational database systems using a standard methodology.

When it was originally developed, the benchmark was used to evaluate the performance of a number of commercial relational database systems. The results were subsequently reported, with the systems under test being named. Consequently, this led to so-called "database wars", since vendors used the benchmark on each new release of their product, highlighting improvements in performance over the previous version, besides claiming superior performance over their competitors. DeWitt [DeWit91] noted that had the products not been named, then the benchmark would simply have been treated as an academic exercise.

The Wisconsin Benchmark is essentially for Selection-Join query processing and therefore not suitable for engineering applications, which generally have higher-level conceptual operations [Ander90]. Additionally, Duhl & Damon [Duhl88] commented that the Wisconsin Benchmark was targeted at measuring operations specific to the Relational Model and was therefore not suitable for object-oriented systems.

Rubenstein et al. [Ruben87] noted that although there was some overlap between the Engineering Database and Wisconsin Benchmarks, the latter was not suitable for engineering applications, since these applications and tools required fast response for *simple* operations. Furthermore, they stated that in relational terms, the queries in the Engineering Database Benchmark were on a single table, with operations on single records or small groups of records representing one logical object. Kim et al. [Kim90c] have also commented that the Wisconsin Benchmark was inappropriate for object database systems, since there was only a partial overlap between operations in relational and object databases. For example, relational databases do not support operations for concepts such as inheritance, methods, object navigation and nested objects. Therefore, there is nothing to compare the overhead of inheritance or the cost of traversing the sub-components of a complex object [Ghand93].

The Wisconsin Benchmark can continue to serve a useful role for evaluating object-relational database systems, since the relational sub-system forms one part of the ORDBMS. This is something that has also been noted in [Asgar97]. However, since its design is geared towards a schema and operations for relational processing, it is unsuitable for measuring the performance of pure object databases.

3.3.2 The DebitCredit/TP1 Benchmark

The DebitCredit Benchmark [Anon85] was developed in response to one of the major deficiencies of the Wisconsin Benchmark, namely that it measured only single-user performance [Khosh92b].

The benchmark was designed to measure the Transactions Per Second (TPS) performance (or system throughput) of different transaction processing systems. Using the TPS measure, various systems could then be fairly compared using a standard price/ performance ($/TPS) ratio.

The benchmark modelled a banking system and was based on an earlier benchmark used by the Bank of America in 1972/73 to select a teller system [Khosh92b]. For the benchmark, a Bank had a number of Branches, each branch had a number of Automated Teller Machines (ATMs) and Customer Accounts. Customers could withdraw from or deposit into their accounts (hence the name DebitCredit) using teller machines either at their own or another branch. The customer account balance, branch total and teller total were updated accordingly and a record of the entire transaction was also maintained in a history file. Figure 3.10 illustrates the database schema.

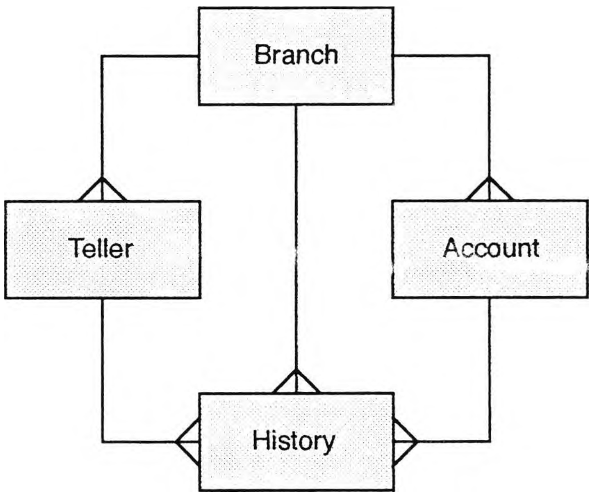

Figure 3.10 - The DebitCredit Database Schema.

In addition to many terminal users, DebitCredit simulated a "think time" of 100 seconds (interaction time between a customer and the ATM before a transaction was submitted) and network traffic time. TP1 was a simplified, informal version of DebitCredit.

The original paper also specified Sort and Scan Benchmarks in addition to DebitCredit. However, these appear to have been largely ignored by users.

TP1 measures system throughput for OLTP applications (large numbers of small transactions). In contrast, engineering applications typically have a small number of long transactions [Ander90]. Duhl & Damon [Duhl88] also comment that TP1 was an inappropriate benchmark, because it measured high volume transaction processing usage, which was not typical of object-oriented systems.

According to Rubenstein et al. [Ruben87], there is some overlap between TP1 and the Engineering Database Benchmark. However, as previously mentioned, TP1 measures system throughput rather than response time. Additional throughput can be achieved by multiprocessing and pipelining, whereas improved response time cannot [Ruben87]. Because TP1 includes a composite of measures used in the Engineering Database Benchmark, TP1 results correlate with results from that benchmark [Ruben87]. However, it has not been explained by Rubenstein et al. what measures were composites and how they established the correlation.

Chapter 7 implements an OLTP benchmark on two object databases and an object-relational database. The reasons for this will be justified in detail in Chapter 4. Briefly, however, the arguments against measuring OLTP performance for object databases are now somewhat dated. As discussed in Chapter 1, object databases are gaining popularity and usage in many non-engineering application domains. For example, one of the case studies presented in Chapter 6 demonstrates the use of an object database for a financial application that exhibits OLTP characteristics.

3.4 ORDBMS Performance Benchmarks

Object-relational databases are a relatively recent development and as such there are few reported benchmarks to measure the performance of such systems. However, since they attempt to combine features of both object and relational databases, it should be possible to use existing benchmarks to measure certain aspects of object-relational database performance. As an example, benchmarks to measure OLTP performance should be applicable, since the familiar rows, columns and tables paradigm from the Relational Model is supported. Furthermore, systems such as UniSQL also support direct physical links, which means that benchmarks for measuring pointer traversal speeds, as are commonly used for object databases, can also be used. Dick & Chaudhri [Dick95b] also argued that new benchmarks for object-relational databases were needed, consisting of mixed workloads with both navigational and declarative query processing.

3.5 Where Benchmarks are Lacking

There are a number of areas where current object database benchmarks are lacking. Future metrics may address one or more of these deficiencies. The headings below follow a similar format to those used in the paper by Rotzell & Loomis [Rotze91].

3.5.1 Representative Benchmark Data and Operations

Benchmarks are only representative if the data and tests coincide with the application in mind [Hohen97a]. However, it is very difficult to design benchmarks that are relevant to applications. Most benchmarks measure low-level features in isolation, whereas application performance depends upon the interplay of sets of low-level features. Databases are holistic: the whole is greater than the sum of the parts [Bradl94]. Developing benchmarks to measure complete operations at the application level has been previously attempted with some success, e.g. [Lakey87; Lakey89; Hohen97a; Hohen97b].

Generic benchmarks try to be suggestive of a number of application areas. This may be possible in some circumstances where the applications exhibit similar characteristics. For example, CAD, CASE and Hypertext have similarities in their requirements for data manipulation [Berre91; Catte94a]. However, benchmarks such as OO1 would not be particularly representative of MIS or business applications. In the past this was not a problem, since many object databases were developed specifically to meet the requirements of engineering applications. Increasingly, however, object database vendors are offering integration and interoperability with relational systems, as well as trying to compete with relational databases in many non-engineering areas, as illustrated in the survey by Everest & Hanna [Evere92]. Therefore, existing benchmarks may not be very appropriate for measuring the performance of these non-engineering applications.

3.5.2 Multiple Applications

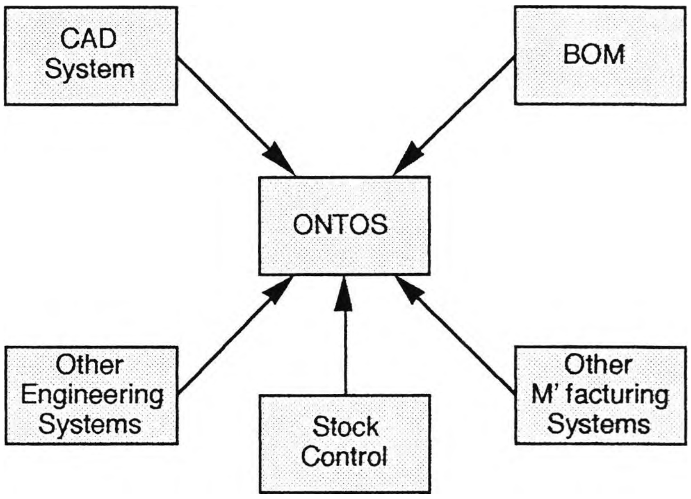

Figure 3.11 - British Aerospace and ONTOS.

Most benchmarks are single-application oriented. It is therefore difficult to extrapolate performance results to environments where multiple-applications are running. For example, consider the British Aerospace case study cited in [Wilki93], which illustrates the use of an object database (ONTOS) as an integrator of disparate tools used in the production of an aircraft wiring harness. A variety of applications access the database, e.g. engineering, manufacturing and stock control, as illustrated in Figure 3.11. If a benchmark were used to measure the performance of just one application, e.g. engineering, this would provide a very limited picture of overall performance, since it would only show the access paths through the database used by that particular application. However, benchmarking multiple applications is non-trivial.

3.5.3 Multiple Users

Most benchmarks are single-user and therefore not suitable for commercial applications which are *always* multi-user [Bradl94]. Benchmarks that attempt to be representative of real-world applications must also be designed to test features such as concurrency control and locking, as these can be disabled with single-user benchmarks. However, current commercial products differ significantly in their approaches to locking, making it difficult to develop fair comparisons between them. For example, some products provide variable granularity, enabling locks to be set at the object-, page-, file- or database-levels, whilst others do not [Rotze91]. However, it is *very* difficult to design multi-user benchmarks due to the number of degrees of freedom, especially in client-server or distributed environments. Therefore, results from multi-user ODBMS benchmarks should be interpreted with caution [Bradl94].

3.5.4 Multiple Databases

Approaches to multiple databases vary considerably between products. Some support multiple databases and distributed transactions, whilst others provide multiple databases but no distributed transactions. Additionally, some products centralise the object location and transaction co-ordination functions, whilst others fully distribute these [Rotze91]. Current benchmarks have been designed with only single databases in mind.

3.5.5 Client-Server

Object databases were developed to take advantage of client-server computing, with DBMS functionality partitioned between the two. Both OO1 and OO7 were designed to test database access across a network. However, client-server architectures vary considerably, as demonstrated by DeWitt et al. [DeWit90], with each architecture having particular strengths and weaknesses. Trying to develop benchmarks that can fairly represent these alternative architectures is again, non-trivial.

3.5.6 Multiple Platforms

Object databases are available on a variety of platforms, ranging from personal computers to UNIX workstations. Some products only support particular combinations of platforms in their client-server configurations. It is also possible that they may have been optimised to run in particular combinations [Rotze91]. However, the majority of object database benchmarks have only reported results for UNIX workstations and have not considered any form of hardware normalisation to provide some kind of price/performance ratio. Furthermore, as reported in Chapter 6, commercial ODBMS applications can consist of mixtures of hardware platforms, running different operating systems.

3.5.7 Functional Differences

There are simply too many functional differences between current commercial products to be easily compared using benchmarks. Relational databases have a standard interface (SQL) and a fairly homogeneous set of application requirements (e.g. OLTP) and products. As yet, there is no standard interface for object databases and there is such diversity in applications, architectures, features, possibilities for optimisation, etc. which makes it extremely difficult to avoid comparing apples with oranges [Bradl94]. However, the efforts of ODMG to provide standard language bindings for Smalltalk, C++ and Java™ may be very beneficial in future object database performance work.

3.5.8 Tuning

Applications are not normally designed in isolation, but are designed with the architecture, features and interface of a particular product in mind. It is possible to achieve two or three orders of magnitude performance differences with the same application on the same product depending on the choice of data model, clustering, concurrency mode, functional distribution between client and server, etc. [Bradl94]. Examples of tuning object database applications and some of the performance benefits achieved are discussed in [Chaud97].

3.5.9 Development and Maintenance

It is often forgotten that object databases are also components of object technology. As such, it should also be possible to benchmark development and maintenance time as well, since raw performance is only a part of object database evaluation [Ander90].

3.6 Chapter Summary

In this chapter, three well-known object database performance benchmarks have been discussed. All three of these benchmarks have focused on operations suggestive of engineering applications, although only one has actually been based on any requirements analysis or performance study.

Table 3.7, adapted from [Khosh92b], summarises the major features of these three benchmarks.

	OO1	HyperModel	OO7
Database Generation	Synthetic, uniform distribution	Synthetic, uniform distribution	Synthetic, uniform distribution
Industry Acceptance	Wide, ODBMS vendors	No	No
Mixed Workload	Yes	Yes	Yes
Number of Users	Single	Single	Single/Multi
Orientation	Engineering	Hypertext	Generic
Performance Metric	Response times of individual queries	Response times of individual queries	Response times of individual queries
Specification	Detailed	Limited	Detailed
System Size	Scalable	Scalable	Scalable
Utility Operations	No	No	No

Table 3.7 - Summary of OO1, HyperModel and OO7.

These three benchmarks have tended to measure raw performance and pointer traversal speeds, although there are many other aspects to object database performance that can be measured. For example, a database panel held in conjunction with OOPSLA '88 identified the following possibilities [Josep89]:

- Navigational (pointer chasing speed).
- Query throughput.
- Query response time.
- Save and retrieve of complex states by Object Identifier (OID).
- Simple bulk operations on all objects.

- Complete operations at the application level.
- Good metrics depend on application.

The workshop on object database performance held at OOPSLA '95 also revealed a wide range of issues in measuring object database performance, as discussed by Zorn & Chaudhri [Zorn95]:

- More performance studies of real applications.
- Holistic benchmarks.
- Need for common trace formats.
- Multi-user benchmarks.
- Time-varying nature of applications.
- Impact of design.

In this chapter, areas where current object database benchmarks are lacking have also been highlighted and discussed. Many of these deficiencies have been directly observed in the case studies, described in Chapter 6. In the next chapter, some of these issues will be considered in the research design for developing performance benchmarks for object databases.

CHAPTER 4 - Research Design

4.1 Introduction

Chapter 1 showed that object databases are now being used for many more application domains than just engineering. Chapter 2 described seven major object and object-relational database products in detail and highlighted a number of architectural and other issues that can affect database performance evaluation. Chapter 3 presented several well-known object database benchmarks, discussed their limitations and, furthermore, suggested areas that future object database performance benchmarks should address. This chapter presents a research design that attempts to overcome some of the limitations described in these three previous chapters. It is organised as follows. Firstly, the problem statement and problem space are presented in sections 4.2 and 4.3, respectively. Then, in section 4.4, the research objectives are described. In sections 4.5 to 4.8, the research approach is presented in detail, including a description of the techniques used. In section 4.9, a brief description of several statistical methods is presented. Finally, section 4.10 contains the chapter summary.

4.2 Problem Statement

As discussed earlier, ODBMSs are now being used in many different application domains. Whilst the history of this technology has been rooted in engineering applications, today it is being used for Network Modelling in Telecommunications, Risk Management and Financial Applications in International Banking, Multimedia, Healthcare Applications, etc. However, this growth in diversity and range of applications has not been matched by the development of adequate performance benchmarks. This is a serious problem, since benchmarking complete applications is known to be expensive [Ander90; Barry94] and there is no indication of the suitability (or otherwise) of particular products for particular classes of applications. The latter is significant, since the architectural choices made by ODBMS vendors vary considerably, unlike RDBMS products which are fairly homogeneous [Bradl94]. Another issue is that most previous object database performance work lacks any real basis, i.e. there have been no performance studies, as discussed in Chapter 3, as well as no methods used to verify benchmark results. What is required, therefore, is a more systematic/programmatic approach, based on sound research techniques, using statistical methods to verify performance results. This research aims to address these deficiencies by using these techniques.

4.3 Problem Space

In the previous section, it was stated that ODBMSs are being used in a large range of applications. This is illustrated in Figure 4.1.

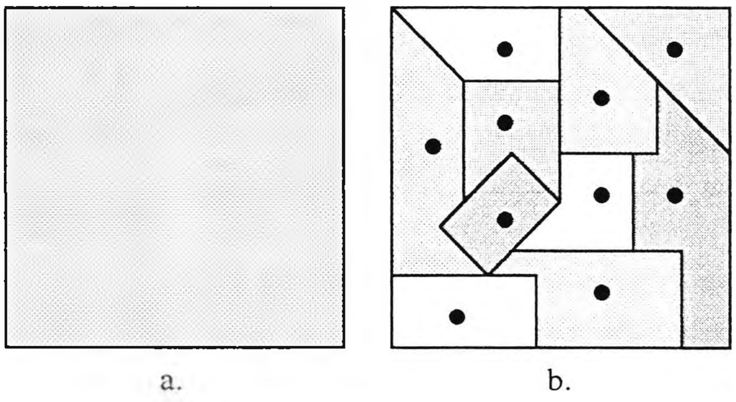

Figure 4.1 - The Problem Space [Wagne91].

In the diagram, a. shows that there are a potentially infinite number of problems in the space and b. shows partitioning of the space into equivalence classes. Since resources are limited and exhaustive testing is impractical, Wagner [Wagne91] proposes that partitioning of the space into equivalence classes allows the testing of just a few cases which, by induction, would be equivalent to testing the entire class. This makes the problem of testing tractable. Furthermore, Wagner divides the problem space into a performance space (a subset of the problem space), which can be explored by either an **Application-Specific** approach (consisting of one or several points) or a **Systematic** approach (consisting of many points). Some of the characteristics of each approach are [Wagne91]:

- **Application-Specific**
  - Based on real-world problems.
  - Measures system performance in the actual context in which the system is employed.
- **Systematic**
  - Has wide-ranging utility.
  - Provides generic evaluation.
  - Can be used to test system variability and sensitivity.

The relative merits and drawbacks of these two approaches are discussed in more detail in [Stone85] and [DeWit85]. Hawthorn [Hawth85] has also suggested that a combination of the two may be the best approach to database performance evaluation. For this project, what is proposed in later sections is using these two approaches together.

4.4 Research Objectives

The first aim of this research is to study the performance of commercial object database applications. This subject is not well documented, although it may be well understood by consultants and vendors who are asked to undertake benchmarking for their customers. There are also few examples of detailed case studies in the object database performance literature and previous benchmarks suffer from a range of limitations and problems, as stated earlier, such as the lack of proper studies and no verification of published results.

The second aim is to attempt to identify which classes of applications are more suitable for particular object database architectures. This will be approached by undertaking some performance benchmarking, based upon the studies and other techniques proposed in the following sections. Again, this area is not well understood for real applications, although some benchmarks, such as OCAD [Kempe95a; Kempe95b], have proposed and reported results for synthetic benchmarks in an attempt to address this area.

The third and final aim is to determine if a generic, simple and accurate performance model for object databases can be derived. This is because scientific theories offer generic models applicable to a variety of systems [Ohkaw93]. A generic model provides a framework for representing a targeted system, but specifics are further provided in each case [Ohkaw93]. Inmon [Inmon89] states, however, that treating database performance and its measurement generically is wrong. This is because, he argues, a benchmark for measuring the performance of an OLTP system, for example, has very different characteristics to a benchmark for measuring the performance of a Decision Support System (DSS) and there is very little in common between them. However, a generic model for testing the performance of ODBMSs would considerably aid benchmark designers, since they can then focus on the specifics needed for a particular target application, rather than designing a benchmark from scratch. However, Weick [Weick84] comments that scientific theories are limited because none of them can be acceptable in terms of generality, accuracy and simplicity. This is because, he argues, a theory can at most meet only two of these criteria, as follows:

1. General, accurate theories are complex.
2. General, simple theories are inaccurate.

3. Simple, accurate theories have no generality.

The work by Youssef [Youss93], however, demonstrated that generality, accuracy *and* simplicity are possible in database performance work, although his work was aimed at OLTP performance evaluation for RDBMSs. More widely, the problem for database performance evaluation could be solved as follows:

1. Generality - a set of core benchmark operations.
2. Accuracy - specifics added to the core for a particular target application.
3. Simplicity - core operations are easier to compare.

4.5 Choice of Research Approach

A discussion and taxonomy of alternative Information Systems (IS) research approaches by Galliers [Galli91] is summarised in Table 4.1, below.

Scientific (empirical)	Interpretivist
Laboratory Experiments	Subjective/Argumentative
Field Experiments	Reviews
Surveys	Action Research
Case Studies	Descriptive/Interpretive
Theorem Proof	
Forecasting	Futures Research
Simulation	Role/Game Playing

Table 4.1 - Information Systems Research Approaches.

From the discussion in [Galli91] of the strengths and weaknesses of the above, the empirical approaches are well suited for this research project, based on the objectives stated in the previous sections. Furthermore, the empirical approaches provide good routes to theory extension, which Galliers feels is the most difficult phase in IS research, as illustrated in Figure 4.2.

Galliers also suggests examining previous work in one's particular field of study from the perspective of the research approaches that have been used. If previous research has used one or two approaches only, it may be useful to adopt another approach. Most previous work on ODBMS performance has used laboratory experiments with little or no evidence of any of the other approaches being used. Case studies are, therefore, a suitable alternative to use, particularly since Youssef [Youss93] used this approach successfully in the development of an OLTP benchmark for RDBMSs.

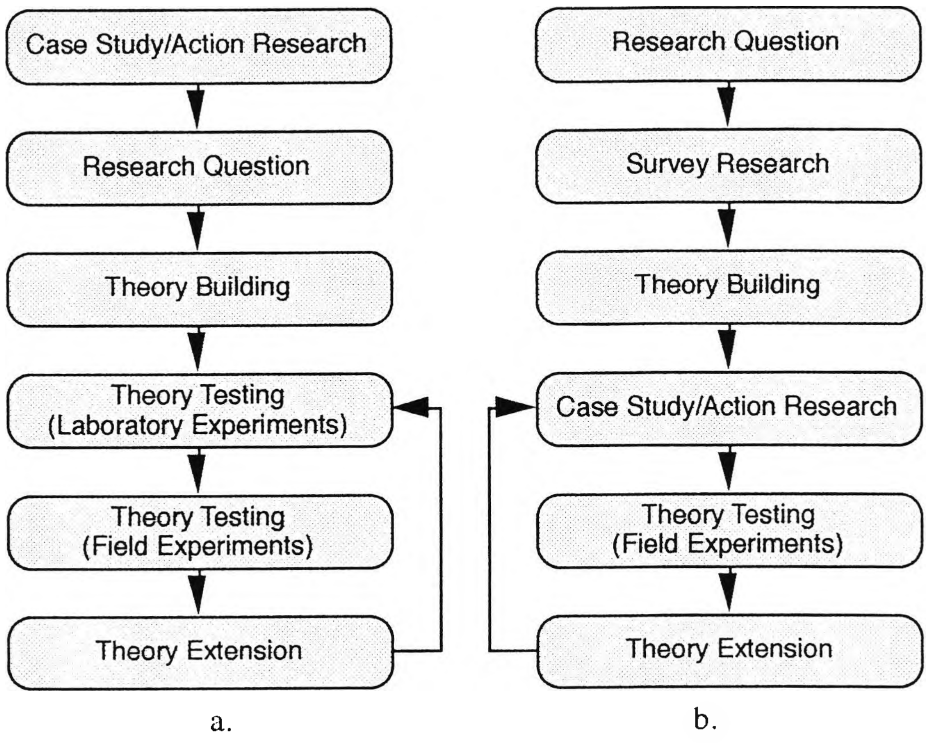
Case Study/Action Research
Research Question
Theory Building
Theory Testing (Laboratory Experiments)
Theory Testing (Field Experiments)
Theory Extension
a.
Research Question
Survey Research
Theory Building
Case Study/Action Research
Theory Testing (Field Experiments)
Theory Extension
b.

Figure 4.2 - Alternative Routes to Theory Extension [Galli91].

Benbasat et al. [Benba87] also cite the following reasons why the case study approach is useful in IS research:

1. The researcher can study information systems in a natural setting, learn about the state of the art and generate theories from practice.
2. The case method allows researchers to answer *how* and *why* questions, that is, understand the nature and complexity of the processes taking place. This view is also shared by Yin [Yin89]. For ODBMSs, the questions include *how* are they being used and *how* can we develop better benchmarks based on these observations? [Chaud94a].
3. A case approach is an appropriate way to research an area in which few previous studies have been undertaken. This is indeed the current situation in ODBMS performance.

Case research is also useful for exploration and hypothesis generation, which are legitimate ways to add to the body of knowledge in the IS field [Benba87]. The research objectives stated earlier are precisely aimed at exploration. Figure 4.2 also shows that several approaches can be combined in the process of theory building, testing and extension. In this research project, only case studies and laboratory experiments will be used. Field experiments are an extension of laboratory experiments into an actual organisation [Galli91]. They have the same strengths and weaknesses as laboratory experiments, but also suffer from two additional weaknesses [Galli91]:

1. It is difficult to find organisations prepared to be experimented on.
2. It is extremely difficult to achieve sufficient control to enable replication of an experiment with only the study variables being altered.

Field experiments are, therefore, rejected for this research project.

4.6 Case Research

In this section, the case research design is outlined in more detail. The section headings are after the style used by Benbasat et al. [Benba87], since they cover the essential elements of the case research approach.

4.6.1 Deciding on Case Research

Benbasat et al. suggest that the following questions should be asked to determine the appropriateness of the case strategy:

1. Can the phenomenon of interest be studied outside its natural setting?
2. Must the study focus on contemporary events?
3. Is control or manipulation of subjects or events necessary?
4. Does the phenomenon of interest enjoy an established theoretical base?

They go on to comment that the case approach is useful when a natural setting or focus on contemporary events is needed, but not suitable when there is a strong theoretical base or manipulation of subjects or events is required. ODBMS performance does not have a strong theoretical base. Furthermore, it is a contemporary topic and would be better studied in natural settings given the many variables that may influence performance, as discussed at the end of Chapter 3.

4.6.2 Unit of Analysis

It would be reasonable to assume that ODBMS technology is still in its infancy in terms of deployed systems and total market share when compared to other types of DBMS technology. Consequently, the number of production systems is likely to be very small and developed to meet very specific problems. This suggests that the unit of analysis will be a specific project or application. The corollary is that it may be difficult to generalise from this to other applications in other organisations.

In the case approach, whilst the researcher may have less *a priori* knowledge of what the variables of interest will be and how they will be measured [Benba87], the prominent processing tasks described by Youssef [Youss93] and according to him also used by other researchers in database performance provide a useful starting point:

- What queries will be run?
- What is the relative frequency of each query?
- What is the size of the database?
- What patterns of behaviour are expected?

Furthermore, a requirements analysis to determine what the characteristics of applications are in the domains of interest would also serve to identify variables. This will be presented in the next chapter. Additional factors, specific to ODBMSs, have also been discussed by Lai & Guzenda [Lai91] and Stein [Stein92]. Lakey et al. [Lakey87] have similarly proposed some factors specific to object database applications from which the following detailed questions can be derived:

- What is the level of data sharing, nesting depth and object size?
- What is the amount of sub-structure that is accessed for each object?
- What are the most frequent operations on? Single objects? Collections of homogeneous objects? Collections of heterogeneous objects?
- What types of graph structures are used? Trees? Directed Acyclic Graphs (DAGs)?
- Do graph structures contain cycles?
- Do path traversals contain objects whose types are not known *a priori*, or are only constrained by a hierarchy of types?

The problem is finding a small number of applications that exhibit the kinds of variability just discussed [Lakey87].

From the case studies, the aim will be to look for generalisations. This research project is interested in measuring both the *quantitative* and *qualitative* aspects of ODBMS performance.

4.6.3 Single-Case vs. Multiple-Case Designs

Chaudhri & Revell [Chaud94a] proposed a multiple-case study approach to measuring ODBMS performance. This is necessary, since the variety of ODBMS applications is wide-ranging. Furthermore, multiple-case designs are useful when the intent of the research is description, theory building or theory testing [Benba87]. Additionally, a multiple-case design yields more general results, since it allows cross-case analysis [Benba87]. As stated earlier, one objective of this research is to discover generalisations.

4.6.4 Site Selection

Most organisations approached for this research were initially contacted as a result of scans in trade publications, through networking at conferences or through friends and colleagues. This follows the approach recommended in [Benba87]. In summary, the main problems experienced in finding suitable organisations were:

1. Few organisations in the UK have developed object database applications of any size. Many are still experimenting with the technology and, as yet, have not committed themselves to it.
2. Some organisations that initially indicated an interest in this research subsequently withdrew due to internal re-organisations, out-sourcing, new priorities, project being abandoned, primary contacts leaving, etc.
3. Those organisations that had developed applications were reluctant to divulge any information, e.g. even which product they were using, mainly for commercial reasons, e.g. they did not wish their competitors to know what they were doing, even though assurances were given to them about confidentiality and the benefits of this research to them.
4. Object database vendors have been unwilling to help, since the market for this technology is still small compared to relational databases [IDC94; OOS96; Stone96] and they are all competing for a share of this. Performance is a major selection factor when user's are evaluating commercial products [Rotze91]. The vendors also proved to be very ineffectual in providing any case studies themselves.

The organisations that were finally selected were chosen due to availability. The research objectives were not factors that influenced site selection, since organisations from many industry sectors are using ODBMSs, as shown by a number of surveys, e.g. [Leach95]. However, a homogeneous sample of organisations, either: (i) developing the same types of applications or (ii) from the same vertical markets or domains would have been preferable, since this research could then have focused on identifying common characteristics. The approach described in (i) was successfully used in the work on RDBMS benchmarks by Youssef [Youss93]. Unfortunately, neither approach was possible, mainly due to point 3. above. The research in this project, therefore, follows theoretical rather than literal replication [Benba87], i.e. contradictory results instead of similar results are predicted. This is confirmed by the case studies described in Chapter 6, which show wide variations from each other.

4.6.5 Data Collection Methods

In the case research approach, multiple methods of data collection are used [Benba87]. This has the benefit that it offers the opportunity for triangulation (i.e. the multiple methods of data collection lead to similar conclusions) [Benba87] and provides external validity for the research findings. According to Yin [Yin89], the sources of evidence include:

- Documentation.
- Archival Records.
- Interviews.
- Direct Observation.
- Physical Artefacts.

Many of these methods were successfully used by Youssef [Youss93]. For this research, the primary methods used were:

- **Analysis of System Documentation**
  System documentation helps to provide insight into static data structures, function definitions, design decisions, etc. [Ohkaw93]. One of the case studies described in Chapter 6 (the MMIS Project at Earth Observation Sciences), for example, provided very detailed design documents, including algorithms and pseudo-code.
- **Informal Interviews**
  For each case study, an initial interview was arranged with the primary contact within an organisation. The purpose of the interview was to explain the nature of the research and its aims and objectives. Contact was also maintained by electronic mail.
- **Data Collection**
  Following [Ohkaw93], the purpose of this activity was to gain insight into the dynamic characteristics of applications, such as access patterns, the actual numbers of each type of object used and the frequency and cost of operations

executed on them. This type of information is generally difficult to obtain, since it relies on actual execution which in certain domains has not been well documented [Ohkaw93]. Some of the organisations described in Chapter 6 provided data that they had already collected from their own performance evaluations. In other cases, the relevant organisation also provided its own data analysis.

4.7 Limitations

Examining real systems using the methods just discussed may have a number of limitations. These are discussed below.

4.7.1 Choice of Data Structures

The flexibility of an object-oriented database architecture renders a variety of design alternatives and it is possible to tune the architecture for targeted applications [Ohkaw93]. This agrees with the views of Lakey et al. [Lakey87] and Bradley [Bradl94]. Furthermore, [Ohkaw93] states that:

> "Some encodings render more compact representation than others, resulting in fewer pages to read for data retrieval and therefore more efficient performance."

As an example, Ohkawa describes the representation of floating-point numbers versus integers in GemStone. The former are stored as objects with an object table used to map their object identifiers to memory locations, whilst the latter are stored as byte sequences. As a result, floating-point numbers consume more space and require an extra level of indirection. In an experiment to test these two representations, however, she found that the differences were surprisingly small (9% for the largest data sets she used). Additional experiments also showed that the overhead of repeated function calls was 12% greater than passing a complete set of statements as a large string and, furthermore, the overhead of parsing and compiling this string was 16% more than using a pre-compiled method. This leads to the conclusion that careful analysis of data structures and frequently performed operations can result in performance improvements which, when added together, can highlight major differences between ODBMS products.

When examining actual applications, decisions regarding application design and choice of data structures will inevitably have been taken by the developers. These will affect any performance measurements.

4.7.2 Choice of Language

In ODBMSs, traversal operations specified in languages such as C++ result in a waterfall of individual object-fetches that is hard to optimise [Boncz96a]. Furthermore, a complex loop programmed in C++ with object-referencing operations inside cannot be analysed easily by the ODBMS to optimise complex traversals - the task of optimisation is the responsibility of the programmer [Boncz96a]. Such considerations do not necessarily extend to other languages.

4.7.3 Hardware Normalisation

Performance measurements are only valid for the particular versions of hardware and software used. In this research project, there is no attempt to define any hardware normalisation (or price/performance ratio) as found in benchmarks from the relational domain, such as the TP1 benchmark discussed in Chapter 3.

4.7.4 System Behaviour

For the benchmarks described in Chapter 7, this research project, following Wagner [Wagne91], assumes well-behaved systems. According to Wagner, this is one that is neither erratic nor random. However, such an assumption may be highly artificial, he goes on to say, since multi-user, multi-tasking environments using shared or distributed databases cannot be considered as having well-behaved performance, because minor changes in the environment can cause major changes in performance. Bradley [Bradl94] holds similar views. However, random fluctuations were controlled in the manner described below.

4.7.5 Random Fluctuations

When modelling using observations from either application-specific or systematic approaches, random fluctuations are likely, since uncontrollable factors affecting performance will always be present [Wagne91]. These random fluctuations can be minimised by taking multiple measurements and running experiments at periods of low system activity, such as during evenings and at weekends - the approach used by Halloran [Hallo93a], proposed by Hohenstein et al. [Hohen97b] and used for the benchmarks described in Chapter 7.

4.7.6 Single-User vs. Multi-User Benchmarks

For this research project, the benchmarks developed and described in Chapter 7 will be single-user only. This is because:

- Benchmarks should initially be run in single-user mode to measure a system's performance under optimal conditions and provide a picture of the resources required by different queries [Boral84].
- Single-user evaluations are a crucial first step for multi-user evaluations and are capable of probing significant portions of the problem space [DeWit85].
- Tests in single-user mode help to eliminate the effects of non-controllable workloads and to isolate benchmark results [Youss93].
- It is useful to understand single-user performance before attempting to measure multi-user performance [Bonne95b].

Single-user tests will also keep the problem space manageable. For example, Carey et al. [Carey94] describe some initial experiences with a multi-user version of OO7 and report considerable variability in observed results when simulating increasing numbers of users. Resource limitations discussed in detail in Chapter 7 also restrict the tests to single-user only in this research project. Future work should investigate multi-user tests.

4.7.7 Real Data vs. Synthetic Data

Lakey et al. [Lakey87] claim that a requirement of an object database benchmark is that it be run using real application data. This is because, they argue, shared sub-objects and their degree of nesting are difficult to generate randomly. Furthermore, creating data sets by hand is unfeasible because of the size of the data sets needed for testing the storage management capabilities of object databases. The advantages and disadvantages of using real data have been discussed elsewhere, e.g. [Stone85] and [DeWit85] respectively. For this research project, the aim is to examine real data sets where possible with a view to finding generalisations, as discussed earlier, but keeping in mind that certain patterns may be peculiar to the real data sets.

4.8 Laboratory Experiments

In a survey of case research papers in IS that appeared in a number of journals spanning several years, Benbasat et al. [Benba87] noted that many of the examples they encountered appeared to be stand-alone, one-shot studies and few authors indicated whether their studies were part of systematic/programmatic research plans. For this research project, case studies are one of several techniques that are used. Other techniques include: (i) laboratory experiments based on the case studies and supplemented by (ii) laboratory experiments based on published work and satisfying the following criteria:

1. The work must be based on a requirements analysis or study.
2. The work must be relevant to object databases.
3. There must be sufficient detail to implement the work.

In effect, the aim is to find examples in the literature that allow entry straight into the Case Study/Action Research path illustrated in Figure 4.2a, but without the need to actually undertake a detailed case study in each case. Based on this, three candidates emerge:

1. The OO1 Benchmark [Catte92].
2. The AFIT Wargame Simulation Benchmark [Hallo93a; Hallo93b].
3. The CITY Benchmark [Youss93].

These choices are now justified.

4.8.1 The OO1 Benchmark

This benchmark has become ubiquitous, since it is frequently referred-to in the object database performance literature. It is included here because:

1. Although this work is not based on any studies and suffers from considerable criticisms, as discussed in Chapter 3, Cattell & Skeen [Catte92] claimed that other work from the CAD domain correlated with their work. It, therefore, meets this criterion by transitivity.
2. The benchmark is primarily designed to test pointer traversals and efficient utilisation of workstation cache. As previously discussed, these are operations that ODBMSs are good at performing. Furthermore, as discussed in the next chapter, many application domains require support for fast traversals. Although OO1 was not designed to compare ODBMSs, it has highlighted considerable differences between some products, as reported by Halloran [Hallo93a]. Additionally, sometimes, testing systems with a simple benchmark can be more useful than using a complex benchmark, as shown by Dewal et al. [Dewal90]. In fact, this is precisely the approach that Halloran used: (i) testing a number of ODBMSs using OO1 and then (ii) developing a simulation benchmark for the system that performed fastest in (i).

3. Reference implementations for a number of commercial systems are available in [Hallo93b].

The OO1 benchmark is also useful in that it can serve to provide a base-line against which to compare other benchmarks or performance tests. The reason for this is that [Catte94b] claimed that, with one exception, most major ODBMS products provided similar levels of performance on OO1. Running the benchmark today would help establish whether this statement was still correct because vendors continue to extend and enhance the capabilities of their products. Furthermore, although OO1 has been superseded by other benchmarks, it still remains simple, portable and easy to understand. These are qualities that many other object database performance benchmarks lack.

4.8.2 The AFIT Wargame Simulation Benchmark

This work is described in detail in Appendix B. Briefly, Halloran [Hallo93a] developed this benchmark to test the capabilities of ODBMSs for *stochastic discrete-event simulation*. The particular example that he used was a wargame scenario consisting of a battlefield with trucks and planes. Performance results were provided for ObjectStore and a non-persistent version using C++. Form this, Halloran concluded that the overhead of using ObjectStore was very small when compared to the non-persistent version and attributed this primarily to the similarity of ObjectStore to C++. In terms of the criteria cited earlier:

1. The work is based on a requirements analysis, as reported in [Hallo93a].
2. It is relevant, since Halloran left open the research question as to whether other ODBMSs might be suitable for this type of application. Furthermore, as discussed in the next chapter, simulation is an important requirement for many application domains, such as Computer Integrated Manufacturing, Engineering, Financial and Scientific Applications. Additionally, there are no published results of the behaviour of other ODBMSs for discrete-event simulation.
3. A detailed reference implementation for ObjectStore is available in [Hallo93b].

4.8.3 The CITY Benchmark

The CITY Benchmark describes the behaviour of OLTP systems. These are not generally associated with ODBMSs. However, using the criteria cited earlier, its choice for inclusion is justified as follows:

1. The CITY Benchmark is based on in-depth studies at three of the largest computer sites in the UK: (i) a large international airline, (ii) a "Big Four" bank

and (iii) a local authority computer centre. It conclusively showed that the industry standard benchmarks from the Transaction Processing Performance Council (TPC) did not represent OLTP application behaviour [Youss93].

2. Many ODBMSs now provide SQL interfaces. Some products even support a dedicated OLTP application server engine, e.g. M.A.T.I.S.S.E. from ADB. For ODBMSs, OLTP represents a worst-case scenario, because data are well structured, of a fixed-format with simple types and little or no navigation is involved. Furthermore, Youssef [Youss93] claims that the CITY Benchmark can be applied to ODBMSs, although he did not actually implement any benchmarks to test this claim. Additionally, there have been no reported results of any OLTP benchmark applied to any of the leading ODBMS products. Therefore, there is no knowledge of the behaviour of these products for this type of application.
3. The CITY Benchmark is reasonably well described and a full source-code listing is provided in [Youss93].

4.9 Data Analysis

4.9.1 Choice of Data Analysis Method

"For many metrics, the mean value is all that is important. However, do not overlook the effect of variability." [Jain91].

As discussed earlier, a limitation of most previous ODBMS performance work is the lack of any verification or sensitivity analysis. As described in [Wagne91], the standard approach to performance modelling is the least-squares regression analysis method, with the workload variables being the independent variables and the system resources required by the workload being the dependent variables. Youssef [Youss93] used tests of variance (including one- and two-way analysis of variance) as well as a difference quotient and factor of relative change, whilst Halloran [Hallo93a] used the small-sample test of hypothesis for the difference between population means, as described in [McCla91]. This research project will also use the small-sample test of hypothesis, as two of the benchmarks implemented (OO1 and AFIT) follow from Halloran's work.

4.9.2 Condensing the Performance Results

This research project will not condense the performance results reported in Chapter 7 into a single number, for the following reasons:

- **It is Contrary to a Research Aim**
  As discussed at the beginning of this chapter, the second aim of this research project is to identify which classes of applications are more suitable for particular object database architectures. This can only be achieved by considering the results of each benchmark independently, since each benchmark is representative of the data manipulation characteristics of a different application domain.
- **The Benchmarks use Different Performance Metrics**
  The OO1 Benchmark measures response time, the CITY Benchmark measures throughput, whilst the AFIT Benchmark measures both response time and throughput. If all the benchmarks used the same performance metric, a single number would be possible. However, it would then face the next problem.
- **There are Alternative Methods to Condense Results**
  The developers of the OO7 Benchmark provided three different techniques to condense their performance results: (i) lowest number for a test, (ii) weighted ranking and (iii) geometric mean. However, this led to confusion, since a different commercial product came top in each approach.
- **It is Statistically Unfeasible**
  As mentioned in Chapter 1, this research project uses statistical techniques to provide rigour. Therefore, this author consulted a statistician, who confirmed that due to the different performance metrics, condensing the results of this research project into a single number would be statistically unfeasible.

4.10 Chapter Summary

This chapter has outlined a research design to tackle some of the limitations of previous object database performance work. The problem statement, problem space, aims and objectives of this research project have been stated.

The problem space is very large for a number of reasons:

- Object-orientation provides extensive flexibility in object database application design. There are currently no published rules on designing object database applications, similar to the normalisation and entity-relationship modelling rules for relational databases.
- Object databases differ in many respects, such as architectures, features, functionality, etc. Vendors have taken a variety of implementation choices, each of which have particular strengths and weaknesses.

- The variability of the applications that object database technology is being used for is far greater than previous generations of database technology.

For the case studies, these factors cannot be controlled, since decisions and choices will already have been taken by a particular organisation and this research will simply report the observed findings. For the laboratory experiments, it should be possible to control many factors, such as the client cache size, client-server configuration, database size, page size, etc. A full factorial experimental design [Jain91] would be appropriate in this case. However, as discussed later in Chapter 7, resource limitations meant that only a partial factorial experimental design [Jain91] was used for each benchmark in this research project. The factors that were varied and their values are discussed further in Chapter 7.

To summarise, the aims of this research project are to:

1. Study the performance of commercial object database applications.
2. Attempt to identify which classes of applications are more suitable for particular object database architectures.
3. Determine if a generic, simple and accurate performance model for object databases can be derived.

It has been argued that a combination of the application-specific and systematic approaches to ODBMS performance evaluation is required and several alternative techniques have been discussed which, when used in combination, will provide a route to theory building, testing and extension. This combination of techniques includes multiple-case studies and laboratory experiments (benchmarks).

Some of the possible limitations and problems that may be encountered have also been highlighted.

Finally, a number of statistical methods that have been successfully used by other database performance researchers were briefly mentioned. The small-sample test of hypothesis will be used to verify the benchmark performance results presented in Chapter 7.

CHAPTER 5 - Application Requirements

5.1 Introduction

This chapter presents an analysis of some of the characteristics of what Cattell [Catte94a] calls *next-generation applications* (discussed below), based upon the framework originally presented in [Ahmed92]. Whilst this framework was originally applied to engineering applications, it will be shown that other applications also have similar requirements in many areas. This is important, since most previous work on ODBMS performance has failed to identify the important characteristics and requirements of these applications. Furthermore, this analysis will serve to identify any generalisations that, as stated in the previous chapter, can assist benchmark designers.

The chapter is organised as follows. Section 5.2 discusses the choice of applications. Section 5.3 then presents an analysis of these applications using the framework originally proposed in [Ahmed92]. Finally, section 5.4 presents the chapter summary.

5.2 Choice of Applications

The choice of applications is based on some of those that Cattell [Catte94a] cites as requiring better support for complex data and complex transactions. The examples discussed here will provide sufficient variety, but also be representative enough, since the problem space for object database applications is very large, as discussed in the previous chapter. Lakey et al. [Lakey87] also point out that these new applications and the object database products themselves show greater variety than traditional applications and DBMSs and the problem in constructing benchmarks is finding a small number of example applications that cover the many kinds of variability possible, both in terms of data and operation. The application domains are:

- **Computer Integrated Manufacturing**
  Manufacturing systems are a natural choice for object-orientation, since generalised frameworks can be developed which can then be tailored for specific organisations. Furthermore, as discussed in [Adiga93a], issues such as

complexity and control, simulation, etc. are easier to support and model using object-orientation. An example manufacturing facility described in [Adiga93c] illustrates that "what if" analysis and simulation are important aspects for many manufacturing sub-systems, since the aim is to maximise both the number of finished products and the utilisation of available equipment.

- **Engineering**
  ODBMSs, as discussed in Chapter 2, have traditionally been associated with engineering applications and all the object database benchmarks discussed in Chapter 3 have been suggestive of these types of applications. However, with several exceptions, most benchmark designers have not undertaken a requirements analysis of the data manipulation and schema characteristics of engineering applications.
- **Financial**
  A data model for financial trading is complex and embodies a large number of data elements with numerous relationships [ODI96a]. Object databases enable these elements and their relationships to be stored directly without the need to decompose them into relational tables. Furthermore, given the volatile nature of the financial markets, new financial instruments may need to be modelled at short notice. This can be achieved by inheritance. Performance is also an important requirement to support split-second decisions by traders - the difference between profit and loss.
- **Geographical Information Systems**
  This is an area that has traditionally not been served well by existing database technology in a number of areas, such as support for spatial data types, e.g. points, lines, polygons, etc. Benchmarking evidence from [Arctu95] also shows the difficulties in storing GIS data in a tabular format, with possible data inconsistencies and complex joins needed to reconstruct map data. In the same paper, the alternative object-oriented representation provided significant benefits in both performance and consistency.
- **Healthcare**
  Patient healthcare data increasingly require support for rich data types, such as the ability to directly store x-ray images and detailed patient histories including medications and treatments. The latter could mean support for queries on historical data and nested list structures. A list of requirements for computerised medical records by Cheung [Cheun92] highlights the nature and complexity of modelling patient data and how semantic data models and object databases can help with this task.

- **Scientific Data**
  In her PhD Thesis, Ohkawa [Ohkaw93] investigated the use of ODBMSs for scientific data management. This covers a broad range of areas, but she initially described examples from molecular biology and chemistry and concluded that scientific data required a range of features currently available only in ODBMSs. A variety of other scientific applications is described in [Frenc90b].
- **Telecommunications**
  Many standards being developed by the International Telecommunications Union (ITU) have an object-oriented flavour. Consequently, many telecoms companies have chosen ODBMSs to store managed telecoms objects as there is no impedance mismatch between the definition and implementation of objects.

Another application domain that has received considerable attention in the literature is Office Information Systems (OISs). For example, Nierstrasz & Tsichritzis [Niers89] discuss how OO techniques, such as inheritance and part hierarchies, are very useful for capturing aspects of OISs. Furthermore, office procedures require support for flexible transactions, since they may have a life-span lasting from a few minutes to the lifetime of the office. Finally, a graphical environment is required, since office objects are represented by desktop interfaces. OISs should be included in a future domain analysis.

5.3 Domain Analysis

5.3.1 Introduction

The difference between a requirements analysis and a domain analysis is that the former is often focused on a particular system, whilst the latter is used to identify the conceptual structures shared by a class of applications [Priet91]. This is appropriate here, since the aim in this chapter is to look for some common characteristics from the domain of next-generation applications.

Domain analysis is important in developing useful modelling abstractions and is defined in [Bodne94] as:

> "... how we look at or choose to describe a system, its important features and the relations between them."

In the paper by Bodner et al. [Bodne94], their aim was to identify important features of manufacturing, but the same approach could be extended to any of the areas discussed in the previous section. In order to do this, a framework would be useful and the one developed by Ahmed et al. [Ahmed92] for engineering applications will be used. This framework uses fifteen categories (fourteen from Ahmed et al. plus one other on Data Manipulation Characteristics) and the results of the analysis are summarised in Table 5.1, with a detailed analysis presented in Appendix C. The table draws its data mainly from published work and table cells marked with a "?" indicate that insufficient information was available to draw a conclusion.

	1	2	3	4	5	6	7
Complex Modelling	✓	✓	✓	✓	✓	✓	✓
Schema Design	✓	✓	✓	✓	✓	✓	✓
Schema Evolution	✓	✓	✓	?	✗	✓	✗
Constraints	✓	✓	✓	✓	✓	✓	✓
Large Volumes of Data	✓	✓	✓	✓	✓	✓	✓
Meta-Data	✓	✓	✓	✓	✓	✓	✓
Data Sharing	✗	✓	✓	?	✗	?	✓
Versioning	✓	✓	✓	?	✓	?	✗
Inter-Client Communication	✓	✓	✓	?	?	✗	✗
Flexible Transactions	✓	✓	✓	✓	✓	✓	✗
Efficient Storage	✓	✓	✓	✓	✓	✓	✓
Computationally Complete	✓	✓	✓	✓	✓	✓	✓
Compatibility	✓	✓	✓	✓	✓	✓	✓
GUI	✓	✓	✓	✓	✓	✓	✓

Table 5.1 - Summary of Domain Analysis.

Key to columns in Table 5.1:

1	CIM
2	Engineering
3	Financial
4	GIS
5	Healthcare
6	Scientific
7	Telecoms

The paper by Bjørner [Bjørn97] also contains many references and pointers to formal domain models for a number of industries, such as transport, manufacturing, financial services, healthcare, etc. However, that level of detail is not attempted in this research.

5.3.2 Data Manipulation Characteristics

This category is concerned with identifying the data manipulation characteristics of the seven application domains described earlier. When developing application-specific database performance benchmarks, it is important to know the patterns of data access and the types of queries:

- **Computer Integrated Manufacturing**
  A recurring requirement for manufacturing systems, as discussed in [Adiga93a; Adiga93b], is that of discrete-event simulations. As mentioned in the previous chapter, this is modelled by the AFIT Wargame Simulation Benchmark and results for this benchmark are presented in Chapter 7.
- **Engineering**
  Adiga and Kolyer [Adiga93d] comment that niche applications, such as engineering, are characterised by object traversals, which are optimised through efficient data storage, e.g. reducing page faults and improved object clustering. Hurson et al. [Hurso93] also note that due to the nature of the underlying applications (typically engineering), the two most frequent operations in an object database are retrieving complex objects and navigating among related objects. Consequently, performance of an object database is best measured by how fast a large set of RAM-resident objects can be traversed, rather than measuring how long it takes to move objects from disk to RAM [Atwoo91]. The OO1 Benchmark is implemented and the results presented in Chapter 7.
- **Financial**
  According to [Chand93a], portfolio managers and traders often perform queries on temporal data. Numerically intensive procedures are also required to calculate portfolio measures. The latter was directly observed in several case studies described in the next chapter. Also, an implementation of a financial benchmark that models a portfolio management system, together with performance results, is described in Chapter 7.
- **Geographical Information Systems**
  GISs are characterised by the need to model large and complex data structures and, according to [Berg96], support for: (i) associative queries that use complex predicates containing spatial and non-spatial elements and the operations on them and (ii) navigational access, since GIS applications are mostly written in

procedural programming languages such as C or C++ in which GIS objects are accessed object-at-a-time. Also, in the GIS-domain, loading spatial features can be a very time-consuming task, depending upon the underlying representation (OO or relational), as demonstrated by the benchmarks in [Arctu95]. These characteristics have been directly observed in one of the case studies described in Chapter 6 and a GIS benchmark is implemented in Chapter 7.

- **Healthcare**
  Giffen [Giffe94] states that one of the most common operations is to "pull" a patient chart to enter a new clinical note when the patient comes in to be seen. During the visit, reference may be made to previous visits, test results, other reports, etc. Furthermore, he goes on to say, the second most common reason for pulling a chart is to follow-up on an item of information that has been received, such as a lab result, x-ray report, consultant report, third party requesting information, etc. Many of the hard facts, such as demographic information, billing information, etc. can easily be stored in a relational database. The clinical note, however, does not fit in a relational database and is usually the most useful [Giffe94]. An entire patient object could end-up being a huge object, but it would only be necessary to access individual components (1 KB - 100 KB) at a time [Giffe94].
- **Scientific Data**
  For the scientific community, there are several recurring forms of query [Frenc90b]: (i) identity to key value, (ii) identity to synonym list to key value, (iii) similarity to key value, e.g. text, number, space and/or time co-ordinates, sub-sequence in a series or sequence, proximity in a mathematical graph, (iv) recursive application of a rule, e.g. moving down a hierarchy to its tips or leaves and (v) recursive sub-component matching. Many of these are also common to other application domains, such as Financial and GIS, as highlighted in the next chapter.
- **Telecommunications**
  Managed objects have pointers to other objects, as a result of the model of the telecoms systems. Therefore, there is much relationship chasing [Perry96]. This is very similar to Engineering.

5.4 Chapter Summary

In this chapter, an analysis of what are often termed as next-generation applications was undertaken, using a framework proposed by Ahmed et al. [Ahmed92]. These example applications were:

- Computer Integrated Manufacturing.
- Engineering.
- Financial.
- Geographical Information Systems.
- Healthcare.
- Scientific Data.
- Telecommunications.

Although the framework used was originally designed to identify the requirements of engineering applications, the analysis showed that other applications also have similar requirements in the following areas:

- Complex Information Modelling Capabilities.
- Semantic Schema Design.
- Dynamic Schema Evolution.
- Rigorous Constraint Management.
- Management of Large Volumes of Data.
- Meta-Data.
- Data Sharing.
- Data Versioning.
- Inter-Client Communication.
- Flexible Transaction Framework.
- Efficient Storage Mechanisms for Fast Data Access and Retrieval.
- Computationally Complete Database Programming Language.
- Compatibility, Extensibility and Integration.
- Graphical Development Environment.

One of the research objectives outlined in Chapter 4 was to look for generalisations. The analysis in this chapter has shown that there are many common requirements amongst a range of applications, but further detailed investigation is required. In particular, how performance is determined or affected by these factors. The next chapter will describe a number of case studies to explore the data manipulation characteristics further.

CHAPTER 6 - Case Studies

6.1 Introduction

Chapter 4 presented a research design, using case studies and benchmarks, to support the following research aims:

1. Study the performance of commercial object database applications.
2. Attempt to identify which classes of applications are more suitable for particular object database architectures.
3. Determine if a generic, simple and accurate performance model for object databases can be derived.

This chapter presents six case studies, undertaken at five organisations, to investigate these three aims and to discover any relevant evidence to support them (the reader is also referred to the additional material from the case studies presented in Appendix D). The results presented below will show that aim number 1. was directly met. Aim number 2. proved difficult, since only one organisation evaluated several object database products and could provide comparative performance results. Finally, aim number 3. was even more difficult, although the evidence indicates that a generic model cannot be derived. Perhaps this was to be expected, given the exploratory nature of the case studies described in this chapter and the issues previously raised in sections 2.5 and 4.10.

Case study site selection was based on availability, as discussed in Chapter 4, since all the major vendors (i.e. GemStone, Illustra, O_2, Object Design, Objectivity, UniSQL and VERSANT) were invited to provide just one case study each, but not one of them was able to do so. The case studies can be grouped into four major sections: (i) **Financial Systems** (HOODINI Project at Nomura, Nomura Treasury Dealing System, DCx Project at Reuters), (ii) **Geographical Information Systems** (MMIS Project at Earth Observation Sciences), (iii) **Healthcare Systems** (Uncle Project at St. Mary's Hospital) and (iv) **X.500 Systems** (Messageware Directory Project at NEXOR).

The remainder of this chapter is organised as follows. Sections 6.2 to 6.7 present the six case studies mentioned above. Each case study will be divided into four sections: (i) introduction, (ii) queries, (iii) performance results and (iv) discussion. The chapter summary in section 6.8 evaluates the case studies in terms of the criteria previously described in section 3.5, to see whether any of those factors appeared within the applications described in this chapter.

Pages 123-141 cannot be published in this thesis at this time, as information from Nomura International, Reuters and Earth Observation Sciences was provided to this author on a commercial-in-confidence basis, requiring the signing of non-disclosure agreements. However, the restrictions to publication may be lifted by these organisations at some future time and the interested reader should contact this author through City University.

6.6 The Uncle Project at St. Mary's Hospital

6.6.1 Introduction

This case study was undertaken with St. Mary's Hospital, Paddington, London. The system studied was a tissue-matching application, based on the Cosmos Clinical Process Model (CCPM) and developed using GemStone and Smalltalk. According to Sherlock [Sherl94], the prototype implementation has 5,000 lines of code.

The CCPM was developed as part of a larger initiative within the National Health Service (NHS) of the UK to develop a conceptual model of all healthcare, called the Common Basic Specification (CBS), which aims to cover all aspects of running a Health Service [Cairn91]. Object-oriented modelling was selected for this task for its ability to represent complex structures [Thurs93]. The object-oriented modelling method chosen was called Ptech (also referred to as Object-Oriented Information Engineering or OOIE), described in [Marti92]. The relationship between Cosmos and CBS is that Cosmos forms the clinical view on the CBS core model through a well-defined mapping [Thurs93].

6.6.2 Queries

Determining the types of queries for this case study was very difficult, since the system was developed as a prototype, with little or no documentation available that described this information in detail. Furthermore, the queries were automatically generated from the graphical user interface, developed in Smalltalk. The current prototype has been designed to support the following activities [Fowle93]:

- Donor/recipient matching in transplantation.
- Some decision support capabilities.
- Some advanced features to support research work in immunology.

These activities have differing data manipulation requirements. Additional problems in identifying queries were caused by the client-server structure, shown in Figure 6.11. The generic model, views and database storage being managed by the server and the applications and presentation services managed by the client.

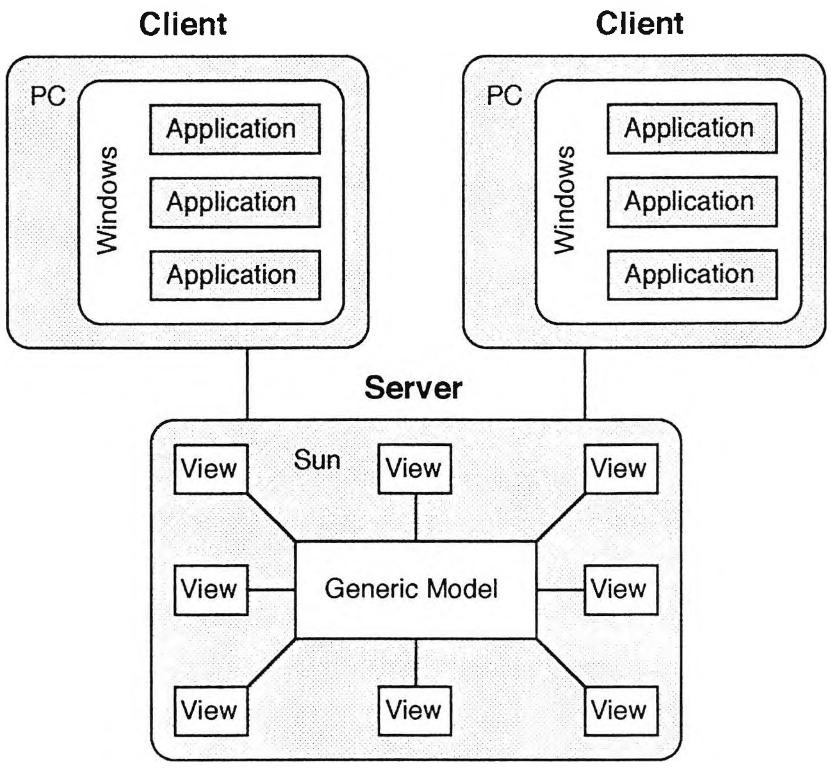

Figure 6.11 - Client-Server Architecture [Thurs93].

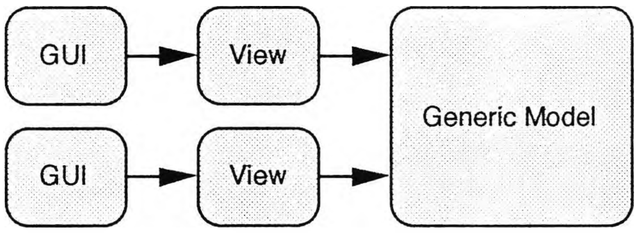

Figure 6.12 - Separating Presentation, Views and the Generic Model.

Views provide a searchlight on the shared CCPM [Fowle93]. Views are also needed because the generic model may be too abstract for user requirements or comprehension [Thurs93]. In Smalltalk, a view would be represented by a window on the client PC, which would be programmed to produce appropriate information, e.g. blood group information. Furthermore, there is a division of tasks between an application view and the CCPM when updating information. Validation is provided in the application view to ensure that legal values for data are being entered. Consequently, the mappings between the view and the CCPM can be complex [Fowle93]. To leave all the code within a window would provide poor reuse, so the approach taken was to separate the GUI code from the mapping code, as shown in Figure 6.12. This approach provides the flexibility to have multiple user-interfaces for the same application view [Fowle93] and results in a four-layer architecture. The classes defined for views and the generic model are also different; view classes have many properties and are structurally simple, whilst CCPM classes have few properties and complex links [Fowle93].

Views use a number of standard access routines defined as part of the CCPM. This simplifies access for common navigations defined across the CCPM [Cairn92a]. Fowler [Fowle94] describes views and their application in healthcare in more detail.

6.6.3 Performance Results

No performance data were available from this case study. Furthermore, the current Uncle prototype does not use any optimisations, such as indexes or structural access to attributes, as provided by GemStone (recall from Chapter 2 that GemStone allows associative access queries by using selection blocks for fast path traversal and comparison, but at the expense of encapsulation). Cairns & Fowler [Cairn92a] suggest that such optimisations should be part of the CCPM to maintain encapsulation of the CCPM framework.

6.6.4 Discussion

This case study has demonstrated that object-oriented techniques can be successfully used in patient healthcare. However, both GemStone and Smalltalk have shortcomings in directly representing the CCPM. For example, bi-directional relationships were modelled using a master-slave technique, with the master being responsible for updating both sides of a relationship. Furthermore, Ptech contains many rich modelling constructs that could not be directly represented in the prototype, but templates and mapping rules had to be developed. These deficiencies again demonstrate that applications are not designed in isolation, but are designed with the features and functionality of a particular product in mind.

6.7 The Messageware Directory Project at NEXOR

6.7.1 Introduction

This case study was undertaken with NEXOR Ltd. in Nottingham. Currently, they are not using any ODBMS products, but are considering the use of this technology for the storage management of X.500 servers (one of their business areas). The existing X.500 servers that they provide at present are based on proprietary storage technology.

6.7.2 Queries

X.500 can be described as a **White Pages Directory Service** and contains 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]. Several types of queries could be posed by users of an X.500 system:

- 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, e.g. hobbies.

The types of queries can, therefore, be quite varied and imprecise.

According to Coulouris et al. [Coulo94], data stored in X.500 servers are organised in a tree structure with named nodes, each holding a range of attributes. Searches on the tree are not only possible by name, but also by a combination of attributes. Furthermore, the name tree is called the **Directory Information Tree** (DIT), which is a virtual hierarchical data structure [SURFn95]. The entire tree with all the attributes is referred to as the **Directory Information Base** (DIB). Using distributed systems principles, only one DIB would exist world-wide, but with parts of it being located in individual X.500 servers [Coulo94].

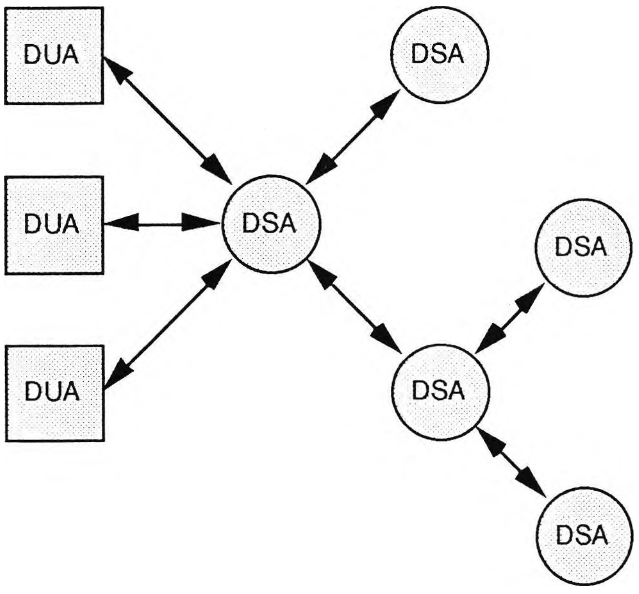

Figure 6.13 - X.500 Service Architecture [Coulo94].

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, using the terminology of the X.500 standard. Figure 6.13 illustrates one possible scenario, with a number of DUAs connected to a particular DSA and the DSA issuing requests to other DSAs to complete the DUAs requests. Communication between DSAs is through a **Directory System Protocol** (DSP), part of the X.500 recommendations.

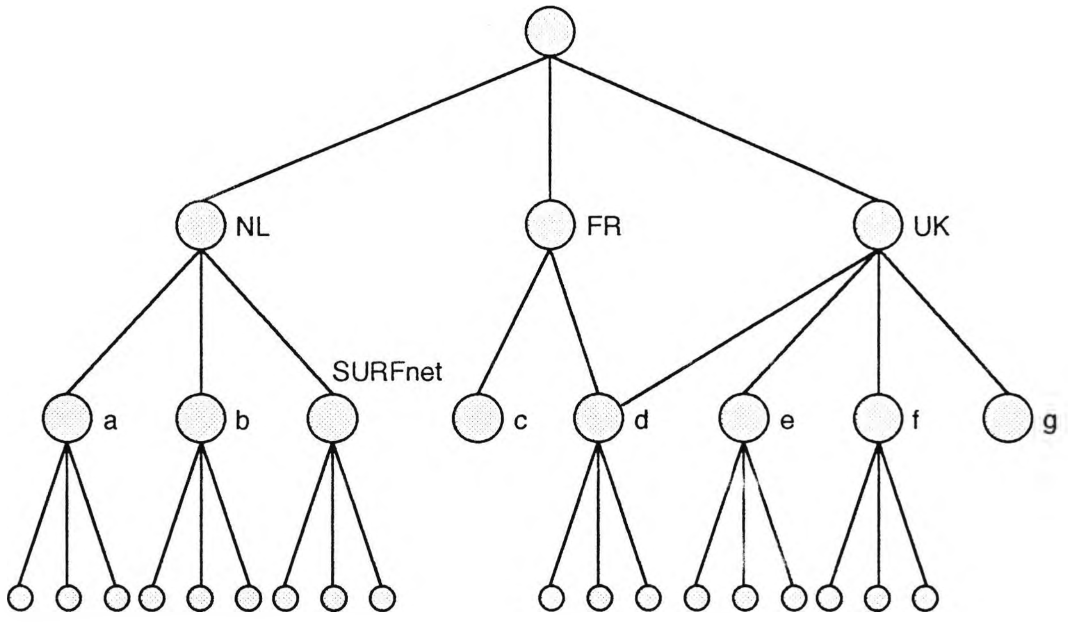

Figure 6.14 - Directory Information Tree [SURFn95].

Figure 6.14 illustrates a simplified DIT, with the **root** of the tree at the top, followed by **countries** (NL, FR, UK), **organisations** (a, b, SURFnet, etc.) and finally **people** at the leaves. There may be another level as well, called **organisational units**, but this has not been shown.

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 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 for common name (Figure 6.15), which has three values.

Attribute type	**Attribute value**
Object Class:	top
	person
Common Name:	Anonymous Nobody Other
	Anonymous N. Other
	A.N. Other
Surname:	Other
Postal Address:	...
	...
	...
	...
	...
Phone Number:	...
Fax Number:	...
E-mail:	...
Hobbies	Eating, Poetry, Crocodile-Wrestling

Figure 6.15 - Example Object Class Entry.

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 **Anonymous Nobody Other** would be the name of this node as it occurs in the DIT.

Coulouris et al. [Coulo94] describe two main methods in which a directory can 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 persons' e-mail address.
2. **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 research students in room A528 at City University.

Both a read and a 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 shown in Figure 6.13.

6.7.3 Performance Results

Some benchmarks have been developed by this author for NEXOR and several database products evaluated, as discussed in the next chapter. These benchmarks were based on data and queries provided by NEXOR.

6.7.4 Discussion

The types of navigations that would favour a general naming service (X.500 is an attribute-based name service) include [Coulo94]:

- Iterative.
- Multicast.
- Recursive server-controlled.
- Non-recursive server-controlled.

In addition, two very important issues are: (i) the use of replication to reduce the need to send requests to other DSAs and (ii) caching for fast performance. It is interesting to see that both these requirements, the model of navigation described above and the tree-like structure (DIT) fits very well with object databases. One important consideration with X.500, however, is that since it is a standard and does not include implementation details, it is conceivable to use a variety of alternative data structures to represent the DIT (which, as mentioned earlier, is a virtual data structure), that provide the same functionality defined in the standard.

6.8 Chapter Summary

This chapter has discussed six case studies. In terms of the level of maturity, i.e. the time that a system has been under continuous development, the case studies can be organised as follows, with the most mature project being at the top and the least at the bottom:

- The Uncle Project at St. Mary's Hospital.
- The MMIS Project at Earth Observation Sciences.
- The Nomura Treasury Dealing System.
- The HOODINI Project at Nomura.
- The DCx Project at Reuters.
- The Messageware Directory Project at NEXOR.

A comparison of the case studies using the set of criteria previously discussed in section 3.5 is shown in Table 6.3. Table cells marked with a "?" indicate that this factor is currently unknown.

	1	2	3	4	5	6
Data and Operations	✓	✓	✓	✓	✓	✓
Multiple Applications	✓	✗	✓	✓	✓	✓
Multiple Users	✓	✓	✓	✓	✓	✓
Multiple Databases	✗	✗	✗	✗	✗	?
Client-Server Architecture	✓	✓	✓	✓	✓	✓
Multiple Platforms	✓	✗	✓	✓	✓	?
Functional Differences	?	?	?	✓	✓	?
Tuning	✗	✓	✓	✓	✓	?
Development and Maintenance	✓	✓	✓	✓	✓	?

Table 6.3 - Comparison of Case Studies.

Key to columns in Table 6.3:

1	Uncle
2	MMIS
3	Dealing
4	HOODINI
5	DCx
6	Messageware

In further detail,

- **Data and Operations** - all the systems use representative data and operations, either because they are live systems (Uncle, MMIS, Dealing), have been benchmarked with realistic data (HOODINI, DCx) or will use real data for prototyping (Messageware Directory).
- **Multiple Applications** - the MMIS system has essentially one interface for query building, as discussed earlier. The other systems have a variety of data manipulation requirements that require different applications.
- **Multiple Users** - all of the systems are already deployed or are planned to be deployed in multi-user environments.

- **Multiple Databases** - all the systems studied use a single database. It is unknown at the moment if the Messageware Directory system may support multiple databases, since some queries may be re-directed to other servers.
- **Client-Sever Architecture** - the architectures of the systems studied use various client-server configurations and multi-tier architectures.
- **Multiple Platforms** - some of the systems (e.g. Uncle, Dealing) use mixed environments of high-end workstations and PCs.
- **Functional Differences** - this is difficult to answer in some cases, since only one product was being used. However, in other cases where a number of products were evaluated, it is apparent that this factor can be important. For example, in HOODINI, one object database was chosen in preference to another because of its in-cache performance.
- **Tuning** - with the exception of Uncle and Messageware Directory, all the other systems looked carefully at various tuning options and identified possible performance improvements based on the features provided by various products.
- **Development and Maintenance** - informally, many of the organisations have indicated that they have used OO techniques to reduce development time and hope that in the medium to long term, maintenance will become easier.

It is apparent from the findings reported in this chapter that none of the public benchmarks discussed in Chapter 3 satisfy the requirements described above. Furthermore, whilst there are similarities in the data manipulation characteristics of some systems (e.g. tree-traversals in HOODINI, MMIS and Messageware Directory), many of the systems have taken advantage of certain product-specific features to improve performance, which a generic benchmark would obviously not be able to do. Furthermore, even systems from the same vertical market have very different requirements, e.g. compare HOODINI with DCx.

Evaluating the findings against the three research aims outlined earlier, it is clear that the first aim of studying the performance of commercial object database applications has been met. The second aim of identifying which classes of applications are more suitable for particular object database architectures has been difficult to determine, since only one of the case studies provided comparative performance data. Finally, the third aim of attempting to determine if a generic, simple and accurate performance model for object databases can be derived has also been difficult to determine, although the evidence available from the case studies strongly indicates that generic high-level benchmarks are simply too difficult to develop and that there is no standard or canonical workload for object databases.

CHAPTER 7 - Benchmarks

7.1 Introduction

Chapter 4 presented a research design, using case studies and benchmarks, to support the following research aims:

1. Study the performance of commercial object database applications.
2. Attempt to identify which classes of applications are more suitable for particular object database architectures.
3. Determine if a generic, simple and accurate performance model for object databases can be derived.

The previous chapter described a series of case studies that met the first aim. However, the second aim was difficult to achieve, since only one case study provided comparative performance numbers from several object database products. The third aim was also difficult to achieve, but evidence from the case studies suggested that a generic model was not possible since, for example, applications were designed to take advantage of product-specific features.

The research design described in Chapter 4 proposed that case studies would be used as part of a systematic/programmatic research plan and would be supported by laboratory experiments to meet the above aims. Therefore, to achieve aim 2. and discover further evidence for aim 3., this chapter describes and reports the results of the following six performance benchmarks:

1. The OO1 Benchmark (Engineering).
2. The AFIT Wargame Simulation Benchmark (Discrete-Event Simulation).
3. The CITY Benchmark (OLTP).
4. The X.500 Benchmark (X.500).
5. The GIS Benchmark (GIS).
6. The OO-Fin Benchmark (Financial Trading).

The justification for the inclusion of the first three benchmarks was previously discussed in Chapter 4. The X.500 Benchmark was directly driven from the case study work undertaken by this author with NEXOR Ltd., as mentioned in Chapter 6. The GIS Benchmark was derived form a customer benchmark that was implemented by Illustra, but the source code was made available to this author. It has some similar data manipulation characteristics, discussed later, to the MMIS project at EOS Ltd. Similarly, the OO-Fin Benchmark is included, since it has some similar data manipulation characteristics, discussed later, to the HOODINI project at Nomura International.

These six benchmarks provide a cross-section of tests, since they produce some of the data manipulation characteristics found in the case studies, such as navigational access and throughput. The diversity of data manipulation characteristics among ODBMS applications also requires a larger range of tests, as suggested by Lakey [Lakey87], hence the use of six benchmarks, instead of just one. A single performance number that condenses the results of these benchmarks is not possible, for reasons previously discussed in section 4.9.2. It was not possible to develop more benchmarks based on the case study work for the reasons previously discussed in sections 2.5, 4.10 and 6.8.

The DBMS software used in this research was provided by the vendors on the basis that their products would not be named, since this research is studying the performance of competitive products. Previous researchers, e.g. [Carey93; Carey94], have also reported legal difficulties in publishing the results of competitive benchmarks. Other researchers, e.g. [Kempe95a; Kempe95b], have published results, but have anonymised the products under test. However, from the descriptions of the systems, it was easy to determine the mapping [Chaud96b]. Anonymising the products under test is, therefore, the only viable approach for this research project. The three products tested were ObjectStore, UniSQL and Objectivity/DB. These will be named as DBMS-1, DBMS-2 and DBMS-3 in the results reported below (without, of course, describing the mapping used between an actual product and its anonymised name).

The remainder of this chapter is organised as follows. Section 7.2 describes the hardware and software test parameters and configurations used for the benchmark tests. Section 7.3 presents the statistical test used to analyse benchmark results. Sections 7.4 to 7.9 present the six above mentioned benchmarks, respectively. Finally, section 7.10 contains the chapter summary. This chapter is somewhat longer than the others, since it contains graphs and tables that summarise the results.

7.2 Factors and Their Values

As mentioned in Chapter 4, it should be possible to control many factors for the benchmark experiments, such as the client cache size, client-server configuration, database size, page size, etc. A full factorial experimental design [Jain91] would be appropriate in this case. However, as discussed below, resource limitations meant that only a partial factorial experimental design [Jain91] was used for each benchmark in this research project. The results and conclusions presented below are, therefore, only valid for the particular factors and their values. The factors that were varied and their values are now discussed in detail.

7.2.1 Summary of Environment Variables

To ensure that fair comparisons between database systems are possible and results are reproducible, the following environment variables need to be reported [Larse92]:

- Hardware Platform and Operating System.
- CPU Type.
- Amount of Memory.
- Disk Type and Size.
- The Name and Version of the DBMS software.

Hardware	sun4
Operating System	SunOS 4.1.2
CPU	Dual SPARC (33 MHz)
Memory	64 MB
Disk Type	Computer International SCSI 1000
Disk Size	2 x 800 MB
Swap Space	200 MB
File System	UNIX
Compiler	SPARCcompiler SC3.0.1 (7/13/94)
DBMS Software	Objectivity/DB 3.8.6 ObjectStore 4.0.2 UniSQL 3.5.2

Table 7.1 - Environment Variables.

To this list, three other factors were added:

- Swap Space [Hallo93a].
- File System.
- The Name, Type and Version of the Compiler.

These environment variables were kept constant throughout all the benchmark tests.

7.2.2 Summary of Factors

Table 7.2 summarises the factors and their values. Most factors were kept constant for all tests. The exceptions were tests that required separate measurements for cold and warm cache numbers and/or using different database sizes.

Caching	Cold and warm cache numbers were reported separately where required by a benchmark, otherwise only a single number was reported.
Client-Server Cache Sizes	The default cache sizes provided by a particular product were used for all tests.
Client-Server Configuration	The DBMS-2 client and server processes were running on the same machine for all tests.
Clustering	The default clustering scheme provided by a particular product was used for all tests.
Database Size	The database size was dependent on the particular benchmark and DBMS internals.
Indexing	Indexing was used as required by a particular benchmark.
Journaling/Logging	Journaling/logging was enabled for all tests. Archiving for DBMS-3 was disabled.
Page Size	The page size was kept constant for all tests.
Query Programmer	The query programmer was used where it was available in preference to hand-coding.
Single/Multi-User	This research project investigated single-user performance only.

Table 7.2 - Summary of Factors.

These factors are discussed in detail below.

7.2.3 Benchmark Runs

During this research, the server with the DBMS software was also servicing other users, so it was necessary to run the benchmarks at off-peak periods, such as overnight and at weekends, to minimise the impact that other users and processes could have on the benchmark results. However, in some cases, such as lengthy database loads, this was not possible. Running the tests at off-peak periods follows the approach used by Halloran [Hallo93a] and is in contrast to the approach used by Larsen [Larse92], who tested his benchmark software in a full multi-user environment with other users and processes running during his experiments.

7.2.4 Database Loads

For DBMS-1 and DBMS-2, a database load was performed from within the benchmark program and it was possible to check the database size by examination of the database file created using simple UNIX commands. For DBMS-3, it was found that attempting to load the database from the benchmark program would take considerable time (possibly several days) and would eventually result in memory faults, even with small database sizes. Consequently, it was necessary to use the DBMS-3 bulk database load utility to populate the database. Various parameters can be passed to this tool, such as the page size and the number of pages to be created. Other DBMS-3 tools can calculate the number of pages required. Using these tools, the database size was estimated and then created, before using the bulk loader. The DBMS-3 load times are, therefore, based on the time to load the database using the bulk loader plus the time to update the database statistics plus the time to create indexes.

The DBMS-3 bulk loader requires an input file in a particular format. This was usually created by simply loading a modified DBMS-1 database and writing a small program to dump this database into a file in the format required by DBMS-3. DBMS-1 was used in this way for the OO1, GIS and OO-Fin (large database) benchmarks. For the CITY, X.500 and OO-Fin (small database) benchmarks, the benchmark code within the respective DBMS-3 programs was modified to generate the load file directly. DBMS-3 was not tested on the AFIT Benchmark, for reasons discussed in section 7.5.3.

7.2.5 Caching

Three of the benchmarks, namely OO1, X.500 and OO-Fin, require both cold and warm cache numbers to be reported. 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.

A precaution that was used was to read a large file after a benchmark test, to clear UNIX file system buffers, as described in [Nag95]. This enabled both cold and warm cache numbers to be more accurately measured for those tests that required it and is also an approach described in [Asgar97].

7.2.6 Client-Server Cache Sizes

Results reported by Hohenstein et al. [Hohen97a; Hohen97b] showed that the client cache size for one ODBMS had very little influence on its performance, whilst for another ODBMS it was very important. However, to keep the problem space manageable, this factor was kept constant for all tests in this research project, by using the default values provided by a particular product in its configuration file. This would be a reasonable decision, since without investigating caching specifically, it would be difficult to determine the best cache sizes for a particular benchmark. Furthermore, this author could not find any references to specific cache sizes for the OO1, AFIT, CITY, GIS or OO-Fin benchmarks.

7.2.7 Client-Sever Configuration

Whilst this research was being undertaken, new servers were being introduced with a new operating system in the School of Informatics at City University. Therefore, it was necessary to confine the running of the DBMS software to one server that was still running the old operating system. Furthermore, it was not possible to use a normal client-server configuration, since a client machine with the minimum memory required by the DBMS software and running the same operating system as the server was not available. To upgrade the DBMS software would have been possible, but time consuming, since there were three products, as mentioned above in Table 7.1.

Of the three products tested, DBMS-2 is the only one that uses a true client-server configuration with separate client application and database server processes. DBMS-1 uses a peer-to-peer architecture and there is no concept of a server process as such, although a lock manager process needs to be running for multi-user applications. DBMS-3 provides libraries for both single- and multi-user applications and during the tests described below, all applications were built with the single-user libraries.

7.2.8 Clustering

To keep the problem space manageable, this factor was kept constant for all tests in this research project, by using the default clustering strategy provided by a particular product. This follows the approach used by other researchers, such as Kempe et al. [Kempe95a; Kempe95b]. This would be a reasonable decision, since without investigating clustering specifically, it would be difficult to determine the best clustering strategy for a particular benchmark. Furthermore, this author could not find any references to special clustering directives for the OO1, AFIT, CITY, GIS or OO-Fin benchmarks.

7.2.9 Database Size

The OO1, AFIT and OO-Fin benchmarks provide specific values for the number of objects to be created. The X.500 and GIS benchmarks are dependent on the quantity of input data available or generated, respectively. The CITY Benchmark leaves the choice of database size open, based on available system resources. The exact database size for each benchmark will also be dependent on DBMS internals.

7.2.10 Indexing

Results reported by Hohenstein et al. [Hohen97a; Hohen97b] showed that some queries for a particular ODBMS were slower with indexes than without! However, to keep the problem space manageable, this factor was kept constant for all tests in this research project, by using indexing (normally a B-Tree), where specified by a benchmark. Specific indexing decisions for some benchmarks are also discussed in later sections.

7.2.11 Journaling/Logging

In all tests, journaling/logging was enabled for both DBMS-1 and DBMS-2. DBMS-3, however, caused serious problems, since it created log and archive files that were as large as the database itself. After some initial tests with the OO1 Benchmark, it was decided to disable archiving since, otherwise, the disk would fill-up very rapidly with numerous and possibly very large archive files, particularly if large transactions were in progress.

7.2.12 Page Size

An 8 KB page size was used as the unit of transfer between the database and application in all cases, since this matches the page size used by the operating system and was the size used by Carey et al. [Carey93] and Halloran [Hallo93a] in their benchmarking work.

Hohenstein et al. [Hohen97b] also comment that using pages smaller than operating system pages has the disadvantage that only parts of them are transferred at a time, increasing the total number of transfers.

7.2.13 Query Programmer

Where a product supported a query programmer, a benchmark was written to take advantage of it, as it was generally easier to express a query rather than trying to hand-code it.

7.2.14 Single/Multi-User

All the benchmarks measured single-user performance only, as discussed in Chapter 4.

7.3 Data Analysis

As mentioned in previous chapters, few object database benchmark efforts have used any verification, sensitivity or statistical techniques on their results. Chapter 4, however, briefly described several techniques used by some database performance researchers. The approach used by Halloran [Hallo93a] is the most relevant to this research project, since several of the same benchmarks are also explored in this research project.

The statistical test used by Halloran was the small-sample test of hypothesis for the difference between population means, as described in [McCla91]. This research project will also use this test to determine if differences exist between the performance results obtained, either:

1. Between two different databases using the same benchmark configuration or
2. Between the same database on two different benchmark configurations.

These comparisons can be performed on the mean times for benchmark operations. The standard statistical approach to performing comparisons of this nature is to form a null hypothesis which states that there is no difference between the population means, against an alternative hypothesis which states that there is a difference. In equation form, these can be represented as (1) and (2), respectively:

$$H_0:(\mu_1 - \mu_2) = 0 \tag{1}$$

$$H_0:(\mu_1 - \mu_2) \neq 0 \tag{2}$$

Although the population mean times (μ_1, μ_2) are unknown, the sample mean times are known, since each benchmark is typically performed five times and an average obtained. Equation 3 shows the formula to calculate the statistic used to perform the test of the null hypothesis versus the alternative hypothesis:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - 0}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}(\frac{1}{n_1} + \frac{1}{n_2})}} \tag{3}$$

Where n is the sample size, $\bar{x}$ is the sample mean and s is the sample standard deviation (calculated using a divisor of $n - 1$ instead of n, since this gives a better estimate of the population standard deviation). Here a two-tailed test is used, with the rejection region being calculated from the Student's t distribution as $t < -t_{\frac{\alpha}{2}}$ or $t > t_{\frac{\alpha}{2}}$ and using 8 degrees of freedom in most cases (with fewer degrees of freedom used in several cases, due to a test failure) with a level of significance (α) equal to 0.05. One of the following three conclusions can be drawn from the results [Hallo93a]:

1. If $t < -2.306$ the first benchmark result is faster than the second benchmark result.
2. If $t > 2.306$ the second benchmark result is faster than the first benchmark result.
3. There is insufficient evidence to reject the null hypothesis at $\alpha = 0.05$.

The same assumption as Halloran, regarding the population standard deviation being equal for both samples, was also used in this research project. This is because, like Halloran, random number variations were controlled by using the same random number stream for all benchmarks and system loading was controlled by running the tests at periods of low activity, such as evenings and weekends.

7.4 The OO1 Benchmark

7.4.1 Introduction

This benchmark has been described in detail in Chapter 3 along with some results for a number of ODBMSs. The main purpose of this benchmark is to measure the performance of *interactive* engineering applications, based on databases of **parts** and **connections**. The schema illustrated in Figure 7.1 shows that parts are connected "to" other parts and connected "from" other parts. The connection is the relationship between parts and also has some attributes. The cardinality of the "to" relationship is exactly three, whilst the cardinality of the "from" relationship is zero or many.

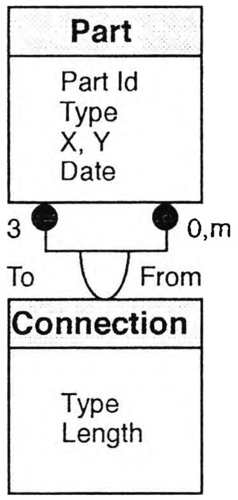

Figure 7.1 - The OO1 Benchmark Schema.

Three operations are specified (Lookup, Traversal and Insert), which are based on those that would typically be used by engineers in such environments [Catte92]. There are two database sizes: (i) a **small** database of 20,000 parts and 60,000 connections (approximately 4 MB) and (ii) a **large** database of 200,000 parts and 600,000 connections (approximately 40 MB).

There are four possible configurations of this benchmark in a client-server environment:

1. **Small Local Database** - running the application and database server on the same machine for the small database.
2. **Small Remote Database** - same as 1. above, but running the application remotely (across a network) on a different machine from the database server.
3. **Large Local Database** - running the application and database server on the same machine for the large database.
4. **Large Remote Database** - same as 3. above, but running the application remotely (across a network) on a different machine from the database server.

According to Cattell & Skeen [Catte92], most ODBMS applications require the second configuration and for the OO1 Benchmark itself, the second and fourth are required, with the others being optional. However, it was only possible to test the first and third due to the problems described in section 7.2.7. Some caution is, therefore, necessary when interpreting the results reported below, since Halloran [Hallo93a] discovered in the case of

ObjectStore, for example, that it performed slightly worse for the small local database than the small remote database. This he attributed to the fact that both the application and database processes were likely to be competing for system resources on the same machine in the first configuration. However, running application and database processes on the same machine removes network latency effects and is an approach that has been used by Dewan & Agarwal [Dewan97] and Hohenstein et al. [Hohen97a; Hohen97b].

One of the requirements of the OO1 Benchmark is that 90% of connections between parts must be randomly linked to 1% of the closest parts, where closeness is defined as a numerically similar part identifier. In his research, Halloran [Hallo93a] removed this restriction and discovered that one of the products he was benchmarking (ObjectStore) was very sensitive to this Locality of Reference (LOR). Therefore, this adds four further configurations to the above list, although Halloran only tested the small local and small remote databases with No Locality of Reference (NLOR). Results are also presented below for the small local NLOR database.

Complete results for the OO1 Benchmark are described in Appendix E, together with a statistical analysis. These results are summarised below. The benchmark was performed five times, following the approach used by Halloran. The source code for ObjectStore described in [Hallo93b] was used as the basis for all three products under test. The DBMS-2 implementation was close to the one listed in [Hallo93b], with minor changes required to replace DML statements with library interface statements. Conversion to the other two products was surprisingly easy and no major problems were encountered. Hohenstein et al. [Hohen97a] also comment that porting code from one ODBMS to another for their performance tests was a mechanical task as far as standard functionality was concerned. Similar experiences have also been reported by Shiers [Shier98].

7.4.2 Factors and Their Values

All factors were kept constant, as mentioned earlier in section 7.2.2, with the exception of the database size. This was varied as follows:

1. **Small Local Database**
2. **Large Local Database**
3. **Small Local Database (NLOR)**

Cold and warm cache numbers were also collected, as required by the OO1 Benchmark.

7.4.3 Small Local Database Results

The following table illustrates database size and load time.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	957.360	260.997	715.300
DB Size (MB)	8.523	4.500	6.070

Table 7.3 - Small Local Database Load Results (Secs).

From Table 7.3, only DBMS-2 comes close to the 4 MB database size, mentioned earlier. The DBMS-3 bulk loader gives better performance than DBMS-1 and its database size is also smaller.

Benchmark results for the Small Local Database are shown in Table 7.4 and Figures 7.2 and 7.3. Reported times are in seconds. The L+T+I metric is simply the sum of the Lookup, Traversal and Insert numbers.

	DBMS-1	DBMS-2	DBMS-3
Lookup Cold	19.020	24.478	57.724
Lookup Warm	12.549	2.812	128.813
Traversal Cold	19.968	35.415	25.942
Traversal Warm	3.916	6.445	9.119
Insert Cold	25.304	21.316	335.230
Insert Warm	26.165	4.745	342.046
L+T+I Cold	64.292	81.209	418.896
L+T+I Warm	42.629	14.002	479.979
R. Traversal Cold	28.786	35.600	31.732
R. Traversal Warm	3.598	6.305	8.650

Table 7.4 - Small Local Database Benchmark Results (Secs).

The OO1 Benchmark requires that each test is performed 10 times. The warm numbers shown in Table 7.4 represent the averages of iterations 2 to 10.

The forward traversal operation follows the "to" relationship from one object to another and is determined as follows. Since each object is connected "to" three other objects, the total number of objects visited for 7 "hops" will be:

$$\sum_{i=0}^{7} 3^i \tag{4}$$

This is equal to 3,280. The reverse traversal operation follows the "from" relationship, which is variable, as mentioned earlier. Therefore, reverse traversal results are normalised, so that they can be correctly compared with the forward traversal results. The formula for this is described in [Catte92; Hallo93a]:

$$T_{rt\_normalised} = T_{rt}\frac{N_{ft}}{N_{rt}} \tag{5}$$

T_{rt} is the elapsed time, N_{rt} is the number of parts found in a single reverse traversal measure and N_{ft} is the number of parts found in a single forward traversal measure, which is always 3,280 parts (with possible duplicates).

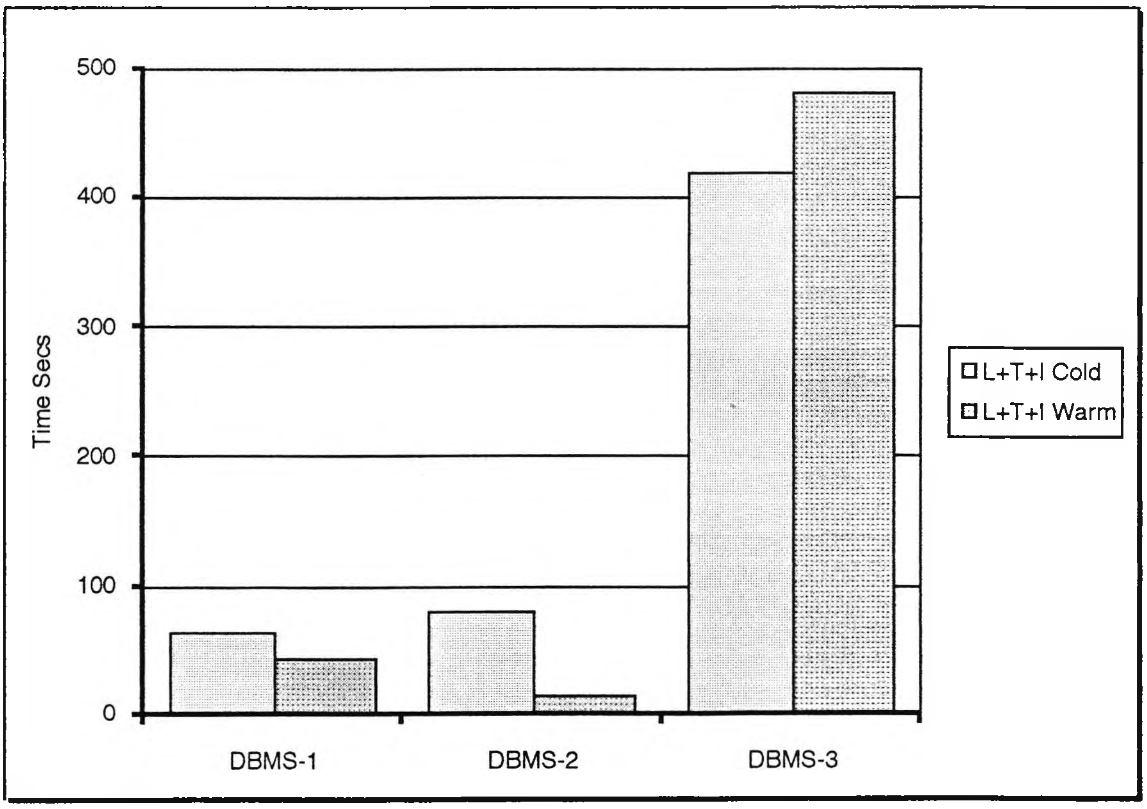

Figure 7.2 - Small Local Database Benchmark Results.

In Figure 7.2, DBMS-2 has high start-up costs (cold cache) when compared to DBMS-1, but its warm numbers are generally better. Traversals in DBMS-3 are comparable to the other two products, but inserts are very expensive and warm lookups are curiously about twice as expensive as cold lookups.

Figure 7.3 shows the L+T+I numbers averaged over the five benchmark tests and DBMS-3 is clearly experiencing memory problems, as the numbers get worse with each benchmark iteration. DBMS-3 technical support were contacted about these strange numbers and were provided with the benchmark code and data. They agreed that the results appeared strange and were able to reproduce similar numbers, but were unable to suggest any reasons for this. For DBMS-2, the traversal and reverse traversal numbers are almost identical, suggesting that the entire database may be memory resident.

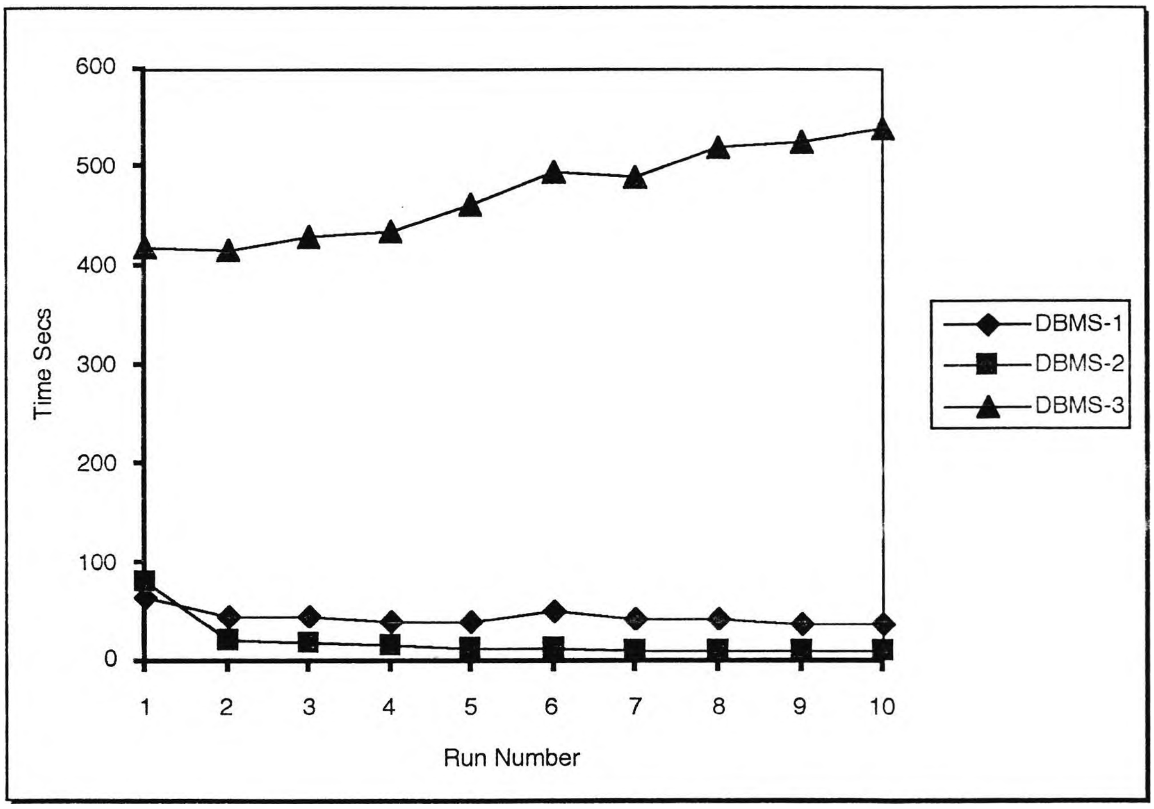

Figure 7.3 - L+T+I Results for Small Local Database.

7.4.4 Large Local Database Results

DBMS-2 comes closest again to the database size requirement (40 MB) for the OO1 Benchmark, as shown in Table 7.5. The database sizes scale-up by a factor of ten for each product when compared to their sizes for the small database. The DBMS-3 bulk loader gives very good performance when compared to the other two products. Load time for

DBMS-2, however, is very high. This is probably because the OO1 Benchmark requires the entire database load to be performed in a single transaction. When the pages are 8 KB and the objects are small, as with the OO1 parts and connections, the cost of updating a page appears high, compared to the data that have actually changed. Furthermore, both DBMS-2 client and server processes are running on the same machine and competing for resources. In contrast, the other products do far better, since there is no server process.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	15,208.068	128,895.250	10,792.000
DB Size (MB)	83.094	43.000	57.664

Table 7.5 - Large Local Database Load Results (Secs).

Benchmark results for the Large Local Database are shown in Table 7.6 and Figures 7.4 and 7.5. Reported times are in seconds.

	DBMS-1	DBMS-2	DBMS-3
Lookup Cold	48.774	176.476	100.969
Lookup Warm	18.612	141.035	210.063
Traversal Cold	53.280	257.402	70.643
Traversal Warm	33.389	240.669	110.803
Insert Cold	38.394	60.138	402.913
Insert Warm	39.171	59.960	424.082
L+T+I Cold	140.448	494.017	574.525
L+T+I Warm	91.171	441.664	744.948
R. Traversal Cold	63.419	543.407	64.873
R. Traversal Warm	51.102	339.906	106.851

Table 7.6 - Large Local Database Benchmark Results (Secs).

The performance of DBMS-2 is more dramatically affected by the database size than DBMS-1 (Figure 7.4). DBMS-3 again provides good performance on some tests, but the results show that inserts still account for the bulk of the L+T+I results. The DBMS-3 warm numbers are worse than the cold numbers in all cases this time.

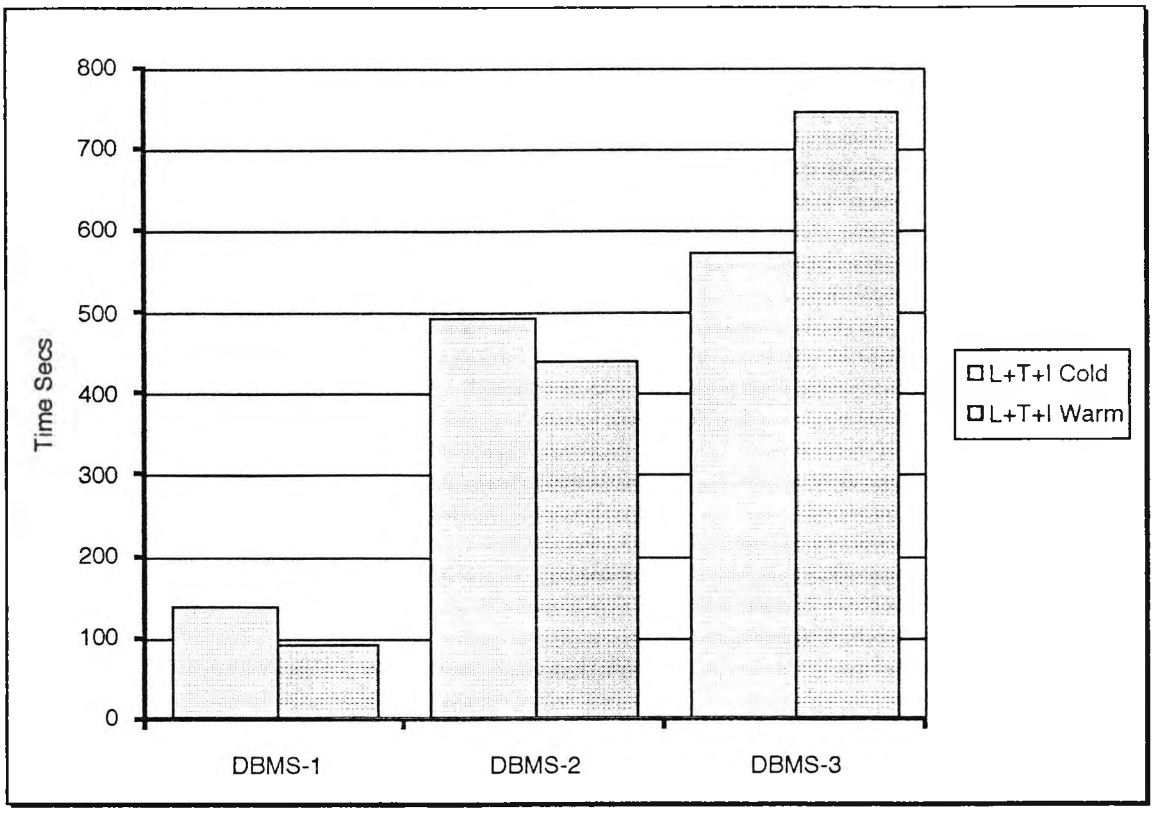

Figure 7.4 - Large Local Database Benchmark Results.

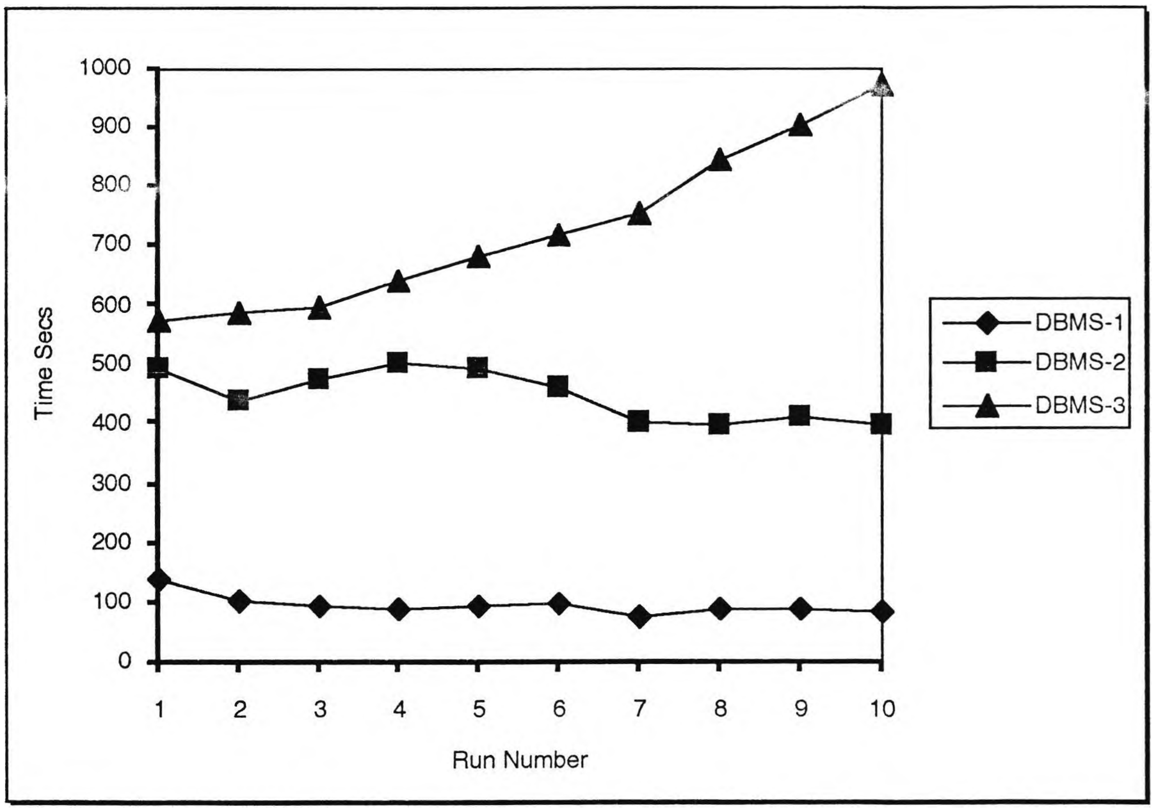

Figure 7.5 - L+T+I Results for Large Local Database.

Figure 7.5 shows the L+T+I times averaged over the five benchmark tests and DBMS-3 again shows poor use of caching. The reverse traversal numbers for DBMS-1 and DBMS-3 are very similar to their respective numbers for the forward traversal. DBMS-2, however, is far more expensive.

7.4.5 Small Local Database (NLOR) Results

The following table illustrates database size and load time for NLOR.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	982.905	223.130	702.560
DB Size (MB)	12.609	4.500	6.070

Table 7.7 - Small Local Database (NLOR) Load Results (Secs).

Both DBMS-2 and DBMS-3 maintain the same database sizes when compared to their sizes for the small Locality of Reference (LOR) database. DBMS-1, however, is 4 MB larger, even though exactly the same quantity of objects is being loaded. This indicates that database loading is an important issue with some ODBMSs, which has implications for very large databases, where disk usage can become significant. Understanding how to "pack" objects more efficiently onto pages would help under such circumstances.

	DBMS-1	DBMS-2	DBMS-3
Lookup Cold	17.516	28.700	56.650
Lookup Warm	12.547	2.535	129.579
Traversal Cold	27.124	37.264	43.956
Traversal Warm	5.132	5.867	12.539
Insert Cold	80.670	51.812	425.074
Insert Warm	73.649	8.194	422.432
L+T+I Cold	125.310	117.777	525.680
L+T+I Warm	91.327	16.596	564.549
R. Traversal Cold	29.208	39.272	43.004
R. Traversal Warm	6.993	5.634	14.814

Table 7.8 - Small Local Database (NLOR) Benchmark Results (Secs).

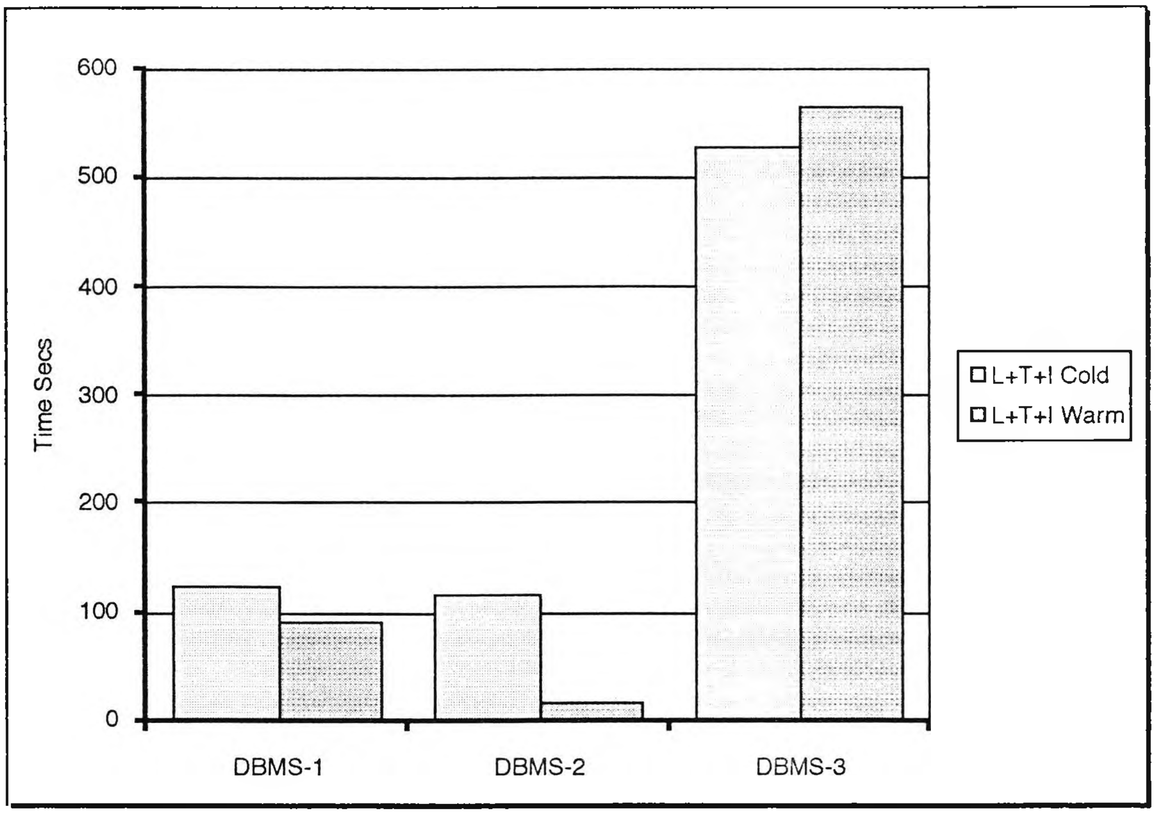

Figure 7.6 - Small Local Database (NLOR) Benchmark Results.

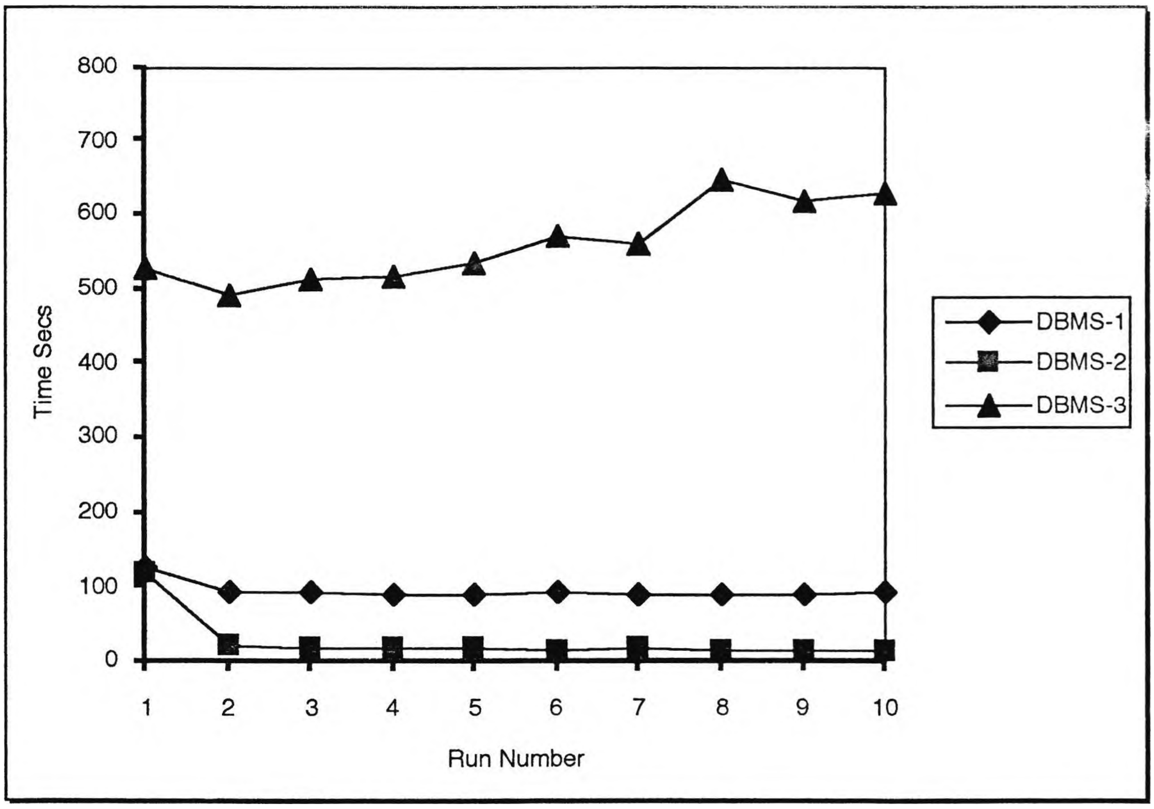

Figure 7.7 - L+T+I Results for Small Local Database (NLOR).

Benchmark results for the Small Local Database (NLOR) are shown in Table 7.8 and Figures 7.6 and 7.7. Reported times are in seconds. The diagrams show that DBMS-1 is far more sensitive to locality of reference than the other two products. DBMS-2 produces slightly worse cold numbers, but the warm numbers are comparable to its locality of reference results. This must again be due to its ability to cache the entire small database in memory. DBMS-3 again performs worse overall. Reverse traversal numbers for all three products are similar to their respective forward traversal numbers.

Raw results and a statistical analysis are presented in Appendix E. Some of the results are significant at $\alpha = 0.05$.

7.5 The AFIT Wargame Simulation Benchmark

7.5.1 Introduction

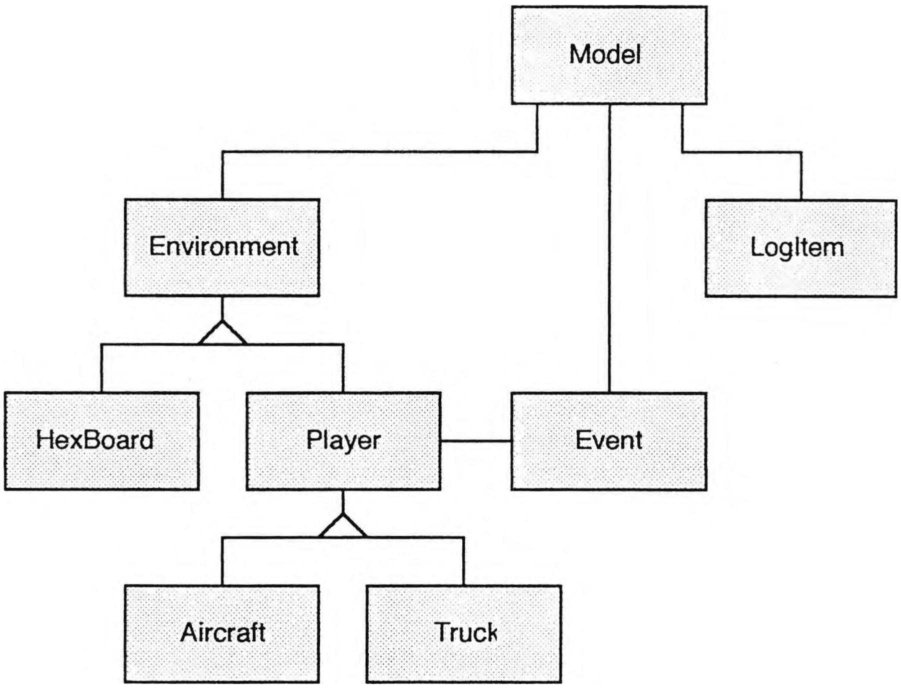

Figure 7.8 - The AFIT Wargame Simulation Benchmark Schema.

This benchmark was developed by Halloran [Hallo93a] as part of his Master's Thesis and a full source code listing for ObjectStore is provided in [Hallo93b]. Its purpose and operations are described in detail in Appendix B. Briefly, the benchmark was designed to test the ability of an ODBMS to directly run a discrete-event simulation model. Simulation is an important requirement for many application domains, as described in Appendix C.

The particular environment being modelled by this benchmark is a wargame scenario, consisting of a battlefield with aircraft and trucks.

Figure 7.8 illustrates the schema. **Environment** and **Player** are virtual classes. **HexBoard** is a passive simulation component, whilst **Aircraft** and **Truck** are active. There is a containment relationship illustrated between **Model** and the three classes Environment, **Event** and **LogItem**. A 1:1 relationship exists between Player and Event, since Event holds schedule details for all Aircraft and Trucks.

The benchmark design deliberately separates the various components that support the user interface from the database and simulation objects. In this way, it would be possible, with minimum effort, to replace either the database or the user interface. In fact, for the results that are reported in the following sections, it was very easy to replace the DBMS and leave the user interface intact. This is the qualitative measure of this benchmark.

In his thesis, Halloran reports results for this benchmark for the small database that he defines and due to time constraints was unable to extend his work to either the large database or the other ODBMS products that he was testing. His work will be revisited and extended in this section (although obviously not all the same ODBMSs are under test). Other justification for the inclusion of this benchmark has been discussed in Chapter 4.

Four possible benchmark configurations are defined:

1. **Small Local Database** - running the application and database server on the same machine with 1,000 trucks, 500 aircraft and a 50 x 50 hex board.
2. **Small Remote Database** - same as 1. above, but running the application remotely (across a network) on a different machine from the database server.
3. **Large Local Database** - running the application and database server on the same machine with 10,000 trucks, 5,000 aircraft and a 100 x 100 hex board.
4. **Large Remote Database** - same as 3. above, but running the application remotely (across a network).

Due to the restrictions previously discussed in section 7.2.7, only results for the small local database can be reported. Furthermore, the large local database was not attempted, since the benchmark uses a Graphical User Interface (GUI) to enter test parameters, that requires continuous monitoring and user interaction. It was found that even running the small local database tests consumed considerable time. Extending the benchmark with a batch capability for larger databases should be a future enhancement. The benchmark measures seven major operations:

1. **Model Creation** - the elapsed time to create a new model.
2. **Scenario Creation** - the elapsed time to create a scenario.
3. **Simulation Execution** - the simulation is run for one hour with the time slice set to 60, 600, 1800 and 3600 seconds.
4. **Simulation Throughput** - the simulation is run as fast as possible with the time slice set to 60 seconds.
5. **Version Creation** - the time to create a complete copy or new version of a model.
6. **Map Creation** - the time to create the map.
7. **Report Creation** - the time to create the summary report.

Furthermore, whilst the movements of Aircraft and Trucks are not particularly interesting, Halloran argues that the load on the system is constant and, therefore, should provide a good indicator of how an ODBMS is able to cope with the running of a simulation model directly within the ODBMS.

7.5.2 Factors and Their Values

All factors were kept constant, as mentioned earlier in section 7.2.2. Only the **Small Local Database** was tested.

7.5.3 Small Local Database Results

	DBMS-1	DBMS-2
Model Creation	0.482	0.140
Scenario Creation	71.682	3.496
Hr Run (TS=60)	7,582.355	621.042
Hr Run (TS=600)	7,941.710	537.571
Hr Run (TS=1800)	8,225.547	532.404
Hr Run (TS=3600)	7,685.126	526.138
Throughput	2.104	0.198
Version Creation	178.372	22.384
Map Creation	9.936	12.812
Report Creation	2.094	1.312

Table 7.9 - Small Local Database Benchmark Results (Secs).

The benchmark design does not provide any distinction between cold and warm numbers and does not require these to be reported separately. The Small Local Database results are presented in Table 7.9 and Figure 7.9. Reported times are in seconds.

Results for DBMS-3 are not available, as the schema for this benchmark is more complex than the other benchmarks and it was felt that creating a dump program would consume considerable time and effort. Similar problems for generating inter-related data for an object-relational database have also been reported by Asgarian et al. [Asgar97].

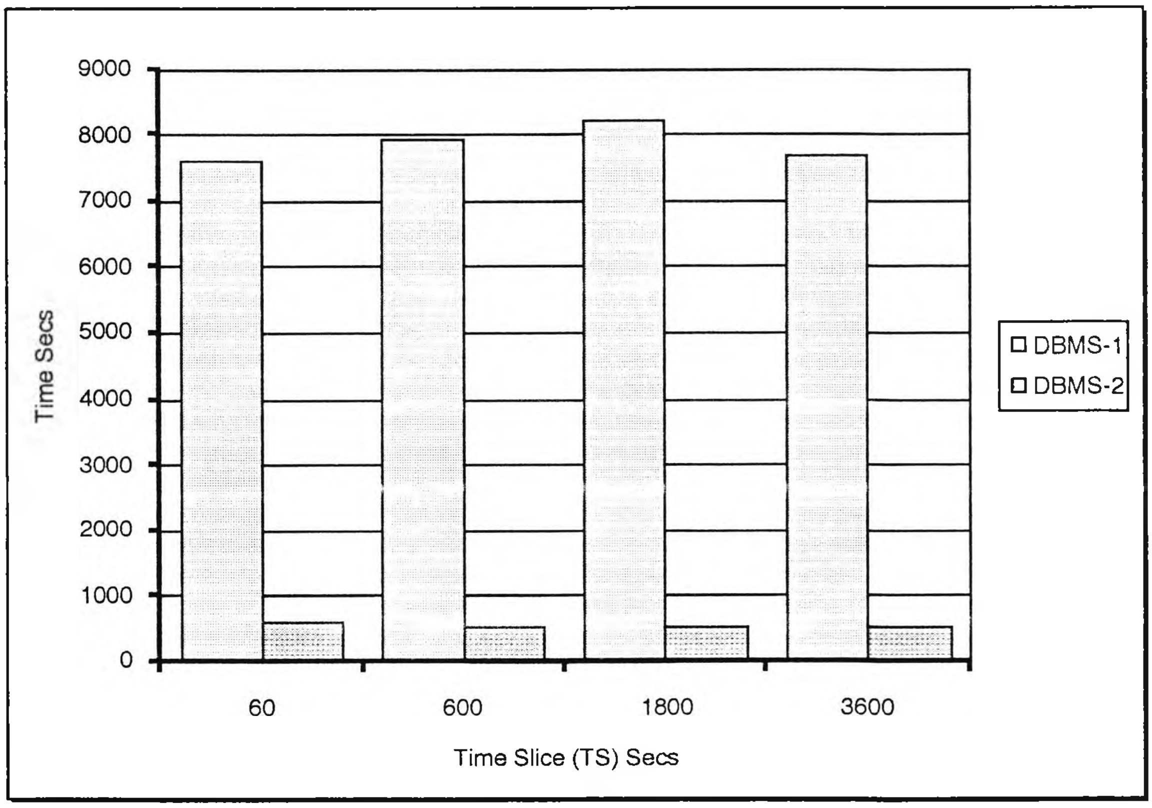

Figure 7.9 - Small Local Database Benchmark Hour Run Results.

The results reported for this benchmark by Halloran [Hallo93a] showed that ObjectStore and a non-persistent (C++) version of the benchmark produced very similar results, leading Halloran to conclude that ObjectStore was well suited for use in simulations. Clearly, the results above show very big differences between DBMS-1 and DBMS-2. One reason for the observed performance of DBMS-1 could be due to the slow updates/writes, as discussed in the last chapter. Frequent writes are needed, since the simulation model constantly updates active simulation players. From the above results, DBMS-1 is less well suited for this type of application than DBMS-2, despite the small size of the database. Further investigation is recommended to determine the reasons for the large differences.

Raw results and a statistical analysis are presented in Appendix F. Some of the results are significant at $\alpha = 0.05$.

7.6 The CITY Benchmark

7.6.1 Introduction

The CITY Benchmark was developed by Youssef [Youss93]. It is based on studies of OLTP systems at three of the largest computer sites in the UK: (i) a large international airline, (ii) a "Big Four" high street bank and (iii) a local authority computer centre. The results of these studies showed a common pattern of OLTP behaviour at these organisations, although they were in totally different business areas. According to Youssef, the OLTP model emerged after analysis of more than 40 Million transactions, 4,800 discrete applications and 5,000 relational tables. The work also showed that the industry-standard TPC benchmarks did not represent OLTP behaviour.

The CITY Benchmark uses five tables, with simple data types (char, varchar, integer, floating point, etc.). The table names, number of attributes and row sizes are shown in Table 7.10.

Table Name	No. of Attributes	Row Size (bytes)
DB100	7	100
DB200	14	200
DB300	21	300
DBUPD	14	200
DBINS	14	200

Table 7.10 - The CITY Benchmark Tables and Sizes.

The relationships between the tables are illustrated in Figure 7.10, using the notation commonly seen in Entity-Relationship (ER) modelling. DBINS is not illustrated, since it has no relationships with any of the other tables and initially starts with no rows, but rows are added during the benchmarking process.

The CITY Benchmark source code listed in [Youss93] is written in C and Embedded SQL (ESQL). Three programs are described:

1. Create Tables Program.
2. Load Tables Program.
3. Benchmark Transactions Program.

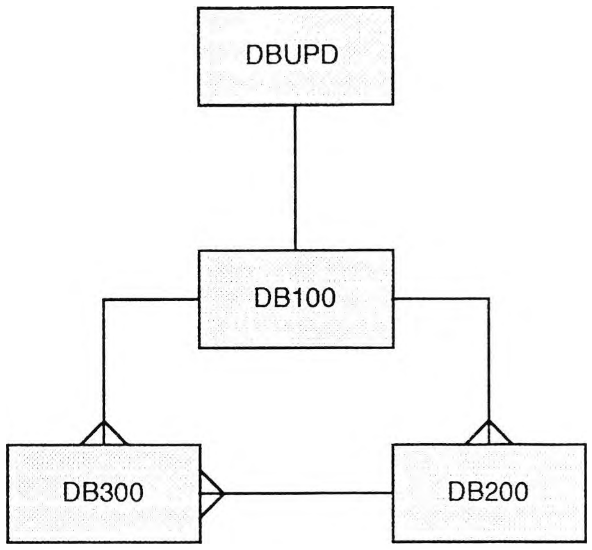

Figure 7.10 - The CITY Benchmark Tables and Relationships [Youss93].

Translation of the source code to DBMS-3 ESQL was very straightforward, requiring minimum amendments. Similarly, a C++ version for DBMS-3 was easily derived. It was decided to write these two versions for DBMS-3 to determine whether any performance differences between the C++ and ESQL versions existed.

Translation of the source code to DBMS-1 and DBMS-2 versions was a little more difficult, requiring more care and attention, since no previous implementation of this benchmark on an ODBMS has been reported. The approach used in this research project was to map a table to a class and each row to an object instance. In the case of DBMS-1, each class was stored in a separate container, since this is the only mechanism available for indexing in this product and is also the approach used by the DBMS-1 SQL interface (not tested on this occasion, since the SQL libraries provided by the vendor would not link with the compiler used in this research project). In the case of DBMS-2, an extent was defined for each class over which the indexes were created. However, only one segment was used. This latter decision was possibly a major reason for DBMS-2's poor performance, discussed shortly. Future work should investigate storing each extent in a separate segment.

7.6.2 Factors and Their Values

All factors were kept constant, as mentioned earlier in section 7.2.2, with the exception of the database size. This was varied, as discussed in the next section. The benchmark was also implemented on DBMS-3 using both C++ and ESQL to test different language bindings.

7.6.3 Benchmark Results

In his thesis, Youssef proposes using table sizes of 1 Million rows, but subsequently recognises that the actual number generated for a benchmark run will depend upon available system resources. Consequently, it was decided to use five different sizes, to test scalability, as illustrated in Figure 7.11.

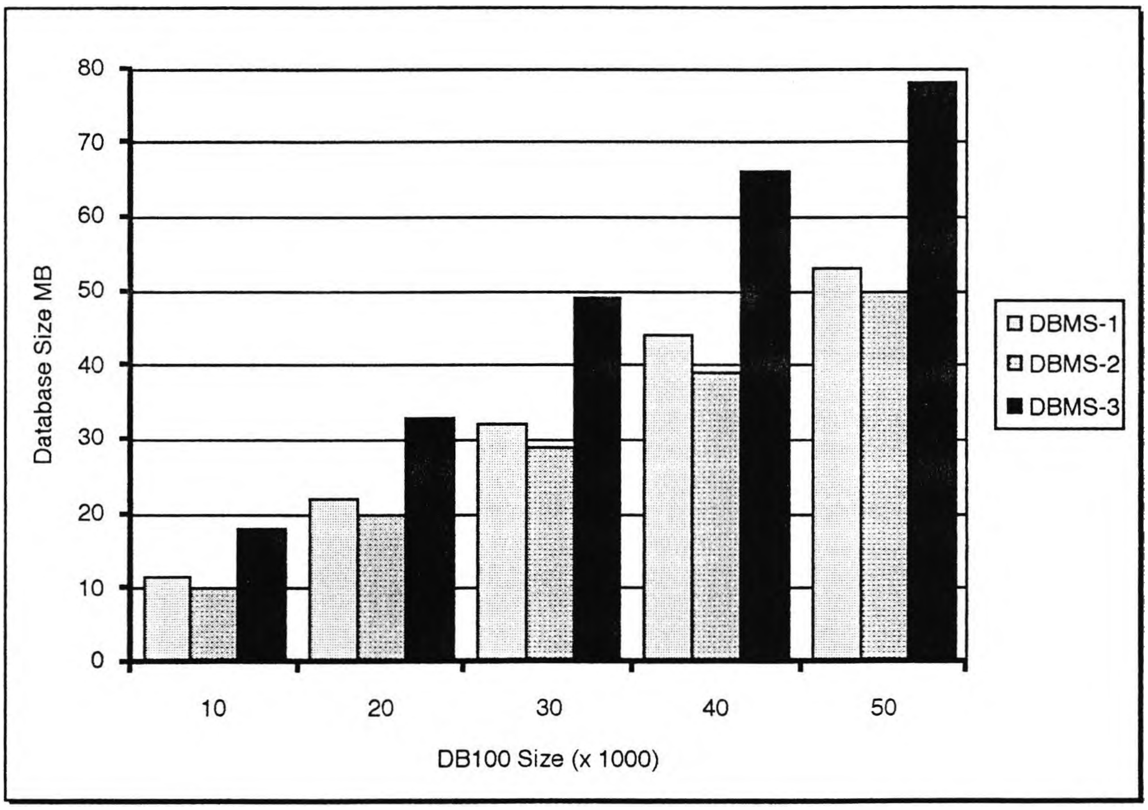

Figure 7.11 - Database Size.

Both DBMS-1 and DBMS-2 have similar storage requirements. For DBMS-3, the same database was used for both the C++ and ESQL applications. In his thesis, Youssef undertook a one-way Analysis of Variance (ANOVA) test of several time periods (900, 3,600 and 9,000 seconds) and found the differences to be statistically insignificant. Consequently, it was decided to use runs of one hour (3,600 seconds). Using this approach, it was possible to test two different database sizes of a product with five runs per database size, giving ten overnight runs. For example, starting a set of tests at 21:00 hours, ten runs would be completed by early next day. The total time was slightly variable, since not each benchmark run would complete to exactly one hour and, as mentioned earlier, each test was separated by reading a large file to flush file system buffers, adding to the overall time.

Table 7.11 shows the total number of rows/objects for each database size (marked 10, 20, 30, 40 and 50 in the extreme left-hand column).

	DB100	DB200	DB300	DBUPD	Total
10	10,000	10,000	10,000	10,000	40,000
20	20,000	20,000	20,000	20,000	80,000
30	30,000	30,000	30,000	30,000	120,000
40	40,000	40,000	40,000	40,000	160,000
50	50,000	50,000	50,000	50,000	200,000

Table 7.11 - Database Size.

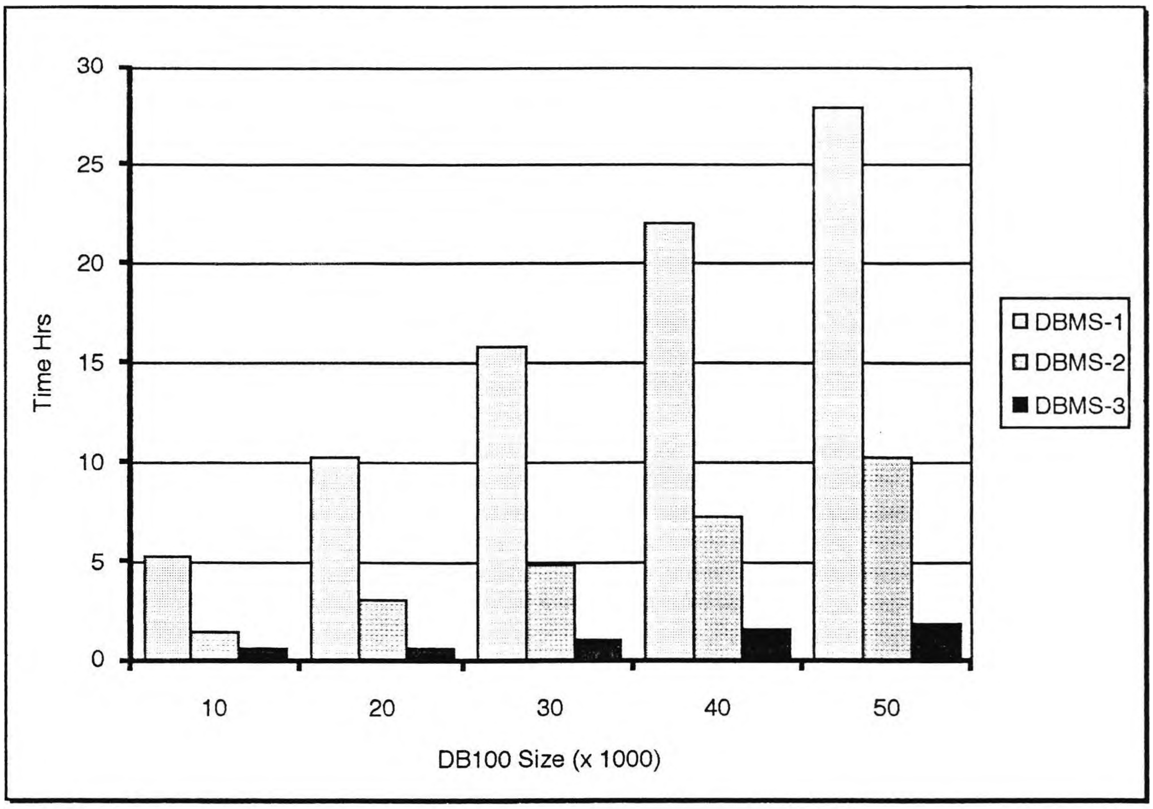

Figure 7.12 - Database Load Times.

Figure 7.12 illustrates large differences in load times among the three products. One of the requirements of the CITY Benchmark is that indexes are created *before* the database is loaded. This seems unusual, since the indexes can easily be created after the load is completed. Updating indexes whilst objects are being loaded must be more expensive in DBMS-1, hence the observed numbers. Problems were encountered with both the C++ and ESQL versions of DBMS-3 whilst trying to load the database. DBMS-3 was running short of workspace to update the indexes. Consequently, it was necessary to use the bulk loader again. The CITY Benchmark load program for DBMS-3 was modified to create a load file in the required format for the bulk loader, rather than attempting to load the database directly. After the file was loaded, the utility to update database statistics on classes/tables was used and then finally the indexes were created. There could be several reasons for the superior load times demonstrated by DBMS-3:

1. Indexes are created *after* the database is loaded. This is necessary, due to the workspace problems just discussed.
2. Using the bulk loader, all rows in DB100 are loaded, followed by all rows in DB200, etc. This might be more efficient than loading one row at a time for each table, as described in the CITY Benchmark.

Future work should investigate the creation of indexes for both DBMS-1 and DBMS-2 *after* the respective databases are loaded.

The CITY Benchmark does not distinguish between cold and warm numbers, although database caching is also available in many RDBMSs. This could be a future enhancement, but the implementation used for this research project remains as true to the original as possible. The results of the benchmark tests are illustrated in Figure 7.13. Clearly, DBMS-1 gives far superior performance. Both the DBMS-3 C++ and ESQL applications give a fairly constant performance level, whilst DBMS-2 gives such poor performance that it barely registers on the graph below. This result is a little surprising, since it shows that ODBMSs can provide high throughput for OLTP applications, but the choice of ODBMS is critical.

Since OLTP is not usually associated with ODBMSs and to get a slightly better picture of performance, some additional code was added to the CITY Benchmark transactions program for each product. The data gathered were on soft and hard page faults (no physical I/O and physical I/O, respectively).

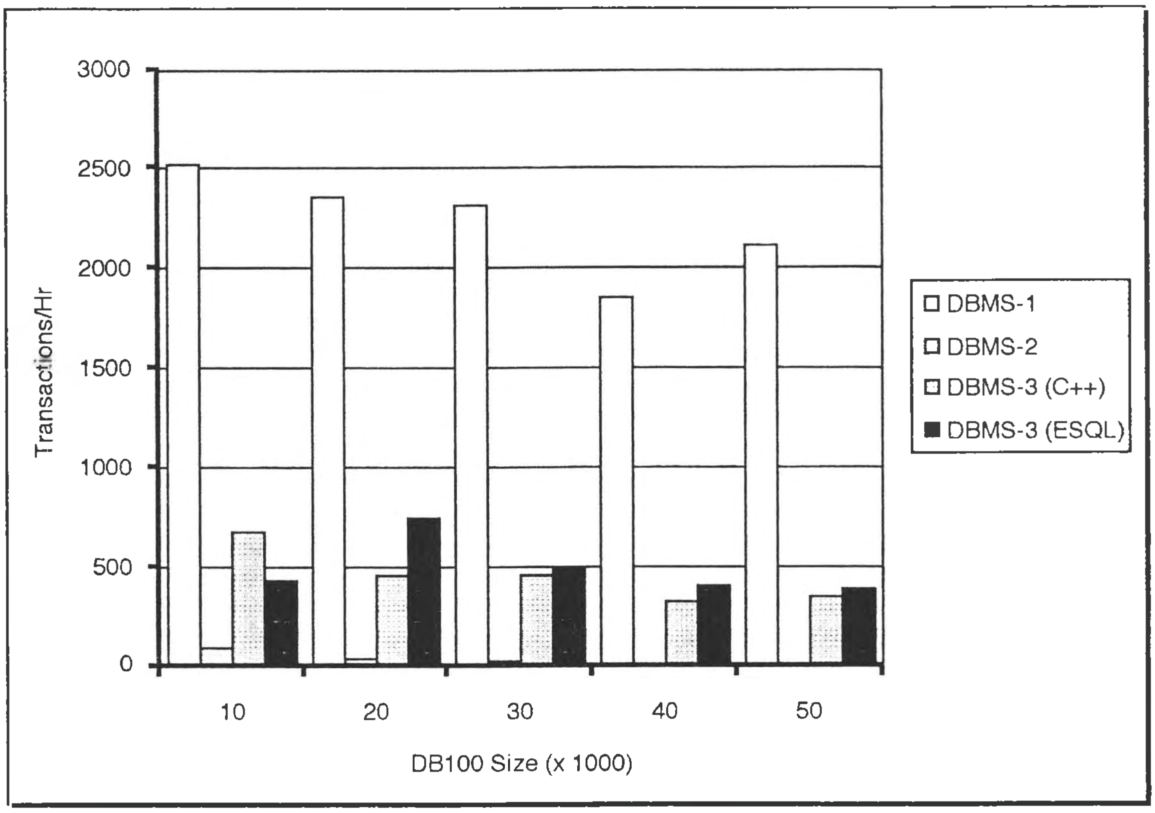

Figure 7.13 - Transactions/Hour.

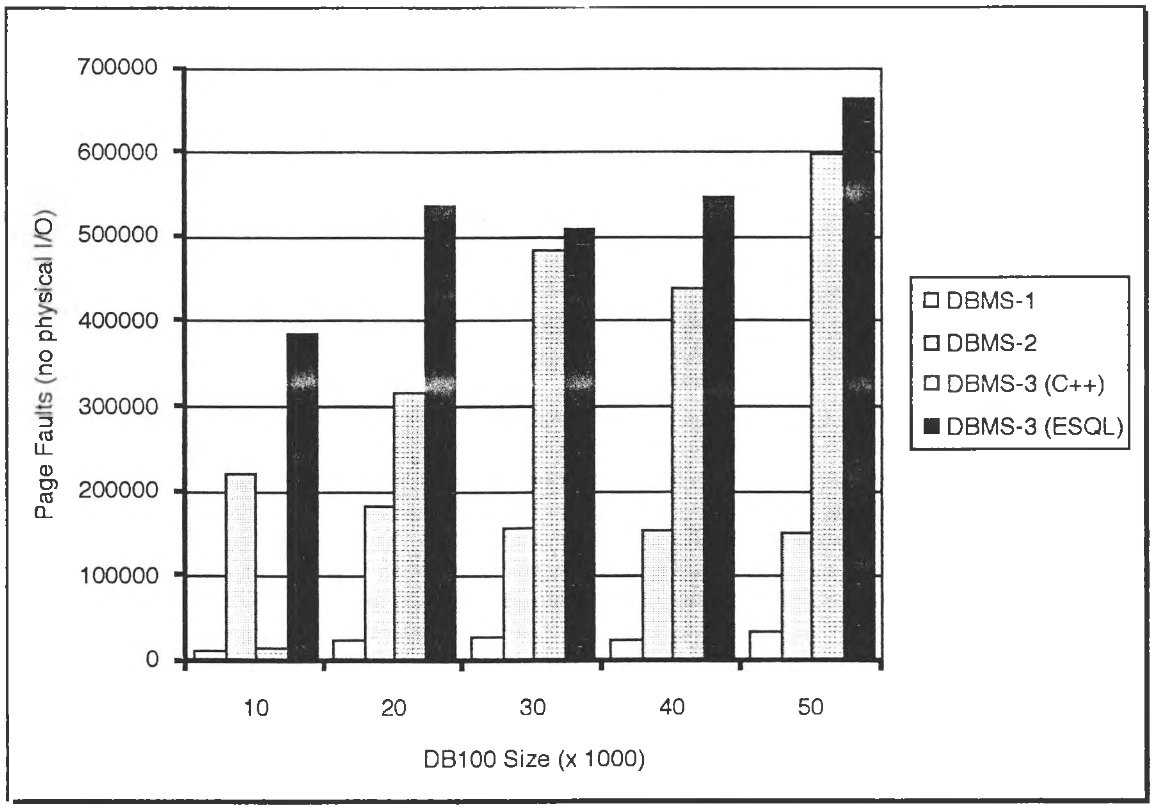

Figure 7.14 - Total Number of Page Faults (no physical I/O).

Figure 7.14 shows that DBMS-2 is performing many more page faults than DBMS-1. Interestingly, the DBMS-3 C++ and ESQL applications have very differing results. The latter is always more costly with page faults. This demonstrates that the choice of language interface has performance implications - something previously observed on the Nomura Treasury Dealing System case study described in Chapter 6.

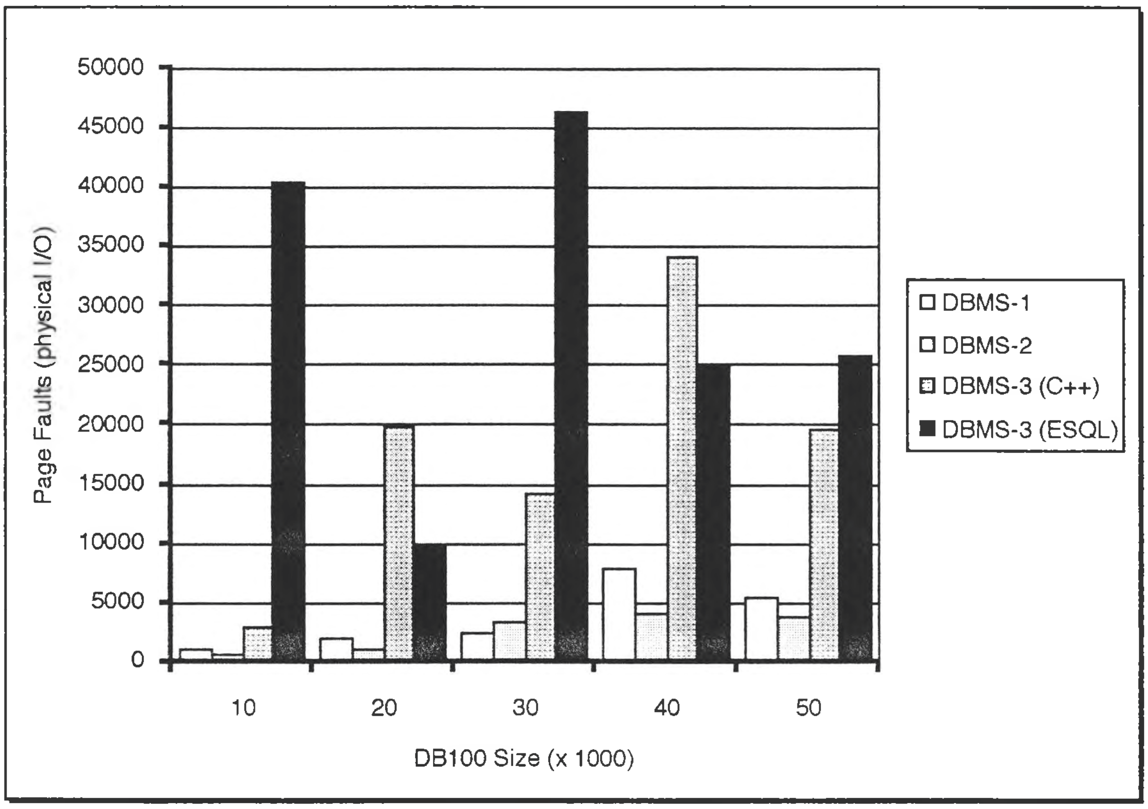

Figure 7.15 - Total Number of Page Faults (physical I/O).

Figure 7.15 illustrates hard page faults. Both DBMS-1 and DBMS-2 perform far less of these than DBMS-3. The DBMS-3 results differ very widely for the C++ and ESQL versions - there is no clear pattern. Further investigation is recommended here.

Raw results and a statistical analysis are presented in Appendix G. Some of the results are significant at $\alpha = 0.05$.

7.7 The X.500 Benchmark

7.7.1 Introduction

One of the case studies presented in the previous chapter was with NEXOR Ltd. This company is currently investigating the development of an X.500 directory system on top of an ODBMS. The case study described some of the characteristics of X.500 and, from the discussion, an ODBMS was well suited to this type of application.

NEXOR were initially interested in trying some simple queries. They provided a data file and test script to his author, based on data and tests they currently use with their proprietary X.500 system. The data file was 4 MB in size and comprised data for several thousand nodes. This quantity of data would be representative of a medium-sized company, for example, which has its own X.500 server.

The tests NEXOR provided used string comparisons, consisting of exact match, partial match and soundex queries on surname fields. So, example queries could be to find all X.500 entries with the surname "smith" (exact match), "s*m*th" (wildcard search, with any characters matching the "*") and "~smith" (soundex). Since none of the products under test provided support for soundex queries, this was omitted in this research project.

Benchmarks were written for each DBMS to load the node data into a database. Each node was indexed, using a B-Tree index, on the surname field, whilst the remaining fields of node data where held as an ordered list. Two types of queries were then investigated: **exact match** and **wildcard**.

For the exact match, several surname values were hard-coded in the benchmark to represent the first node entry in the file, the last node entry in the file and several others chosen arbitrarily. A surname value was also used that was not present in the data file to represent a node that did not reside on the current server (and in which case, a request might be sent to another server, for example). A surname was chosen at random from the list of hard-coded values. Several of the DBMSs had enhancements to their query programmer to manage some string comparisons. DBMS-2, for example, allows the use of the C string comparison function "strcmp", which is specially recognised by its query optimiser.

The wildcard queries were similarly hard-coded with a set of values. These values were "*A*", "*E*", "*I*", "*O*" and "*U*". One of these values was selected at random for a query. DBMS-1 cannot use indexes for this type of query in its query programmer.

Similarly, DBMS-2 is unable to manage this type of query well, since the C string comparison function "strstr" is required, which its query optimiser has no knowledge of. In both these cases, the entire list of surnames must be searched (worst-case scenario), so a value that was not from the above list of values was not used, since the entire database must be searched anyway. DBMS-3 supports an OQL which can manage this type of wildcard query, but the following results show that its performance gets worse with warm cache numbers again, indicating memory management problems.

7.7.2 Factors and Their Values

All factors were kept constant, as mentioned earlier in section 7.2.2. The database size was dependent on the input file. Cold and warm cache numbers were also collected, as discussed below.

7.7.3 Benchmark Results

Table 7.12 shows that both DBMS-1 and DBMS-2 are more economical with disk space than DBMS-3. Overall, however, there is a high overhead in storing the 4 MB of raw data. The DBMS-3 database load was again not possible from the C++ program, but a load file needed to be generated from the NEXOR data file, which was read into the database using the bulk loader.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	103.710	204.520	319.890
DB Size (MB)	8.766	7.750	13.086

Table 7.12 - Load Results (Secs).

	DBMS-1	DBMS-2	DBMS-3
Exact Cold	0.510	2.812	1.518
Exact Warm	0.100	0.130	0.274
Wildcard Cold	10.546	33.916	56.933
Wildcard Warm	9.746	5.060	200.015

Table 7.13 - Benchmark Results (Secs).

In Table 7.13, reported times are in seconds. Cold and warm numbers are reported, since caching frequently accessed node data is an important requirement for X.500 directory systems, as discussed in section 6.7.4.

Figures 7.16 and 7.17 show the tabulated results graphically. DBMS-1 is faster for both exact and wildcard cold queries, with DBMS-2 suffering high start-up costs again. However, the warm cache numbers for DBMS-2 improve more dramatically than the other two products. The performance of DBMS-3 on the exact cold and warm tests is comparable to the other two products, but its warm cache numbers for the wildcard query are very high.

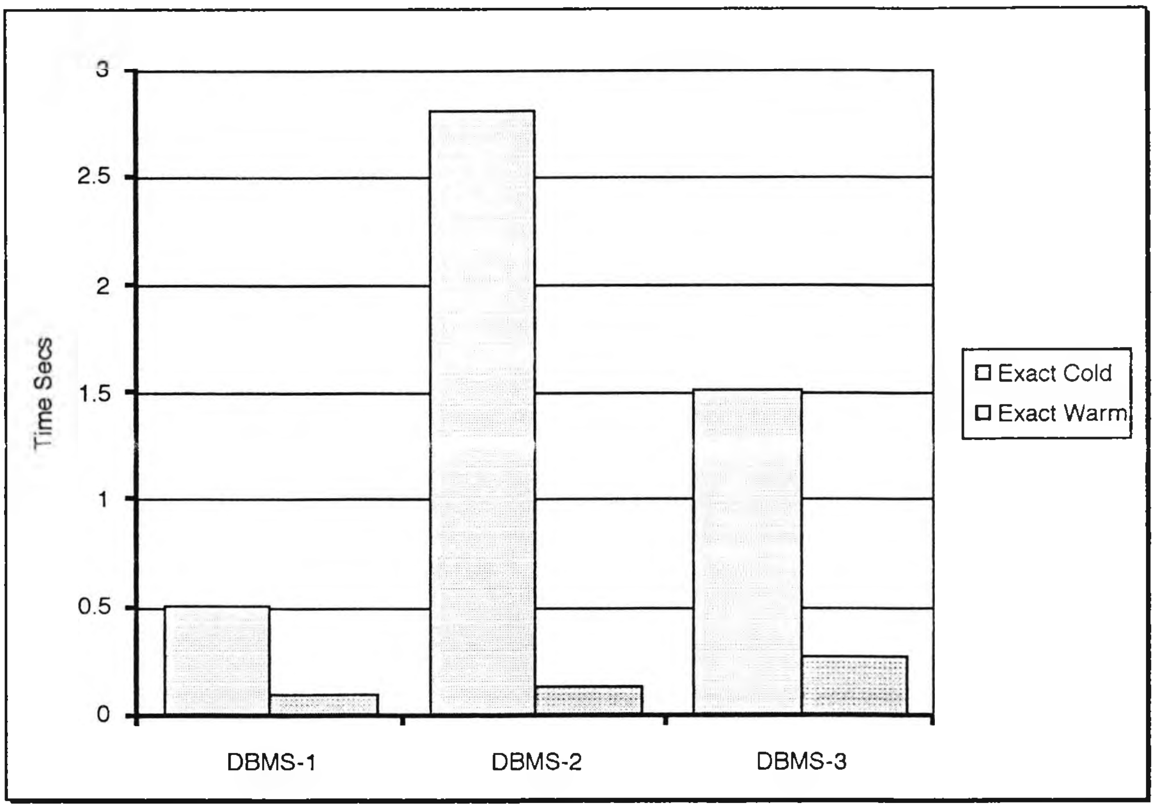

Figure 7.16 - Exact Results.

These preliminary results show that for small data sets, an ODBMS and ORDBMS can manage exact match queries in under 1 second - NEXOR would eventually like to scale to a system where it's possible to search 1 Million node entries in under 1 second. Wildcard queries are more expensive and indicate that a dedicated text search component in conjunction with a DBMS would be more appropriate. The Verity text search engine, for example, can be integrated with ObjectStore. Similarly, the Informix Universal Server has plug-in datablade modules from Excalibur, Verity and Picdar that provide text content search.

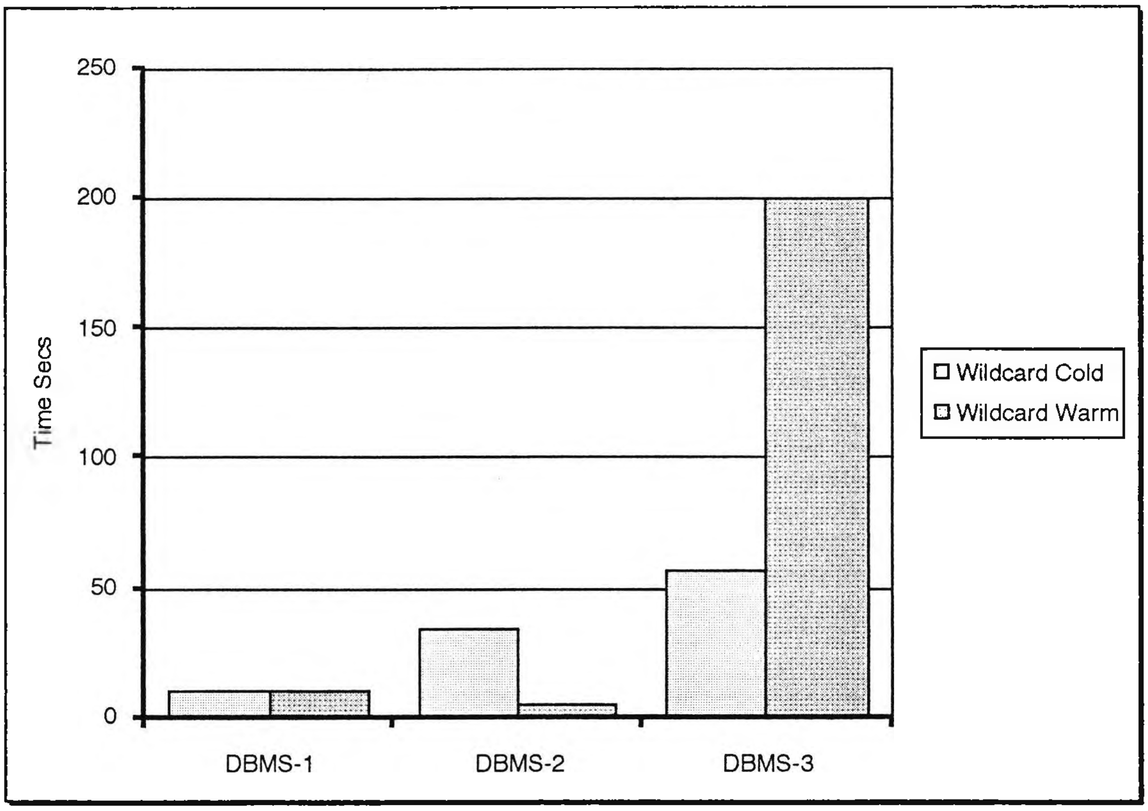

Figure 7.17 - Wildcard Results.

Raw results and a statistical analysis are presented in Appendix H. Some of the results are significant at $\alpha = 0.05$.

7.8 The GIS Benchmark

7.8.1 Introduction

This benchmark was originally developed by Illustra Information Technologies (now part of Informix) for a potential customer in the United States. The potential customer was interested in creating a yellow-pages type service with a mixture of spatial and textual information. A user of this system would be able to enter a set of keywords, e.g. 'restaurant', 'take away service', 'no smoking', etc. as well as a geographical region of interest, e.g. 'within 2 miles of my home', to be able to pose queries of the form 'show me all the non-smoking restaurants with a take away service within 2 miles of my home'. The database would be constructed by collecting information about businesses and their locations using spatial co-ordinates expressed as Latitude and Longitude points.

The original aim of the benchmark was to demonstrate that Illustra was able to handle this type of query better than RDBMSs or RDBMSs with spatial extensions. However, this type of querying is perfectly suited to ODBMSs as well and similar queries may be possible in the MMIS system in the future, e.g. 'show me all the active volcanoes over 14,000 feet in the pacific rim'.

The benchmark comes with several data generation programs and query scripts. One of the programs takes the number of businesses and generates locations for them according to a binomial distribution, using either x and y co-ordinates (to represent a point) or pairs of x and y co-ordinates (to represent a Minimum Bounding Rectangle or MBR). The keywords are drawn from a large file and each business is allocated a set of these. The size of this set is also binomially random, below a maximum value. Table 7.14 shows the parameters for the benchmark used by Illustra.

Total Number of Businesses	200,000 +/- 20,000
Spatial Area	100,000 x 100,000
Total Number of Keywords Available	50,000 +/- 2,500
Number of Keywords per Business	~17
Total Number of Keywords Used	3,400,000

Table 7.14 - GIS Benchmark Parameters.

The benchmark script developed by Illustra consists of extended SQL statements and overloaded functions that try to simulate the types of queries that users would pose to the database. Since Illustra supports various indexing structures depending upon the datablade that is being used (datablades have been discussed previously in section 2.4.6), two different versions of the benchmark schema are described. The first uses the following:

```
CREATE TABLE VendorTmp (
     Location   Pnt         NOT NULL,
     VendorId   int         NOT NULL,
     Keywords   setof(text) );
```

This schema supports a B-Tree index on the Keywords, which are actually stored in a backing table. The second version is:

```
CREATE TABLE VendorTmp (
     Location   Pnt         NOT NULL,
     VendorId   int         NOT NULL,
     Keywords   doc );
```

This creates a D-Tree (document tree) provided by the document datablade. According to Illustra, using this version of the schema provides an order of magnitude performance improvement over the B-Tree approach.

The total number of queries is 30 for the first schema and 41 for the second and comprise combinations of spatial and textual predicates. The benchmark requirement is that the total number of tuples returned is 1100-1200 and that any single query does not return more than 100 tuples. There is no requirement in the benchmark documentation to measure cold and warm numbers separately and elapsed time and number of queries per second are reported for 1 to 5 concurrent users.

Converting the benchmark to ODBMSs was reasonably easy, with the overloaded SQL functions being represented as C++ member functions. The B-Tree schema was adapted, since none of the three products used in this research supported a D-Tree. The "Pnt" data type was replaced with pairs of x and y co-ordinates, since the code to check for overlapping MBRs was available from the R-Tree example program developed in the GiST research project [GiST97], whereas Illustra supports "Box" and "Circle" data types and operations. It was felt that using overlapping MBRs was a reasonable approximation.

When implementing the schema, it was decided to use two object classes: one to hold the vendor MBR co-ordinates and vendor identifier and a second to hold keywords with each keyword holding a list of references to those vendors that had that keyword. This is similar to implementing a backing table, as used in Illustra. Since some queries are by keyword, it was felt that this approach would provide fast access to vendors, as keywords can be indexed. For those queries involving keywords and spatial co-ordinates, a search on keywords will produce a small working set, which can then be queried on spatial co-ordinates. This follows the approach used in the MMIS project at EOS Ltd. Finally, for those queries that consist entirely of spatial co-ordinates, a search of the entire database would be needed anyway, since no spatial indexing was available in the three products under test.

Since there was insufficient detail in the benchmark documentation to establish how the concurrent users were simulated, the benchmarks were confined to just one user in keeping with the aims of this research project, outlined in Chapter 4.

7.8.2 Factors and Their Values

All factors were kept constant, as mentioned earlier in section 7.2.2. The database size was dependent on the quantity of data generated, as discussed in the next section.

7.8.3 Benchmark Results

The parameters used are shown in Table 7.15. The number of businesses was reduced to 100,000 and the number of keywords to 25,000 with all other parameters remaining the same, giving a data file of 11 MB. The reason for this was that the load process became very lengthy using the original parameters. This is because as each keyword is read from the input file, it is necessary to check whether that keyword is already in the database. Furthermore, since the number of vendors that have a particular keyword is not known in advance, each time that a new vendor is added to the set of vendor references for a keyword, the set for that keyword becomes larger, which means that the DBMS needs to move it in virtual memory if there is insufficient space in the current location.

Total Number of Businesses	100,000
Spatial Area	100,000 x 100,000
Total Number of Keywords Available	25,000
Number of Keywords per Business	~17
Total Number of Keywords Used	1,700,000

Table 7.15 - Benchmark Parameters.

The following table illustrates database size and load time.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	104,514.997	108,820.267	2,340.940
DB Size (MB)	29.977	21.000	29.391

Table 7.16 - GIS Database Load Results (Secs).

Table 7.16 shows the high cost of database loading for both DBMS-1 and DBMS-2 in comparison to DBMS-3. DBMS-3 was loaded by dumping a DBMS-1 database, as previously discussed. The results appear to support the assumption that memory relocation is taking place with the other two products, hence the observed times. If the data are already in a suitable format, however, the load is comparatively fast, as in the case of DBMS-3.

DBMS-2 is clearly the top performer, as illustrated in Table 7.17. There is a detailed breakdown of the results by the type of query (i.e. keyword, spatial, keyword and spatial) in Appendix I, although the benchmark only requires the overall time and throughput (number of queries per second) for the complete set of queries to be reported, which have been tabulated below.

	DBMS-1	DBMS-2	DBMS-3
30 Queries (Secs)	113.918	36.338	430.837
Throughput	0.263	0.826	0.070

Table 7.17 - GIS Database Benchmark Results (Secs).

One of the major problems encountered with this benchmark was trying to get the number of returned objects to be consistent with the actual requirements. Since keywords are generated randomly, it is difficult to know in advance which keywords actually appear. Furthermore, the spatial co-ordinates (pairs of x and y co-ordinates) are even more difficult to determine, since they are also generated randomly. Therefore, to pose intelligent queries that return the stated number of objects becomes very difficult. Use of actual spatial data and the context in which they are used would help in this case.

Raw results and a statistical analysis are presented in Appendix I. Some of the results are significant at $\alpha = 0.05$.

7.9 The OO-Fin Benchmark

7.9.1 Introduction

Dewan & Agarwal [Dewan97] describe the OO-Fin Benchmark for object-relational trading systems. The original goal of this benchmark was to measure the performance and scalability of Persistence against a proprietary in-house solution built by Morgan Stanley. Persistence is an example of an object mediator [Thomp93] and provides automatic mapping of objects to relational tables. The benchmark simulates portfolio analysis operations found in financial trading systems and was designed to answer the following three questions [Dewan97]:

1. **Read Penalty** - is the overhead for mapping relational information into objects acceptable?
2. **Cache Benefit** - what is the performance benefit for using an object cache to navigate object structures rather than querying the database?

3. **Cache Scalability** - how is the performance of an object cache affected as the number of objects in the cache are increased?

For pure ODBMSs, the first question would obviously not apply. For ORDBMSs, the first question will apply if a relational storage manager is being used and objects need to be "flattened" to save them into the database.

Three classes are defined, with the relationships between these classes illustrated in Figure 7.18. These classes represent [Dewan97]:

- **Portfolio** - total investments for a particular customer.
- **Position** - profit and loss in a particular financial instrument.
- **TaxLot** - result of a trade for recording profit and loss on a transaction basis.

The cardinality of the relationships is that a Portfolio is associated with 20 Positions and each Position with 50 TaxLots.

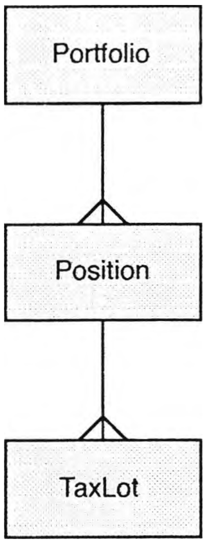

Figure 7.18 - The OO-Fin Database Schema.

The structure of these classes is illustrated in Table 7.18.

Class Name	No. of Attributes	Instance Size (bytes)
Portfolio	10	133
Position	16	222
TaxLot	41	452

Table 7.18 - The OO-Fin Attributes and Sizes.

A detailed breakdown of the attributes is not provided, other than indicating that keys are used over which indexes are created. Also, database sizes are not specifically reported, except the largest one, which comprises 100 Portfolios, 2,000 Positions and 100,000 TaxLots, giving a size of approximately 44 MB. Cold and warm measurements are taken to read a Portfolio object and navigate through all its Positions and TaxLots in a depth-first manner to simulate a calculator operation traversing portfolios to determine total risk [Dewan97].

In implementing the OO-Fin Benchmark on DBMS-1, DBMS-2 and DBMS-3, two database sizes were used: (i) a **small** database of 10 Portfolios, 200 Positions and 10,000 TaxLots and (ii) a **large** database of 100 Portfolios, 2,000 Positions and 100,000 TaxLots. Furthermore, each object instance was allocated a unique identifier to allow indexing, although an index was only created on Portfolio objects, since embedded references (pointers) to sub-objects are supported by all three of the products under test. Since the types of the attributes for each class were unknown, an appropriate-sized filler was used after the approach used on the DCx project at Reuters. It was decided to test cold and warm measures for two operations. Firstly, a **lookup** operation to find a random portfolio and traverse all its sub-objects and secondly a **traversal** operation to find all portfolios in a database and traverse all their sub-objects. These are analogous to the Trader Browser and Portfolio Browser operations used on the HOODINI project.

7.9.2 Factors and Their Values

All factors were kept constant, as mentioned earlier in section 7.2.2, with the exception of the database size. This was varied as follows:

1. **Small Local Database**
2. **Large Local Database**

Cold and warm cache numbers were also collected, as required by the OO-Fin Benchmark.

7.9.3 Small Local Database Results

DBMS-1 and DBMS-2 have similar database sizes (Table 7.19). The DBMS-3 load time is high, since this was one of the few occasions where it was able to load objects directly from the C++ program, rather than using the bulk loader. Clustering in DBMS-3 should also be optimum - for each portfolio, its positions are created and for each position, its taxlots. This should provide fast access to the entire sub-tree for any portfolio object. This would be in contrast to the bulk loader which would load all portfolios, followed by all positions and finally all taxlots.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	34.630	59.430	496.002
DB Size (MB)	4.953	5.000	5.953

Table 7.19 - Small Local Database Load Results (Secs).

Reported times in Table 7.20 are in seconds. From this table, the in-cache performance of DBMS-2 is again evident. DBMS-3 is also faster than DBMS-1 on the warm traversal - one of the few cases where it has performed better than another product. The results also agree with the OO1 Benchmark results - the speed of traversals in DBMS-3 is comparable to the other two products.

	DBMS-1	DBMS-2	DBMS-3
Lookup Cold	1.832	7.908	5.756
Lookup Warm	1.413	0.795	2.529
Traversal Cold	9.636	15.654	37.378
Traversal Warm	5.113	1.429	2.058

Table 7.20 - Small Local Database Benchmark Results (Secs).

Figure 7.19 shows that DBMS-3 cold numbers are better than DBMS-2 this time, but once the objects are loaded into the cache, DBMS-2 performs better than the either of the other two products.

As mentioned earlier, the DBMS-3 warm numbers are better than DBMS-1 (Figure 7.20).

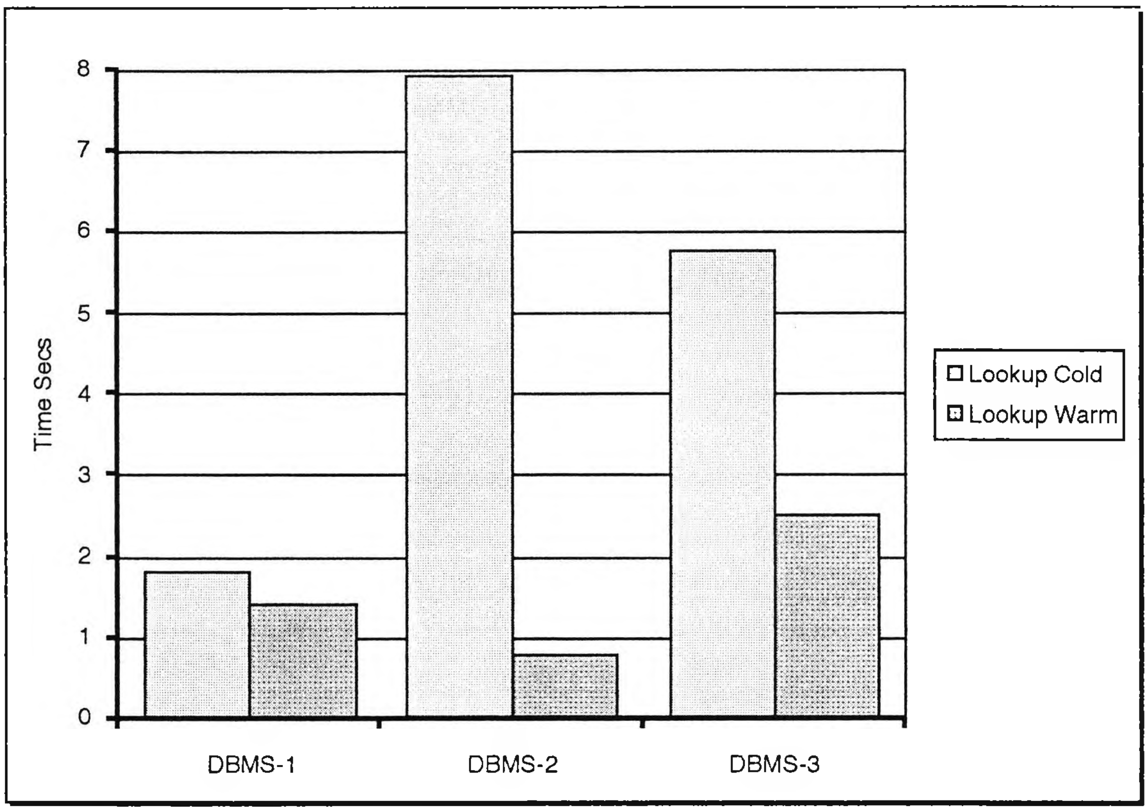

Figure 7.19 - Small Local Database Benchmark Lookup Results.

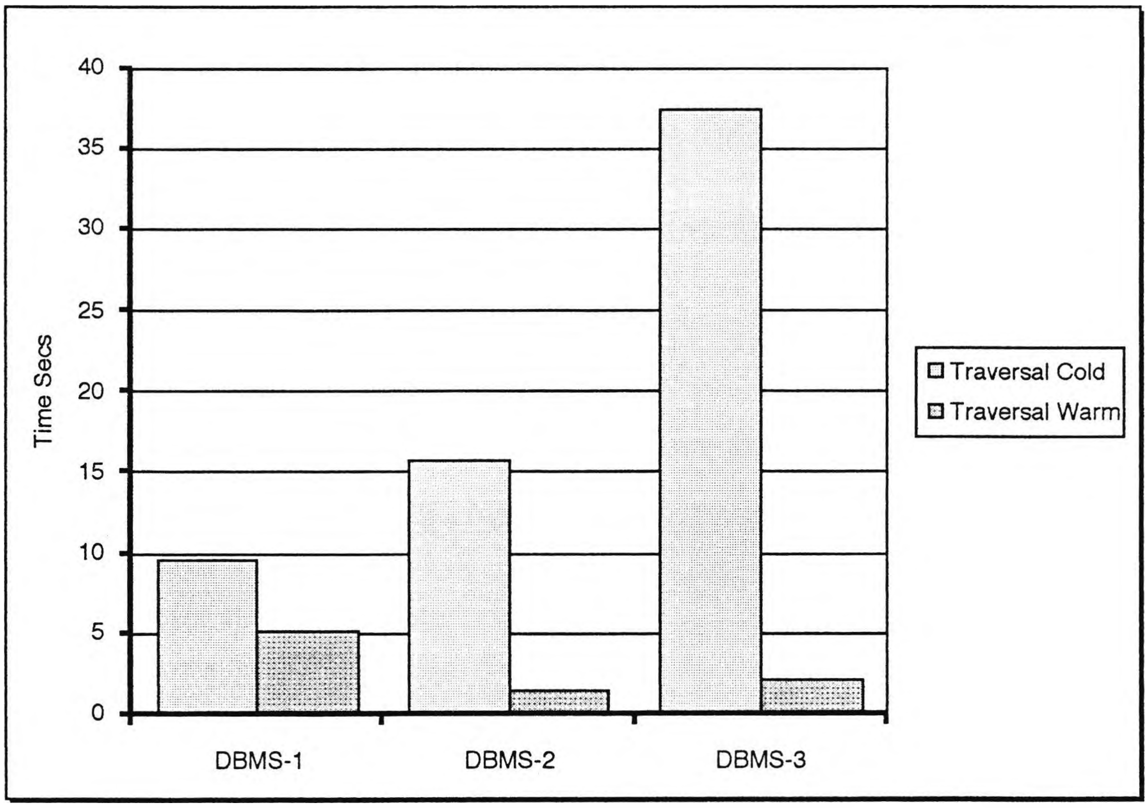

Figure 7.20 - Small Local Database Benchmark Traversal Results.

7.9.4 Large Local Database Results

The following table illustrates database size and load time.

	DBMS-1	DBMS-2	DBMS-3
Load Time (Secs)	589.127	753.731	5,203.310
DB Size (MB)	48.164	47.000	56.781

Table 7.21 - Large Local Database Load Results (Secs).

In Table 7.21, the DBMS-3 load time is an order of magnitude worse than the other products. The load was performed using the bulk loader, since memory problems were occurring in the C++ program. This result is a little surprising, since DBMS-3's bulk loader has given good performance on other benchmark tests. The differences between DBMS-1 and DBMS-2 are comparatively small in terms of database size.

Warm traversal results are not available for DBMS-3 in Table 7.22, as the product was suffering memory management problems again. Only one cold traversal result could be obtained, as shown. It was predicted that warm numbers would take considerably longer, based on experience with other benchmarks on DBMS-3. However, lookup results are again comparable to the other two products.

	DBMS-1	DBMS-2	DBMS-3
Lookup Cold	1.638	7.364	5.386
Lookup Warm	1.532	2.025	3.190
Traversal Cold	51.904	212.454	16,305.694
Traversal Warm	55.745	134.463	N/A

Table 7.22 - Large Local Database Benchmark Results (Secs).

In Figures 7.21 and 7.22, DBMS-2 does worse than DBMS-1 on both the cold and warm numbers. In Figure 7.22, this is reminiscent of the Large Local Database results for the OO1 Benchmark and is probably caused by the larger working set. Strangely, DBMS-1 performs slightly worse on the warm test, whereas its warm cache numbers on all other benchmarks have always been better than its cold numbers. DBMS-3 warm numbers are missing for reasons mentioned earlier. The DBMS-3 cold traversal number in Table 7.22 is not shown in Figure 7.22, as it swamps the results for the other two products.

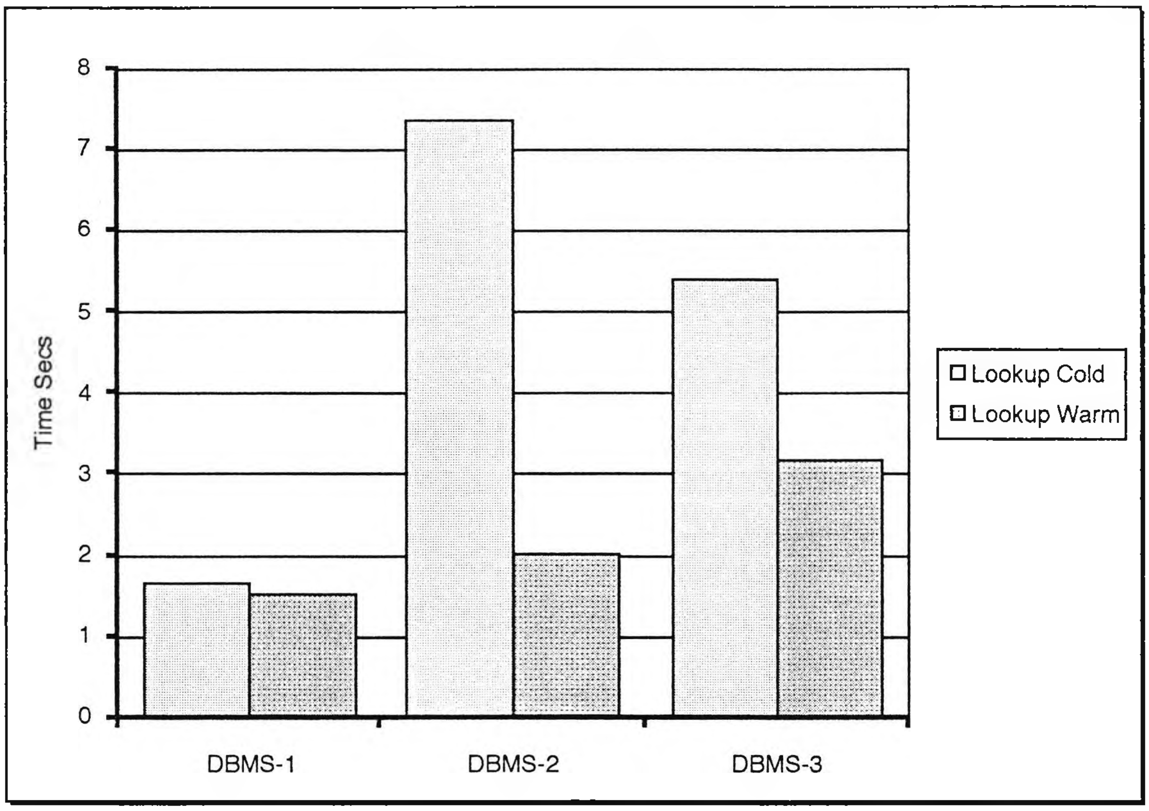

Figure 7.21 - Large Local Database Benchmark Lookup Results.

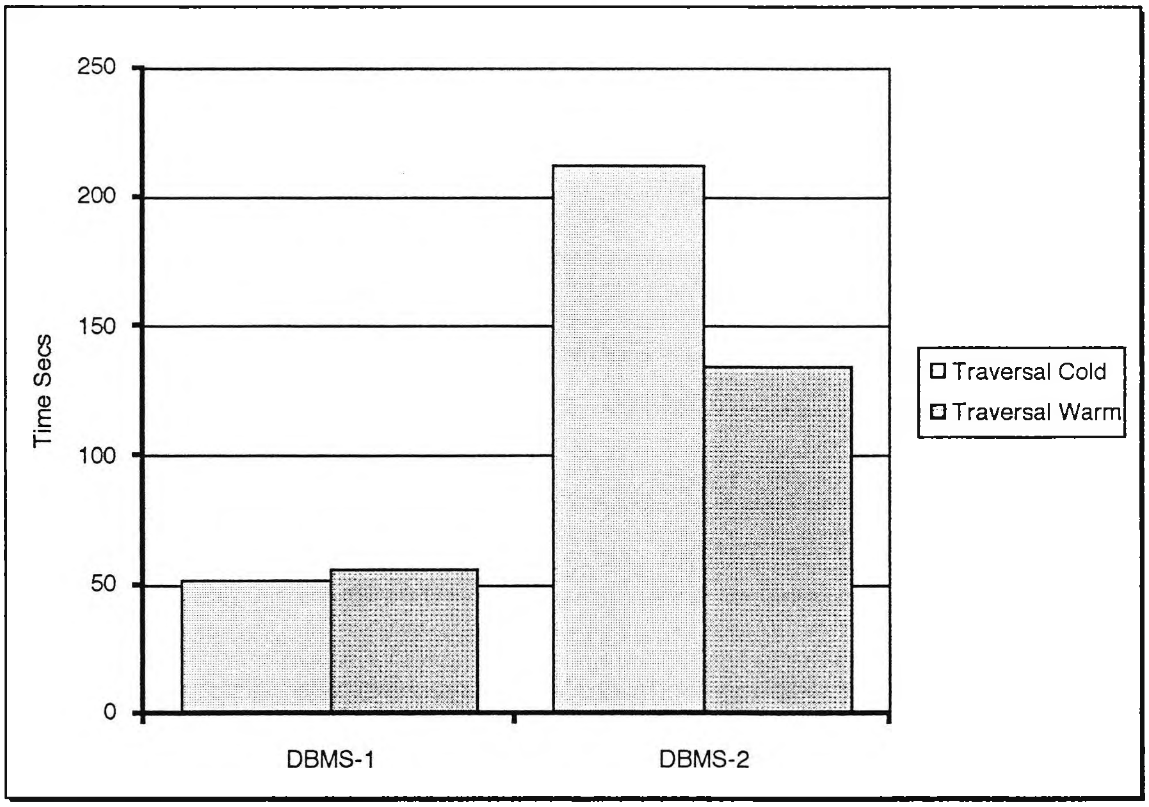

Figure 7.22 - Large Local Database Benchmark Traversal Results.

Raw results and a statistical analysis are presented in Appendix J. Some of the results are significant at $\alpha = 0.05$.

7.10 Chapter Summary

This chapter has presented the following six database performance benchmarks:

1. The OO1 Benchmark (Engineering).
2. The AFIT Wargame Simulation Benchmark (Discrete-Event Simulation).
3. The CITY Benchmark (OLTP).
4. The X.500 Benchmark (X.500).
5. The GIS Benchmark (GIS).
6. The OO-Fin Benchmark (Financial Trading).

The aims of each benchmark have been discussed, as well as results presented from running these benchmarks on various configurations for three database systems. In many cases, it is the first time that implementations of these benchmarks have been attempted for these three products. The results obtained have shown some interesting insights into the suitability of each product for the application-domain whose data manipulation characteristics each benchmark represents. As mentioned earlier, the second aim of this research project was to identify what classes of applications were more suitable for particular object database architectures, which has been met by undertaking this performance study. The third aim has also been met, since it has been determined that a generic, simple and accurate performance model for object databases cannot be derived (supporting the evidence from the previous chapter). Specifically, the results have shown:

- **The OO1 Benchmark** - this benchmark has shown that there are major performance differences between pure object databases, although it was not originally designed for this type of comparison. DBMS-1 provides good performance on both cold and warm numbers and sustains this performance independent of database size. DBMS-2's warm cache numbers are exceptionally good for small database sizes. However, its cold numbers indicate that it has a high start-up cost. Furthermore, DBMS-2 is more sensitive to the database size than DBMS-1. The results for DBMS-3 have been very disappointing. It has demonstrated serious memory management problems, which vendor technical support has been able to reproduce, but unable to diagnose.

- **The AFIT Wargame Simulation Benchmark** - this benchmark was only tested on two products, since the schema was more complex than the other benchmarks and it was felt that the probability of success for implementing it on DBMS-3 was low. The results for DBMS-1 and DBMS-2 differed widely, possibly showing the high cost of updates/writes in the former, but significantly better performance in the latter. One recommendation is that this benchmark should be extended with a batch capability, since it requires considerable user interaction and monitoring, which can be very time-consuming.
- **The CITY Benchmark** - this benchmark is possibly the first time that an OLTP benchmark has been implemented on pure object databases. The results for DBMS-1 showed that it provided superb OLTP performance. DBMS-2's performance, however, was so poor that it barely registered on some of the graphs. These results show that object databases can support high-volume transaction processing applications, but the choice of product is critical. Two implementations for DBMS-3 (C++ and ESQL) demonstrated performance differences between language interfaces. This confirmed the results reported in Chapter 6 for the Nomura Treasury Dealing System, where performance differences were observed between two Smalltalk variants.
- **The X.500 Benchmark** - this was a benchmark directly developed as a result of a case study undertaken in Chapter 6. Example data and operations were provided by NEXOR Ltd. and exact match and wildcard queries were implemented on all three products. The results showed that all the products could manage exact match queries on cached data in under 1 second, meeting one of the objectives of the testing for this company. However, wildcard queries were more costly, since neither DBMS-1 nor DBMS-2 could use any indexing and DBMS-3 again demonstrated memory management problems.
- **The GIS Benchmark** - this benchmark describes queries consisting of predicates that involve: (i) keywords, (ii) spatial co-ordinates and (iii) a mixture of both keywords and spatial co-ordinates. Implementations of the benchmark on all three products were influenced by the lack of spatial indexing. Consequently, a schema was used that was not ideal. Furthermore, loading was very lengthy on DBMS-1 and DBMS-2. The benchmark results showed that DBMS-2 was the clear winner. The DBMS-3 bulk loader again demonstrated that data could be loaded quickly if they were already in a suitable format.
- **The OO-Fin Benchmark** - this benchmark was designed to represent portfolio analysis in financial applications. DBMS-1 demonstrated good performance on both cold and warm tests and the numbers were also less affected by database size than the other products. DBMS-2 again showed a high start-up cost (cold cache numbers), but superb warm cache performance for

small database sizes. As with the OO1 Benchmark, DBMS-2 showed more sensitivity to the database size than DBMS-1. This could be due in part to the client-server configuration (both running on the same machine) and its virtual memory mapping architecture. DBMS-3 also showed that it was scalable in terms of database size, but its performance for the tests on the larger database was very mixed.

These benchmarks have demonstrated that there is great variation in the performance of the three products under test. Moreover, the tests have highlighted certain product characteristics, such as the apparent high cost of updates/writes for DBMS-1 and the unmatched warm cache performance of DBMS-2 when the working set fits entirely in the client cache. Furthermore, the results have shown that DBMS-3 has serious memory management problems, as its performance worsens with more test iterations, whilst the other products generally provide superior performance on the same tests. Another interesting discovery has been the DBMS-3 bulk loader, which has demonstrated that if the data are already in a particular format, then loading them into a database using a dedicated utility can save considerable time in some cases.

CHAPTER 8 - Conclusions

8.1 Introduction

This chapter summarises the research findings and contributions of this work, presents the major conclusions and suggests areas that require further investigation for future work.

The remainder of this chapter is organised as follows. Section 8.2 summarises the major areas where contributions have been made by this work and some of the difficulties experienced during this research project. Section 8.3 presents the major conclusions. Section 8.4 suggests areas for future work in object database performance. Finally, Section 8.5 presents the chapter summary.

8.2 Summary

This research project has described an approach to object database performance evaluation, using multiple-case studies and benchmarks. This work has been driven by the lack of suitable performance studies and the increasing use of object database technology for mainstream commercial applications.

This research work has identified deficiencies and problems with existing object database benchmarks. Some of those deficiencies have been directly observed in the case studies. For example, many previous public benchmarks have not considered database tuning as an important issue, whilst the results of this research have shown that commercial systems are rarely evaluated without considering such issues and that applications are designed with the features and architecture of a particular object database product in mind.

Six case studies were undertaken at five organisations. The case studies proved to be very difficult to undertake, for four main reasons:

1. Few organisations in the UK have developed object database applications of any size. Many are still experimenting with the technology and, as yet, have not committed themselves to it.

2. Some organisations that initially indicated an interest in this research subsequently withdrew due to internal re-organisations, out-sourcing, new priorities, project being abandoned, primary contacts leaving, etc.
3. Those organisations that had developed applications were reluctant to divulge any information, e.g. even which product they were using, mainly for commercial reasons, e.g. they did not wish their competitors to know what they were doing, even though assurances were given to them about confidentiality and the benefits of this research to them.
4. Object database vendors have been unwilling to help, since the market for this technology is still small compared to relational databases [IDC94; OOS96; Stone96] and they are all competing for a share of this. Performance is a major selection factor when user's are evaluating commercial products [Rotze91]. The vendors also proved to be very ineffectual in providing any case studies themselves.

It has also been very difficult to draw out standard performance factors from the case studies, as the level of information provided by each organisation varied considerably. Other issues, such as the capabilities of each product also made this difficult, as discussed in previous chapters.

The benchmark tests have highlighted some performance characteristics for each of the products used in this research. The tests have provided useful insights into the suitability of these products for a number of different application domains, represented by each benchmark.

DBMS-1 consistently provided superior cold cache performance than either of the other two products. Although it does appear to have problems with updates/writes, as shown by the AFIT Wargame Simulation Benchmark, it does provide far superior throughput, on the CITY Benchmark, than the other two products. This was a little surprising, since object databases are generally not associated with OLTP systems.

DBMS-2's warm cache performance is generally better than the other two products, although does degrade more significantly with larger databases, as demonstrated by the OO1 and OO-Fin benchmarks and observed in one of the case studies. DBMS-2 also has a high start-up cost, relative to the other products. Loading a DBMS-2 database for testing can also be very lengthy, as shown by the OO1 Benchmark.

Product literature from DBMS-3's vendor claims that it provides balanced performance for both navigational and OLTP applications. However, the results presented in this research project show that it generally performs worse than two other products across a range of tests that include both navigational and OLTP access. DBMS-3 also has serious memory problems, particularly when attempting to load the database from a C++ application. For relatively simple data, the bulk loader gives very good performance, but the data need to be in a particular format, which could be difficult to generate with complex schemas and was the major reason why the AFIT Benchmark was not attempted with DBMS-3. Code portability is also affected to some degree, since it is easy to use a combination of SQL, OQL and C++ constructs within a single program. SQL is useful, since it is possible to define a cursor on a table/class and retrieve rows/objects one at a time for further processing. OQL does not support this, but is able to retrieve single objects or groups of objects that satisfy some selection criteria. Retrieving large numbers of objects into the local cache using OQL is not appropriate, particularly given the memory problems that were observed on several benchmarks. DBMS-3 also requires more work in creating a database, since several tools are needed to calculate the number of database pages required. During this research, it was found that calculating this number for each benchmark test was a non-trivial task, requiring considerable work, despite the tools available. A further problem with DBMS-3 was the large database log files that it created. There was no mechanism to disable this feature, although archiving could be stopped.

8.3 Conclusions

The three main objectives for this research were outlined in Chapter 4:

1. Study the performance of commercial object database applications.
2. Attempt to identify which classes of applications are more suitable for particular object database architectures.
3. Determine if a generic, simple and accurate performance model for object databases can be derived.

The first objective was met by undertaking six case studies. The studies highlighted the following issues:

1. Representative benchmark data and operations are important for commercial object database evaluations. None of the organisations participating in this research project ever considered using any public benchmarks.
2. Multiple applications are found in deployed object database systems, but appear less frequently in commercial benchmarking work due to the time and cost involved.

3. Multiple users are a common feature of deployed object database systems, but are again generally not simulated in commercial benchmarks also due to time and cost.
4. Commercial object database applications do not use multiple databases.
5. Client-server architectures are standard in commercial object database systems.
6. Multiple platforms are sometimes used in commercial environments, in contrast to most public benchmarks, that use a homogeneous environment.
7. Functional differences between object databases can cause substantial performance differences, as demonstrated by one of the case studies.
8. Tuning is an important factor for commercial object database evaluations and rarely do such evaluations proceed without considering tuning issues.

The second objective was met by undertaking laboratory experiments. Six benchmarks were implemented and each benchmark represented the data manipulation characteristics of a particular application domain. The benchmark results showed the following:

1. Major performance differences among the three products under test were observed. These differences varied according to the benchmark, demonstrating that a single generic benchmark was difficult to develop.
2. Statistical analyses showed that the results from many tests were significant, giving greater credibility to this research project. This is something that most previous object database performance researchers have not attempted.
3. DBMS-1 consistently provided the best cold cache numbers and also demonstrated superior performance for OLTP applications, but updates/writes appear to be a bottleneck.
4. DBMS-2 has high start-up costs, but once data were available in the client cache, its warm cache numbers were very good for small working sets. Its performance is also more significantly affected by the database size. For simulations its performance is very good, but for OLTP, its performance is very poor.
5. DBMS-3 behaved very strangely on some tests, with its warm cache numbers getting worse with more benchmark iterations, which vendor technical support was unable to explain. On other tests, its performance was comparable to the other two products. Its bulk loader also provided a useful way to load large quantities of simple data.

The third objective was met as a result of both the case studies and benchmarks. This research project has determined that a generic, simple and accurate performance model for object databases cannot be derived at this time. From the case studies, the reasons for this are as follows:

1. The quantity, quality and type of information available from each organisation varied considerably. This is expanded further below.
2. Access to system documentation varied. For example, some organisations provided substantial system design documents, whilst others generally didn't have such information available, since their development work was undertaken on a part-time, ad-hoc basis.
3. It was too difficult to find common performance factors although, superficially, tree traversal was a common operation in many of the studied systems. Additionally, some of the difficulty stemmed from the more complex design due to object-orientation and lack of facilities for data collection. Future work should consider using techniques for measuring object-oriented designs to get accurate measurements about factors such as inheritance, distribution of attributes, etc. for creating representative benchmark schemas.
4. Obtaining information on transactions and application access patterns from systems was generally not possible, since some organisations were still evaluating products, whilst others were not interested in performance issues as such. Furthermore, collecting application tracing and profiling patterns by hand proved to be impossible from just the supplied documentation. A tool is really needed to collect this kind of information automatically and at least one of the vendors does sell such a tool, but was unable to make it available to this author.
5. It proved to be impossible to get time to see primary contacts at each case study site beyond the initial contact, which was used to describe the purpose and objectives of this research. Generally, people were extremely busy.

The results of the benchmarks, discussed above, also showed wide variations in performance between the three database systems under test. It was not possible to find common characteristics from these benchmarks that could lead to a generic model. For example, discrete-event simulation has very different data manipulation requirements to OLTP. However, one of the suggestions for future work described in the next section proposes a larger study. Better support from users and vendors in addition to improved tools for collecting data about access patterns should enable future researchers to determine whether a standard workload can be developed.

8.4 Future Work

Some areas for future performance work in object databases have already been suggested by Zorn & Chaudhri [Zorn95]. From this research project, a number of issues have also emerged, which are discussed below.

8.4.1 Analytical Modelling and Simulations

Jain [Jain91] states that the techniques that may be used for performance evaluation include analytical modelling, simulation and measurement. This research used measurement only. Object database performance work using analytical modelling has been reported by Rabitti et al. [Rabit93], although no model verification was described. Future work should, therefore, focus on simulations. For example, by constructing an application system model and then varying factors, such as page size, pointer swizzling, etc.

8.4.2 Database Features

This research project showed the importance of a number of database features in Chapter 5, such as schema evolution and versioning. However, these were not observed in the case studies described in Chapter 6. They were also not considered in the benchmarks undertaken in Chapter 7, to keep the problem space manageable. Future work should investigate the performance implications of such issues, since object database systems use different mechanisms, as discussed in Chapter 2.

8.4.3 Larger Databases

Larger databases should be used for future object database performance benchmarks. This is because some large scale object database systems are now under development, that involve tens or hundreds of GB of data as well as hundreds of concurrent users, as described by Loomis & Chaudhri [Loomi98] and supported by survey evidence from Barry [Barry97]. Most previous benchmarks have focused on small databases of a few or tens of MB. This has also been the case with this research project, since disk space was a limiting factor. The difficulty in performing such large-scale tests is that there may be large overheads involved, which could make them expensive [Dick95b; Loomi96]. Obviously, there is a trade-off between attempting a large-scale benchmark effort versus implementing an actual application.

8.4.4 Larger Studies

Undertaking a larger study of commercial object database applications should now be considered. This is because the market is more mature and many more organisations are using or considering this technology than when this author began his research. Evidence for the larger take-up comes from a number of studies, such as [Barry97]. Other survey evidence from [DBWor97] shows that many organisations are considering this technology for future applications. Figure 8.1, for example, shows that almost 50% of respondents to the DB World survey of IT professionals in the UK indicated that their organisations may use object data management in the future.

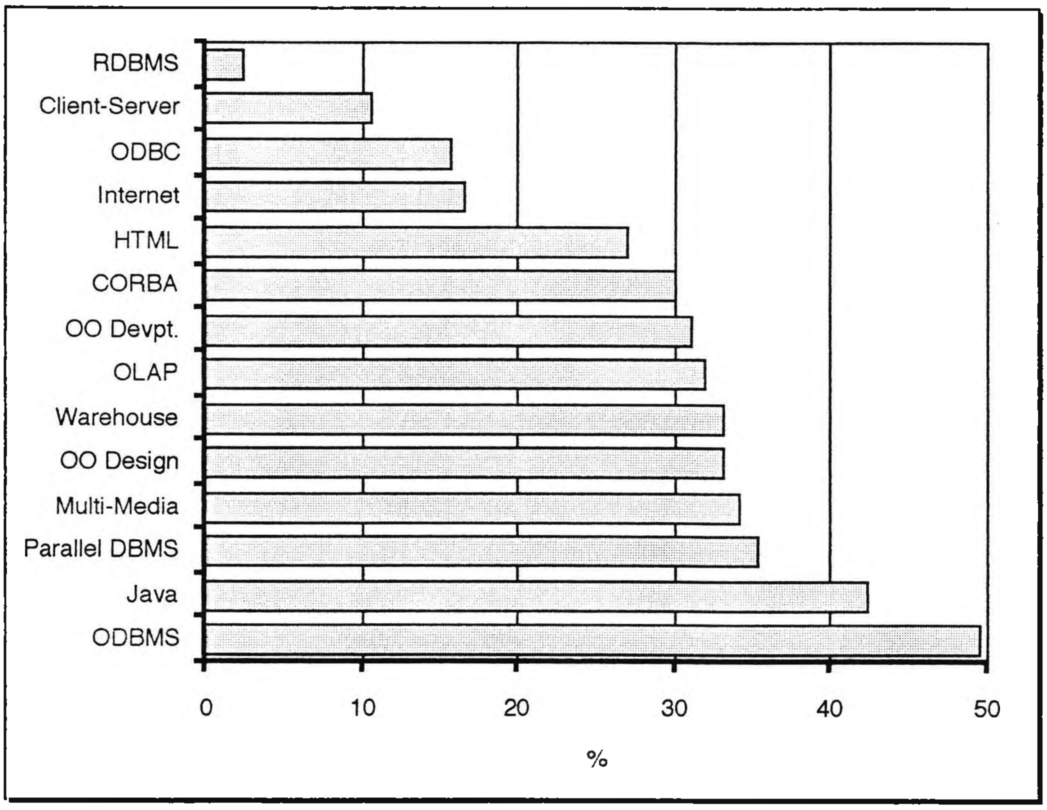

Figure 8.1 - Technology Areas Companies May Use [DBWor97].

A new study should consider a range of industry sectors, since there is interest in a variety of domains, as illustrated in Figure 8.2. The case studies and benchmarks used in this research project have already demonstrated that data manipulation characteristics differ greatly across domains. In-depth studies in particular vertical markets will produce benchmarks that are more representative of the data manipulation characteristics of those markets. However, a larger study with better support from users and vendors should also be able to establish common characteristics, as suggested in Chapter 4:

1. Generality - a set of core benchmark operations.
2. Accuracy - specifics added to the core for a particular target application.
3. Simplicity - core operations are easier to compare.

To achieve this goal, future researchers may wish to consider techniques such as Ethnographic Workflow Analysis (EWA) to study usage of actual object database applications. This approach has been successfully used to study the work practices of physicians in the design of a Physician's Workstation, as described in [Fafch91].

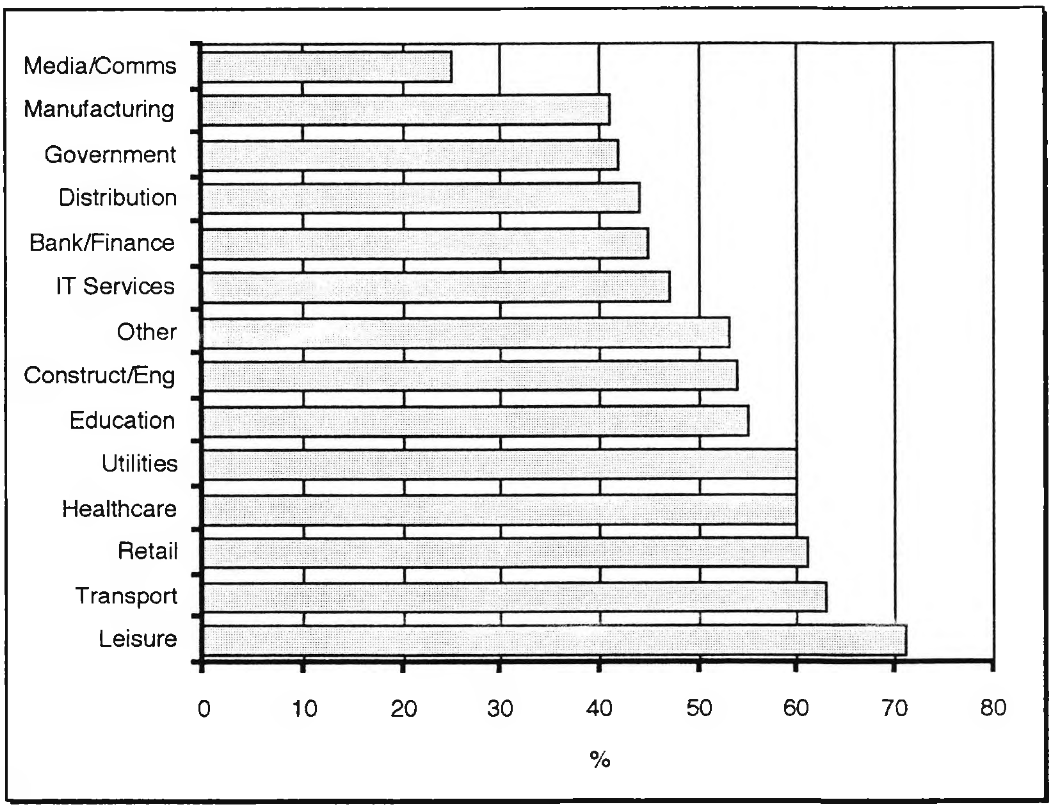

Figure 8.2 - Industry Areas Considering ODBMSs [DBWor97].

8.4.5 Multi-User Benchmarks

The mechanisms for locking and transaction management for various object database systems were described in Chapter 2, but were not considered in the performance benchmarks undertaken in this research, as justified in Chapter 4. The results of the case studies presented in Chapter 6 did show that multi-user tests were sometimes performed, but cost was often a limiting factor for its more widespread use. Furthermore, the organisations participating in this research indicated that they would always prefer to undertake their own multi-user performance tests, demonstrating that public multi-user benchmarks have little or no utility as far as commercial evaluations are concerned. Some researchers, e.g. [Carey94], have also reported difficulties in collecting multi-user object database benchmark results. This casts doubt over the utility of a generic multi-user benchmark, but is an area that should be investigated further in collaboration with industry.

8.4.6 New Benchmarks

The classification of benchmarks presented in Chapter 3 (Figure 3.2) revealed a number of gaps. These are obvious candidates for future database performance work.

8.4.7 Programming Language Bindings

Some issues were highlighted in Chapters 6 and 7 and suggested for future research. For example, a more detailed analysis of the trade-offs between alternative object database language bindings. One of the case studies showed that the choice of language can be important. Additional evidence came from the two alternative implementations of the CITY Benchmark on DBMS-3.

8.4.8 Rigorous Domain Analysis

The domain analysis in Chapter 5 is a candidate for enhancement. The categories selected were based on those used by Ahmed et al. [Ahmed92] for engineering applications. Although the analysis showed that many other applications have similar requirements, a formal approach, as discussed by Bjørner [Bjørn97], should provide greater accuracy.

8.4.9 Rigorous Taxonomies

As suggested in Chapter 3, future research should investigate rigorous taxonomies, as the benchmark survey in this thesis was mainly presented as lists and trees. These provide useful starting points, but better forms of categorisation may be possible. The reader is directed to Worlton [Worlt93] for a discussion on taxonomies for performance metrics.

8.4.10 Tuning

A number of factors, such as cache contents, cache size, special fetch policies, size of database commits, etc. were not investigated within this research. Many tuning factors were kept constant to keep the problem space manageable. Furthermore, the focus of this research work was on high-level benchmarks to represent specific application domains. However, there is evidence to suggest that low-level issues can influence performance significantly. For example, Moorley [Moore96] discusses some results obtained for the

OO1 Benchmark on a commercial object database system. He observed that the commit time appeared to have a high overhead. He speculated that this was possibly because each of the OO1 measures was completed within a single transaction. Modifying the lookup test, so that each lookup was done within its own transaction, increased the observed time from 6.5 seconds to 27 minutes, causing Moorley to conclude that 99.6% of the time was spent on transactional overheads. Recent work reported by Hohenstein et al. [Hohen97a; Hohen97b] has also demonstrated the importance of tuning. Each of the three object database products they tested was more or less sensitive to particular factors. The importance of tuning and its impact on performance has also been discussed elsewhere by Chaudhri [Chaud97].

8.5 Chapter Summary

This research project has demonstrated that a pragmatic approach to object database performance evaluation can provide important insights into how object database systems are being used in commercial environments. By using a systematic/programmatic research plan, consisting of multiple-case studies and laboratory experiments, this work has highlighted issues that organisations consider when evaluating object database systems, as well as identify significant differences in the suitability of products for certain applications. Furthermore, the evidence and results presented in this work have shown that a generic model of performance for object databases cannot be determined at this time. The reasons for this are varied and numerous and include issues such as the flexibility of the object-oriented approach to modelling, the architectural and feature differences between products and the lack of a canonical or standard workload, unlike OLTP for relational systems.

The commercial products under test in this research project have, as mentioned above, been subjected to systematic performance evaluation. Also, this author is not aware of any previous efforts to compare object and object-relational database systems. Overall, the results have shown that the "best of both worlds" approach used by an object-relational product do not provide any significant benefits for either transaction processing or navigational access. However, ORDBMSs are still in their infancy and as the technology matures, improvements will occur. These will require suitable performance benchmarks that can test system components, such as optimisers that can manage more complex data, as well as tests to represent the requirements of particular applications. Bulk loading of data is also an issue that object database vendors should investigate, since database loading is currently very time-consuming in some pure ODBMSs.

In Chapter 1, it was stated that new performance benchmarks for object databases were needed for the following reasons:

1. Performance is typically among the top three selection criteria for users when deciding which object database to purchase [Rotze91].
2. The cost of implementing a complete application to test performance is expensive [Ander90].
3. A benchmark is only a valid yardstick for applications that are similar to the benchmark [Dietr92; Hallo93a].
4. Treating performance and its measurement generically is wrong and can lead to incorrect conclusions [Inmon89].

Taking these points in turn, this research project has demonstrated that performance is a very important factor for users when deciding which object database product to purchase. However, benchmarks have also been used to investigate design alternatives. This research has also shown that organisations often consider performance benchmarking before implementing complete applications, because the cost of testing a complete application is prohibitive. Furthermore, the case studies have shown the importance to users of identifying the data manipulation characteristics of applications as well as considering product-specific features. The case studies have also demonstrated that a generic performance model for object databases is difficult to determine and the benchmark experiments have confirmed this.

These conclusions demonstrate that benchmarks continue to fulfil very useful roles, but considerable care and attention are required by both academics and users when studying object database performance, since the performance problem space is far greater than with previous generations of database technology.

Finally, for the future, this author encourages the establishment of an organisation to develop and promote object database benchmarks. Such an organisation should comprise representatives of academics, users and vendors to ensure that the interests of all these communities are adequately served.

APPENDIX A - Object Databases

A.1 Introduction

Chapter 2 described some object database products in detail. This appendix provides support material for that chapter and would be suitable for the reader that is not familiar with object or data management concepts.

A.2 Object Requirements for ODBMSs

"My Cat is Object-Oriented." [King89].

"If I hear the phrase 'everything is an object' once more, I think I will scream." [Stone88].

The concepts presented here are not exhaustive and the terminology used should be compared with [OODBT91], which provides general characteristics of object models. An additional useful source is [Stefi86], which provides a comprehensive discussion of object concepts from a programming language perspective.

A.2.1 Object

An object is a software representation of a real-world object. Objects can be physical items, such as a computer, disk drive, etc. or abstract concepts, such as a University Board of Studies meeting. Software objects are self-contained modules that include data (sometimes called instance variables) and code that acts on that data (called methods). Instance variables are "everything an object knows" and methods are "everything an object can do" [Taylo91].

An object is an instance of a class, where "classes are descriptions of things, objects are the things themselves" [Winbl90]. However, in some languages, such as Smalltalk, even classes are considered as objects.

The state of an object can be defined as the values of an object's instance variables at any point in time. Alternatively, it may be defined as the result(s) returned after an object has performed certain operations. An object's state can be modified by sending it messages to invoke one or more of its methods.

An object's instance variables can be simple base types, such as integer, char, etc., more complex types, such as arrays, sets, other aggregates, etc. or complex objects themselves, such as a Computer Aided Design (CAD) drawing.

An object can participate in a number of different (orthogonal) relationships. Some examples described by Kim [Kim90b] include:

- class and object (instance-of).
- generalisation (is-a).
- aggregation (class-composition).
- composition (part-of).
- versioning (version-of).

Each object can, therefore, have a number of different *roles* depending on which relationship we view it from. Taylor [Taylo91] has noted that an ODBMS can support any number of alternative structures for the same set of data and that these structures are not simply *views* of the data superimposed on a single underlying model (viz. relational), but are all equally valid and exist independently of each other. Prabhu [Prabh92] has also commented that existing data models provide poor relativism (alternative ways of looking at the same thing). With ODBMSs, better support for multiple ways of viewing the same information have become possible.

A.2.2 Object Identity

A unique Object Identifier (OID) is associated with every object and distinguishes one object from another. This logical identifier is invariant across all possible modifications of an object's state [Dawso89]. An OID would also be invariant across physical address, object structure and object name. This is in contrast to tuples (rows) in the Relational Model (RM), which are value based.

Given the uniqueness of an OID, it is then possible to compare objects for several forms of equality. Furthermore, the OIDs, instance variables, methods or even object hierarchies of two objects can be compared to determine whether they are the same. If the OIDs of two objects are the same, it is the same object.

OIDs permit the referential sharing of objects. Also, because a relationship between objects is explicitly defined, the existence of the relationship is ontologically dependent on the prior existence of the objects that participate in the relationship [Atwoo90]. In contrast, RM supports implicit relationships, whose semantics need to be re-created at run-time. SQL also incorporates additional capabilities, such as foreign key constraints, to recapture lost semantics [Khosh92a].

An OID is system generated, in contrast to RM where explicit identifiers need to be created by users. However, an alternative to using pure OIDs was suggested in [Commi90], where identifiers are system generated only when meaningful user generated values are not available, e.g. social security number, student number, employee number, etc.

Some ODBMSs do not use OIDs as have been described so far. ObjectStore [Lamb91], for example, uses virtual memory addresses instead.

Ullman [Ullma88] has argued that pre-relational database languages can be classified as object-oriented, since they support the notion of object identity. Although we can view hierarchical, network and relational models as special cases of a general object model, object identity is a semantic concept, whereas virtual addresses or pointers (as used in hierarchical and network systems) represent memory locations on an underlying von Neumann machine [Khosh92a]. Furthermore, as Duhl & Damon [Duhl88] comment:

> "An object identifier also contains information about the object's type. This differentiates simple connectivity, as with pointers, found in the network model, from what could be called typed direct connectivity. This additional type information embedded in the object reference can be used to provide information for semantic validation."

Pre-relational languages also do no support concepts such as methods, classes or inheritance.

For a more detailed discussion of the similarities and differences between object-oriented and earlier generations of database systems, the reader is referred to [Kim90a] and [Kim90b].

A.2.3 Method

A method is equivalent to a procedure, function or subroutine in traditional programming environments. An object's methods define its interface (or protocol).

A.2.4 Message

A message is a request to a specific object to invoke one of its methods to perform some operation. It is equivalent to a procedure, function or subroutine call in traditional programming. As noted by Atkinson et al. [Atkin89], the syntax to send a message (invoke a method) may vary between programming paradigms. For example, the following would be equivalent:

- "john hire".
- "john.hire".
- "hire john".
- "hire(john)".

A.2.5 Type and Class

Type and class are often used interchangeably. However, one of the few ODBMSs that differentiates between a type and a class is O_2 [Deux90; Deux91]. In O_2, a class encapsulates attributes and behaviour. Objects are instances of a class. Types are components of a class and describe the structure of its instances. Values are instances of a type and are not encapsulated. An O_2 object may, therefore, have a complex structure consisting of values *and* objects.

The class construct can be either intentional (object factory) or extensional (object warehouse). ODBMSs that attempt to provide the seamless integration of a programming language and database system by extending an Object-Oriented Programming Language (OOPL) with DBMS capabilities fall into the former category. An example is GemStone, which extends Smalltalk.

A.2.6 Inheritance

Classes can inherit the attributes and behaviour of other classes. They are then organised into an inheritance hierarchy, where classes further down the hierarchy (subclasses) are specialisations of those above them and classes higher in the hierarchy (superclasses) are generalisations of those below them. New subclasses can be created by programming only the differences from their superclass(es). Also, subclasses can override attributes and methods inherited from superclasses.

A subclass may have only one superclass (single inheritance) or two or more superclasses (multiple inheritance). Single inheritance is similar to the concept of parent/child records in hierarchical database systems. Multiple inheritance can similarly be compared to owner/member records and sets in network databases. Multiple inheritance provides the capability to model greater complexity, but at the expense of more complex software. Conflict resolution, error logging or some other strategy will be necessary, if the superclasses of a class contain attributes and/or methods with the same names. Smalltalk supports single inheritance and C++ multiple inheritance.

A.2.7 Encapsulation

The manipulation of an object is only possible through its defined external interface (strict encapsulation). The implementation of the instance variables and methods is hidden. As a result, the implementation can be changed without affecting existing program code that uses an object's interface. This provides logical data independence [Atkin89]. Encapsulation, to paraphrase [Winbl90], makes boundaries among objects clear, communication among objects explicit and hides implementation details.

It has been suggested in [Atkin89], that strict encapsulation may not be desirable for ODBMSs and that access to implementation details may be required, e.g. by the query optimiser, which is a *trusted* component of the DBMS. Also, structural access to instance variables may be needed for performance reasons when using an ad-hoc query language.

A.2.8 Abstract Data Type (ADT)

The ability to distinguish between an object's interface and its implementation results in an Abstract Data Type (ADT). The class construct specifies an ADT.

A.3 Data Management Requirements for ODBMSs

This section examines how the data management paradigm has been influenced by the object paradigm and some of the problems and challenges that have appeared.

A.3.1 Persistence

Within OOPLs and languages in general, most data are of a transient nature and cease to exist when the process(es) that created them terminate, unless explicit commands are issued to make the data persistent. An ODBMS, however, would enable data to be saved implicitly. Khoshafian & Abnous [Khosh90] have discussed a number of alternative strategies for persistence: persistence extensions and persistence through reachability.

ODBMSs that define persistence extensions use the class construct to specify structure, extension and persistence, similar to a relational table. ODBMSs using persistence through reachability specify an object space with a persistent root. Objects that can be reached from this root are also persistent.

A.3.2 Transactions

The properties of a transaction are [Loomi90]:

1. It is application-defined. It obeys the application's rules of consistency.
2. It is all-or-nothing. All parts of a transaction complete and any updates are committed to the database, otherwise the entire transaction aborts.
3. After a transaction has committed, changes to the database cannot be undone, except by running another transaction.

The second property has certainly had to be re-examined for ODBMSs, since in many application areas, such as CAD, transactions could be very lengthy. For example, data could be checked-out of a central database into a local workstation, worked on for hours or even days and then checked-in again. To lock data for such lengthy periods of time would result in poor levels of performance if concurrent access to the data were required by many users.

A new model to support these *long transactions* for ODBMSs has been suggested in [Rotze90]. Brown [Brown91] has also recognised that new models are required to support conversational, long duration, complex and non-atomic transactions.

A.3.3 Concurrency

One of the major benefits of a DBMS is that data can be shared by multiple users and applications. However, with multiple transactions attempting to access the same data at the same time, some form of control is required to ensure that the database is always in a consistent state. To support these two competing demands upon the database, a serialisable order of execution for transactions is usually imposed.

Khoshafian & Abnous [Khosh90] discuss three possible strategies (time-stamp ordering, optimistic algorithms, pessimistic algorithms) to ensure the serialisability of transactions. Some additional important characteristics of transactions have been noted in [OODBT91].

Stone & Hentchel [Stone90a] note that ODBMSs offer "long transactions, gaining optimistic concurrency", although some commercial ODBMS systems, for example GemStone, offer several of the above mentioned strategies.

A.3.4 Recovery

Recovery is used to return the database to a consistent state after a failure, e.g. transaction, system, media.

This is another area that has required re-examination since, according to Michael Stonebraker (cited in [Hazza90]), most ODBMSs run their data manager in the same protection environment as the user program (usually the client workspace). Consequently a protection boundary does not have to be crossed for data lookup. Although this improves performance, it results in the loss of a protected database. This could cause problems for recovery if there is a failure.

A.3.5 Querying

A criticism of data manipulation in object databases is that record-at-a-time navigational access is required [Date90]. However, many ODBMSs provide declarative SQL-like DDL/DMLs, e.g. OPAL (GemStone), OQL (Zeitgeist), OSQL (IRIS). These languages vary in their support for data abstraction, ranging from the direct manipulation of attributes to access through an objects' public interface. Some of these languages are also computationally complete. Additionally, commercial ODBMS vendors now provide rich graphical tools for database design, administration, examination and the development of user applications, therefore making the direct manipulation of the underlying structures unnecessary.

A.3.6 Security and Authorisation

Barry [Barry91] has stated that some ODBMSs provide very extensive support for security, whilst others do not even provide a log-on password. This has been confirmed by a review of some commercial products in [OOS92].

Security and authorisation have not been adequately addressed by object database vendors, since many commercial products were developed for work-group applications, where such issues were perhaps not considered to be of prime importance. However, these issues need to be addressed for other application domains, where access to sensitive data must be restricted. In contrast, many relational systems offer certifiable security levels [Loomi92].

Further discussion of issues related to this subject can be found in [Kim90a].

A.4 Objects + Persistence - The Choices

ODBMSs seem attractive in that they can be combined directly with OOPLs and Object-Oriented Analysis and Design (OOAD) Methodologies to provide a full object-oriented environment [Chaud96a]. However, many organisations may not be able to take advantage of this potential. They may, for example, have adopted a particular design methodology (or combined several methodologies) already and may also be using languages, such as C++ and Smalltalk, for new developments. To manage persistent objects, there are a number of possibilities. These are now discussed in more detail.

A.4.1 File Systems

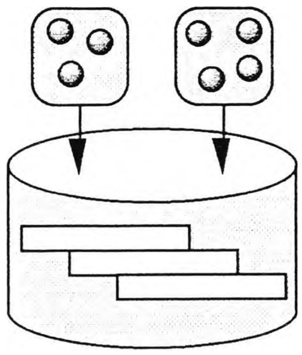

Figure A.1 - File Systems.

This option is suitable for single-user systems and enables object attributes to be stored in standard operating system files (Figure A.1). These files can be shared between applications, but will prove difficult to share between more than one user at a time. In such cases, users will effectively have to "roll-their-own" DBMS - a non-trivial task.

A.4.2 Relational Database Views

"You can do anything with an RDBMS that you can with an OODBMS, except you have to roll your own." Mark Hanner, cited in [Hodge89].

This approach provides a way to store object attributes in relational tables and has been successfully used by at least one international airline. An object class is mapped to a relational view and object attributes are then stored in one or more relational tables (Figure A.2). However, problems arise when dealing with inheritance, where there are a number of different approaches that can be used, such as horizontal or vertical partitioning (discussed later). Some of the issues concerning the use of object-oriented techniques with relational systems are more fully discussed in [Burle93].

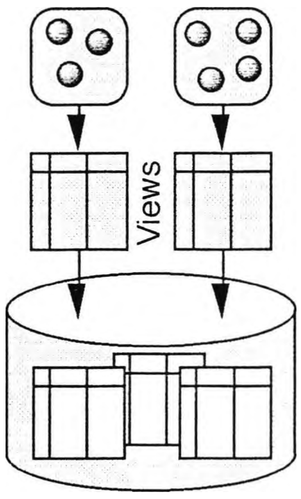

Figure A.2 - Relational Database Views.

A.4.3 Relational Database BLOBs

Most relational systems provide support for Binary Large Objects (BLOBs). However, the semantics of such objects are unknown to the database. Typically, a column in a table can be defined to support a BLOB type, but this would be a pointer to an external file that contains the actual object (Figure A.3). Other restrictions may also apply, such as limiting the number of BLOBs per table.

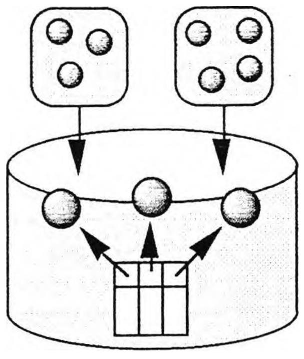

Figure A.3 - Relational Database BLOBs.

A.4.4 Object Mediators

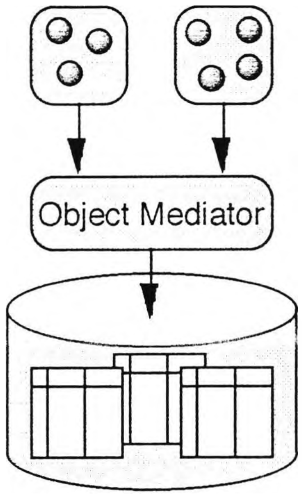

Figure A.4 - Object Mediator.

The term "object mediator" was previously used in [Thomp93] to describe a new set of tools that provide automatic mapping of objects to relations (Figure A.4). Perhaps the best example of this is Persistence. However, even such tools as these have restrictions. For example, Persistence does not currently support multiple inheritance or aggregation [Kelle93]. Other alternatives that might also be included under this section are Subtleware and Rogue Wave's Tools.h++. Tools.h++ provides classes that enable developers to write to an abstract interface that hides the Application Programming Interfaces (APIs) of RDBMSs.

A.4.5 Object Server

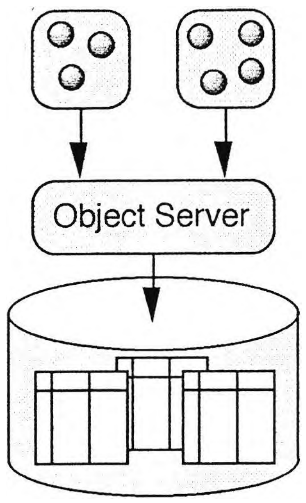

Figure A.5 - Object Server.

An approach that attempts to provide a more seamless integration of objects and relations is taken by Hewlett-Packard with its OpenODB/Odapter technology. This uses an object server sitting on top of a relational storage manager (Figure A.5). OpenODB [Ahad92] uses HP's own Allbase storage manager, whilst Odapter can be used with the Oracle RDBMS. With any layered approach, however, performance will always be a major issue.

A.4.6 Extended Relational Databases

The phrase *extended relational* refers to a range of database products with various enhancements/extensions. Sometimes the phrase *post-relational* is also used. However, there is no standard definition of what is meant and enhancements/extensions vary widely between products. Perhaps one of the best examples is OpenIngres, which provides a library supporting spatial data types.

A.4.7 Object-Relational Databases

The phrase *Object-Relational DBMS* can be attributed to Stonebraker [Stone93a] and refers to those products where there is strong integration between object-oriented and relational concepts (Figure A.6). Two products that exemplify this approach are Illustra and UniSQL. 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 OO, such as better abstraction. However, even these two products differ markedly from each other, as discussed in Chapter 2.

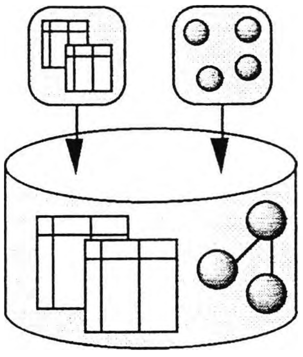

Figure A.6 - Object-Relational Database.

Larry Ellison of Oracle announced several years ago that Version 8 of Oracle would be "fully object-oriented" [Lauch92]. Since then, there has been much speculation and debate about the exact nature of the object-oriented support that Oracle would provide. However, given the significant customer base Oracle already has, any extensions or enhancements will need to ensure backward compatibility with existing customer applications. A layered approach would be simpler, but would have performance drawbacks, whilst re-architecting the database engine would provide a better long-term solution. Additionally, Oracle is actively involved in standards efforts that are concerned with object-oriented extensions to SQL. The end result may, therefore, be a combination of several approaches, such as server support for an Object SQL and some object-oriented layering over Oracle 7.

A.4.8 Object Databases

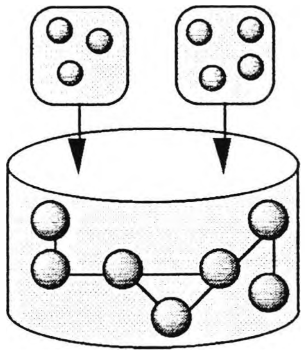

Figure A.7 - Object Database.

Using the ODMG definition, this category contains those products that provide DBMS extensions to one or more OOPLs (Figure A.7). Whilst, in the past, some products were closely tied to one OOPL, vendors are increasingly providing support for multiple OOPLs. Many products now offer similar functionality and features and even some architectural differences are being eroded. For example, the distinction between what has been termed as "active" or "passive" object databases [Manol94] has become blurred, as discussed by Wilcox [Wilco94]. Taylor [Taylo92] differentiates active and passive object databases primarily in terms of where methods are stored.

An active object database (Figure A.8) stores methods inside the database and this, according to Taylor, provides the following benefits:

- Dynamic binding at run-time.
- "Active" data dictionary.
- Can be updated dynamically.
- Concurrency control for methods.
- More secure (protected by DBMS).
- Methods are treated the same as data (first-class).

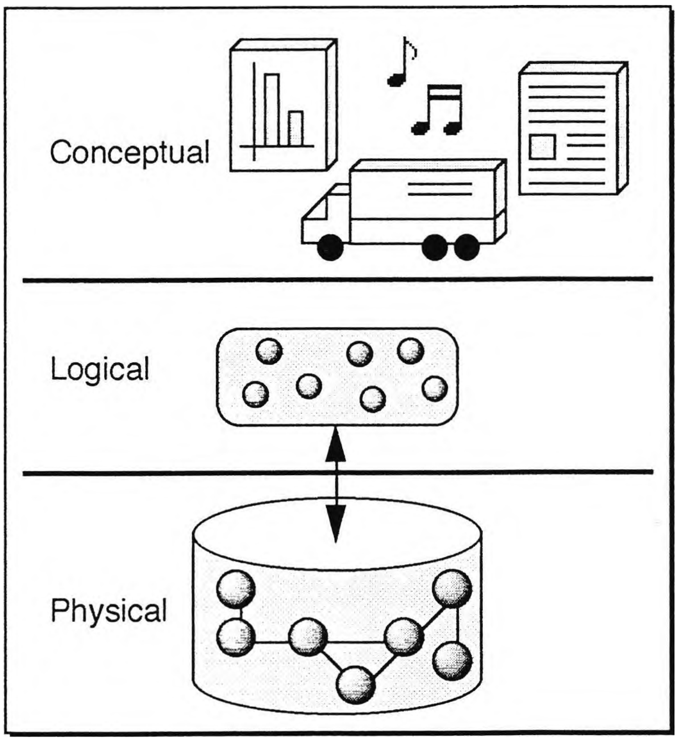

Figure A.8 - Active Object Database.

A passive object database (Figure A.9) stores methods outside the database which Taylor claims has the following disadvantages:

- Methods are bound at compile/link time.
- More complicated, since changes require re-linking and re-deployment.
- Possibility of corruption to database.
- No concurrency control for methods, as they are held in external files.
- Breaks encapsulation and treats methods as second-class, denying them the protection and services of the DBMS.

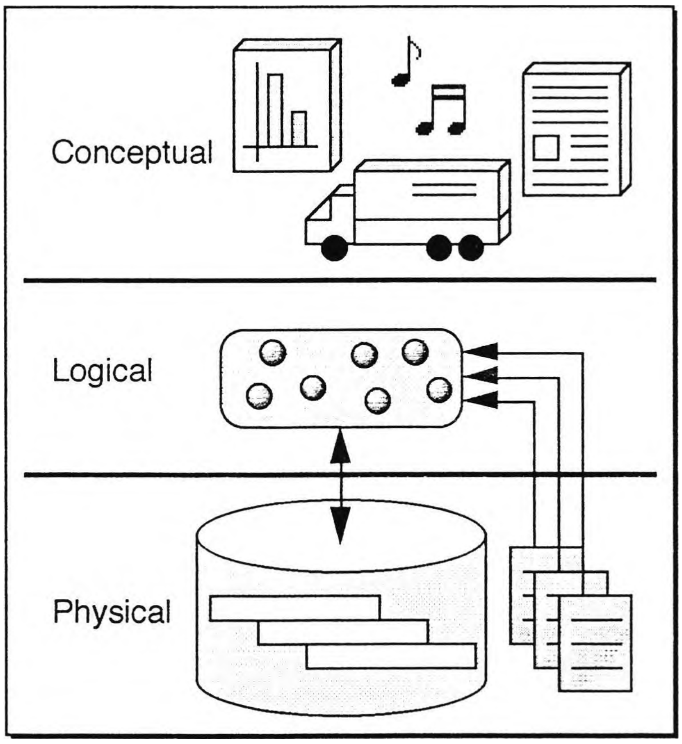

Figure A.9 - Passive Object Database.

	Object Server	Page Server
Active	ITASCA	O_2 GemStone
Passive	VERSANT	Objectivity/DB ObjectStore

Table A.1 - Method Execution vs. Server Architecture [Manol94].

This differentiation, however, is somewhat simplistic, since some products such as ObjectStore (which could be classified as "passive" according to Taylor's approach), for example, can be configured with the client application running on the server, able to execute methods and act as a network server for the ultimate client [Manol94]. Another important issue is the distinction between various server architectures (discussed later) and where methods are executed, as shown in Table A.1.

A.4.9 Object + Relational

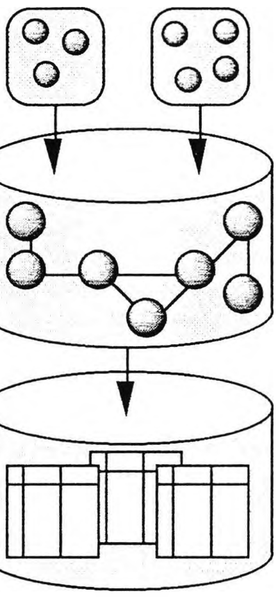

Figure A.10 - Object + Relational.

For a variety of reasons, including the dominance of relational technology for most new MIS applications (there is still a great deal of data held in hierarchical and network systems as well), it may not be possible to replace existing databases with new ODBMSs. A co-existence approach may provide a way forward. One method could be to use an object database as a fast cache for a relational database (Figure A.10). This approach has recently been used by one commercial bank (discussed in Chapter 6). The benefit is that full DBMS capabilities are provided by the object system and existing *legacy* applications can still access the relational system. Another example was reported by Hewlett-Packard in the development of a Physician's Workstation, where OpenODB was used to cache some frequently referenced patient data from an existing medical system, resulting in performance improvements for some queries [DeSme94].

A.4.10 Relational Gateways

Gateway products are currently offered by a number of object database vendors. GemStone Systems, for example, provide a product that connects GemStone with Sybase. Similarly, Object Design have developed a product that provides access to DB2. The use of a gateway also enables organisations to keep their investment in existing database technology, whilst developing applications using new tools and techniques with object databases (Figure A.11).

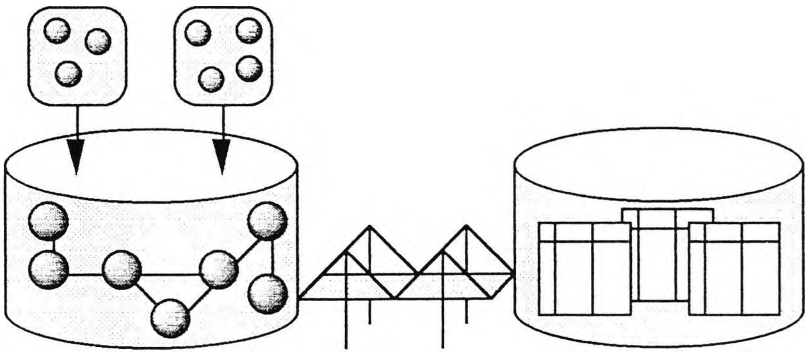

Figure A.11 - Relational Gateway.

A.4.11 Objects + Persistence Summary

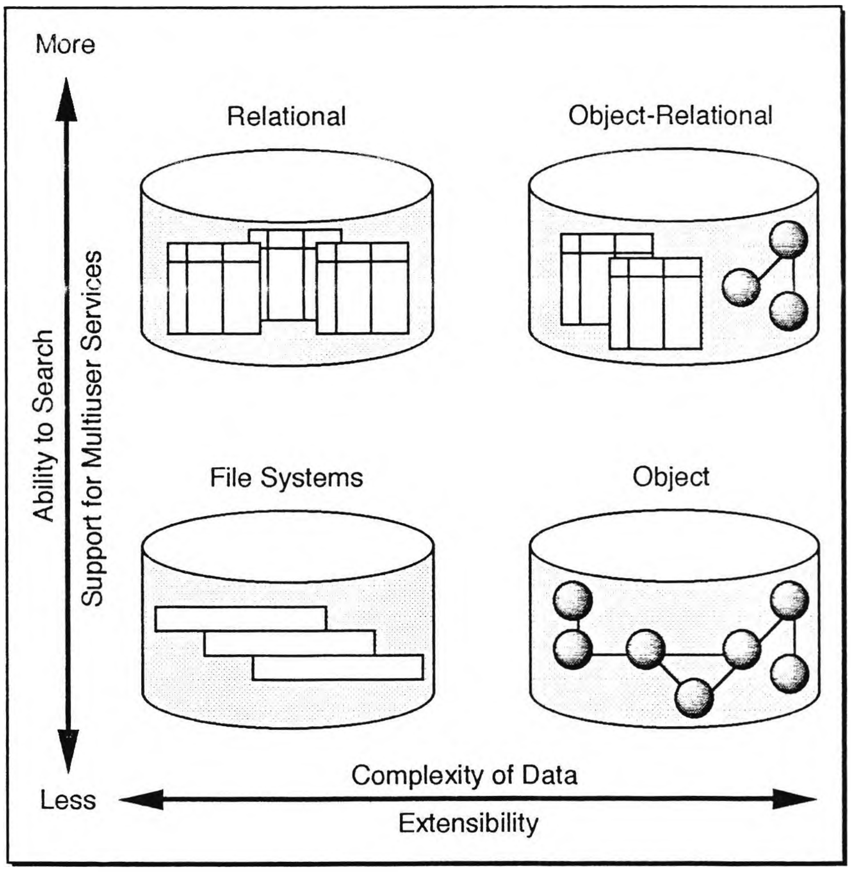

Figure A.12 - Summary of Persistence Approaches [DBMS94].

There are several ways that the various approaches to managing persistent objects just discussed can be summarised. One is illustrated in Figure A.12. This shows that file systems provide very limited or no support for DBMS facilities, whilst relational databases have good multi-user capabilities and query support (SQL) and object databases provide extensibility and support for complex structures. Object-Relational DBMSs provide the best of all worlds. The relational vendors are moving across to the right as all the major players have announced a commitment to producing ORDBMSs. Similarly, object vendors are moving upwards and many already provide some relational capabilities, such as SQL++ from Objectivity and DBConnect from Object Design.

A.5 Classification Models of ODBMSs

A number of ODBMS classification models have been suggested in the literature. Although there is considerable overlap among the models, a summary of these models is now presented. This work has also been previously reported in [Chaud93].

A.5.1 Data Models

This approach has been discussed by a number of authors, e.g. [Khosh90; Catte94a].

- **Non-First Normal Form (NF^2) Models**
  NF^2 models extend RM, whilst trying to maintain a strong mathematical foundation. The FNF constraint of RM is relaxed to allow repeating groups. POSTGRES [Stone90b; Stone91] is an example of a product following this approach. Another example is DASDBS [Schek90] where a DBMS kernel is augmented with application-specific front-ends.
- **Object-Oriented Languages**
  There are more than 80 object-oriented languages in the world [McClu92]. ODBMS products that are based on object-oriented language models include GemStone, O_2, ORION [Kim89; Kim90a; Kim90c], ObjectStore and ONTOS.
- **Functional Models**
  A number of ODBMSs have been built based on the DAPLEX Functional Data Model [Shipm81]. Attributes, methods and relationships are all represented by functions in these models. Subtypes and referential integrity are also supported. Example products include OpenODB based upon the IRIS research prototype [Fishm89; Wilki90], PROBE [Manol86] and VISION.
- **Semantic Models**
  There are many semantic models, but SDM, developed by Hammer & McLeod [Hamme81], was one of the earliest attempts to add more semantics (e.g. objects, type hierarchy, etc.) to RM. SDM models structural abstractions (like frames in AI systems) and does not support behavioural abstractions. It uses a diagrammatic representation similar to semantic networks, with nodes and

links, to depict entity types and relationships. SIM [Fritc90] is perhaps the best example of a database system based on a semantic model (SDM). Further discussion of semantic models and their database implementations can be found in [Prabh92].

A.5.2 Architectures

Classification by data models is a useful way to distinguish the genealogy of a system, i.e. which previous systems and philosophies have contributed, but a better method is by architecture [Catte94a]. Cattell [Catte91b; Catte94a] has suggested the following architectural classification.

- **Object Managers**
  These are extensions of existing file systems or virtual memory, have a limited data model, no query language and provide the most basic functionality (storage of persistent objects). Examples include POMS, Mneme and ObServer. Persistent-Data Servers [Simme92] are another development akin to object managers. They attempt to provide an alternative persistence mechanism to ODBMSs and file systems. They offer some DBMS features and perform like file systems, whilst maintaining the structure of data on disk, independent of the application that created them.
- **Extended Database Systems**
  These systems attempt to provide multi-programming language access by supporting a database language neutral model, sacrificing performance for data independence. New or extended database query languages that provide richer modelling concepts such as classes, inheritance, types and functions are provided by these systems, e.g. OSQL (IRIS), POSTQUEL (POSTGRES). Also, there can be full support for concurrency control, transaction management, etc. Examples include IRIS, POSTGRES, PROBE, Starburst [Haas90; Lohma91], SIM and VISION.
- **Database Programming Languages**
  Existing programming languages (e.g. C++, Smalltalk, Lisp) are extended to provide DBMS features such as persistence, concurrency, etc. The database query language and application programming language execute in the same workspace and share the same type system. Examples include GemStone, O_2, ObjectStore, ORION and ONTOS.
- **Database System Generators**
  Since no DBMS can provide all the specialised functions and operations needed by a specific application area (e.g. text processing, GIS, etc.), these toolkits

enable customisable DBMSs to be built. Examples include EXODUS [Carey86; Carey89] and GENESIS [Bator86]. Further discussion of the data management requirements for environments such as project management, office documents, geographical and spatial information can be found in [Oxbor91].

- **Relational Object Shells**
  Backward compatibility with RDBMSs is the main motivation for the development of these systems [Catte91b]. Rasmus [Rasmu92] has also noted:

  > "... to prove useful, object-oriented databases must be able to be integrated into a relational world."

  This integration could be achieved by interfacing or encapsulation. The latter approach is more elegant and requires a wrapper to be built around a relational database, thus providing object-level interfaces and hiding the syntax and semantics of the underlying database. In this scenario, the relational database simply becomes an abstract data definition [Rasmu92]. A shell would store objects as virtual tables [Winbl90]. There is, however, a performance trade-off, since retrieving complex structures using search-and-match can be time-consuming in RDBMSs. Storing the same structures in a full ODBMS would enable them to be retrieved in a single query. Other approaches to integrating existing databases with object systems have been proposed in [Ahad88; Nelso90; Preme90]

A.5.3 Storage Server Models

This approach was proposed by Joseph et al. [Josep91].

- **Typeless Page Servers**
  These systems do not directly manipulate objects, but manipulate virtual memory pages on which objects reside. A user application and the server share transient and persistent virtual memory. This approach is, therefore, architecture specific, since page formats differ between hardware platforms. Examples include EXODUS and ObjectStore.
- **Typeless Object Servers**
  Object servers control access to objects or groups of objects. They have very limited knowledge about the objects themselves (e.g. whether objects have a type, whether objects may be related, etc.). They cannot execute methods or access the states of objects. Examples include Mneme, ObServer and Zeitgeist.

- **Class-Based Object Servers**
  Such systems manipulate objects or groups of objects and can interpret an object's state to provide additional services, such as queries. These systems are typically built on relational storage managers, mapping objects onto tables. Two alternatives to representing inheritance are through either horizontal or vertical partitioning. In horizontal partitioning, the inheritance graph is flattened, leading to class evolution problems. Using vertical partitioning, each inherited definition is represented by one table, requiring the use of joins. The storage manager provides support for full DBMS features, such as backup, recovery, concurrency, etc. Examples include IRIS and POSTGRES.
- **Type-Based Object Servers**
  These object servers can execute methods and enable computations to be moved from client to server. Any of the other three server types discussed can be enhanced with extra layers of software to become type-based object servers. Examples include O_2 and ORION.

A.5.4 Alternative Object Database Strategies

Khoshafian & Abnous [Khosh90] have proposed six different approaches to ODBMSs.

- **Novel Database Data Model/Data Language Approach**
  Khoshafian & Abnous [Khosh90] have suggested that many research projects have pursued this approach. Of the commercial systems available, SIM has been cited because of its novel DDL/DML.
- **Extend an Existing Database Language with O-O Capabilities**
  With the ANSI X3H2 committee currently working towards the definition of SQL3 (SQL with object extensions), this is the path that is most likely to be pursued by existing relational database vendors wishing to provide greater support for object concepts.
- **Extend an O-O Programming Language with Database Capabilities**
  An OOPL already supports object concepts, but lacks DBMS facilities, e.g. querying, transactions, persistence, etc. With this approach, the language is extended to support these facilities. One example is GemStone.
- **Extendible ODBMS Client Libraries**
  An alternative to the previous approach provides libraries that extend the facilities of a language to provide classes of aggregates (e.g. sets, lists, arrays, etc.), types and methods for transaction handling, etc. ObjectStore, ONTOS and VERSANT are good examples of this approach.

- **Embed Object Database Language Constructs in a Host Language**
  This approach is similar to using Embedded SQL (ESQL) in a host language. O_2 uses this approach to provide database access from C.
- **Application-Specific with an Underlying ODBMS**
  The example cited in [Khosh90] for this approach is TeamOne, which is a configuration management system for engineering applications. This system supports an object repository for project design files, with access and modification achieved by manipulating encapsulated objects.

A.5.5 Language Data Models

Wells et al. [Wells92] refer to three approaches to ODBMSs, based on language data models.

- **Programming Language Neutral**
  The data model in this approach has no direct relationship with the programming languages that manipulate data within programs. As mentioned earlier, such an approach aims to provide maximum data independence, although inevitably leads to the "impedance mismatch" problem. Examples include IRIS, O_2, ORION, POSTGRES and PROBE.
- **Database Programming Language**
  This approach provides the highest level of transparency of database access from a programming language, since the language has been specifically designed with database facilities in mind. Examples are Galileo [Alban86] and Taxis.
- **Programming Language Specific**
  The data model in this approach is an extension of the type system of an existing programming language. Although the main objective is to ameliorate the "impedance mismatch" problem, difficulties can arise when users wish to use another language for their application programs - most ODBMSs do not provide direct language interfaces for conventional languages, such as COBOL, FORTRAN, etc.

A.5.6 Client-Server Architectures

A classification method based on client-server architectures and the unit of transfer between client and server was described by DeWitt et al. [DeWit90]. Three approaches were suggested - *object-server*, *page-server* and *file-server*. Cattell [Catte94a] added that relational databases can be viewed as a fourth architecture, called a *query-server*. It is worth noting that the object-server and query-server architectures use logical units of transfer (objects and tables respectively).

- **Object-Server**
  This architecture uses objects as the unit of transfer between client and server. VERSANT is an example of a commercial ODBMS that is based on this approach.
- **Page-Server**
  Pages are the unit of transfer in this architecture. This approach is used by commercial products such as GemStone and ObjectStore.
- **File-Server**
  This architecture is a special case of the page-server approach and uses a remote file service, such as NFS, to directly manipulate pages. Objectivity/DB is the best example of a commercial product using this approach.
- **Query-Server**
  This approach is included here for completeness, since some ORDBMSs use it. SQL requests are transmitted from client to server. The server responds with relational tuples.

A.5.7 Evolution vs. Revolution

Ultimately, perhaps all the classification models discussed so far can be generalised as evolutionary - extending existing (relational) databases with support for object concepts or revolutionary - abandoning existing database technology in favour of a fresh start. These two approaches have received some attention in the literature, e.g. [Brodi89; Spier91]. A number of market research reports, e.g. [Jeffc91], indicate that there is room for both types of database systems, although ultimately it may not matter, since the two database technologies appear to be converging. For example, POSTGRES uses a set-oriented query language (POSTQUEL), but navigational access is also possible, since each record has an OID [Stone91]. With this in mind, an important point noted by Taylor [Taylo92] is that as soon as relational database vendors begin adding pointer navigation to their products, they bring into question the entire mathematical foundation of relational technology.

A.6 Standards

The OODBTG [OODBT91] reported that there were strong arguments in favour of a standard object data model. This has been difficult to achieve due to reasons previously discussed, such as the lack of a single object-oriented paradigm [Catte94a]. However, a reference model was defined in [OODBT91]. The purpose of the reference model was:

1. To provide a common language of ODBMS definitions.
2. To provide a means to differentiate ODBMS systems from other DBMS systems.
3. To provide a means to differentiate ODBMS systems from one another.
4. To enable future standards work in the related programming and data management areas to have a basis to work from.

More recently, the task of developing standards for ODBMSs has commenced under the direction of ODMG [ODMG93], which was specifically formed for this task by the ODBMS vendors themselves.

A.6.1 Areas for Standardisation

Cattell [Catte94a] suggested standardisation efforts in a number of areas that are now discussed in more detail.

- **Object Data Model**
  It is difficult to say how many ODBMSs there are in the world today. Everest & Hanna [Evere92], for example, have cited more than 80 organisations world-wide that are:

  > "... believed to have some form of an Object-Oriented Database Management system (ODM) implemented or in development."

  Their report also highlights major differences in terminology for many things, such as exactly what constitutes an object. ODMG has taken many definitions from those used by the Object Management Group [Soley92] to help standardise terminology across ODBMS vendor products.
- **Object Query Language (OQL)**
  Currently one ODBMS vendor's query language cannot be used on another vendor's system. However, a common standard is being developed by ODMG. Outside of ODMG, there has also been debate as to what form this language should take [Works91] and whether SQL with object extensions is suitable [Beech90] or not [Orens90]. Some of the arguments in favour of using SQL as the basis for an OQL are:

- It already has an ANSI defined standard.
- SQL with object extensions would allow backward compatibility with existing applications that used standard SQL.
- It would provide a language neutral model, rather than tying down the query language to a particular programming language.

Additionally:

- It is "intergalactic dataspeak" [Commi90].
- "Our perspective is that SQL is the standard language for database access. Like democracy, it may not be perfect, but it's better than anything else that's around." Ken Jacobs, cited in [Hazza90].
- "It is socially irresponsible to invent new languages if an existing language is a good approximation to what is required." [Beech88].

However, Gray et al. [Gray92] describe the following drawbacks to using an Object SQL (based upon DAPLEX):

- Syntax of OSQL may be similar to SQL, but the semantics may be quite different, e.g. implicit joins, unexpected behaviour of familiar constructs, etc.
- Limited computational power.
- Restrictive structure.
- Awkward syntax.

In terms of query formulation, Kim [Kim90a] notes that the structure of an object-oriented query is basically that of a relational query. There are, however, significant differences between ODBMS query languages. For example, most languages support path expressions in query formulation, but vary in the degree of encapsulation. In ORION, attributes can be directly referenced, whereas in IRIS, functions are used. Access through a behavioural interface obviously provides better data abstraction than access through state.

- **Programming Language**

  Portability of application code between ODBMSs is also desirable. Lack of portability is something that relational systems have suffered from, since most use proprietary 4GLs. The ODMG approach has been to define precise language interfaces for Smalltalk and C++. Adherence to these interfaces should ensure application portability.

- **SQL**

  With the advent of heterogeneous distributed databases, a common form of communication between various types of DBMSs is required. SQL may be the best choice, since it has already become a standard communications protocol for many client-server applications. Also, so-called legacy systems cannot be ignored as significant quantities of data are already held in tables, accessed by SQL. In fact, some ODBMSs are already offering gateways that allow SQL access to the popular relational systems and permit SQL queries to be viewed as objects. Differences in SQL implementations, however, will cause problems with interoperability. For example, Edelstein [Edels91] has noted that RDBMS vendor implementations of SQL are known to suffer from the following major differences:

  - Syntactical differences.
  - Semantic differences.
  - Dictionary tables.
  - Return codes.
  - Host language interface.

  Presumably these are in addition to other "minor problems" [Commi90].

A.6.2 Achieving Standards

There are several ways standards may be achieved [Kim91].

- **Industry-Wide Efforts**

  Since the lack of a standard for object databases was seen as a limitation for their more widespread use [ODMG93], the ODMG effort was undertaken. The focus of the work by ODMG has been to provide application portability between object databases at the levels of schema, language bindings and query language. The major components of ODMG-93 include:

  - Object Model.
  - Object Definition Language (ODL).
  - Object Query Language (OQL).
  - Object Manipulation Language (OML).

  The Object Model is based on the OMGs Core Object Model and adds a DBMS profile, as shown in Figure A.13. The Object Definition Language is based on the OMGs Interface Definition Language (IDL) for specifying interface signatures and is equivalent to Data Definition Languages in other DBMSs. Currently, Smalltalk and C++ bindings to the ODL syntax have been defined. The Object Query Language is strongly typed and is similar to SQL, but it is possible to query lists, arrays, bags, etc. as well as sets. There was also some work underway to provide convergence of OQL and SQL3, as discussed by Manola [Manol94]. The Object Manipulation Language extends C++ and Smalltalk to support operations, such as create, delete, access and update, on database objects as well as provide support for transactions. Some criticisms of the ODMG effort can be found in [Kim94a; Alagi97a; Alagi97b; Alagi98].

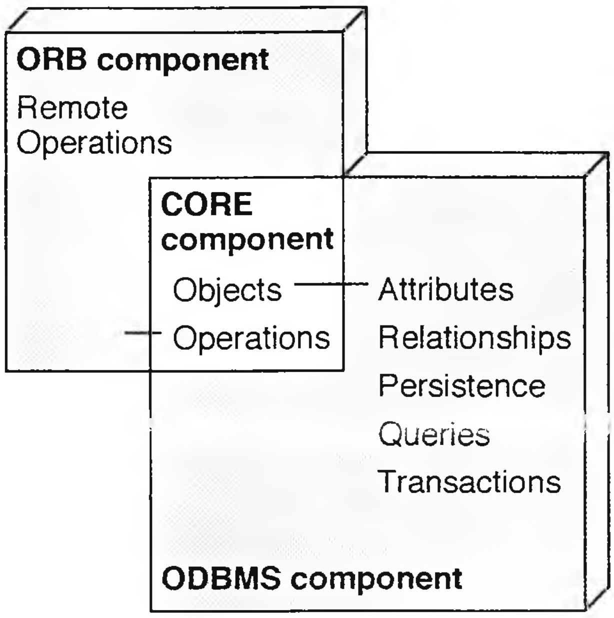

Figure A.13 - ODMG Object Model.

- **De Facto Standard**
  Of the current commercial vendors, Object Design, Inc. (ODI) has approximately a 30% market share of world-wide ODBMS sales. Whether this can be sustained in the long-term remains to be seen, particularly with the major RDBMS vendors beginning to offer significant object extensions. Furthermore, the SQL3 standard provides for some support of object-oriented concepts.
- **Major Companies**
  IBM currently has business partnership agreements with several of the commercial ODBMS vendors. It also has its own research efforts such as Starburst and Cloris [Evere92]. English [Engli92] has mentioned some other products from major systems vendors. As previously mentioned, Oracle 8 contains object extensions.

APPENDIX B - Survey of Benchmarks

B.1 Introduction

Chapter 3 described and critiqued the three most well-know object database benchmarks: OO1, HyperModel and OO7. However, as Figure 3.1 showed, many more benchmarks have been developed to test the performance of object and object-relational databases. Since the classification presented in Figure 3.2 results in some benchmarks appearing in multiple categories, they will be ordered according to the following scheme in this appendix:

1. **Application Benchmarks**
   - **CAD/Engineering**
     - Behavioural [Kempe90]
     - Engineering Database Benchmark (EDB) [Ruben87]
     - OCAD [Kempe95a; Kempe95b]
     - Object Operations 1 (OO1) [Catte88; Catte91a; Catte92]
     - Sequent [Seque93]
   - **Data Warehousing**
     - Siemens [Hohen97a; Hohen97b]
   - **Financial Trading**
     - OO-Fin [Dewan97]
   - **Geographical Information Systems (GISs)**
     - SEQUOIA [Stone93b]
   - **Hypertext**
     - HyperModel [Berre88; Ander90; Berre91]
     - Lakey [Lakey87; Lakey89]
     - Test Evaluation Procedure (TEP) [Larse92]
   - **Language-Based Editors**
     - Opus-Merlin [Emmer92; Emmer93]
     - Recognition Editors [Peder93; Peder94a; Peder94b; Peder94c]
   - **Telecommunications**
     - British Telecom (BT) [Baker91]

- **Wargame Simulation**
  - Air Force Institute of Technology (AFIT) [Hallo93a; Hallo93b]
- **Workflow Management**
  - LabFlow-1 [Bonne95a; Bonne95b; Bonne96a; Bonne96b]
- **World Wide Web**
  - Quantum Objects [STR97]

2. **System Benchmarks**
   - **Analytical Modelling**
     - Performance Evaluation System for Object Stores (PESOS) [Rabit93]
     - Teeuw [Teeuw93a; Teeuw93b]
   - **Client-Server Architectures**
     - Altaïr Complex Object Benchmark (ACOB) [DeWit90]
   - **Clustering**
     - CluB-0 [Harru91]
     - Grid [Gerlh92]
   - **Generic/Parameterised**
     - BEAST [Geppe94]
     - BUCKY [Asgar97]
     - JUSTITIA [Schre94; Schre95]
     - Kim & Garza [Kim94b]
     - OO7 [Carey93; Carey94]
     - Simple [Kelte89; Dewal90; Dewal92]
     - University of Southern California (USC) [Ghand93]

Future research should investigate rigorous taxonomies, as discussed by Worlton [Worlt93].

B.2 Application Benchmarks

B.2.1 A Benchmark to Scale Behaviourally Object-Oriented Databases

Most well-known benchmarks for object databases focus on the structural dimension of object modelling and do not consider the behavioural dimension. Kemper & Chriesten [Kempe90] described several benchmarks that attempted to model both.

The object types and behaviour modelled were based on actual CAD/CAM applications and could be considered as typical of those found in many engineering applications. The object types were referred to as **cuboid** (a small object with no references to other objects), **brep** (an object with highly interconnected sub-objects) and **robot** (a large object with many sub-objects).

The benchmarks were implemented on a relational database (SQL/DS) and an experimental non-first normal form database system that incorporated the notion of ADTs, called R^2D^2. R^2D^2 was configured in a client-server architecture, with a local database and local cache on the client. Kemper & Chriesten argued that since engineering applications exhibited reference locality, the overhead of loading objects into the client cache should be compensated by the high hit-rate - a view shared by Kim et al. [Kim90c].

The metrics used included the storage requirements for the two systems (reasonably comparable), transfer time (the time to transfer objects from the global database to the local database and subsequently to transform the objects into main memory representation), repeated sequential access to cached objects, varying the cache sizes, random access and select (direct access to an attribute versus predicate expressed with a user-defined function).

Although R^2D^2 did perform worse than SQL/DS on some tests, the benchmark developers concluded that improvements in performance could be achieved by the close integration of object behaviour into the database query language. This is an approach that has been used by most commercial ODBMS vendors that have seamlessly extended object-oriented programming languages with database capabilities.

B.2.2 The Engineering Database Benchmark (EDB)

This benchmark is discussed in section 3.2.1.

B.2.3 The OCAD Benchmark

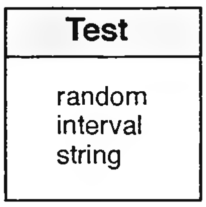

Figure B.1 - The OCAD LargeVolume Database Schema.

Kempe et al. [Kempe95a; Kempe95b] described the OCAD Benchmark for measuring the performance of CAD applications on ODBMSs. In fact, three benchmarks were actually specified: a **LargeVolume** database benchmark for testing scalability and the ability of an

ODBMS to manage large amounts of data (Figure B.1); a **ClassHierarchy** database benchmark to test the composite object modelling capability of an ODBMS (Figure B.2); and finally a benchmark that was used to implement a simple spatial index. The first two benchmarks are reminiscent of the work reported in [Kempe90].

Single-user performance was tested and warm measures taken, typical of CAD applications. The C++ interface of the products under test (Objectivity/DB, ObjectStore, ONTOS DB and VERSANT) was used, with the underlying clustering strategy provided by each product being utilised. For legal reasons, the four products were not directly named in specific performance results, but from the descriptions of these systems, it is possible to determine which results refer to which product.

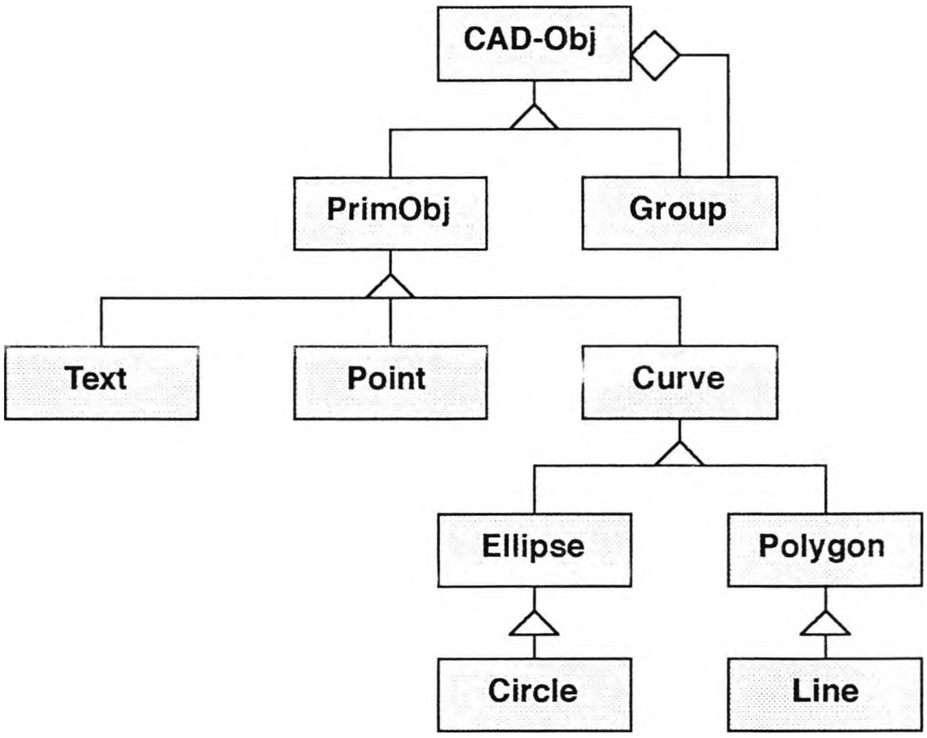

Figure B.2 - The OCAD ClassHierarchy Database Schema.

Several of the LargeVolume operations are similar to the operations of OO7 and results for two database sizes (6 MB and 60 MB) revealed that all the systems exhibited similar performance levels for the small database (this agrees with results from OO7), but for the larger database, major differences were observed. For example, for one ODBMS the database was very large due to indexes, whilst another failed to load due to memory problems. Other observations revealed that query optimisers for some ODBMSs were still very primitive.

The ClassHierarchy database operations comprised two traversal operations (one with updates) that touched 488 parts of the complex object hierarchy in a depth-first manner. Again, two database sizes (700 KB and 6 MB) revealed very interesting results. For example, one ODBMS that uses logical identifiers provided similar performance to one that uses memory pointers (a particular selling point used by the manufacturers of the latter product). Furthermore, one system failed the large database tests (this is the same product that was mentioned as failing in the previous paragraph). There was also noticeable degradation of performance with several products when moving from the small to the large database, even though the size of the complex object traversed was the same. This has implications for many production environments, where the database may be growing over a period of time and underlines the importance of testing scalability.

Thirdly, the spatial index was not directly supported by any ODBMS and had to be simulated using B-Trees. These B-Tree indexes were created on the x and y co-ordinates of a bounding box. A set of spatial operations on two database sizes (600 KB and 6 MB) again revealed interesting results. One system again failed the large database tests (the same product as mentioned in previous paragraphs). Results also revealed major differences in the intelligence of ODBMS query optimisers. A further observation was that an object-server architecture provides superior performance to page- and file-server approaches when data are not clustered according to the spatial index. The benchmark developers did not explain this further.

Finally, some results were also reported for tests to measure the sensitivity of an ODBMS to changes in object size. By varying the string attribute used in the LargeVolume database from 500 to 1400 bytes, the OCAD developers reported that an object-server architecture was highly sensitive to object size, whereas the performance of a page-server was superior, since the growth in object size does not necessarily result in equal growth in the number of pages transferred from server to client.

To summarise, the OCAD Benchmark has provided some very interesting results for a number of the major ODBMSs. It has shown how scalability is still a major problem for some products and that query optimisers are still very primitive. Perhaps most startlingly, the OCAD developers could not recommend without reservations any of the products for CAD applications. This is very surprising, since engineering and CAD applications have been the driving forces in the early adoption of object database technology, as discussed previously.

B.2.4 The Object Operations 1 (OO1) Benchmark

This benchmark is discussed in section 3.2.3.

B.2.5 The Sequent Benchmark

This is a multi-user version of the OO1 Benchmark.

B.2.6 The Siemens Benchmark

There are very few examples in the object database performance literature of application-specific benchmarks. Generally, such benchmarks are undertaken by organisations for their own internal use, as shown by the case studies in Chapter 6. The results are also rarely made public. However, one recent example of an application-specific benchmark developed at Siemens is described by Hohenstein et al. [Hohen97a; Hohen97b].

The motivation for the work by Hohenstein et al. was that they were interested in measuring the performance of complete operations at the application level, rather than low-level operations in isolation. They evaluated three object database systems - two based on the page-server architecture and the third on the object-server architecture. The application that they used for their tests was based on a relational implementation of a data warehouse. The specific steps that they used to perform their evaluation were as follows:

1. Finding Characteristic Transactions.
2. Defining an Object-Oriented Database Schema.
3. Migrating Data.
4. Reimplementing the Application in Object-Oriented Terms.
5. System-Specific Adaptation and Tuning.
6. Measurements and Feedback.

The first product that was evaluated was one of the two page-server systems. This allowed some tuning of the client cache and the unit of transfer (page size). Some tests were performed varying the page size and running queries with and without indexes. The results showed the importance of choosing good parameters, since the best performance number obtained was one-third of the time of the worst performance number. Other observations for this particular product were as follows. Firstly, page sizes above 1 KB were better for standard queries, since more objects could be transferred to the client for query evaluation. Secondly, small page sizes were sufficient for lookups of objects, since an index could do a direct selection. Thirdly, some queries were actually slower with indexes than without!

Finally, the size of the client cache was not a major influence on the performance of this system.

The second product was the other page-server system. This also provided tuning of the client cache size and unit of transfer, as well as some server-specific parameters, such as the sizes of log files. Investigating alternative query forms with and without indexes revealed a number of interesting observations, as follows. Firstly, preprocessing of frequently used queries resulted in worse, rather than better, performance. Secondly, indexing could be very effective. Finally, the size of the client cache was the most important influence on performance in this system.

The third product tested used an object-server architecture. It provided many tuning parameters, but no benefits were found from varying the queries, as with the two other products. The most effective optimisations were as follows. Firstly, indexes on every subclass gave the best query performance (and implied, therefore, that a deep class hierarchy would have a negative impact on performance). Secondly, three different commit options were available: (i) invalidate the client cache and release locks, (ii) keep the client cache and release locks and (iii) keep the client cache and keep locks (which provided the best performance). Finally, several options were also available to flush the transaction log file, such as after every transaction or asynchronously to disk.

A comparison of the three products showed that the page-server systems were better at complex searches, especially when large quantities of objects needed to be transferred to the client. However, the object-server provided the best overall performance, particularly using the third commit option, mentioned above.

A comparison with the original relational database implementation showed that the object databases were good at performing simple queries. Also, traversals of relationships in the two page-server systems were better than the equivalent SQL queries and additional benefits could be derived by using clustering. Furthermore, code savings were possible with the object database implementations due to the seamless integration of an object-oriented programming language with the DBMS, as well as support for powerful modelling constructs, such as inheritance and polymorphism. However, only the object-server was competitive in performance with the relational database, mostly due to its low commit overhead. The benchmark developers also speculated that in a multi-user environment, the object-server would perform better than the page-server systems, as it supported object-level locking.

To summarise, the benchmark developers concluded that the performance differences they observed were substantially different from results that had been reported by some object database performance researchers. Also, exploring many of the tuning options had shown that considerable performance improvements could be achieved. Furthermore, one of the most important issues that emerged was the choice of architecture, with the object-server clearly providing the best performance. Finally, the benchmark developers concluded that standard benchmarks were not always meaningful and application-specific tests were essential. This agrees with the results of this research project.

B.2.7 The OO-Fin Benchmark

This benchmark is described by Dewan & Agarwal [Dewan97] and was primarily designed to measure the performance and scalability of Persistence (mentioned in the previous appendix) against an in-house object to relational mapping tool used at Morgan Stanley. The schema is illustrated in Figure B.3 and is designed to represent a financial trading application.

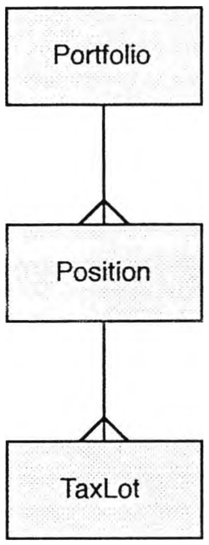

Figure B.3 - The OO-Fin Database Schema.

The cardinality of the relationships is that a **Portfolio** is connected to 20 **Positions** and each position to 50 **TaxLots**. The benchmark was used to determine the overhead of mapping relational information into objects (performed by Persistence and the in-house tool), the benefits of caching and cache scalability. Various database sizes were tested, with the largest being 100 Portfolios, 2,000 Positions and 100,000 TaxLots (approximately 44 MB in total). The tests were performed on the same machine and measurements were taken for both cold and warm reads. The results of the cold read showed that both Persistence and the in-house tool reported similar performance numbers, with the access time per object instance being almost independent of the number of instances in the cache. For the warm read, Persistence again demonstrated constant performance, whilst the performance of the in-house tool degraded with database size, possibly due to the different techniques it used to manage objects [Dewan97].

Besides the superior performance demonstrated by Persistence, the results also showed that it was 250 times faster than using just an RDBMS. This is because Persistence provides a mechanism, called a Live Object Cache (LOC), that enables objects constructed from relational database tuples to be held locally in a client cache and the LOC effectively behaves like an object database cache.

The benchmark developers also proposed a number of extensions, which they indicated they would address in future tests:

- **Scalability** - use data sets greater than 1 Million objects.
- **Updates** - test various update operations, such as batching changes until a transaction committed.
- **Distribution** - perform tests using an Object Request Broker (ORB).

B.2.8 The SEQUOIA 2000 Benchmark

The SEQUOIA 2000 Benchmark [Stone93b] is based on real data and queries typical of those found in GISs. The authors claim that the benchmark is also representative of other applications such as engineering and can therefore be used in a more generic manner.

Geographic systems are characterised by three major features - large database size, complex data types (such as spatial data) and queries on complex data. Existing benchmarks such as OO1 and OO7 are, therefore, not appropriate since they focus on very small database sizes and the efficiency of pointer traversals in main memory on simple objects.

The benchmark consists of ten queries, oriented towards four data types (raster, point, polygon, directed graph) and one operation for database load. Three database sizes were discussed, but results were reported only for the smallest database (1 GB) with a larger database (18 GB) reported as being in preparation.

Results were reported for GRASS (a public domain GIS), IPW (a raster image processing workbench) and POSTGRES (a DBMS research prototype investigating the feasibility of extending the Relational Model with objects). The most complete benchmark results were available for the POSTGRES implementation, which found a number of problems and limitations of this system. It is difficult to draw further conclusions until the benchmark has been more widely implemented - a significant problem, since most commercial relational and object database systems do not support all the complex data types required. However, the authors have realised this and partial results may also be reported. The benchmark is also criticised by Boncz et al. [Boncz96b] for the simplicity of the queries and lack of thematic data.

SEQUOIA 2000 is a storage benchmark, but network and visualisation benchmarks have also been developed as part of this project.

B.2.9 The HyperModel Benchmark

This benchmark is discussed in section 3.2.4.

B.2.10 Benchmarks Proposed by Lakey

Lakey et al. [Lakey87] and Lakey [Lakey89] proposed a number of benchmarks to compare object and relational database systems based on complete operations at the application level, rather than measuring low-level primitives. By using conceptual schemas and operations on those schemas, it was argued that it was possible to develop benchmarks that could be implemented on any database system. Furthermore, it was suggested that since ODBMSs themselves differed markedly in terms of implementation and there was a lack of consensus on a standard object-oriented model (at the time this work was undertaken), the proposed approach could also be used to compare ODBMSs. Three major factors were also proposed that distinguished object database benchmarks from other data management systems:

1. Operations in object database applications are more likely to follow conceptual access paths, rather than logical or physical ones [Stein92]. For example, in conventional DBMS applications, an operation may include a sequential scan of a relation for report generation or single record lookups using an indexed field. However, for object databases, a typical operation may be to compute the weight of an object by combining the weights of all its sub-components [Stein92].
2. In an object database, a programming language is more tightly integrated with data management. The language acts as the Data Definition Language (DDL), Data Manipulation Language (DML) and Data Control Language (DCL), besides being computationally complete. This contrasts with SQL, for example,

which provides DDL, DML and DCL facilities only, requiring a host language to provide general computation. Consequently, it is better to evaluate how well the language and data management features work together, rather than as individual components.

3. Traditional metrics tend to focus on individual query response time or transaction throughput. Other factors, more difficult to quantify, such as rapid prototyping, ease of implementation, maintenance, etc. are not considered.

Since the aim of this work was to compare object and relational database systems, GemStone and University INGRES were chosen. The principal reason being the availability of suitable software. Furthermore, *relative performance* was measured within each system, since the two products could not be tested on identical hardware configurations. Five candidate applications were considered:

- Block Structured Programs.
- Document Processing.
- Hypertext.
- Persistent LISP Objects.
- VLSI Design.

These applications were chosen as they were considered to be of an object-oriented nature and published work describing relational schemas was already available. The **Document Processing** and **Hypertext** applications were actually implemented and benchmarked.

After implementing and running her benchmarks, Lakey [Lakey89] concluded that schema design was a major performance factor for both systems - improvements might be possible with alternative designs. Other quantitative and qualitative observations she made were that GemStone was superior to INGRES in several areas:

- I/O management (it accessed the disk only when objects were needed).
- Data model (closer to the real world).
- Ease of implementation (model definition, method implementation and statistics generation could be defined using just one language).
- It could better manage information for applications within the database itself, such as class definitions and application objects.

However, modifying schemas and bulk database loading and unloading were easier to undertake in INGRES than GemStone.

B.2.11 The Test Evaluation Procedure (TEP)

Larsen [Larse92] describes a Test Evaluation Procedure (TEP) to compare object and relational database systems using a simplified version of the HyperModel Benchmark. The TEP was partly based on a requirements analysis for two different applications termed **administrative** and **technical**. Interestingly, there appear to be significant similarities in the requirements for the two application types.

After a discussion of object and relational database technologies and in a survey of database performance benchmarks, Larsen critiqued HyperModel:

- The benchmark description was too generalised and details on how to create test applications were left unspecified.
- The overall benchmark schema was described, but not how queries should be performed.
- There were no guidelines on how measurements should be made, e.g. whether measurements should be taken inside or outside transactions.
- The two database systems under test in [Ander90] had different programming language bindings which may have meant that any hidden behaviour was not revealed.
- The benchmark was expensive to implement, perhaps requiring four to six man months per system (this included the time required to learn each product and to create the implementation and operating procedures).
- Most of the operations appeared to measure the same things.
- The benchmark was very weak in its cold measurements, since it performed a very large number of benchmark iterations, which meant that considerably more warm measurements were taken.

Following the critique, Larsen proposed a simplified benchmark schema and operations. This is reminiscent of the approach used on the original Engineering Database Benchmark [Ruben87], which was also simplified in its design and number of operations, as reported by Cattell [Catte88]. The modified schema used by Larsen is illustrated in Figure B.4.

Figure B.5 shows the **parent/children** hierarchy with the number of nodes at each level. Non-terminal nodes are of type **Node** and terminal nodes of type **TextNode**. Each node (of type Node or TextNode) is connected to five other random nodes in the database for the **refTo/refFrom** relationship. As with the original HyperModel, tests were run against level 4, 5 and 6 databases (781, 3,906 and 19,531 nodes, respectively).

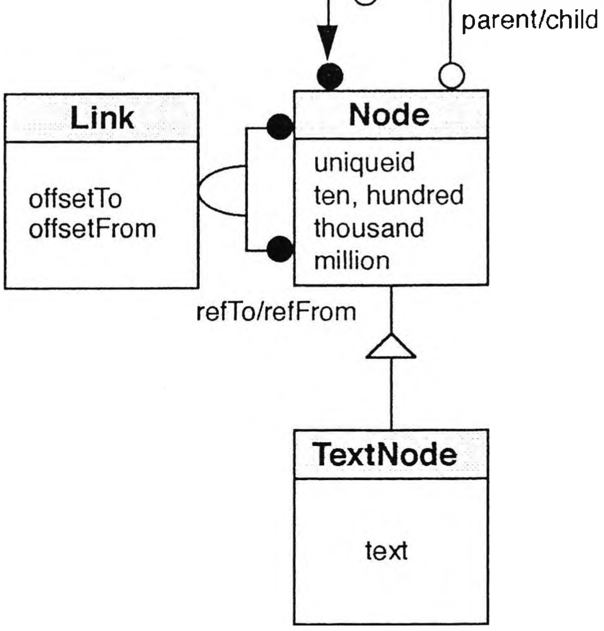

Figure B.4 - The TEP Database Schema.

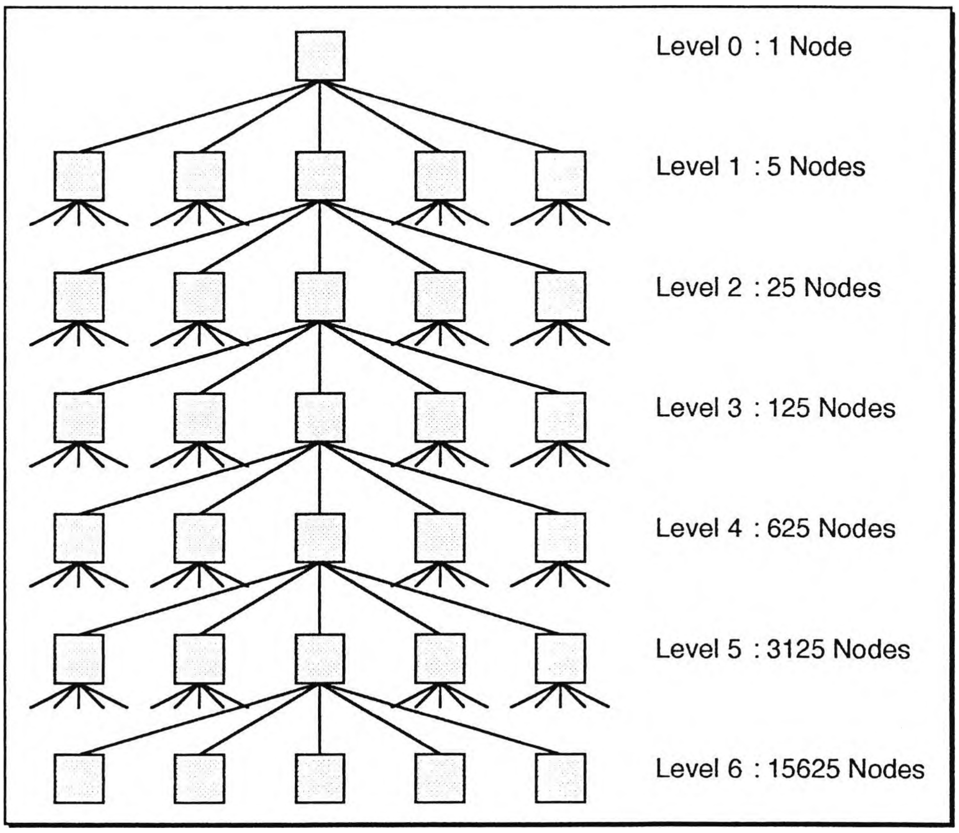

Figure B.5 - Network of Nodes used for HyperModel Database.

The proposed benchmark operations consisted of six retrieval and traversal operations (with and without updates) of varying complexity for both single- and multi-user. Indexing and transaction mechanisms were also tested. The multi-user tests were configured so that applications were executed simultaneously on different client computers against the same database, which was held on a remote server. The aim was to test the ability of the DBMS to support concurrent access to the same data structures as well as how it was able to cope with deadlock situations. Interestingly, all the tests were conducted with other users logged-on to the computer systems, leading to the obvious conclusion that the tests were not undertaken in a controlled manner and casting doubt over the reproducibility of the results.

Test measurements were reported for the RDBMS and one ODBMS, with the results for a second ODBMS included as an appendix. The times for the database load operations showed that the ODBMS was faster in all cases. The ODBMS was also smaller for each level after loading (ranging from a quarter to a half of the size of the RDBMS). This was probably due to the direct connectivity between objects in the ODBMS, rather than requiring multiple tables to recreate the link structures as used in the RDBMS. For the single-user tests, the ODBMS did provide better performance than the RDBMS for many operations for level 4 and level 5 databases, but was noticeably worse for level 6 databases. Larsen speculated that this was due to client cache evictions, since the database at level 6 could not be stored entirely in workstation memory. The multi-user results followed similar patterns to the single-user results, with differences in performance between the two database systems becoming larger as the number of users was increased. Overall, however, Larsen commented that the performance differences in many cases were not of the one or two orders of magnitude in favour of the ODBMS over the RDBMS as had been suggested elsewhere.

On the basis of his results, Larsen concluded that an ODBMS would not be suitable for the administrative application, but would be more appropriate for the technical one, although the two ODBMSs and RDBMS did well against the requirements analysis undertaken as part of his research. Additional conclusions were that good ODBMS performance was very dependent on a number of issues, such as object clustering and that scalability should also be tested as demonstrated by the cache evictions that were observed with the level 6 databases. However, since the two object databases used by Larsen were not named (or even described in enough detail to determine their architectures), it is difficult to determine whether the reported results contradict results presented elsewhere.

B.2.12 The Opus-Merlin Benchmarks

The ESPRIT-III project GoodStep was aimed at enhancing the O_2 object database to make it particularly suitable as a database for Software Development Environments (SDEs). An integrated SDE is described in [Emmer93] as one that includes a number of tools to support many of the life-cycle stages, such as document development (with possibly many connections between documents at varying levels of granularity). Typically in an SDE, there will be tools to provide graphical representations of system design (e.g. Entity-Relationship diagrams) and other tools to provide code generation. These tools will exchange information between each other. For such environments, the response time must be below 1 second for the SDE to be considered user friendly [Emmer93]. To assess the performance of an object database (which dominates the performance of an SDE built on top of it), the Opus-Merlin Benchmarks were defined.

The difficulty in developing an appropriate benchmark, however, is complicated by the differences between commercial ODBMSs in areas such as functionality, features, lack of a standard interface, etc. Consequently, an *abstract benchmark* was proposed that could then be implemented on any object database. The approach was to take two documents that represented certain module information. One contained details of modules and their import interfaces. The other defined the import and export information in detail. Next, a series of transformation rules was applied to convert the textual representation to an Extended Entity-Relationship (EER) model. Further simplification rules were applied to reduce the size of this model, which could then be implemented on a specific system. Furthermore, four sets of benchmark operations were proposed, divided into two groups called **increment operations** and **traversal operations**. The former being those that created, changed or deleted objects and the latter being those that performed traversals. Interestingly, running these tests in the specified order will leave the database unaltered and so additional runs can be undertaken without the need to create a new database. The complete benchmark was also designed in a modular manner consisting of sub-systems, enabling significant code re-use for implementations on a variety of different products.

Detailed results for this benchmark were described in [Emmer92] for GemStone and a prototype graph storage system called GRAS [Kiese92]. The results showed that although GemStone was functionally very rich (e.g. support for schema definition, transaction management, distribution, etc.), it provided superior performance to GRAS in all but one case. Furthermore, it utilised disk space more efficiently and its benchmark times were within 1 second. Additionally, using GemStone in a remote client-server configuration didn't add any significant response time overheads. In [Emmer93] it was reported, however, that the pessimistic transaction management approach used by O_2 was superior to the optimistic approach used by GemStone, since additional time was required in the latter at commit time for conflict detection. The benchmark developers concluded in [Emmer92] that with further improvements in ODBMSs (something that usually occurs with each software upgrade) and faster hardware platforms, performance could be improved where benchmark times may be under 500 milliseconds (a figure arbitrarily chosen by the benchmark designers), making object databases even more suitable for SDEs.

B.2.13 Benchmarks for Recognition Editors

Pedersen [Peder93; Peder94a; Peder94b; Peder94c] describes some work to investigate the feasibility of utilising ODBMSs for LBEs. According to Pedersen [Peder93], the definition of an LBE is as follows:

> "Language-based editors facilitate construction of documents that are correct from the syntactic and static semantic viewpoint of the language concerned. The language may be any structured language, such as a programming language or a specification language."

Following a discussion of tree-based editors and recognition editors in [Peder93], three strategies for persistence were considered:

1. **Direct Persistence** - the LBE is coded in a persistent language.
2. **Direct Integration** - a full ODBMS is used.
3. **Loosely-Coupled Integration** - an ODBMS is used as a back-end server.

The third option was considered as being the most favourable, since it was able to provide the benefits of a DBMS, such as persistence, whilst providing data-sharing with other tools.

In [Peder94b], a requirements analysis for a recognition editor benchmark was undertaken. Some strategies used to develop a benchmark for recognition editors as a first step towards migrating them to a persistent storage environment were then described. The approach used was based on the work reported by Emmerich & Kampmann [Emmer92]. The steps required to develop an abstract benchmark for the UQ1 editor developed at the University of Queensland were subsequently described in [Peder94a]:

1. Undertake a conceptual analysis of the UQ1 document-representation structure to highlight the significant objects.
2. Use quantitative analysis techniques to determine the average occurrence of the objects identified in 1. in a UQ1 document.
3. Use the figures from 2. to create an initial database of objects in the persistent storage environment.

4. Undertake an analysis of the typical operations performed by the UQ1 editor such as the number of objects created, visited or destroyed.

From the initial results Pedersen obtained, he proposed that the approach described above could provide a useful technique to develop benchmarks for a wider range of ODBMS applications. This work should also be compared with that reported in [Emmer92; Emmer93].

B.2.14 The British Telecom (BT) Benchmark

Baker & Salman [Baker91] report on the results of some work to compare a relational database, a relational database with object extensions and a pure object database. 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 object database 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 relational and object systems. Not surprisingly, the performance gap between relational and object became wider as queries became more complex, although the cost of retrieving a single tuple was discovered to be equivalent to retrieving a single object. 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 object database to two main reasons. Firstly, 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. Secondly, since the object database 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.

B.2.15 The Air Force Institute of Technology (AFIT) Benchmark

There are few published benchmarks that attempt to model some type of simulation domain, although many benchmarks could be described as simulating the data manipulation requirements of particular applications. One ODBMS performance benchmark specifically targeted at *stochastic discrete-event computer simulation* is described by Halloran

[Hallo93a]. This benchmark simulates a battlefield environment with aircraft searching and logging any trucks found. It is described by Halloran as a quantitative *and* qualitative benchmark and measures elapsed time and throughput.

The benchmark is unique in a number of ways with its design. For example, a Graphical User Interface (GUI) is included as part of the benchmark. This is a qualitative measure of an ODBMS to interface with a graphics library.

The benchmark consists of two database sizes:

1. **Small** - 1,000 trucks, 500 aircraft and a 50 x 50 hex board.
2. **Large** - 10,000 trucks, 5,000 aircraft and a 100 x 100 hex board.

It measures seven major operations (Model Creation, Scenario Creation, Simulation Execution, Simulation Throughput, Version Creation, Map Creation and Report Creation). Furthermore, three simulation events are modelled [Hallo93a]:

1. **Aircraft Move** - moves an aircraft into a randomly selected adjacent hex.
2. **Search** - aircraft searches hex it is located-in and logs any trucks found.
3. **Truck Move** - moves a truck into a randomly selected adjacent hex.

The benchmark was implemented on both ObjectStore and C++. The latter, it was argued, was not a valid implementation since it was non-persistent, but was used for comparison purposes to determine the performance overhead due to the extra functionality provided by the ODBMS, such as database management services.

Local and remote results for the small ObjectStore database and small non-persistent C++ database showed that the ODBMS provided comparable performance to the non-persistent version. Interestingly, the work reported in this thesis is one of the few published efforts that performs detailed statistical analyses of the results. From the results obtained, Halloran concluded that the ODBMS provided extra functionality without significant performance overheads and cited the following advantages of using ObjectStore for simulation systems:

- **Similarity to the C++ Programming Language**
  A simulation system developed using C++ can easily be ported to ObjectStore, since the data models are so similar. For other OO or non-OO languages, however, this may be a drawback.
- **Motif Interface**
  ObjectStore worked well with the Motif GUI.

- **Multi-User Access to Model Data**
  ObjectStore provided good support for multi-user access, with no consistency problems encountered. For simulation systems implemented in a programming language, additional code to manage multiple users would be required.
- **Browser Tool Use**
  A graphical database browser tool with ObjectStore enabled the database to be examined in an ad-hoc manner. Such tools may be useful for simulation systems.

The conclusions suggest that some ODBMSs may be suitable for simulation systems, since they provide the added functionality that may be required by such systems with only a small loss in performance, but that further work was necessary in some areas, such as version management (which was not evaluated as part of this work).

B.2.16 The LabFlow-1 Benchmark

The LabFlow-1 Benchmark [Bonne95a; Bonne95b; Bonne96a; Bonne96b] was designed to measure both the performance and functional characteristics of a DBMS to support workflow activity. According to the authors, this requires the database to manage audit trails and event histories, together with queries, indexing and dynamic schema evolution. The benchmark described in this work was based on observed data and workflow activity at the Human Genome Project at MIT, although the benchmark itself was synthetic, since a benchmark based on real data and workloads would be complex, hard to understand and difficult to scale [Bonne95b]. However, the essential components of observed workflow activity, such as cycles, success and failure states and multiple co-operating production lines were all simulated in this benchmark. Existing benchmarks were found to be unsuitable, since workflow deals with a small number of complex, correlated events [Bonne95b].

The benchmark was implemented on the commercial ODBMS product ObjectStore and the research prototype Texas (similar to ObjectStore in its virtual memory mapping architecture) and measured single-user performance, since it was useful to understand this before attempting to measure multi-user performance [Bonne95b]. Five variations of ObjectStore and Texas were used:

- OStore (full ObjectStore).
- Texas (full Texas).
- Texas+TC (full Texas with extra clustering capabilities implemented in client code).

- OStore-mm (ObjectStore running in main memory only).
- Texas-mm (Texas running in main memory only).

The latter two obviously having no storage managers. Three database sizes were used and corresponded to half, full and twice physical memory. The benchmark operations were grouped into the following five categories:

1. Retrieve materials in a given state.
2. Query results of workflow activities.
3. Insert new material instances.
4. Insert new step instances.
5. Modify the state of materials.

As mentioned earlier, the workflow activity used as a basis for this work was drawn from the Human Genome. DNA sequencing was the specific activity being modelled.

The major results of the benchmarks showed that thrashing became a significant issue the larger the database size, from which the benchmark developers concluded that clustering frequently accessed objects was critical to obtaining good performance. Similar observations on the importance of locality of reference for ObjectStore have been reported by Halloran [Hallo93a]. Furthermore, a functional comparison of the two products in terms of concurrency control, class libraries, tuning options and database administration tools revealed large differences in favour of ObjectStore. However, this is not really surprising, since Texas is a single-user research prototype and ObjectStore a full-featured commercial ODBMS.

B.2.17 The Quantum Objects Benchmark

Most performance benchmarks for object databases have been written in C++. Several of these benchmarks have also been implemented in Smalltalk. However, the increasing interest and commercial use of Java would be a natural candidate for new benchmarks, as many object database products now provide Java language bindings. One effort to measure the performance of Java data management is described in [STR97]. This paper compared two alternatives to Java data management using ObjectStore and JDBC/Oracle. The comparison included both code sizes and performance for the two database technologies, based on a simplified version of an on-line arts and entertainment guide for the city of Chicago. This application was a web-based system that dynamically generated HTML pages from a collection of components stored in the database. These components included images, text and HTML pages. Components could themselves have further elements, resulting in a hierarchical structure.

A code comparison of the entire application favoured ObjectStore, which was three-quarters of the size of the Oracle implementation. Some code examples also demonstrated the virtually seamless approach possible with ObjectStore, whilst mapping objects to relational tables was more complex and required additional considerations, such as whether to represent each class as a table. The relational implementation was also very rigid, since it was difficult to add new object types without adding new tables and additional code to manage them. In contrast, new object types could be added very easily to the object database.

A performance comparison favoured ObjectStore over Oracle by a far bigger margin than the code comparison. This was attributed to a number of reasons. For example, there was no translation overhead with the object database, as programming language and database objects were the same, whilst the relational implementation required objects to be copied into memory and reconstructed. Additionally, the object database was able to use client caching to improve performance.

B.3 System Benchmarks

B.3.1 The Performance Evaluation System for Object Stores (PESOS)

Rabitti et al. [Rabit93] describe a software tool for estimating the cost of associative retrieval in very large and complex object stores. The tool, called PESOS, can model various object storage organisations with respect to specific characteristics of the secondary memory hardware and provides a simple language to specify queries on the object store structures and returns quantitative evaluations, using analytical methods, of the access performance. The advantage of this tool is that these evaluations can be undertaken without the need to actually implement any database (which can be a costly and time-consuming process).

PESOS can describe:

- Object storage structures.
- Value-based and navigational indexes.
- Hardware characteristics.
- Set-oriented and navigational queries.

The interface to the tool is kept simple to promote portability across hardware platforms.

No results are reported in this paper, although some detail is provided about its design and the types of parameters that can be specified. Furthermore, there is no evidence of any model verification. For example, how does PESOS compare in its evaluations with actual databases?

B.3.2 Performance Work Reported by Teeuw

Recent work using ACOB (discussed below) has been reported in [Teeuw93a; Teeuw93b], which described an analytical model to estimate disk I/Os. The model was validated by experiments using four storage models for complex objects.

B.3.3 The Altaïr Complex-Object Benchmark (ACOB)

The Altaïr Complex-Object Benchmark (ACOB) was developed to test the performance of alternative workstation-server (client-server) architectures for object database systems [DeWit90]. The three approaches explored were:

1. **Object-Server** - individual objects are transferred between workstation and server using a Remote Procedure Call (RPC) mechanism.
2. **Page-Server** - individual disk pages are transferred using an RPC, with the server buffering these pages.
3. **File-Server** - disk pages are transferred and accessed by the workstation using a remote file service, such as NFS.

The object-server architecture is used in systems such as ORION and VERSANT. EXODUS and ObjectStore are examples of the page-server architecture. Finally, Objectivity/DB uses the file-server configuration.

According to [DeWit90], the design of the benchmark was influenced by the data model of the O_2 system, which distinguishes between values and objects.

A complex object is composed of seven records (to represent values), with each record being 112 bytes long. The records are connected in a hierarchical manner, as illustrated in Figure B.6. The four leaf records contain references to eight other complex objects to simulate 1:N aggregation relationships (e.g. part/subpart) or M:N relationships (e.g. suppliers/parts). The attached objects are termed "components" [DeWit90]. A total of five sets of 1,500 objects were used to create the test database.

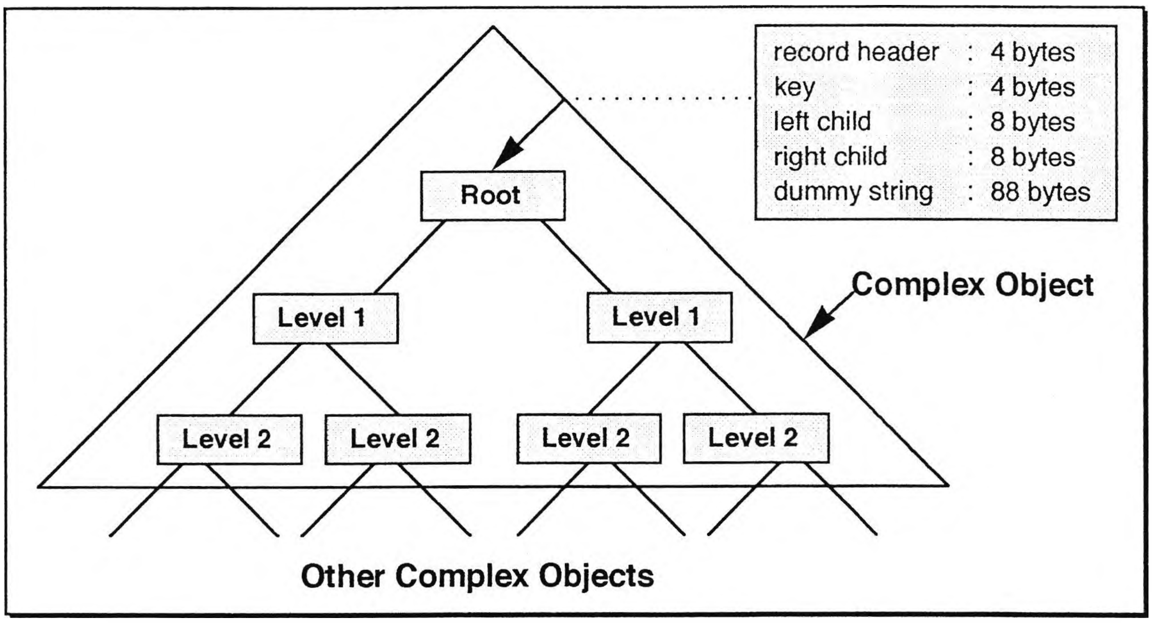

Figure B.6 - ACOB Complex Object Organisation.

The benchmark consists of three queries:

1. **Sequential Scan**
   This query reads all complex objects (but not their components) in their physical order. It simulates reading all instances of a class and is therefore similar to queries for relational systems that read all rows of a table.
2. **Random Read**
   This query randomly selects 300 complex objects and their components. A partial traversal is then undertaken and an average of 44 records are read. The selectivity factor chosen was based on observed access patterns for VLSI tools. The query simulates the user checking-out a complex design into a local workstation at the start of a transaction or session.
3. **Random Update**
   This is an update version of 2. In-place updates are performed on 17 records, thus preserving the structure of the database. This query was primarily used to observe the effects of transferring objects between workstation and server on the three architectures mentioned earlier.

In running the queries, a number of parameters were varied, including the degree of clustering, “smearing” (to simulate the dispersal of records over different data pages caused by updates) and workstation and buffer pool sizes.

Briefly, the results showed that the object-server architecture was not really affected by clustering. Workstation buffer size was a factor to a point, beyond which the cost of RPCs was dominant. For sequential scan and update, the page-server architecture was very sensitive to the workstation buffer pool and clustering. The file-server was more sensitive to the workstation buffer pool than the page-server. Read operations performed well, but page writes were slow using NFS, since it uses non-buffered writes. Overall, there was no clear winner [DeWit90].

To summarise, ACOB has been successfully used to observe the effects of clustering, "smearing" and buffering on three different workstation-server architectures. Although it could be criticised for being a very simple benchmark, it showed that a small set of focused queries can be successfully used and that a very large and complex benchmark is not necessarily required.

B.3.4 The CluB-0 Benchmark

Harrus et al. [Harru91] used a derivative of the HyperModel Benchmark to evaluate alternative clustering strategies. They proposed performance indices to measure the efficiency of clustering decisions and reported upon their implementation on an early version of O_2, together with detailed performance results.

B.3.5 The Grid-Benchmark

The Grid-Benchmark [Gerlh92] was developed to test a new clustering strategy for ODBMSs based on: (i) workflow analysis techniques to determine access patterns, followed by (ii) heuristics for graph partitioning to determine which objects should be clustered together.

The workflow analysis phase extracts access patterns from object behaviour. This technique is referred to as **decapsulation** and is a static approach in that schema and object instances are analysed (versus the dynamic approach where real applications are monitored). The dynamic approach is inferior to the static approach in several respects [Gerlh92]:

- There is a run-time penalty imposed on any running applications, since monitoring must involve some resource utilisation.
- There is a time penalty for the data collection used by the monitoring process to reach a "steady state".

- Only average system load is measured and rare (but critical) operations are ignored.

The static approach allows weighted measurements to be assigned to inter-object references. Those paths that are likely to be traversed more frequently are assigned correspondingly higher weights.

The OO1 Benchmark was used together with a new Grid-Benchmark in the experiments, as example applications to test the new clustering strategy. The OO1 Benchmark was criticised in the following areas, though:

- The simple schema did not reflect real-world applications, which contain more complex types and operations with diverging access patterns.
- The benchmark operations and results were difficult to interpret due to the random connections between objects.

The Grid-Benchmark was designed to address some of these deficiencies, specifically:

1. Use of recursion to truly stress clustering strategies.
2. Support for a complex schema with a variety of operations and access patterns.
3. Inclusion of both regular and random connections between objects.

The test results showed that the new clustering approach did provide superior performance than alternative object clustering approaches.

The conclusions suggest that determining static access patterns could provide a useful way forward for the development of other ODBMS benchmarks. The difficulty for database performance researchers, of course, is gaining access to the necessary schema and design documentation!

B.3.6 BEAST

BEAST is a derivative of the OO7 Benchmark, for active database systems.

B.3.7 The BUCKY Object-Relational Benchmark

This benchmark is described by Asgarian et al. [Asgar97] and is a query-oriented benchmark that is designed to test features of object-relational databases, such as:

- Row types with inheritance.
- Inter-object references.
- Set-valued attributes.
- Methods attached to row objects.
- User-defined ADTs and their methods.

The schema is illustrated in Figure B.7, with generalisation represented by a triangle and other lines illustrating other relationships (for brevity, the names of these relationships have been omitted in the diagram).

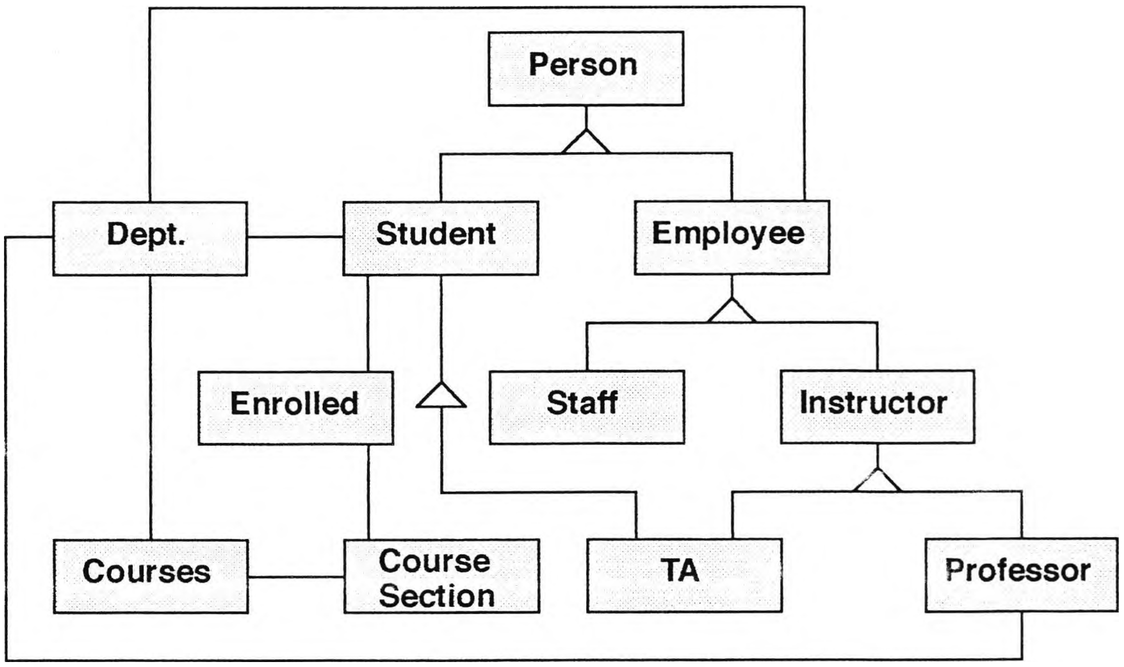

Figure B.7 - The BUCKY Database Schema.

The focus of the work was to test those features provided by ORDBMSs that were above and beyond those provided by just RDBMSs. Furthermore, the benchmark developers also implemented a purely relational schema on the system under test, to see what differences this might make with query optimisation, etc.

The results showed that the object-relational implementation did provide benefits in some cases, such as smaller code, but object-relational query processors were still weak in managing the object extensions they provided. Other observations they reported with the object-relational database when compared to the relational implementation of their benchmark were:

- The object-relational database size was larger.
- Loading the object-relational data set was more complex and time consuming for both the benchmark developers and the system under test.
- Some queries for the object-relational database needed more care and attention.
- Inheritance, when using abstract base classes in the object-relational system, caused unnecessary work.
- Loading method code, written in a high-level language in the object-relational system, could be expensive.

The relational implementation performed faster than the object-relational implementation on most of the queries described in [Asgar97].

The benchmark developers also proposed two metrics: an **efficiency index** (for comparing object-relational and relational implementations) and a **power rating** (for comparing different object-relational database systems). The former is a ratio of the Geometric Means of all test times of object-relational to relational and a value of less than one indicates that the object-relational version of the benchmark is faster than the relational version. The latter is a measure of the absolute performance of an object-relational system and is the value 100 divided by the Geometric Mean of all an object-relational systems' test times.

To conclude, the benchmark developers felt that current object-relational database systems provided both benefits and drawbacks. The benefits were realised by better expressiveness and more compact queries than relational equivalents, whilst the drawbacks were mainly due to the larger design space, more implementation choices available and generally poorer performance than relational equivalents.

B.3.8 The JUSTITIA Generic Object Database Benchmark

The JUSTITIA Benchmark [Schre94; Schre95] was designed to be a flexible and user-configurable ODBMS benchmark. It draws upon previous ODBMS benchmarks, such as OO1, HyperModel and OO7, for some of its schema design and operations. Figure B.8 illustrates the schema components, which can be configured through parameters to represent the schemas of other ODBMS benchmarks. For example, JUSTITIA can be configured to represent the HyperModel benchmark structure.

Nodes representing **Container Objects** are configured to take the appearance shown in Figure B.9. Not illustrated is that **Primitive Objects** are also connected to other Primitive

Objects within the ring. This is equivalent to the approach used by OO7 with its AtomicParts.

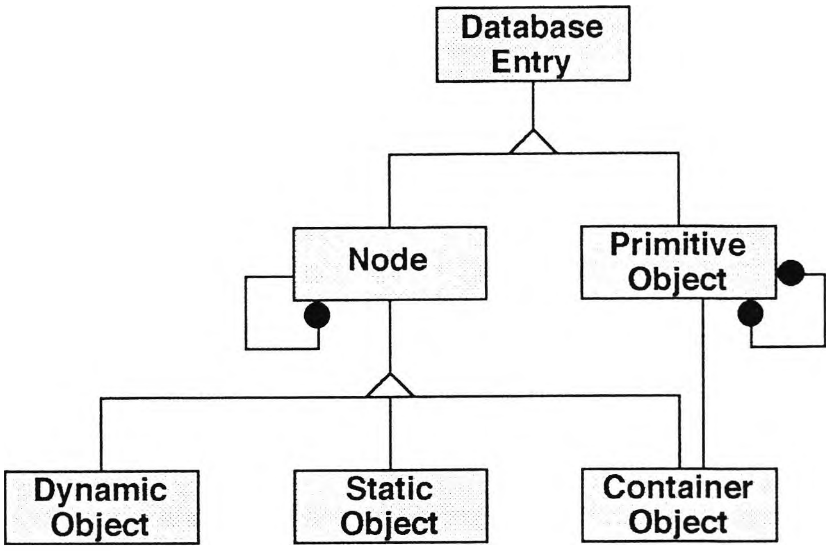

Figure B.8 - The JUSTITIA Database Schema.

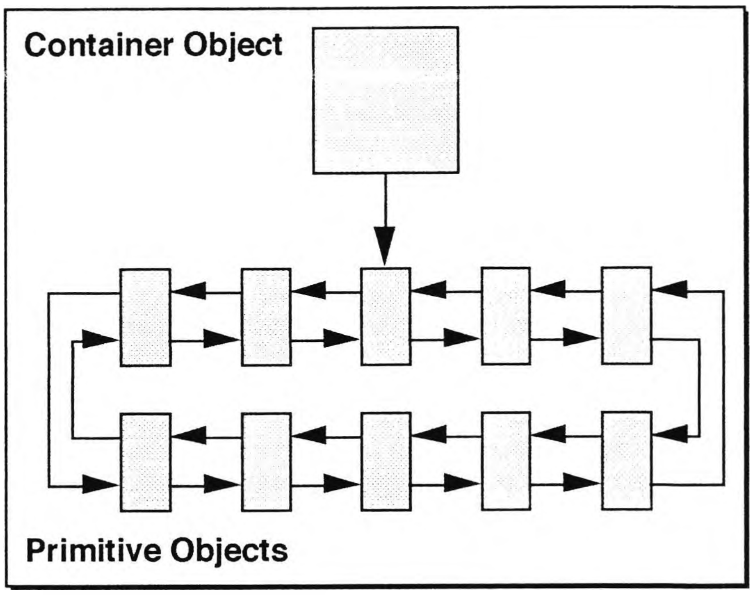

Figure B.9 - JUSTITIA Container Object with Primitive Objects.

As with OO1 and OO7, the database operations appear suggestive of engineering applications and the example described in [Schre94] is based on CAD parameters from the ship building industry. The major distinguishing factors between JUSTITIA and other ODBMS benchmarks, however, are its support for multi-user tests and structure preservation in the face of dynamic changes to the physical database. For example, Schreiber [Schre95] described some results obtained using multi-user and structure preservation with simple read/write and extended read/write operations. These results would not have been observed using other benchmarks. One set of results, for example, showed how an ODBMS based on a virtual memory architecture provided superior performance to one based on NFS for simple multi-user read/write operations, but this situation was totally reversed for more complex read/write operations. Wade [Wade96] suggests two reasons for the poor performance of the virtual memory architecture ODBMS. Firstly, the product lacks object-level access (i.e. there is no object manager), so there is no co-ordination of access to objects by multiple users - once objects are swapped into virtual memory, they are gone from the DBMS and from other users. Secondly, the call-back mechanism results in exponentially growing number of messages, which are slow as users are added. However, results from multi-user benchmarks should be interpreted with caution [Bradl94].

To summarise, JUSTITIA has been designed for comparisons between ODBMSs and provides a user-configurable database schema and some user-configurable operations. It can, therefore, be used to generate quite complex schemas. The "generic" label used in [Schre94], however, is misleading, since the author indicated that the benchmark is based on some work to identify the characteristics of engineering applications. Furthermore, there is no evidence of any model verification, to determine how representative the benchmark schema and operations are of actual engineering applications or sensitivity analysis, to determine what the impact of varying the sequence of benchmark operations to the number of concurrent users has.

B.3.9 Benchmarks Proposed by Kim & Garza

Kim & Garza [Kim94b] describe some requirements for what they deem to be a suitable benchmark for object-relational systems, but these are not based on any reported requirements analysis and are more likely to be based on their experiences with their own UniSQL product. Strictly speaking, the proposed benchmark is described as a unified benchmark for comparing object and relational database systems. However, many of the features are drawn from those that are directly supported by UniSQL. The proposed benchmark operations include:

- Fetching a single object from the database using its OID.
- Traversing a nested object in the database using embedded OIDs.
- Navigation of memory-resident objects.
- Traversing an inheritance hierarchy.
- Reverse navigation of objects in the database and in memory.
- Dynamic loading of methods.

- Retrieval and update of large objects (BLOBs).
- Various queries (path, class-hierarchy, set-valued attributes, methods and regular expressions) and updates.
- Dynamic schema changes.
- Transaction commit and abort.

They also propose a set of guideline parameters for such a benchmark, including using a 1 GB database and 20 concurrent users. However, as mentioned earlier, the set of parameters is rather arbitrary and no justification is provided.

Finally, they conclude with some observations about meaningful benchmarks. Firstly, it is suggested that the set of benchmark operations should be comprehensive and against a large database. However, this need not necessarily be the case, as the case studies show in Chapter 6. Secondly, that benchmarks should be multi-user. However, the Research Design in Chapter 4 argues against this and proposes that single-user benchmarks can provide useful insights into database performance. Finally, they suggest that comparisons of feature-rich and feature-poor systems should include some mechanism to penalise the feature-poor system by using "feature not supported" for a missing feature or recording "infinity" as the time to complete an operation. Among the benchmarks described in Chapter 3 and this appendix, only OO7 is known for certain to use hand-coded queries on some object database systems, whilst using the query programmer of others, for the same tests.

B.3.10 The OO7 Benchmark

This benchmark is discussed in section 3.2.5.

B.3.11 The Simple Benchmark

The Simple Benchmark [Kelte89; Dewal90; Dewal92] was so named because it focused on measuring simple operations (create, delete, read, write) on a simple data model, rather than complex operations on a complex data model, such as found in the HyperModel Benchmark. The benchmark developers suggested that, ideally, a benchmark should simulate complex operations on large, complex data structures, but this was difficult to do, because: (i) it was expensive to implement and (ii) the choice of data structures may not be obvious. They felt that it would be easier to model simple operations common to all systems as a first step in an evaluation process.

The benchmark schema is illustrated in Figure B.10 and is derived from [Emmer93]. This shows three types - **DIR**, **SMALL**, **BIG** and two relationships - **DIRREL** (cardinality 1:N), **MNREL** (cardinality M:N). The string attributes have length 10, 80 and 160 bytes and the longfield can be either 10 or 128 KB. The initial database required by the benchmark is loaded with 3,000 SMALL and 400 BIG objects.

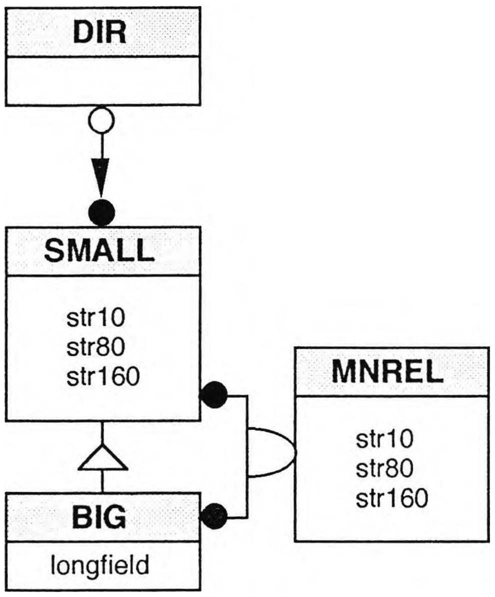

Figure B.10 - The Simple Benchmark Database Schema.

The benchmark operations are grouped as follows [Dewal92]:

1. Open and Close database.
2. Create and Delete SMALL and BIG objects without initialising the attributes.
3. Write and Read an attribute of a SMALL object.
4. Write and Read all attributes of a SMALL object.
5. Write and Read attributes of a BIG object.
6. Create and Delete MNREL without initialising attributes.
7. Write and Read an attribute of an MNREL.

A full description of the benchmark and these operations can be found in [Kelte89]. Results for a number of research prototypes (DAMOKLES, GRAS, OBJECT-BASE, PCTE, PROMOD) showed that these systems [Dewal90]:

> "... differ substantially in their architectural and functional properties and that the time required to perform similar basic functions can differ by orders of magnitude."

This seems to confirm the assumptions made by the benchmark developers that testing elementary operations could provide insights into system performance without the need to implement complex benchmarks (at least not initially). Obviously, the next step would be to develop more comprehensive benchmarks (to test additional capabilities, such as complex object modelling, for example), but this task will now require less time (e.g. [Dewal90] stated that two to four months were required per system to implement a complex benchmark on a number of products, whilst [Larse92] suggested it would take between four to six months - both were referring to the HyperModel Benchmark), since the number of candidate systems for a particular application should have been reduced using the simple tests first.

B.3.12 The University of Southern California (USC) Benchmark

Ghandeharizadeh et al. [Ghand93] described a synthetic benchmark consisting of queries that referenced inherited functions and traversed complex object sub-components. Queries were described using terminology found in the Functional Data Model [Shipm81]. The tests were designed for systems supporting object-based constructs, such as unique object identifiers, complex objects, a type hierarchy and inheritance. The benchmark developers argued against using empirical databases because:

1. They were difficult to scale.
2. The type hierarchies and complex objects (with their associated attributes) were not flexible enough to permit systematic benchmarking.
3. There were too many diverse applications to claim that just one was representative.

According to the developers, their benchmark consisted of 260 queries at one stage. However, after some initial experiences with using their benchmark on a research prototype system, they reduced this number by focusing on queries that appeared more interesting than others, such as selection queries and traversals. No benchmark results were reported, but the developers speculated on the behaviour of IRIS, ObServer/ENCORE, O_2 and GemStone using their benchmark, based on their understanding of the architectures of each of these products.

APPENDIX C - Application Requirements

C.1 Introduction

This appendix presents in detail the analysis of application requirements that were summarised in Table 5.1. The reader is reminded that the categories were:

- Complex Information Modelling Capabilities.
- Semantic Schema Design.
- Dynamic Schema Evolution.
- Rigorous Constraint Management.
- Management of Large Volumes of Data.
- Meta-Data.
- Data Sharing.
- Data Versioning.
- Inter-Client Communication.
- Flexible Transaction Framework.
- Efficient Storage Mechanisms for Fast Data Access and Retrieval.
- Computationally Complete Database Programming Language.
- Compatibility, Extensibility and Integration.
- Graphical Development Environment.

These are used for the application domains: Computer Integrated Manufacturing (CIM), Engineering, Financial, Geographical Information Systems (GISs), Healthcare, Scientific Data and Telecommunications.

C.2 Complex Information Modelling Capabilities

C.2.1 CIM

Manufacturing systems are complex and varied in nature [Adiga93a]. Discrete-event simulation is useful for CIM and, as reported in [Adiga93a], has been used by others to support all stages of the development of CIM systems, from specification to implementation. Narayanan et al. [Naray92b] add that simulation is the only viable approach to modelling the complex interactions between components in manufacturing systems, such as semiconductor fabrication.

C.2.2 Engineering

Engineering data are complex, since they have to model complex physical systems [Ahmed92]. Design elements are also interconnected by various links of the type described in section A.2.1. Simulation is also an important requirement, since it allows engineers to experiment with alternative designs [Ahmed92].

C.2.3 Financial

According to [Objec95a], trading systems:

> "... demand real-time analysis of portfolios of securities, currency, mortgages, evaluation of complex investment units such as derivatives, and simulation of various future scenarios based on sets of assumptions ..."

ODI [ODI96a] comment that financial applications use simple data, but that the relationships among financial instruments are complex. Furthermore, ODI agree that performing simulations quickly is a major characteristic of this domain. Chandra & Segev [Chand93a] also support the view that financial products are complex and a database that efficiently supports financial applications would need to provide a range of features, such as ADTs, user-defined functions, inheritance, etc.

C.2.4 GIS

Geographical data management requires complex data structures to support geometrical and topological data [David93]. Some previous approaches to the use of objects in GIS have been briefly mentioned in [David93]. From this discussion, concepts such as inheritance and aggregation are described as being useful for geographical data.

C.2.5 Healthcare

In his MSc work, Cheung [Cheun92] describes the many aspects to patient data that need to be modelled. This includes the capture of raw data which could be in the form of coded terms, text, numbers, drawings, graphs, tables, images, bio-signals (e.g. ECGs), sound and video. Furthermore, he concludes that only semantic models offer the capability to directly support these types. Other modelling constructs, such as inheritance and aggregation, were also found to be useful.

C.2.6 Scientific Data

DNA and proteins exhibit complex structures. For example, DNA is translated into nucleotides, cleavage sites, ligand-binding sites, etc. and these have further substructure [Ohkaw93]. Proteins are composed of liner chains of amino acids, which in turn are composed of helices, sheets and turns. These are further grouped into tertiary and quaternary structures [Lathr87]. Simulations are also important in scientific research.

C.2.7 Telecommunications

Telecommunications applications are very complex with sophisticated information models that incorporate many relationships among the defined object classes [Objec95b]. Telecoms resources (managed objects) are modelled using the Guidelines for the Definition of Managed Objects (GDMO) notation, which is inherently object-oriented [Perry96].

C.3 Semantic Schema Design

C.3.1 CIM

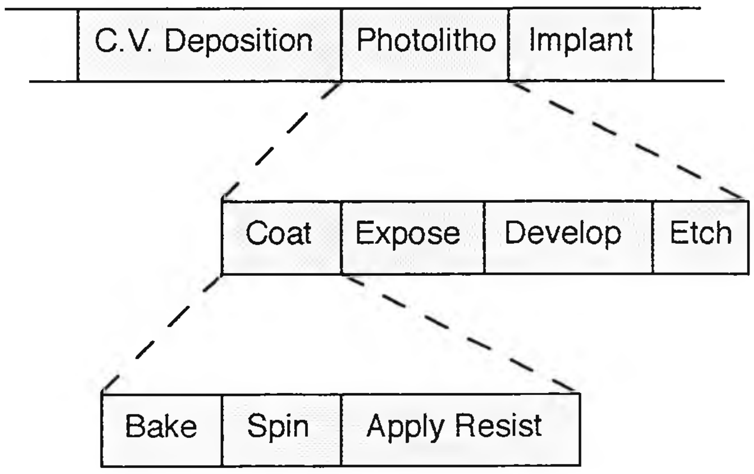

Figure C.1 - Aggregations in Representation of Steps in Wafer Processing [Adiga93a].

Hierarchical data structures are required to represent factory objects, which could be physical entities (e.g. labourers, equipment, inventory storage locations) or logical entities (e.g. formulae, process instructions, quality assurance test plans) [Lozie93]. Aggregation is a critical feature of good software design in manufacturing [Adiga93a]. For example, the top level in Figure C.1 would be seen by production planning, the second level by a short interval scheduling application and the bottom level by a real-time material handling system.

Bodner et al. [Bodne94] describe a framework for the development of a modelling architecture for CIM in which they also describe the benefits of aggregation, such as being able to view modelling abstractions (e.g. shop, cell, device) at various levels of granularity.

C.3.2 Engineering

Complex interconnections and dependencies between objects need direct schema representation to aid maintenance and design extension [Ahmed92]. This is can be provided by mechanisms such as inheritance and composition hierarchies, for example.

C.3.3 Financial

Inheritance of attributes and functions is a desirable feature for financial databases [Chand93a].

C.3.4 GIS

OO data structures are useful for modelling arbitrarily deep hierarchical composition applications such as CAD and GIS [Arctu95]. As mentioned above, inheritance is a mechanism that allows hierarchical composition.

C.3.5 Healthcare

Rich constructs, based on semantic and object-oriented models, have been used in the development of patient healthcare models, e.g. [Cairn92b; Cheun92]. Semantic models provide more flexibility, economy of expression and higher-level modelling [Cheun92].

C.3.6 Scientific Data

Scientific models exhibit hierarchical decomposition and nested structures that require direct schema representation. Ohkawa [Ohkaw93] also comments that large flat-vectors of numbers and deeply nested tree structures are common in scientific applications.

C.3.7 Telecommunications

As mentioned earlier, managed objects are modelled using GDMO. Furthermore, hierarchical and nested structures are commonly used in telecoms networks.

C.4 Dynamic Schema Evolution

C.4.1 CIM

CIM projects often require rapid deployment of a small, prototype implementation and rapid reconfiguration of software [Lozie93], which suggests the need for dynamic schema evolution.

C.4.2 Engineering

This is required since design is an incremental and evolutionary process - dynamic facilities are required to modify inheritance structures, class definitions, object attributes and methods [Ahmed92].

C.4.3 Financial

Financial markets are very dynamic and new products are constantly being introduced to cater for different investment needs [Chand93a]. The result of this is that it is difficult to know in advance what types of financial products may make up a particular portfolio (a combination of financial products, such as stocks, bonds, etc.) [Chand93a]. Dynamic Schema Evolution would, therefore, be very useful in this domain.

C.4.4 GIS

David et al. [David93] comment that at the conceptual level, there should be stability to keep investment in data capture, but internal data structures may change. The latter point is important, since a variety of standard data formats are available and conversion between one form and another cannot be performed automatically [David93].

C.4.5 Healthcare

One requisite of a computerised medical record is that it must be extensible and be able to cope with the evolution of the information requirements of its users and, furthermore, new types and new linkages must be definable, without affecting existing data [Cheun92].

Although this would imply that schema evolution is useful, dynamic schema changes may not be essential.

C.4.6 Scientific Data

Type evolution and type extensibility are required to merge new discoveries with existing knowledge. It should also be possible to add an arbitrary type as a new model is formed in scientific research [Ohkaw93]. For example, in High Energy Physics (HEP), dynamic schema evolution is important, since the data models will evolve with time [CERN96].

C.4.7 Telecommunications

In applications such as Advanced Intelligent Networks (AINs), flexibility is required to configure new services and delete old services quickly in response to customer demands. However, according to [Objec96], modifiable structures are a low priority for Network Management systems.

C.5 Rigorous Constraint Management

C.5.1 CIM

Manufacturing equipment (e.g. lathes, wafer testing, etc.) needs to operate at high levels of accuracy and within certain prescribed limits. Although control may be enforced by machine-level Programmable Logic Controllers (PLCs) [Adiga93c], such mechanisms could also be directly supported by objects.

C.5.2 Engineering

According to [Ahmed92], consistency constraints apply to data types, valid ranges, design conditions, safety limits, etc. Such mechanisms, they go on to say, can be directly supported by objects.

C.5.3 Financial

Numerical stability and accuracy are very important in financial applications [Chand93a]. As well as the types of constraints that were described above for engineering applications, other types of constraints may include, for example, the requirement that not more than 10% of a portfolio's market value may be invested in any one security or that the market value of a portfolio must be US$500 Million [Chand93a].

C.5.4 GIS

Constraints on spatial data types and spatial geometry may be implemented by ADTs. Furthermore, data types could be assigned particular interpretations, processing and referencing methods for different user groups. Such constraints could be implemented as object methods [Zingl95].

C.5.5 Healthcare

Providing legal values for patient data entry and validation of data could be directly supported by methods defined in the database. In fact, one such mechanism, termed view classes [Fowle94] has already been described which contains mapping rules for retrieving and updating objects in a generic model of health.

C.5.6 Scientific Data

Constraints are needed to ensure that data types, data ranges, etc. are valid, since accurate experimental results are critical in scientific applications.

C.5.7 Telecommunications

Telecoms equipment, for example, also need to operate at high degrees of accuracy and the conditions applicable to other domains also apply here, e.g. valid data ranges.

C.6 Management of Large Volumes of Data

C.6.1 CIM

Manufacturing systems need to keep historical data for analysis by Statistical Process Control (SPC) applications to discover undesirable trends in equipment performance, machine utilisation, yield analysis, etc. This historical data could be significant in large production environments.

C.6.2 Engineering

Engineering data applications are highly data-intensive and may involve very large quantities of data, e.g. consider the thousands of individual parts in a modern aircraft. Efficient management of this data will be essential to good performance.

C.6.3 Financial

Financial applications also need to manage very large quantities of data, since information about trades has to be kept for legal reasons. The diversity in the types of financial instruments and the volumes of trades possible would also support the importance of this requirement.

C.6.4 GIS

An example GIS described by Zingler [Zingl95] discusses very large data volumes, which he concludes will only increase as satellite sensors used to collect data will continue to improve. There is no doubt that the increasing number of satellites and the large volumes of data collected will impose serious data management problems. Managing large spatial indexes, such as Quad-Trees and R-Trees, must also be considered.

C.6.5 Healthcare

As mentioned earlier, raw patient data can consist of a variety of data types, some of which demands large storage overheads (e.g. images, sound). Furthermore, entries in a medical record, once made, should not be altered or removed [Cheun92], for historical, legal or other reasons. This means that large quantities of computerised data will accumulate and grow, particularly for large populations.

C.6.6 Scientific Data

Advances in laboratory technology have resulted in large quantities of data being produced and good storage management techniques are required to manage this data. For example, the human Genome comprises some 3 Billion nucleotide bases and, similarly, the Magellan planetary probe will generate a Trillion bytes of data over its five year life [Frenc90a].

C.6.7 Telecommunications

The models used in telecoms applications can be huge. Perryman [Perry96], for example, describes an application with 500,000 telephone lines, with each line comprising five to six objects and many associated objects representing equipment, customer information, etc.

C.7 Meta-Data

C.7.1 CIM

In manufacturing systems, meta information such as device characteristics need to be maintained, since discrete-event simulation is a central activity to production management and would need access to such information.

C.7.2 Engineering

The DBMS needs to keep track of information such as design element ownership, time and purpose of creation, update histories, client dependencies, lock status, versioning, etc. [Ahmed92].

C.7.3 Financial

For this domain, important meta-data could include portfolio ownership, calendar information, change histories for instruments, etc.

C.7.4 GIS

Zingler [Zingl95] describes meta-data as processed Earth Observation (EO) data. He goes on to say that this includes:

> "... all information for the user to understand the properties of a particular satellite product and allows them to judge if data sets are suitable for their application."

C.7.5 Healthcare

This is an important requirement as with other domains and perhaps scientific data in particular, such as units for values (e.g. blood pressure), who did what and when (e.g. which doctor prescribed which medication), etc.

C.7.6 Scientific Data

Meta-data is fundamental to the effective use of scientific data [Frenc90a]. The DBMS needs to manage meta-information that supplements actual scientific data such as units for values and relevant publications [Ohkaw93], as well as other properties, such as who did what and when, device characteristics, transform definitions, etc. [Frenc90a].

C.7.7 Telecommunications

Information about the Open Systems Interconnection (OSI) X.722 definitions of managed objects include attributes, management operations and notifications [ODI96b]. An ODBMS can directly support these requirements [ODI96b].

C.8 Data Sharing

C.8.1 CIM

Traditionally, CIM applications (e.g. process control and production scheduling) have developed independently [Adiga93a]. A single database for all applications has generally not been used, since this would difficult to model and would seriously affect response time. Data sharing between applications would, therefore, appear to be a less important issue for this domain.

C.8.2 Engineering

Mechanisms to support collaborative and group-working are required.

C.8.3 Financial

Data sharing in the form of that required for engineering would, until recently, have been unlikely in this domain, since portfolio's would have been managed on a per trader basis and the contents (i.e. which instruments were being held at any point in time) would be confidential. However, a recent development described in [Objec96], is that of *team trading* whereby a group of traders has joint responsibility for a portfolio. In such cases, advanced data sharing mechanisms would be needed.

C.8.4 GIS

Insufficient information available to draw a conclusion.

C.8.5 Healthcare

Sharing data is viewed as a requirement for a computerised medical record in [Cheun92], but it is also recognised that this could be difficult to achieve due to differing data formats used by institutions. Sharing in the sense of collaborative and group-working would probably not be applicable, as standard mechanisms would be sufficient.

C.8.6 Scientific Data

This varies between applications, since some data are never updated and read-only access may be sufficient (e.g. epidemiological data) whilst in other applications (e.g. drug design), concurrency control is essential [Ohkaw93].

C.8.7 Telecommunications

Telecoms models can be huge and each management scenario may involve accessing a working set of a few objects for as brief a period as possible [Perry96]. This would imply that locking would be comparable to traditional mechanisms used in environments such as OLTP, for example.

C.9 Data Versioning

C.9.1 CIM

Versioning could be used as a mechanism to track the evolution of prototype CIM systems, as well as enabling customisable systems to be built for each organisation.

C.9.2 Engineering

Versions provide a mechanism to keep track of design changes and the ability to return to a previous design state. Such mechanisms are not well served by existing DBMS technology.

C.9.3 Financial

As mentioned earlier, new financial instruments are constantly being introduced. These could be derived by inheritance from existing instruments, but versioning could also be used to derive new instruments from previous ones.

C.9.4 GIS

Insufficient information available to draw a conclusion.

C.9.5 Healthcare

Versioning could be used to keep track of patient histories, previous medications, etc.

C.9.6 Scientific Data

Insufficient information available to draw a conclusion.

C.9.7 Telecommunications

Since modifiable structures are not a characteristic of this domain [Objec96], versioning would have less utility here.

C.10 Inter-Client Communication

C.10.1 CIM

Manufacturing systems are often composed of smaller sub-systems, which have their own controllers and computers. Communication between sub-systems, however, is important. For example, Process Control will feed information regarding the state(s) of equipment and materials to Tracking as described in an example in [Adiga93c].

C.10.2 Engineering

Collaborative and group-working requires better mechanisms for designers to communicate with each other. This includes communication about locks, change notification, conflict resolution, etc. [Ahmed92].

C.10.3 Financial

Team trading was mentioned earlier and would require advanced communication techniques between financial traders.

C.10.4 GIS

Insufficient information available to draw a conclusion.

C.10.5 Healthcare

Insufficient information available to draw a conclusion.

C.10.6 Scientific Data

French at al. [Frenc90a] comment that much scientific data are characterised by large volume, low update frequency and indefinite retention. This suggests that inter-client communication is less important for this domain.

C.10.7 Telecommunications

For Network Management, the transaction profile described in [Objec96] is typically many readers and few writers, with no indication that inter-client communication is required.

C.11 Flexible Transaction Framework

C.11.1 CIM

A general scheme illustrating the factory organisational hierarchy and timing requirements in [Adiga93b] shows that time horizons vary from milli-seconds at the machine/device level to minutes and days at the workstation and work cell level, respectively. These require more flexible transaction mechanisms.

C.11.2 Engineering

Design transactions may span a long duration (when compared to traditional applications), requiring support for long locks and check-out/check-in mechanisms. Nested transactions are also required, according to [Ahmed92].

C.11.3 Financial

Chandra & Segev [Chand93a] propose a concurrency control paradigm where the unit of locking is a user-defined class, such as a portfolio. Other mechanisms to support querying of historical and time-series data would also be required.

C.11.4 GIS

The major concerns for some GIS applications are to manage streams of continuous data, construct complex index structures and save everything to the database. This requires the need for more flexible transactions (e.g. ingesting data and concurrently supporting user queries), which is implied by the example in [Zingl95].

C.11.5 Healthcare

Experimental analysis of data would require more flexible transaction mechanisms, as with scientific data.

C.11.6 Scientific Data

Simulations or experimental analysis of data would require more flexible transaction mechanisms. For example, in HEP environments, transactions may be short where performance is a key factor or very long where an experimental run is being processed [CERN96]. Furthermore, nested transactions could also be useful for reconstructions of experimental runs.

C.11.7 Telecommunications

Short transaction mechanisms are adequate for many telecoms applications, as discussed earlier.

C.12 Efficient Storage Mechanisms for Fast Data Access and Retrieval

C.12.1 CIM

In manufacturing, not all applications have the same degree of urgency. The requirements for historical data were mentioned earlier. For some applications, such as the scheduling system, an operator expects an immediate answer [Adiga93b]. In such cases, keeping the current state of the physical system in RAM may be a solution [Adiga93b]. ODBMSs were designed to better utilise workstation memory.

C.12.2 Engineering

Clustering parts related to one-another through some relationship enable data to be retrieved more quickly from secondary storage. This could be through a part/sub-part relationship, for example, which would allow sub-parts of a particular assembly to be retrieved together.

C.12.3 Financial

Applications may have differing storage and access requirements. For example, for computation of moving window aggregates, data need to be stored contiguously and sorted by real-time [Chand93a]. In portfolio management, on the other hand, prices of instruments are subject to very frequent change, requiring other storage and access techniques.

C.12.4 GIS

In some GIS research described in [David93], good performance is considered as a main objective, since geographical data may consist of vast quantities of point and line data that needs to be managed efficiently. Another example described in [Zingl95], called the Multi-Mission Inventory System (MMIS), also required good performance, consisting of Near-Real-Time (NRT) updates and transaction volumes reaching up to 10,000 queries a day.

C.12.5 Healthcare

Performance is an important requirement for medical data, but applications may have differing needs. A system built around the Cosmos Clinical Process Model [Cairn92b], for example, was designed to allow the integration of multiple systems [Thurs93], which means that deciding on accelerators, such as indexing and clustering, pose significant challenges.

C.12.6 Scientific Data

Scientific data also require good performance. According to [Frenc90a], a major activity in scientific data is discrete sampling of functions across several dimensions and often the results are sequences in which order is important. Some examples include [Frenc90a]:

- Earth science data are analysed statistically; time sequenced, multidimensional tables are common.
- Biological Genome databases are characterised by elaborate pattern matching over liner, character data.
- Space sciences apply transformations to very large two- and three-dimensional arrays.

This implies that clustering is more difficult to determine due to the multiple dimensions but, according to [CERN96], is regarded as key technique to achieve performance in HEP environments.

C.12.7 Telecommunications

Performance is a critical factor for telecoms applications and techniques such as physically clustering groups of objects together, to be accessed together, are well supported by ODBMSs.

C.13 Computationally Complete Database Programming Language

C.13.1 CIM

SPC and time series analysis to determine undesirable trends in equipment performance, for example, are not well supported by existing database languages.

C.13.2 Engineering

Engineering applications involve complex mathematical computations which languages such as SQL do not provide [Ahmed92].

C.13.3 Financial

As mentioned earlier, time-series (sequence) data are common. Most languages, however, do not provide support to query sequences efficiently [Chand93a]. There have been a number of research attempts to extend SQL with temporal capabilities, but it is difficult to express temporal conditions using these [Chand93a].

C.13.4 GIS

Object querying and navigational access are supported by most ODBMSs, which remove the limitations of complexity and the large processing overhead incurred by SQL for expressing spatial queries [Zingl95].

C.13.5 Healthcare

A healthcare system described in [Thurs93], uses a client front-end built with Smalltalk. The query builder is implemented directly in this environment and does not support SQL. Queries might also be difficult to express using SQL anyway, since the entire system is built using pure OO techniques.

C.13.6 Scientific Data

Generic analysis techniques such as statistical analysis, time series analysis and liner algebra are common to a range of scientific disciplines [Frenc90a]. In contrast, SQL queries are limited to simple data access, arithmetic and aggregate operations [Ohkaw93].

C.13.7 Telecommunications

Managed object queries are expressed in the form of predicate expressions - some of which cleanly maps to SQL notation, but a lot of it does not [Perry96].

C.14 Compatibility, Extensibility and Integration

C.14.1 CIM

Manufacturing sub-systems and modules can be better integrated by common representations. Objects can provide such a common representation, as described in [Naray92a; Naray92b].

C.14.2 Engineering

Design tools may need to interface and communicate with other design tools. The use of common representations and languages provide a possible solution to this. The DBMS must also be flexible to accommodate changes and new requirements.

C.14.3 Financial

Using common representations would allow new financial analysis applications and packages to be integrated more easily.

C.14.4 GIS

The example system described in [Zingl95] proposes a modular and extensible system with new functionality and capabilities added when available. Object-orientation provides the common framework to enable this to be realised.

C.14.5 Healthcare

Medical practice is very diverse [Cheun92]. Systems must be able to cope with new demands that are the consequence of changing information needs. Furthermore, integration of departmental systems through networks is possible by a generic model of healthcare based on OO, as proposed by Thursz et al. [Thurs93].

C.14.6 Scientific Data

Rather than embed domain-specific operators in a DBMS, it is more appropriate to create an integrated analysis environment with the DBMS and able to interact with a variety of tools [Frenc90a].

C.14.7 Telecommunications

Network Management really consists of three types of complex data structures [Objec96]:

1. **Managed Objects** - hardware, software and virtual components of the network.
2. **Network Connections** - chains of nodes and links used to construct connections.
3. **Performance and Billing**.

These can use the standard notations (OO in nature) developed by the International Telecommunications Union (ITU) to support compatibility, integration and extensibility.

C.15 Graphical Development Environment

C.15.1 CIM

Good graphical tools are required to manage the design of manufacturing processes and sub-systems as well as providing a standard way to interact, potentially easing the burden on users [Adiga93b].

C.15.2 Engineering

Engineering applications require good tools for database browsing, inspection and modifying data structures and dependencies.

C.15.3 Financial

Graphical tools would enable traders to actively monitor portfolio performance and add or delete new components based on price movements.

C.15.4 GIS

User tools to manage a variety of information systems components are required for many scientific disciplines that may need access to EO data.

C.15.5 Healthcare

The design of a good user interface is important in representing patient information in multiple forms, such as tables, graphs, charts, etc. Furthermore, the use of form fill-in tools allows some data to be verified at the time of entry, providing improved data quality.

C.15.6 Scientific Data

Good data browsing tools for scientific data are required for locating data sets and then scanning them for indications of probable interest [Frenc90a].

C.15.7 Telecommunications

Graphical browsers and tools would enable users to traverse networks, identify bottlenecks, locate points of failure, etc.

APPENDIX D - Case Studies

D.1 Introduction

The discussion of the six case studies presented in Chapter 6 focused on performance issues. This appendix presents additional information about the organisations and the systems that they are developing.

Pages 288-300 cannot be published in this thesis at this time, as information from Nomura International, Reuters and Earth Observation Sciences was provided to this author on a commercial-in-confidence basis, requiring the signing of non-disclosure agreements. However, the restrictions to publication may be lifted by these organisations at some future time and the interested reader should contact this author through City University.

D.6 The Uncle Project at St. Mary's Hospital

The Cosmos Clinical Process Model (CCPM) was developed as part of a larger initiative within the National Health Service (NHS) of the UK to develop a conceptual model of all healthcare, called the Common Basic Specification (CBS), which aims to cover all aspects of running a Health Service [Cairn91]. Originally, three projects were the focus of the work [Fowle93]:

- **PACE** - Paediatrics at the Hospital for Sick Children, Great Ormond Street.
- **Cosmos** - Renal medicine at St. Mary's Hospital, Paddington.
- **Diabptech** - Diabetes at St. Thomas's Hospital.

These three projects were brought together in 1990 to form the Cosmos Project, as commonalties were found between them and the basic clinical process remains constant

[Fowle91]. Besides the above healthcare specialities, the model has also been shown to be applicable to general practice and epidemiology [Fowle95].

The focus of Cosmos is with the development of clinical systems for use by clinicians at the point of care to: (i) improve the quality of patient care delivered and (ii) to form a conceptual foundation to enable the integration of independent clinical systems [Fowle93]. The latter being an important requirement, since clinical systems have tended to be developed independently in the past leading to costly translation and interface mechanisms [Fowle93]. The relationship between Cosmos and CBS is that Cosmos forms the clinical view on the CBS core model through a well-defined mapping [Thurs93].

After early implementation and performance set-backs were encountered with a semantic database called Generis, GemStone and Smalltalk were selected. Smalltalk was chosen since it was found to be a better language for non-professional programmers to use and due to its prototyping nature which better suited the project requirements. The approach to developing a working prototype followed that illustrated in Figure D.11. This enables portions of the conceptual model to be selected and tested according to predetermined criteria [Gold92].

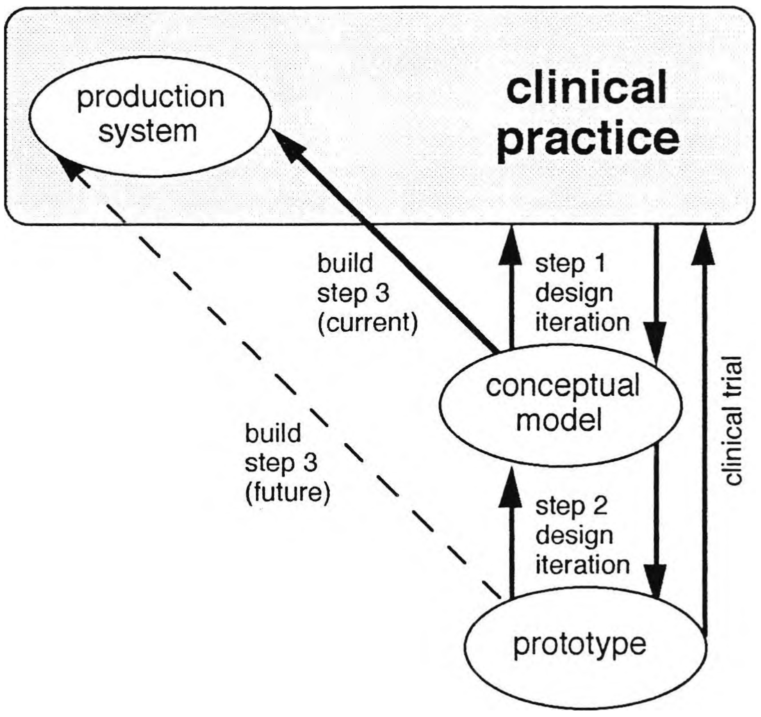

Figure D.11 - Systems Evolution [Gold92].

Templates were developed to enable translation of the model to Smalltalk. The advantage of this is that it is possible to predict the structure of the code from the analysis specification [Fowle93].

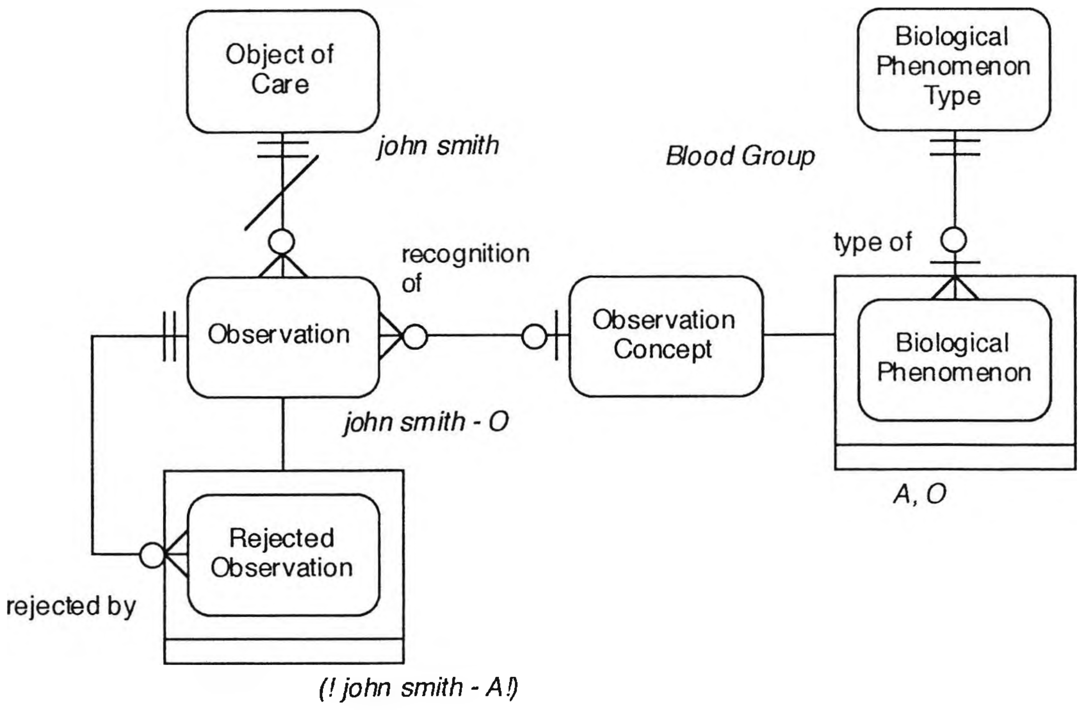

Figure D.12 - Recording Blood Group Information [Fowle93].

To fully understand the design of the CCPM (and its implementation into software), an example of OOIE notation may now be useful. Figure D.12 shows some of the major diagramming conventions used to represent the recording of blood group information. The structural view of a system in Ptech is called a Concept Diagram. The level of abstraction that this provides enables the representation of all disparate clinical specialities [Thurs93].

Fowler et al. [Fowle93] note the similarities between the OOIE notation and that of Entity-Relationship (ER) modelling, such as the crows feet, but also the differences, such as the use of subclasses and greater emphasis on identifying business objects rather than grouping data elements together.

Subclasses are represented by partitions. A partition would include all possible (disjoint) subclasses. In the example above, there are two incomplete partitions that show only the subclasses of interest, namely **Biological Phenomenon** and **Rejected Observation**, respectively. Biological Phenomenon is a subclass of Observation Concept and Rejected Observation is a subclass of Observation. Relationships between two classes are described by a function and its inverse. For example, a rejected observation would have the person that rejected that as its inverse. These diagrams are a formal notation with each element having an equivalent expression in predicate calculus [Fowle91]. Fowler [Fowle91] provides a complete description of the notation.

When implementing the model in Smalltalk, two particular areas that had to be addressed were: (i) how to implement associations and (ii) dynamic and multiple classification. The solutions taken are summarised in Tables D.2 and D.3, respectively.

OOIE	Smalltalk	Solution
Supports bi-directional associations between object types, typical of many ER modelling techniques.	Supports only uni-directional pointers, which make it difficult to represent bi-directional associations.	Model each association with two sets of pointers.

Table D.2 - Handling Associations [Fowle93].

OOIE	Smalltalk	Solution
Allows objects to belong to multiple classes (multiple classification) and classifications can change (dynamic classification).	Cannot directly implement multiple and dynamic classification using inheritance.	Combine CCPM object types into a single class and distinguish between types using flags.

Table D.3 - Handling Dynamic and Multiple Classification [Fowle93].

Figure D.13 illustrates how bi-directional associations can be modelled with the parents side being the master and the children's side being the slave. The master contains all the code for updating the association and any call on the slave simply calls the master operation with the relevant arguments. This enables the pointers to be kept up-to-date and maintains referential integrity [Fowle93].

These transformations show that moving from Object-Oriented Analysis and Design (OOAD) to implementation is not as straightforward as it may first appear, even without considering performance tuning [Fowle93].

The CCPM was implemented in a three schema architecture (external, conceptual, storage) as shown in Figure D.14. Using this approach, more flexibility results, since the generic CCPM could be implemented in a number of different ways. For example, one implementation may wish to use one set of units for recording clinical observations about a patient, whilst another may wish to use a different set. The three schema architecture provides the flexibility to allow both (and many other) approaches. The external schema is described as an application view [Fowle93], which allows one user to see only the parts of the model that are of interest to them.

```
childrenAdd:  aValue

     "Reverse call to parent"
     aValue parentsAdd: self

childrenRemove:  aValue

     "Reverse call to parent"
     aValue parentsRemove: self

parentsAdd:  aValue

     "Add a parent action"

     "If there isn't a set there already, make one"
     parents isNil ifTrue: [ parents := Set new ]

     aValue friend_children add: self
     parents add: aValue

parentsRemove:  aValue

     self remove: self fromSet: (aValue friend_children)
     self remove: aValue fromSet: parents

friend_children

     "Friend access to the children - should only be used
     by parents modifiers"

     "If children set not yet there, make it"
     children = nil ifTrue: [ children := Set new ]

     ^children
```

Figure D.13 - Modelling Bi-Directional Associations [Cairn92a].

Separating the storage schema means that any kind of database model could be used, such as relational, object-oriented, etc. However, since Uncle uses an ODBMS, the storage schema is the same as the conceptual schema, subject to the transformations described earlier [Fowle93].

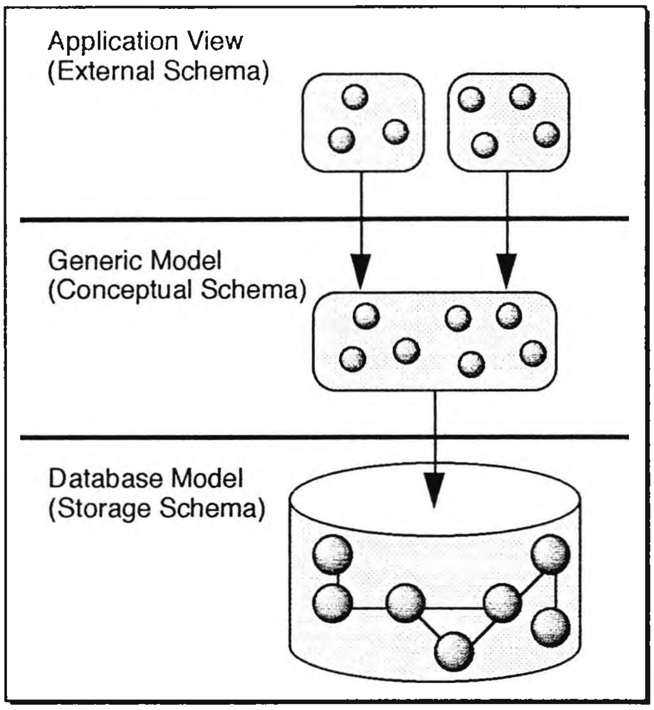

Figure D.14 - CCPM Three Schema Architecture [Cairn92a].

APPENDIX E - The OO1 Benchmark

E.1 Introduction

This appendix contains support material for Chapter 7. Raw results and a statistical analysis for the OO1 Benchmark are presented.

E.2 Small Local Database

E.2.1 DBMS-1 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	20.440	14.550	17.400	12.120	12.080	12.300	13.000	11.770	11.820	12.080
	Traversal	26.300	9.920	4.300	3.730	3.480	3.250	2.970	3.480	3.150	2.700
	Insert	23.880	24.380	28.520	23.550	24.840	33.030	27.310	25.400	22.350	22.000
	R. Traversal	29.480	7.670	7.780	2.470	4.500	2.650	4.360	3.140	3.680	1.460
2	Lookup	18.350	11.950	12.330	11.950	11.970	12.630	11.970	12.170	11.880	12.850
	Traversal	18.190	7.610	5.150	3.820	3.000	3.230	3.050	2.820	2.900	2.580
	Insert	26.130	25.360	27.320	23.830	26.260	35.400	26.350	26.350	22.840	23.380
	R. Traversal	28.480	5.530	4.870	2.000	3.470	2.750	3.600	2.260	3.740	1.400
3	Lookup	19.140	13.300	13.420	13.530	13.240	13.610	13.690	13.610	14.100	13.770
	Traversal	18.500	7.170	5.160	3.590	3.230	4.100	3.500	2.850	2.870	2.700
	Insert	24.710	26.180	31.160	23.470	25.650	37.240	36.620	27.050	23.609	23.299
	R. Traversal	30.350	5.820	5.160	2.890	4.010	2.350	4.480	2.450	3.600	1.620
4	Lookup	18.900	12.800	11.620	11.430	11.920	12.220	11.730	12.600	11.660	12.270
	Traversal	18.700	6.510	4.040	3.430	2.980	4.450	3.290	3.110	3.030	2.620
	Insert	23.800	23.820	27.480	22.430	24.450	34.770	25.630	25.980	22.530	22.650
	R. Traversal	28.450	6.980	5.140	2.000	3.560	2.440	3.980	2.620	3.710	1.340
5	Lookup	18.270	11.330	12.950	11.470	11.560	11.650	11.530	13.220	11.920	11.710
	Traversal	18.150	7.740	4.900	4.130	3.500	3.230	4.200	3.040	3.110	2.570
	Insert	28.000	23.000	26.880	22.900	23.630	32.230	27.010	25.610	21.550	22.130
	R. Traversal	27.170	5.580	5.770	2.330	3.350	2.980	3.850	2.140	3.180	1.270

E.2.2 DBMS-2 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	25.710	2.150	2.970	8.910	2.320	2.500	2.230	2.020	2.050	1.960
	Traversal	33.202	8.330	6.280	11.720	5.160	5.340	5.150	5.260	5.670	5.750
	Insert	23.270	6.780	11.560	3.550	4.360	3.970	2.720	4.060	3.650	3.670
	R. Traversal	36.340	24.080	10.680	9.510	5.540	4.620	0.650	1.610	2.600	1.960
2	Lookup	24.870	9.220	2.440	2.120	2.110	2.200	2.200	2.110	2.110	2.130
	Traversal	37.770	7.600	6.810	6.120	4.950	7.850	4.850	5.060	5.160	4.840
	Insert	24.370	4.590	12.340	3.880	3.800	5.140	2.910	3.540	3.260	6.030
	R. Traversal	32.909	22.610	6.490	9.720	5.030	5.610	0.730	1.780	2.800	2.270
3	Lookup	20.350	7.870	1.910	1.910	1.970	1.950	2.260	3.100	2.040	1.960
	Traversal	38.921	10.470	6.710	5.760	5.230	8.760	4.880	5.200	4.970	4.840
	Insert	18.500	6.660	3.670	8.440	4.590	4.090	4.070	3.420	3.260	2.850
	R. Traversal	35.741	24.380	7.060	10.030	6.100	4.650	0.700	1.730	3.490	2.930
4	Lookup	24.710	2.300	2.110	2.030	2.190	8.290	2.010	2.030	2.070	2.210
	Traversal	33.580	8.790	14.450	6.710	5.080	5.300	5.080	5.270	6.070	5.050
	Insert	20.210	13.150	3.780	3.370	3.760	4.410	3.390	3.920	6.740	3.030
	R. Traversal	40.330	19.670	6.330	10.200	5.240	4.270	0.720	2.000	3.430	1.950
5	Lookup	26.751	1.910	1.920	2.090	2.020	2.030	2.440	3.380	3.050	3.760
	Traversal	33.600	8.730	6.800	6.480	9.700	5.520	5.290	5.680	5.330	5.960
	Insert	20.231	4.150	4.380	4.400	7.290	4.040	2.770	3.300	3.800	2.980
	R. Traversal	32.681	12.630	11.470	9.140	4.770	5.070	0.700	1.760	3.070	1.930

E.2.3 DBMS-3 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	59.240	64.001	83.240	101.481	117.771	134.580	147.260	159.581	172.710	183.731
	Traversal	27.440	13.640	10.750	8.910	6.700	10.240	9.680	8.490	8.010	7.190
	Insert	343.073	343.881	327.932	320.741	332.131	349.591	348.960	347.911	351.681	338.080
	R. Traversal	32.720	12.150	13.880	5.810	10.540	7.700	9.940	6.660	10.470	3.410
2	Lookup	55.990	64.330	81.720	100.570	115.501	132.300	145.100	159.021	170.880	182.121
	Traversal	25.860	13.390	9.310	9.030	7.000	9.900	9.390	9.080	8.150	6.010
	Insert	312.431	335.777	314.954	344.492	330.411	352.350	343.920	374.913	349.922	339.399
	R. Traversal	32.440	11.430	12.190	5.270	10.210	7.570	9.970	6.860	9.340	3.250
3	Lookup	58.710	64.341	82.670	101.050	116.831	131.301	146.050	158.271	170.950	180.461
	Traversal	24.860	14.010	9.390	9.370	6.760	9.500	9.420	8.830	8.110	6.000
	Insert	345.732	339.931	355.451	315.361	353.290	355.411	328.781	355.120	346.451	394.762
	R. Traversal	32.230	10.980	12.220	4.890	10.620	7.800	9.700	6.100	9.430	3.490
4	Lookup	56.990	64.611	83.800	101.560	117.290	132.610	145.700	159.770	171.411	183.401
	Traversal	26.040	14.620	9.350	8.900	7.310	9.570	9.490	8.680	8.420	6.490
	Insert	358.350	357.220	345.011	329.871	354.422	365.921	322.360	350.711	345.299	338.451
	R. Traversal	30.660	11.790	12.020	5.010	10.450	7.380	9.930	7.630	10.470	3.410
5	Lookup	57.690	65.370	84.280	102.071	116.760	134.811	148.771	159.860	172.341	184.339
	Traversal	25.510	12.100	9.260	9.070	6.860	9.840	9.810	8.420	9.790	6.140
	Insert	316.563	322.002	348.491	319.301	323.881	335.831	332.831	335.001	329.731	344.150
	R. Traversal	30.610	11.650	12.420	5.130	10.990	7.830	10.340	7.400	10.100	3.400

E.2.4 DBMS-1 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	20.440	13.013	26.300	4.109	23.880	25.709	70.620	42.831	29.480	4.190
2	18.350	12.189	18.190	3.796	26.130	26.343	62.670	42.328	28.480	3.291
3	19.140	13.586	18.500	3.908	24.710	28.253	62.350	45.747	30.350	3.598
4	18.900	12.028	18.700	3.718	23.800	25.527	61.400	41.273	28.450	3.530
5	18.270	11.927	18.150	4.047	28.000	24.994	64.420	40.968	27.170	3.383
Average:	19.020	12.549	19.968	3.916	25.304	26.165	64.292	42.629	28.786	3.598
Sample SD:	0.874	0.721	3.547	0.164	1.774	1.263	3.702	1.901	1.199	0.352

E.2.5 DBMS-2 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	25.710	3.012	33.202	6.518	23.270	4.924	82.182	14.454	36.340	6.806
2	24.870	2.960	37.770	5.916	24.370	5.054	87.010	13.930	32.909	6.338
3	20.350	2.774	38.921	6.313	18.500	4.561	77.771	13.648	35.741	6.786
4	24.710	2.804	33.580	6.867	20.210	5.061	78.500	14.732	40.330	5.979
5	26.751	2.511	33.600	6.610	20.231	4.123	80.582	13.244	32.681	5.616
Average:	24.478	2.812	35.415	6.445	21.316	4.745	81.209	14.002	35.600	6.305
Sample SD:	2.446	0.196	2.711	0.356	2.423	0.403	3.679	0.600	3.111	0.516

E.2.6 DBMS-3 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	59.240	129.373	27.440	9.290	343.073	340.101	429.753	478.764	32.720	8.951
2	55.990	127.949	25.860	9.029	312.431	342.904	394.281	479.882	32.440	8.454
3	58.710	127.992	24.860	9.043	345.732	349.395	429.302	486.430	32.230	8.359
4	56.990	128.906	26.040	9.203	358.350	345.474	441.380	483.583	30.660	8.677
5	57.690	129.845	25.510	9.032	316.563	332.358	399.763	471.235	30.610	8.807
Average:	57.724	128.813	25.942	9.119	335.230	342.046	418.896	479.979	31.732	8.650
Sample SD:	1.305	0.838	0.951	0.120	19.841	6.406	20.638	5.756	1.017	0.244

E.2.7 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	19.020	12.549	19.968	3.916	25.304	26.165	64.292	42.629	28.786	3.598
	Sample SD:	0.874	0.721	3.547	0.164	1.774	1.263	3.702	1.901	1.199	0.352
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	24.478	2.812	35.415	6.445	21.316	4.745	81.209	14.002	35.600	6.305
	Sample SD:	2.446	0.196	2.711	0.356	2.423	0.403	3.679	0.600	3.111	0.516
	t =	-4.699	29.141	-7.737	-14.412	2.969	36.141	-7.247	32.114	-4.571	-9.691

E.2.8 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	19.020	12.549	19.968	3.916	25.304	26.165	64.292	42.629	28.786	3.598
	Sample SD:	0.874	0.721	3.547	0.164	1.774	1.263	3.702	1.901	1.199	0.352
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	57.724	128.813	25.942	9.119	335.230	342.046	418.896	479.979	31.732	8.650
	Sample SD:	1.305	0.838	0.951	0.120	19.841	6.406	20.638	5.756	1.017	0.244
	t =	-55.085	-235.216	-3.638	-57.150	-34.790	-108.183	-37.817	-161.320	-4.192	-26.356

E.2.9 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	24.478	2.812	35.415	6.445	21.316	4.745	81.209	14.002	35.600	6.305
	Sample SD:	2.446	0.196	2.711	0.356	2.423	0.403	3.679	0.600	3.111	0.516
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	57.724	128.813	25.942	9.119	335.230	342.046	418.896	479.979	31.732	8.650
	Sample SD:	1.305	0.838	0.951	0.120	19.841	6.406	20.638	5.756	1.017	0.244
	t =	-26.812	-327.423	7.373	-15.904	-35.118	-117.510	-36.020	-180.030	2.643	-9.184

E.3 Large Local Database

E.3.1 DBMS-1 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	51.690	27.850	21.800	19.000	19.050	18.750	18.120	18.250	18.330	18.269
	Traversal	54.710	35.240	27.640	30.980	34.510	32.990	25.310	35.240	35.830	37.350
	Insert	38.920	40.260	44.330	39.181	41.950	51.591	40.110	41.950	36.400	36.450
	R. Traversal	64.980	20.700	53.461	83.340	46.310	74.980	25.140	38.680	35.300	58.001
2	Lookup	48.070	27.430	20.370	17.430	16.820	17.530	16.350	16.750	17.000	16.770
	Traversal	52.270	35.930	34.721	34.070	35.060	29.780	22.500	34.500	36.833	34.807
	Insert	37.880	37.180	41.230	35.010	36.960	45.360	35.400	37.110	33.260	32.540
	R. Traversal	60.801	21.940	59.810	91.981	47.450	82.320	30.231	41.180	45.920	63.301
3	Lookup	46.390	25.090	19.360	16.770	16.630	15.890	16.000	15.710	16.150	15.570
	Traversal	51.080	36.480	28.200	32.521	39.031	32.370	24.480	32.530	40.220	35.510
	Insert	37.649	37.080	40.640	35.197	39.840	46.000	35.633	37.832	33.252	32.631
	R. Traversal	64.401	18.800	56.580	86.420	47.580	82.750	30.370	43.830	37.110	61.891
4	Lookup	48.640	26.310	20.190	17.810	17.500	17.450	17.880	16.920	16.920	16.410
	Traversal	53.590	38.430	29.950	35.210	36.951	31.900	24.740	33.010	36.730	35.740
	Insert	40.231	41.401	48.471	41.350	42.351	48.030	36.891	38.670	34.340	35.370
	R. Traversal	63.561	19.270	56.210	87.520	49.830	81.530	27.400	40.950	33.550	52.550
5	Lookup	49.080	26.330	20.340	17.480	16.650	17.710	16.170	16.770	19.150	16.510
	Traversal	54.751	35.120	30.180	36.280	38.170	32.371	23.880	33.700	36.980	38.520
	Insert	37.290	39.680	44.299	37.361	39.787	48.501	36.310	38.720	33.540	33.220
	R. Traversal	63.350	18.350	55.550	89.360	48.900	87.261	27.770	47.380	34.200	56.620

E.3.2 DBMS-2 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	199.830	118.330	128.610	118.040	126.560	113.980	122.671	98.291	106.600	96.632
	Traversal	286.909	268.772	242.277	226.389	444.933	357.573	235.730	214.980	208.101	215.109
	Insert	54.231	63.472	52.291	58.960	53.921	56.081	54.890	49.100	63.730	93.380
	R. Traversal	548.548	453.672	122.760	285.422	448.431	614.671	380.730	125.870	460.849	131.121
2	Lookup	155.119	264.359	266.951	175.830	139.189	104.149	120.851	131.390	149.291	123.470
	Traversal	241.490	220.992	282.701	434.583	277.530	254.152	208.749	223.129	183.061	199.152
	Insert	59.730	73.510	69.408	71.161	54.870	67.390	51.300	77.079	72.620	66.141
	R. Traversal	559.081	483.432	126.050	288.189	604.736	485.708	372.861	128.140	475.942	135.351
3	Lookup	139.459	122.471	122.930	124.870	121.490	100.020	112.970	118.160	120.521	117.391
	Traversal	297.310	208.729	235.289	224.830	408.652	375.713	210.452	233.273	182.081	212.733
	Insert	55.150	47.740	42.950	39.190	62.680	44.000	44.970	65.820	51.161	54.281
	R. Traversal	538.731	475.249	110.929	433.811	445.043	469.262	367.021	127.791	466.261	173.699
4	Lookup	200.332	111.921	125.039	158.692	158.121	113.141	115.871	114.210	128.390	186.250
	Traversal	231.421	195.629	206.371	238.710	198.829	274.000	328.651	210.611	200.930	168.622
	Insert	69.571	44.760	81.300	179.601	59.590	66.020	49.240	46.861	63.529	61.031
	R. Traversal	538.265	646.662	100.891	286.671	443.041	473.871	376.791	123.481	482.922	128.440
5	Lookup	187.641	182.832	281.164	224.732	143.251	120.220	122.129	152.909	144.810	196.868
	Traversal	229.881	203.080	173.599	173.451	168.431	187.021	202.211	215.140	331.372	163.773
	Insert	62.010	54.740	53.010	57.040	49.750	70.490	38.310	40.710	39.570	40.562
	R. Traversal	532.411	615.943	122.490	281.751	435.071	479.011	374.752	131.930	468.850	130.211

E.3.3 DBMS-3 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	96.001	105.311	131.552	159.892	172.470	206.971	236.901	286.421	285.891	303.741
	Traversal	81.480	71.601	62.580	82.407	101.900	108.400	96.681	150.230	187.051	236.680
	Insert	370.313	348.152	375.790	358.672	389.551	414.530	519.247	481.011	502.658	585.371
	R. Traversal	64.721	16.860	71.620	121.990	81.921	150.761	62.480	120.180	110.830	179.381
2	Lookup	105.783	102.792	149.615	210.735	182.163	223.819	253.001	299.878	284.112	322.960
	Traversal	72.500	75.940	55.980	83.861	100.310	103.500	102.070	154.631	172.720	174.900
	Insert	468.919	459.098	474.115	441.395	452.941	470.072	488.583	457.770	486.762	496.007
	R. Traversal	61.752	17.791	76.002	126.523	87.102	143.482	70.761	108.471	117.631	163.812
3	Lookup	100.249	101.981	120.982	154.381	176.622	194.691	221.671	275.719	334.119	324.011
	Traversal	67.583	60.032	56.432	76.302	86.333	97.082	88.581	139.501	168.832	190.331
	Insert	414.953	455.918	442.519	396.346	432.492	418.303	381.039	422.081	436.212	399.700
	R. Traversal	64.620	17.200	69.740	126.520	86.410	161.290	75.540	107.900	122.499	186.111
4	Lookup	108.581	105.960	124.930	152.601	175.009	201.231	223.249	251.061	296.969	325.010
	Traversal	68.430	61.560	61.560	77.880	90.731	90.530	92.520	134.100	155.481	190.210
	Insert	399.069	434.352	372.822	387.582	394.812	381.830	374.581	387.170	394.390	406.940
	R. Traversal	66.673	16.801	71.822	123.203	79.351	142.233	80.401	187.301	195.112	203.049
5	Lookup	94.231	98.620	122.310	155.600	171.642	203.831	222.163	244.732	262.941	292.578
	Traversal	63.220	56.610	57.850	72.510	86.371	89.620	81.360	142.151	158.161	202.050
	Insert	361.311	399.971	378.670	397.991	407.200	393.701	386.971	405.641	395.311	397.418
	R. Traversal	66.601	16.950	71.220	131.442	83.921	154.231	68.310	112.311	110.850	179.000

E.3.4 DBMS-1 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	51.690	19.935	54.710	32.788	38.920	41.358	145.320	94.081	64.980	48.435
2	48.070	18.494	52.270	33.133	37.880	37.117	138.220	88.744	60.801	53.793
3	46.390	17.463	51.080	33.483	37.649	37.567	135.119	88.513	64.401	51.703
4	48.640	18.599	53.590	33.629	40.231	40.764	142.461	92.992	63.561	49.868
5	49.080	18.568	54.751	33.911	37.290	39.047	141.121	91.526	63.350	51.710
Average:	48.774	18.612	53.280	33.389	38.394	39.171	140.448	91.171	63.419	51.102
Sample SD:	1.923	0.878	1.594	0.438	1.193	1.879	3.924	2.493	1.603	2.038

E.3.5 DBMS-2 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	199.830	114.413	286.909	268.207	54.231	60.647	540.970	443.267	548.548	335.947
2	155.119	163.942	241.490	253.783	59.730	67.053	456.339	484.778	559.081	344.490
3	139.459	117.869	297.310	254.639	55.150	50.310	491.919	422.818	538.731	341.007
4	200.332	134.626	231.421	224.706	69.571	72.437	501.324	431.769	538.265	340.308
5	187.641	174.324	229.881	202.009	62.010	49.354	479.532	425.687	532.411	337.779
Average:	176.476	141.035	257.402	240.669	60.138	59.960	494.017	441.664	543.407	339.906
Sample SD:	27.684	27.016	32.206	26.809	6.173	10.150	31.192	25.346	10.504	3.263

E.3.6 DBMS-3 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	96.001	209.906	81.480	121.948	370.313	441.665	547.794	773.519	64.721	101.780
2	105.783	225.453	72.500	113.768	468.919	469.638	647.202	808.859	61.752	101.286
3	100.249	211.575	67.583	107.047	414.953	420.512	582.785	739.134	64.620	105.912
4	108.581	206.224	68.430	106.064	399.069	392.720	576.080	705.008	66.673	122.141
5	94.231	197.157	63.220	105.187	361.311	395.875	518.762	698.219	66.601	103.137
Average:	100.969	210.063	70.643	110.803	402.913	424.082	574.525	744.948	64.873	106.851
Sample SD:	6.159	10.253	6.897	7.091	42.749	32.313	47.897	46.673	2.003	8.734

E.3.7 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	48.774	18.612	53.280	33.389	38.394	39.171	140.448	91.171	63.419	51.102
	Sample SD:	1.923	0.878	1.594	0.438	1.193	1.879	3.924	2.493	1.603	2.038
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	176.476	141.035	257.402	240.669	60.138	59.960	494.017	441.664	543.407	339.906
	Sample SD:	27.684	27.016	32.206	26.809	6.173	10.150	31.192	25.346	10.504	3.263
	t =	-10.290	-10.128	-14.155	-17.286	-7.734	-4.504	-25.148	-30.772	-101.011	-167.875

E.3.8 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	48.774	18.612	53.280	33.389	38.394	39.171	140.448	91.171	63.419	51.102
	Sample SD:	1.923	0.878	1.594	0.438	1.193	1.879	3.924	2.493	1.603	2.038
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	100.969	210.063	70.643	110.803	402.913	424.082	574.525	744.948	64.873	106.851
	Sample SD:	6.159	10.253	6.897	7.091	42.749	32.313	47.897	46.673	2.003	8.734
	t =	-18.089	-41.603	-5.485	-24.367	-19.059	-26.591	-20.197	-31.277	-1.268	-13.899

E.3.9 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	176.476	141.035	257.402	240.669	60.138	59.960	494.017	441.664	543.407	339.906
	Sample SD:	27.684	27.016	32.206	26.809	6.173	10.150	31.192	25.346	10.504	3.263
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	100.969	210.063	70.643	110.803	402.913	424.082	574.525	744.948	64.873	106.851
	Sample SD:	6.159	10.253	6.897	7.091	42.749	32.313	47.897	46.673	2.003	8.734
	t =	5.953	-5.342	12.679	10.472	-17.745	-24.039	-3.149	-12.769	100.067	55.892

E.4 Small Local Database (NLOR)

E.4.1 DBMS-1 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	18.140	11.500	11.460	11.750	11.370	11.630	11.580	11.560	11.740	11.470
	Traversal	27.900	6.130	4.950	5.030	5.270	4.950	5.100	5.350	5.880	5.000
	Insert	80.760	73.841	75.320	70.801	75.080	75.420	73.690	76.210	73.069	76.940
	R. Traversal	28.380	2.670	5.020	5.050	11.900	5.180	2.470	17.280	5.830	7.140
2	Lookup	20.530	13.990	14.080	13.870	14.160	13.950	14.020	14.080	14.140	13.860
	Traversal	26.070	5.660	4.890	5.230	5.800	5.770	5.130	4.980	4.750	4.910
	Insert	85.080	71.300	74.930	70.510	71.851	74.280	72.631	72.450	71.410	76.034
	R. Traversal	28.690	2.700	5.810	4.880	11.600	5.550	2.500	17.000	5.810	7.450
3	Lookup	16.949	13.450	13.270	13.270	13.150	13.350	13.110	13.300	13.100	13.200
	Traversal	28.520	5.610	4.740	4.880	4.920	5.210	4.870	5.200	4.630	4.850
	Insert	76.141	72.851	73.010	68.710	70.011	72.150	70.630	69.161	70.470	73.270
	R. Traversal	26.950	2.640	5.010	4.800	12.100	5.350	2.450	17.340	6.230	7.250
4	Lookup	15.360	11.650	11.250	11.340	11.600	12.780	11.850	11.500	11.800	11.680
	Traversal	25.601	6.420	4.770	4.860	5.390	4.880	4.920	4.960	4.680	4.800
	Insert	82.190	76.520	78.440	74.150	74.431	76.500	74.950	74.511	75.580	77.481
	R. Traversal	29.601	2.700	5.400	4.800	11.710	5.250	2.450	17.600	5.880	7.250
5	Lookup	16.600	12.060	12.350	12.100	12.350	12.120	11.930	12.380	12.150	12.290
	Traversal	27.530	5.860	4.840	5.180	5.170	5.000	5.000	5.000	4.710	4.800
	Insert	79.181	72.781	75.280	69.479	72.690	73.650	75.580	77.470	75.221	73.450
	R. Traversal	32.420	2.700	4.910	5.300	11.860	5.130	2.420	17.150	5.730	7.420

E.4.2 DBMS-2 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	23.940	1.890	2.100	2.550	2.420	1.960	8.830	1.910	1.890	1.920
	Traversal	40.760	5.690	5.260	5.390	8.880	5.230	5.170	5.400	5.400	5.330
	Insert	74.320	15.730	8.540	13.270	5.540	5.910	5.690	5.420	9.520	5.430
	R. Traversal	35.651	13.930	6.830	6.000	7.170	4.480	3.630	4.790	6.200	4.160
2	Lookup	20.030	1.990	8.010	1.960	1.900	1.900	1.960	1.890	1.930	1.910
	Traversal	36.850	5.700	5.350	5.770	5.280	7.350	5.300	5.350	5.500	5.320
	Insert	39.810	7.460	6.570	6.130	5.630	6.170	11.700	5.490	5.530	5.310
	R. Traversal	42.100	6.180	6.450	5.550	8.050	4.350	3.680	3.950	6.000	4.050
3	Lookup	54.790	3.540	2.430	2.100	2.100	2.150	2.320	4.770	2.230	2.170
	Traversal	35.420	5.580	5.040	10.590	5.160	5.110	4.990	5.050	5.160	5.020
	Insert	51.560	10.970	13.920	7.900	12.050	6.760	14.390	6.690	13.040	6.780
	R. Traversal	40.770	6.200	6.630	5.950	9.900	4.380	3.740	4.000	6.100	4.130
4	Lookup	25.370	1.860	1.940	1.970	1.910	1.850	1.920	1.860	1.910	1.860
	Traversal	40.211	5.790	5.160	5.670	5.130	7.090	5.110	5.220	5.360	5.140
	Insert	47.641	8.330	7.380	5.950	9.760	5.480	6.010	6.420	4.840	11.660
	R. Traversal	37.740	5.800	6.340	8.560	6.840	4.550	3.580	3.820	5.930	4.010
5	Lookup	19.370	1.920	1.980	1.970	1.960	2.040	8.240	2.070	2.100	1.970
	Traversal	33.080	15.410	5.390	5.860	5.310	5.420	6.160	5.400	5.680	5.340
	Insert	45.731	8.600	7.210	6.730	6.510	13.560	7.170	13.620	5.570	6.390
	R. Traversal	40.101	5.820	6.430	5.500	6.780	4.660	4.340	4.090	5.960	4.050

E.4.3 DBMS-3 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	56.570	63.790	82.721	101.570	117.200	132.811	149.460	160.831	173.740	185.191
	Traversal	45.890	17.720	15.330	13.500	13.130	11.240	10.730	11.060	9.370	9.080
	Insert	384.092	352.132	359.001	371.071	366.631	390.282	367.361	416.011	377.832	388.849
	R. Traversal	45.281	7.700	13.750	13.070	28.520	11.870	5.580	31.330	10.640	11.800
2	Lookup	55.820	63.941	82.400	102.500	117.941	132.931	147.960	161.671	175.100	184.430
	Traversal	43.440	18.790	15.960	14.470	13.430	11.600	10.810	11.040	10.070	9.310
	Insert	366.961	378.030	390.052	373.971	382.510	393.531	384.300	451.072	481.956	511.745
	R. Traversal	43.020	6.490	14.630	13.640	28.730	11.450	6.200	31.130	10.700	12.280
3	Lookup	56.031	63.510	82.070	99.960	117.771	133.020	146.290	160.811	173.300	184.531
	Traversal	41.991	18.560	14.690	13.460	12.590	11.360	10.750	10.660	9.360	9.290
	Insert	492.391	492.860	526.063	459.262	453.422	496.680	467.613	447.430	478.175	451.553
	R. Traversal	41.850	7.180	13.560	12.370	29.320	12.680	5.610	33.661	10.330	11.660
4	Lookup	57.901	63.930	82.720	102.150	116.391	133.300	147.500	159.831	175.820	184.001
	Traversal	41.700	18.360	15.350	14.240	13.360	11.470	10.870	10.830	9.460	9.060
	Insert	469.742	424.751	414.472	421.863	431.892	414.732	406.812	424.361	419.978	425.980
	R. Traversal	44.201	6.220	13.200	12.300	28.280	10.930	5.470	30.200	10.680	11.570
5	Lookup	56.930	64.071	84.040	100.680	117.751	133.861	147.300	160.231	174.160	185.851
	Traversal	46.760	18.080	16.370	13.960	13.100	11.310	10.870	10.970	9.590	9.630
	Insert	412.182	398.102	387.681	378.332	386.050	433.752	387.542	628.201	416.391	399.161
	R. Traversal	40.670	6.660	13.310	12.130	28.260	11.900	5.900	30.430	10.570	12.730

E.4.4 DBMS-1 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	18.140	11.562	27.900	5.295	80.760	74.486	126.800	91.343	28.380	6.949
2	20.530	14.017	26.070	5.236	85.080	72.822	131.680	92.075	28.690	7.033
3	16.949	13.245	28.520	4.990	76.141	71.140	121.610	89.375	26.950	7.019
4	15.360	11.717	25.601	5.075	82.190	75.840	123.151	92.632	29.601	7.004
5	16.600	12.192	27.530	5.062	79.181	73.956	123.311	91.210	32.420	6.958
Average:	17.516	12.547	27.124	5.132	80.670	73.649	125.310	91.327	29.208	6.993
Sample SD:	1.955	1.053	1.240	0.128	3.335	1.773	4.036	1.234	2.032	0.037

E.4.5 DBMS-2 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	23.940	2.830	40.760	5.750	74.320	8.339	139.020	16.919	35.651	6.354
2	20.030	2.606	36.850	5.658	39.810	6.666	96.690	14.930	42.100	5.362
3	54.790	2.646	35.420	5.744	51.560	10.278	141.770	18.668	40.770	5.670
4	25.370	1.898	40.211	5.519	47.641	7.314	113.222	14.731	37.740	5.492
5	19.370	2.694	33.080	6.663	45.731	8.373	98.181	17.730	40.101	5.292
Average:	28.700	2.535	37.264	5.867	51.812	8.194	117.777	16.596	39.272	5.634
Sample SD:	14.804	0.366	3.240	0.455	13.277	1.370	21.658	1.728	2.568	0.427

E.4.6 DBMS-3 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
1	56.570	129.702	45.890	12.351	384.092	376.574	486.552	518.627	45.281	14.918
2	55.820	129.875	43.440	12.831	366.961	416.352	466.221	559.058	43.020	15.028
3	56.031	129.029	41.991	12.302	492.391	474.784	590.413	616.115	41.850	15.152
4	57.901	129.516	41.700	12.556	469.742	420.538	569.343	562.610	44.201	14.317
5	56.930	129.772	46.760	12.653	412.182	423.912	515.872	566.337	40.670	14.654
Average:	56.650	129.579	43.956	12.539	425.074	422.432	525.680	564.549	43.004	14.814
Sample SD:	0.825	0.334	2.281	0.218	54.198	34.952	53.055	34.659	1.830	0.333

E.4.7 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	17.516	12.547	27.124	5.132	80.670	73.649	125.310	91.327	29.208	6.993
	Sample SD:	1.955	1.053	1.240	0.128	3.335	1.773	4.036	1.234	2.032	0.037
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	28.700	2.535	37.264	5.867	51.812	8.194	117.777	16.596	39.272	5.634
	Sample SD:	14.804	0.366	3.240	0.455	13.277	1.370	21.658	1.728	2.568	0.427
	t =	-1.675	20.087	-6.536	-3.479	4.714	65.325	0.765	78.721	-6.871	7.080

E.4.8 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	17.516	12.547	27.124	5.132	80.670	73.649	125.310	91.327	29.208	6.993
	Sample SD:	1.955	1.053	1.240	0.128	3.335	1.773	4.036	1.234	2.032	0.037
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	56.650	129.579	43.956	12.539	425.074	422.432	525.680	564.549	43.004	14.814
	Sample SD:	0.825	0.334	2.281	0.218	54.198	34.952	53.055	34.659	1.830	0.333
	t =	-41.247	-236.932	-14.496	-65.500	-14.182	-22.285	-16.825	-30.511	-11.279	-52.198

E.4.9 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	28.700	2.535	37.264	5.867	51.812	8.194	117.777	16.596	39.272	5.634
	Sample SD:	14.804	0.366	3.240	0.455	13.277	1.370	21.658	1.728	2.568	0.427
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	56.650	129.579	43.956	12.539	425.074	422.432	525.680	564.549	43.004	14.814
	Sample SD:	0.825	0.334	2.281	0.218	54.198	34.952	53.055	34.659	1.830	0.333
	t =	-4.215	-573.371	-3.776	-29.583	-14.958	-26.481	-15.916	-35.308	-2.646	-37.885

E.5 Small Local Database (LOR vs. NLOR)

E.5.1 DBMS-1 vs. DBMS-1 (NLOR) Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	19.020	12.549	19.968	3.916	25.304	26.165	64.292	42.629	28.786	3.598
	Sample SD:	0.874	0.721	3.547	0.164	1.774	1.263	3.702	1.901	1.199	0.352
DBMS-1 (NLOR)	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	17.516	12.547	27.124	5.132	80.670	73.649	125.310	91.327	29.208	6.993
	Sample SD:	1.955	1.053	1.240	0.128	3.335	1.773	4.036	1.234	2.032	0.037
	t =	1.571	0.004	-4.259	-13.045	-32.771	-48.779	-24.914	-48.056	-0.400	-21.443

E.5.2 DBMS-2 vs. DBMS-2 (NLOR) Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	24.478	2.812	35.415	6.445	21.316	4.745	81.209	14.002	35.600	6.305
	Sample SD:	2.446	0.196	2.711	0.356	2.423	0.403	3.679	0.600	3.111	0.516
DBMS-2 (NLOR)	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	28.700	2.535	37.264	5.867	51.812	8.194	117.777	16.596	39.272	5.634
	Sample SD:	14.804	0.366	3.240	0.455	13.277	1.370	21.658	1.728	2.568	0.427
	t =	-0.629	1.494	-0.979	2.237	-5.053	-5.402	-3.722	-3.172	-2.036	2.240

E.5.3 DBMS-3 vs. DBMS-3 (NLOR) Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm	Insert Cold	Insert Warm	L+T+I Cold	L+T+I Warm	R. Traversal Cold	R. Traversal Warm
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	57.724	128.813	25.942	9.119	335.230	342.046	418.896	479.979	31.732	8.650
	Sample SD:	1.305	0.838	0.951	0.120	19.841	6.406	20.638	5.756	1.017	0.244
DBMS-3 (NLOR)	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	56.650	129.579	43.956	12.539	425.074	422.432	525.680	564.549	43.004	14.814
	Sample SD:	0.825	0.334	2.281	0.218	54.198	34.952	53.055	34.659	1.830	0.333
	t =	1.555	-1.898	-16.297	-30.718	-3.481	-5.059	-4.194	-5.382	-12.040	-33.368

APPENDIX F - The AFIT Wargame Simulation Benchmark

F.1 Introduction

This appendix contains support material for Chapter 7. Raw results and a statistical analysis for the AFIT Benchmark are presented.

F.2 Small Local Database

F.2.1 DBMS-1 Average and Sample Standard Deviation

Benchmark	Model Creation	Scenario Creation	TS=60	TS=600	TS=1800	TS=3600	Version Creation	Map Creation	Report Creation
1	0.500	79.850	8517.116	8489.064	8781.745	8714.206	172.060	9.890	2.030
2	0.470	72.480	7767.054	7586.754	7918.035	7381.211	172.760	9.600	2.040
3	0.420	69.771	7247.652	7709.894	8930.609	7348.673	173.830	10.040	1.990
4	0.540	70.110	7343.732	7907.623	7447.964	7431.251	189.180	9.900	2.390
5	0.480	66.201	7036.223	8015.213	8049.384	7550.290	184.031	10.250	2.020
Average:	0.482	71.682	7582.355	7941.710	8225.547	7685.126	178.372	9.936	2.094
Sample SD:	0.044	5.087	586.327	348.530	619.819	580.346	7.759	0.238	0.167

Throughput Samples	Value
1	1.875
2	1.998
3	2.101
4	1.887
5	1.946
6	1.983
7	1.900
8	1.973
9	2.031
10	2.127
Geometric Mean	1.980
90th Percentile	2.104

F.2.2 DBMS-2 Average and Sample Standard Deviation

Benchmark	Model Creation	Scenario Creation	TS=60	TS=600	TS=1800	TS=3600	Version Creation	Map Creation	Report Creation
1	0.150	4.110	632.093	546.080	531.932	525.581	20.310	11.090	1.580
2	0.130	4.290	636.960	544.581	534.700	530.522	23.810	10.730	1.270
3	0.140	3.070	593.628	527.111	517.571	516.470	22.860	8.910	1.180
4	0.160	3.030	599.938	524.943	518.181	514.771	22.240	24.110	1.190
5	0.120	2.980	642.590	545.141	559.635	543.344	22.700	9.220	1.340
Average:	0.140	3.496	621.042	537.571	532.404	526.138	22.384	12.812	1.312
Sample SD:	0.016	0.647	22.565	10.580	17.098	11.600	1.292	6.385	0.163

Throughput Samples	Value
1	0.198
2	0.181
3	0.190
4	0.199
5	0.186
6	0.193
7	0.196
8	0.197
9	0.195
10	0.190
Geometric Mean	0.192
90th Percentile	0.198

F.2.3 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Model Creation	Scenario Creation	TS=60	TS=600	TS=1800	TS=3600	Version Creation	Map Creation	Report Creation
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	0.482	71.682	7582.355	7941.710	8225.547	7685.126	178.372	9.936	2.094
	Sample SD:	0.044	5.087	586.327	348.530	619.819	580.346	7.759	0.238	0.167
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
	Average:	0.140	3.496	621.042	537.571	532.404	526.138	22.384	12.812	1.312
	Sample SD:	0.016	0.647	22.565	10.580	17.098	11.600	1.292	6.385	0.163
	t =	16.417	29.732	26.529	47.481	27.743	27.578	44.344	-1.006	7.497

APPENDIX G - The CITY Benchmark

G.1 Introduction

This appendix contains support material for Chapter 7. Raw results and a statistical analysis for the CITY Benchmark are presented.

G.2 10 Database

G.2.1 DBMS-1 Average and Sample Standard Deviation

10 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	2554	0.709	1.410	13186	986	6709248
2	3600	2578	0.716	1.396	13035	994	6709248
3	3600	2476	0.688	1.454	12330	1003	6709248
4	3600	2464	0.684	1.461	12872	1012	6676480
5	3600	2510	0.697	1.434	12660	1001	6684672
Average:	3600	2516.400	0.699	1.431	12816.600	999.200	6697779.200
Sample SD:	0	49.059	0.014	0.028	334.861	9.783	15969.148

G.2.2 DBMS-2 Average and Sample Standard Deviation

10 Benchmark	Loop Time	Total Time	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	95	0.026	37.895	218714	702	15597568
2	3600	96	0.027	37.500	221036	702	15630336
3	3600	95	0.026	37.895	218972	708	15630336
4	3600	97	0.027	37.113	223561	784	15654912
5	3600	98	0.027	36.735	225607	714	15638528
Average:	3600	96.200	0.027	37.428	221578.000	722.000	15630336.000
Sample SD:	0	1.304	0.000	0.505	2977.105	35.014	20885.584

G.2.3 DBMS-3 (C++) Average and Sample Standard Deviation

10 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	623	0.173	5.778	12091	5384	16203776
2	3600	598	0.166	6.020	12812	2200	16039936
3	3600	716	0.199	5.028	15247	2223	17416192
4	3600	712	0.198	5.056	15156	2139	17637376
5	3600	721	0.200	4.993	15400	2120	17620992
Average:	3600	674.000	0.187	5.375	14141.200	2813.200	16983654.400
Sample SD:	0	58.724	0.016	0.487	1565.829	1437.744	793639.506

G.2.4 DBMS-3 (ESQL) Average and Sample Standard Deviation

10 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	354	0.098	10.169	307183	58023	9527296
2	3600	428	0.119	8.411	384822	42277	9887744
3	3600	458	0.127	7.860	413562	41299	9887744
4	3600	428	0.119	8.411	389627	35758	9879552
5	3600	466	0.129	7.725	432549	24675	9838592
Average:	3600	426.800	0.119	8.516	385548.600	40406.400	9804185.600
Sample SD:	0	44.195	0.012	0.976	47833.348	12080.201	156121.553

G.3 20 Database

G.3.1 DBMS-1 Average and Sample Standard Deviation

20 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	2323	0.645	1.550	25389	2043	6692864
2	3600	2318	0.644	1.553	25077	2235	6651904
3	3600	2412	0.670	1.493	24838	1785	6684672
4	3600	2484	0.690	1.449	25405	1799	6709248
5	3600	2281	0.634	1.578	24249	1810	6676480
Average:	3600	2363.600	0.657	1.525	24991.600	1934.400	6683033.600
Sample SD:	0	82.748	0.023	0.052	477.275	198.924	21204.466

G.3.2 DBMS-2 Average and Sample Standard Deviation

20 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	32	0.009	112.500	145028	1940	15646720
2	3600	37	0.010	97.297	167475	1373	15581184
3	3600	42	0.012	85.714	189995	1034	15654912
4	3600	46	0.013	78.261	208290	839	15654912
5	3600	45	0.013	80.000	203623	841	15679488
Average:	3600	40.400	0.011	90.754	182882.200	1205.400	15643443.200
Sample SD:	0	5.857	0.002	14.255	26441.687	464.796	36909.483

G.3.3 DBMS-3 (C++) Average and Sample Standard Deviation

20 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	404	0.112	8.911	277747	23839	14163968
2	3600	464	0.129	7.759	320104	25182	15384576
3	3600	475	0.132	7.579	330549	19739	15589376
4	3600	399	0.111	9.023	277776	16013	14155776
5	3600	531	0.148	6.780	373588	14775	16048128
Average:	3600	454.600	0.126	8.010	315952.800	19909.600	15068364.800
Sample SD:	0	54.757	0.015	0.949	40215.821	4604.756	863442.241

G.3.4 DBMS-3 (ESQL) Average and Sample Standard Deviation

20 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	749	0.208	4.806	537150	10492	9977856
2	3600	668	0.186	5.389	479020	10298	9977856
3	3600	762	0.212	4.724	547497	8765	9977856
4	3600	789	0.219	4.563	566810	11400	9977856
5	3600	765	0.213	4.706	550013	8986	9977856
Average:	3600	746.600	0.207	4.838	536098.000	9988.200	9977856.000
Sample SD:	0	46.253	0.013	0.321	33637.019	1100.408	0.000

G.4 30 Database

G.4.1 DBMS-1 Average and Sample Standard Deviation

30 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	2050	0.569	1.756	24104	2541	6676480
2	3600	2361	0.656	1.525	27472	2591	6701056
3	3600	2372	0.659	1.518	27722	2595	6725632
4	3600	2401	0.667	1.499	27949	2562	6701056
5	3600	2366	0.657	1.522	27610	2588	6701056
Average:	3600	2310.000	0.642	1.564	26971.400	2575.400	6701056.000
Sample SD:	0	146.169	0.041	0.108	1612.412	23.180	17377.856

G.4.2 DBMS-2 Average and Sample Standard Deviation

30 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	23	0.006	156.522	153817	3885	15654912
2	3600	23	0.006	156.522	153886	4010	15638528
3	3600	25	0.007	144.000	167005	2398	15638528
4	3600	23	0.006	156.522	154109	2749	15704064
5	3600	24	0.007	150.000	160591	3281	15638528
Average:	3600	23.600	0.007	152.713	157881.600	3264.600	15654912.000
Sample SD:	0	0.894	0.000	5.630	5858.660	699.576	28377.920

G.4.3 DBMS-3 (C++) Average and Sample Standard Deviation

30 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	455	0.126	7.912	472888	16198	15671296
2	3600	452	0.126	7.965	470488	14832	15917056
3	3600	470	0.131	7.660	491209	12867	15785984
4	3600	471	0.131	7.643	491629	13512	15556608
5	3600	468	0.130	7.692	487863	13869	16162816
Average:	3600	463.200	0.129	7.774	482815.400	14255.600	15818752.000
Sample SD:	0	8.983	0.002	0.152	10297.189	1297.668	234296.929

G.4.4 DBMS-3 (ESQL) Average and Sample Standard Deviation

30 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	560	0.156	6.429	580771	31680	9977856
2	3600	485	0.135	7.423	493195	47070	9977856
3	3600	484	0.134	7.438	491236	47146	9977856
4	3600	477	0.133	7.547	485597	53438	9977856
5	3600	478	0.133	7.531	487023	51653	9977856
Average:	3600	496.800	0.138	7.274	507564.400	46197.400	9977856.000
Sample SD:	0	35.506	0.010	0.476	41039.062	8582.046	0.000

G.5 40 Database

G.5.1 DBMS-1 Average and Sample Standard Deviation

40 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	2128	0.591	1.692	31817	3536	6692864
2	3600	1642	0.456	2.192	22758	4440	6692864
3	3600	1664	0.462	2.163	20741	13645	6651904
4	3600	1881	0.523	1.914	27336	8975	6660096
5	3600	1972	0.548	1.826	28111	8599	6701056
Average:	3600	1857.400	0.516	1.957	26152.600	7839.000	6679756.800
Sample SD:	0	206.588	0.057	0.217	4419.556	4051.134	22133.565

G.5.2 DBMS-2 Average and Sample Standard Deviation

40 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	17	0.005	211.765	153294	3871	15728640
2	3600	17	0.005	211.765	153408	4032	15712256
3	3600	17	0.005	211.765	153281	3986	15728640
4	3600	18	0.005	200.000	162302	4196	15753216
5	3600	17	0.005	211.765	153463	3825	15720448
Average:	3600	17.200	0.005	209.412	155149.600	3982.000	15728640.000
Sample SD:	0	0.447	0.000	5.261	3999.048	146.015	15325.829

G.5.3 DBMS-3 (C++) Average and Sample Standard Deviation

40 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	291	0.081	12.371	391937	31462	13058048
2	3600	313	0.087	11.502	418213	37261	13008896
3	3600	333	0.093	10.811	447465	40658	14073856
4	3600	337	0.094	10.682	455148	27484	14049280
5	3600	360	0.100	10.000	486519	33235	14303232
Average:	3600	326.800	0.091	11.073	439856.400	34020.000	13698662.400
Sample SD:	0	26.061	0.007	0.900	36172.599	5109.942	615502.201

G.5.4 DBMS-3 (ESQL) Average and Sample Standard Deviation

40 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	436	0.121	8.257	593787	33705	9994240
2	3600	418	0.116	8.612	571566	26610	9977856
3	3600	371	0.103	9.704	509132	20103	9994240
4	3600	397	0.110	9.068	543238	23905	9986048
5	3600	376	0.104	9.574	515802	19771	9986048
Average:	3600	399.600	0.111	9.043	546705.000	24818.800	9987686.400
Sample SD:	0	27.592	0.008	0.617	36102.389	5716.796	6853.919

G.6 50 Database

G.6.1 DBMS-1 Average and Sample Standard Deviation

50 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	2061	0.573	1.747	33559	5269	6709248
2	3600	2163	0.601	1.664	35362	4871	6709248
3	3600	2130	0.592	1.690	33747	5172	6709248
4	3600	2105	0.585	1.710	33380	5502	6709248
5	3600	2083	0.579	1.728	33219	5702	6701056
Average:	3600	2108.400	0.586	1.708	33853.400	5303.200	6707609.600
Sample SD:	0	39.847	0.011	0.032	866.089	317.704	3663.574

G.6.2 DBMS-2 Average and Sample Standard Deviation

50 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	14	0.004	257.143	156423	3882	15802368
2	3600	14	0.004	257.143	156447	3960	15810560
3	3600	13	0.004	276.923	145301	3716	15810560
4	3600	13	0.004	276.923	145392	3797	15826944
5	3600	14	0.004	257.143	156450	3918	15785984
Average:	3600	13.600	0.004	265.055	152002.600	3854.600	15807283.200
Sample SD:	0	0.548	0.000	10.834	6076.254	97.989	14881.507

G.6.3 DBMS-3 (C++) Average and Sample Standard Deviation

50 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	350	0.097	10.286	595338	20300	14090240
2	3600	340	0.094	10.588	576703	23766	14090240
3	3600	362	0.101	9.945	617631	17795	14188544
4	3600	355	0.099	10.141	605105	18446	14278656
5	3600	350	0.097	10.286	596338	18212	14196736
Average:	3600	351.400	0.098	10.249	598223.000	19703.800	14168883.200
Sample SD:	0	8.050	0.002	0.236	14991.825	2465.156	79971.712

G.6.4 DBMS-3 (ESQL) Average and Sample Standard Deviation

50 Benchmark	Loop Time	Total Trans	Trans/Sec	Response Time	Page Faults (no physical I/O)	Page Faults (physical I/O)	Working Set
1	3600	407	0.113	8.845	694700	23862	9986048
2	3600	411	0.114	8.759	699536	26372	9961472
3	3600	387	0.108	9.302	658992	27907	9994240
4	3600	369	0.103	9.756	628065	25321	9994240
5	3600	378	0.105	9.524	642941	25020	9994240
Average:	3600	390.400	0.108	9.237	664846.800	25696.400	9986048.000
Sample SD:	0	18.188	0.005	0.429	31470.612	1525.340	14188.960

G.7 All Databases

G.7.1 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		10 Trans	20 Trans	30 Trans	40 Trans	50 Trans
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000
	Average:	2516.400	2363.600	2310.000	1857.400	2108.400
	Sample SD:	49.059	82.748	146.169	206.588	39.847
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000
	Average:	96.200	40.400	23.600	17.200	13.600
	Sample SD:	1.304	5.857	0.894	0.447	0.548
	t =	110.271	62.622	34.976	19.918	117.541

G.7.2 DBMS-1 vs. DBMS-3 (C++) Small Sample Test

DBMS		10 Trans	20 Trans	30 Trans	40 Trans	50 Trans
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000
	Average:	2516.400	2363.600	2310.000	1857.400	2108.400
	Sample SD:	49.059	82.748	146.169	206.588	39.847
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000
(C++)	Average:	674.000	454.600	463.200	326.800	351.400
	Sample SD:	58.724	54.757	8.983	26.061	8.050
	t =	53.839	43.020	28.199	16.437	96.644

G.7.3 DBMS-1 vs. DBMS-3 (ESQL) Small Sample Test

DBMS		10 Trans	20 Trans	30 Trans	40 Trans	50 Trans
DBMS-1	N:	5.000	5.000	5.000	5.000	5.000
	Average:	2516.400	2363.600	2310.000	1857.400	2108.400
	Sample SD:	49.059	82.748	146.169	206.588	39.847
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000
(ESQL)	Average:	426.800	746.600	496.800	399.600	390.400
	Sample SD:	44.195	46.253	35.506	27.592	18.188
	t =	70.763	38.141	26.954	15.640	87.703

G.7.4 DBMS-2 vs. DBMS-3 (C++) Small Sample Test

DBMS		10 Trans	20 Trans	30 Trans	40 Trans	50 Trans
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000
	Average:	96.200	40.400	23.600	17.200	13.600
	Sample SD:	1.304	5.857	0.894	0.447	0.548
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000
(C++)	Average:	674.000	454.600	463.200	326.800	351.400
	Sample SD:	58.724	54.757	8.983	26.061	8.050
	t =	-21.996	-16.819	-108.884	-26.560	-93.617

G.7.5 DBMS-2 vs. DBMS-3 (ESQL) Small Sample Test

DBMS		10 Trans	20 Trans	30 Trans	40 Trans	50 Trans
DBMS-2	N:	5.000	5.000	5.000	5.000	5.000
	Average:	96.200	40.400	23.600	17.200	13.600
	Sample SD:	1.304	5.857	0.894	0.447	0.548
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000
(ESQL)	Average:	426.800	746.600	496.800	399.600	390.400
	Sample SD:	44.195	46.253	35.506	27.592	18.188
	t =	-16.720	-33.871	-29.791	-30.986	-46.304

G.7.6 DBMS-3 (C++) vs. DBMS-3 (ESQL) Small Sample Test

DBMS		10 Trans	20 Trans	30 Trans	40 Trans	50 Trans
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000
(C++)	Average:	674.000	454.600	463.200	326.800	351.400
	Sample SD:	58.724	54.757	8.983	26.061	8.050
DBMS-3	N:	5.000	5.000	5.000	5.000	5.000
(ESQL)	Average:	426.800	746.600	496.800	399.600	390.400
	Sample SD:	44.195	46.253	35.506	27.592	18.188
	t =	7.521	-9.109	-2.051	-4.289	-4.385

APPENDIX H - The X.500 Benchmark

H.1 Introduction

This appendix contains support material for Chapter 7. Raw results and a statistical analysis for the X.500 Benchmark are presented.

H.2 4 MB Data File from NEXOR Ltd.

H.2.1 DBMS-1 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Exact	0.440	0.100	0.060	0.170	0.080	0.050	0.120	0.150	0.050	0.050
	Wildcard	10.530	10.350	9.550	10.750	9.800	9.740	9.830	8.430	10.000	10.320
2	Exact	0.470	0.100	0.060	0.250	0.090	0.050	0.080	0.120	0.060	0.050
	Wildcard	10.620	10.230	9.630	10.300	8.970	9.010	10.040	7.280	9.280	10.800
3	Exact	0.610	0.170	0.100	0.170	0.200	0.050	0.110	0.100	0.050	0.050
	Wildcard	10.450	10.140	9.030	11.080	9.620	11.300	9.700	7.530	10.820	10.610
4	Exact	0.580	0.100	0.080	0.170	0.080	0.050	0.090	0.100	0.060	0.130
	Wildcard	11.180	9.870	9.800	11.650	10.160	10.400	9.820	7.680	8.920	9.430
5	Exact	0.450	0.100	0.080	0.280	0.090	0.050	0.130	0.100	0.050	0.050
	Wildcard	9.950	9.300	9.600	11.450	9.430	9.140	11.330	7.170	9.080	10.200

H.2.2 DBMS-2 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Exact	2.890	0.040	0.120	0.590	0.160	0.040	0.190	0.090	0.030	0.050
	Wildcard	36.490	3.690	3.010	3.530	19.710	4.020	4.450	2.680	3.240	3.380
2	Exact	3.550	0.040	0.120	0.560	0.130	0.050	0.250	0.110	0.050	0.100
	Wildcard	27.891	21.390	3.340	3.810	3.150	3.320	3.220	2.560	3.130	3.530
3	Exact	2.190	0.030	0.100	0.360	0.110	0.030	0.140	0.070	0.030	0.050
	Wildcard	25.550	3.050	21.939	3.500	3.060	4.520	3.270	2.870	3.790	3.450
4	Exact	2.760	0.040	0.100	0.470	0.100	0.030	0.170	0.090	0.030	0.050
	Wildcard	35.230	3.260	3.360	3.460	3.290	3.620	3.220	22.410	3.200	3.380
5	Exact	2.670	0.040	0.110	0.460	0.100	0.040	0.220	0.090	0.030	0.050
	Wildcard	44.418	4.780	5.180	4.360	2.900	2.970	3.140	2.760	3.180	4.630

H.2.3 DBMS-3 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Exact	1.380	0.160	0.400	0.280	0.360	0.120	0.110	0.100	0.860	0.160
	Wildcard	56.430	83.049	84.161	172.480	625.721	321.630	565.643	230.020	248.820	311.810
2	Exact	1.700	0.160	0.390	0.300	0.340	0.110	0.110	0.100	0.190	0.160
	Wildcard	57.400	74.129	33.460	147.350	153.380	182.471	258.130	234.431	255.271	281.220
3	Exact	1.450	0.170	0.370	0.320	0.380	0.160	0.150	0.140	0.110	0.160
	Wildcard	57.010	81.680	32.360	143.940	157.891	204.039	204.872	169.300	220.271	281.069
4	Exact	1.280	0.160	0.360	1.530	0.350	0.130	0.120	0.120	0.110	0.110
	Wildcard	56.890	76.150	29.990	131.951	147.510	191.531	203.581	144.510	212.381	304.350
5	Exact	1.780	0.170	0.350	0.250	0.340	0.110	1.410	0.130	0.080	0.130
	Wildcard	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

DBMS-3 failed benchmark run 5 of the **Wildcard** test.

H.2.4 DBMS-1 Average and Sample Standard Deviation

Benchmark	Exact Cold	Exact Warm	Wildcard Cold	Wildcard Warm
1	0.440	0.092	10.530	9.863
2	0.470	0.096	10.620	9.504
3	0.610	0.111	10.450	9.981
4	0.580	0.096	11.180	9.748
5	0.450	0.103	9.950	9.633
Average:	0.510	0.100	10.546	9.746
Sample SD:	0.079	0.008	0.439	0.187

H.2.5 DBMS-2 Average and Sample Standard Deviation

Benchmark	Exact Cold	Exact Warm	Wildcard Cold	Wildcard Warm
1	2.890	0.146	36.490	5.301
2	3.550	0.157	27.891	5.272
3	2.190	0.102	25.550	5.494
4	2.760	0.120	35.230	5.467
5	2.670	0.127	44.418	3.766
Average:	2.812	0.130	33.916	5.060
Sample SD:	0.490	0.022	7.499	0.730

H.2.6 DBMS-3 Average and Sample Standard Deviation

Benchmark	Exact Cold	Exact Warm	Wildcard Cold	Wildcard Warm
1	1.380	0.283	56.430	293.704
2	1.700	0.207	57.400	179.982
3	1.450	0.218	57.010	166.158
4	1.280	0.332	56.890	160.217
5	1.780	0.330	N/A	N/A
Average:	1.518	0.274	56.933	200.015
Sample SD:	0.213	0.060	0.400	63.006

DBMS-3 failed benchmark run 5 of the **Wildcard** test.

H.2.7 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Exact Cold	Exact Warm	Wildcard Cold	Wildcard Warm
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	0.510	0.100	10.546	9.746
	Sample SD:	0.079	0.008	0.439	0.187
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	2.812	0.130	33.916	5.060
	Sample SD:	0.490	0.022	7.499	0.730
	t =	-10.370	-3.004	-6.957	13.903

H.2.8 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Exact Cold	Exact Warm	Wildcard Cold	Wildcard Warm
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	0.510	0.100	10.546	9.746
	Sample SD:	0.079	0.008	0.439	0.187
DBMS-3	N:	5.000	5.000	4.000	4.000
	Average:	1.518	0.274	56.933	200.015
	Sample SD:	0.213	0.060	0.400	63.006
	t =	-9.906	-6.492	-163.571	-6.877

The number of degrees of freedom for **Wildcard** is 7, so for $\alpha = 0.05$ the critical value of t is $\pm$ 2.37.

H.2.9 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Exact Cold	Exact Warm	Wildcard Cold	Wildcard Warm
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	2.812	0.130	33.916	5.060
	Sample SD:	0.490	0.022	7.499	0.730
DBMS-3	N:	5.000	5.000	4.000	4.000
	Average:	1.518	0.274	56.933	200.015
	Sample SD:	0.213	0.060	0.400	63.006
	t =	5.414	-5.064	-6.046	-7.045

The number of degrees of freedom for **Wildcard** is 7, so for $\alpha = 0.05$ the critical value of t is ± 2.37.

APPENDIX I - The GIS Benchmark

I.1 Introduction

This appendix contains support material for Chapter 7. Raw results and a statistical analysis for the GIS Benchmark are presented.

I.2 11 MB Data File

I.2.1 DBMS-1 Average and Sample Standard Deviation

Benchmark	Spatial	Keyword	Keyword and Spatial	Total
1	100.723	1.670	1.080	103.473
2	117.527	1.540	1.080	120.147
3	101.979	1.600	1.060	104.639
4	126.669	2.960	2.000	131.629
5	107.031	1.590	1.080	109.701
Average:	110.786	1.872	1.260	113.918
Sample SD:	11.075	0.610	0.414	11.890

I.2.2 DBMS-2 Average and Sample Standard Deviation

Benchmark	Spatial	Keyword	Keyword and Spatial	Total
1	13.270	11.500	10.830	35.600
2	20.590	7.620	2.160	30.370
3	22.060	23.730	3.530	49.320
4	15.740	17.420	2.620	35.780
5	14.300	14.580	1.740	30.620
Average:	17.192	14.970	4.176	36.338
Sample SD:	3.908	6.102	3.779	7.709

I.2.3 DBMS-3 Average and Sample Standard Deviation

Benchmark	Spatial	Keyword	Keyword and Spatial	Total
1	408.370	15.410	11.650	435.430
2	420.020	16.830	11.780	448.630
3	396.354	15.250	11.680	423.284
4	381.140	15.700	11.630	408.470
5	409.492	16.750	12.130	438.372
Average:	403.075	15.988	11.774	430.837
Sample SD:	14.855	0.750	0.207	15.426

I.2.4 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Spatial	Keyword	Keyword and Spatial	Total
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	110.786	1.872	1.260	113.918
	Sample SD:	11.075	0.610	0.414	11.890
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	17.192	14.970	4.176	36.338
	Sample SD:	3.908	6.102	3.779	7.709
	t =	17.820	-4.776	-1.715	12.242

I.2.5 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Spatial	Keyword	Keyword and Spatial	Total
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	110.786	1.872	1.260	113.918
	Sample SD:	11.075	0.610	0.414	11.890
DBMS-3	N:	5.000	5.000	5.000	5.000
	Average:	403.075	15.988	11.774	430.837
	Sample SD:	14.855	0.750	0.207	15.426
	t =	-35.273	-32.645	-50.806	-36.386

I.2.6 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Spatial	Keyword	Keyword and Spatial	Total
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	17.192	14.970	4.176	36.338
	Sample SD:	3.908	6.102	3.779	7.709
DBMS-3	N:	5.000	5.000	5.000	5.000
	Average:	403.075	15.988	11.774	430.837
	Sample SD:	14.855	0.750	0.207	15.426
	t =	-56.175	-0.370	-4.489	-51.154

APPENDIX J - The OO-Fin Benchmark

J.1 Introduction

This appendix contains support material for Chapter 7. Raw results and a statistical analysis for the OO-Fin Benchmark are presented.

J.2 Small Database

J.2.1 DBMS-1 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	1.300	1.100	0.760	2.050	1.400	1.550	2.390	1.660	0.470	0.450
	Traversal	9.010	10.620	14.400	7.070	9.660	4.720	5.920	5.010	3.700	2.790
2	Lookup	1.390	1.050	0.430	1.350	1.180	0.920	1.880	2.040	0.730	0.600
	Traversal	13.950	10.040	5.250	3.860	4.720	2.870	3.050	2.410	2.640	2.480
3	Lookup	1.520	1.250	0.560	2.000	1.240	1.230	1.980	1.950	1.270	0.620
	Traversal	8.310	5.540	4.580	3.270	2.180	2.200	2.730	2.270	2.170	2.180
4	Lookup	2.150	1.770	0.480	2.200	0.950	1.200	2.900	1.280	0.490	0.450
	Traversal	8.300	6.070	7.030	4.820	4.430	5.970	4.400	4.550	10.950	14.420
5	Lookup	2.800	1.600	0.520	3.080	1.420	1.530	4.030	3.090	1.330	1.170
	Traversal	8.610	5.940	7.810	6.820	5.150	4.230	2.600	2.370	2.080	2.100

J.2.2 DBMS-2 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	9.310	1.130	0.180	0.980	1.070	0.970	1.240	1.240	0.210	0.200
	Traversal	12.330	1.350	1.340	1.350	2.610	1.370	1.370	1.370	1.380	1.370
2	Lookup	6.760	1.090	0.210	1.120	0.900	1.020	0.930	0.630	0.170	0.180
	Traversal	14.170	1.530	1.440	1.510	1.540	1.480	1.680	1.640	1.660	1.600
3	Lookup	10.520	1.340	0.190	1.160	1.030	0.930	1.190	1.030	0.240	0.190
	Traversal	17.850	1.340	1.380	1.330	1.350	1.320	1.310	1.320	1.380	1.340
4	Lookup	6.830	1.170	0.180	1.160	1.450	1.190	1.330	0.950	0.220	0.300
	Traversal	18.000	1.440	1.380	1.350	1.320	1.310	1.310	1.330	1.350	1.430
5	Lookup	6.120	1.380	0.250	1.360	0.980	0.830	0.940	1.000	0.170	0.170
	Traversal	15.920	1.370	1.400	1.410	1.380	1.370	1.370	1.370	1.370	1.410

J.2.3 DBMS-3 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	5.920	2.910	0.300	3.110	2.780	3.130	3.740	3.370	0.630	0.630
	Traversal	37.640	1.970	1.940	1.950	1.960	1.950	2.030	1.960	1.980	1.930
2	Lookup	6.390	3.550	0.320	4.770	3.180	3.180	4.210	3.980	0.700	0.630
	Traversal	31.570	2.030	2.020	1.970	2.030	1.990	1.990	2.020	1.960	1.990
3	Lookup	5.700	3.170	0.270	3.640	3.150	3.040	3.010	3.150	0.710	0.650
	Traversal	35.600	2.180	2.000	1.920	2.150	2.280	2.110	1.920	1.950	2.020
4	Lookup	4.900	3.520	0.300	3.680	4.160	4.180	3.080	3.130	0.660	0.650
	Traversal	33.860	2.240	2.350	2.090	1.980	1.910	1.930	1.950	1.930	1.940
5	Lookup	5.870	2.820	0.270	3.840	4.110	4.120	3.850	4.080	0.780	0.660
	Traversal	48.220	2.290	2.190	2.030	2.600	2.190	2.540	2.360	1.940	1.940

J.2.4 DBMS-1 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
1	1.300	1.314	9.010	7.099
2	1.390	1.131	13.950	4.147
3	1.520	1.344	8.310	3.013
4	2.150	1.302	8.300	6.960
5	2.800	1.974	8.610	4.344
Average:	1.832	1.413	9.636	5.113
Sample SD:	0.635	0.324	2.429	1.823

J.2.5 DBMS-2 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
1	9.310	0.802	12.330	1.501
2	6.760	0.694	14.170	1.564
3	10.520	0.811	17.850	1.341
4	6.830	0.883	18.000	1.358
5	6.120	0.787	15.920	1.383
Average:	7.908	0.795	15.654	1.429
Sample SD:	1.902	0.068	2.431	0.098

J.2.6 DBMS-3 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
1	5.920	2.289	37.640	1.963
2	6.390	2.724	31.570	2.000
3	5.700	2.310	35.600	2.059
4	4.900	2.596	33.860	2.036
5	5.870	2.726	48.220	2.231
Average:	5.756	2.529	37.378	2.058
Sample SD:	0.543	0.216	6.459	0.103

J.2.7 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	1.832	1.413	9.636	5.113
	Sample SD:	0.635	0.324	2.429	1.823
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	7.908	0.795	15.654	1.429
	Sample SD:	1.902	0.068	2.431	0.098
	t =	-6.776	4.167	-3.915	4.512

J.2.8 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	1.832	1.413	9.636	5.113
	Sample SD:	0.635	0.324	2.429	1.823
DBMS-3	N:	5.000	5.000	5.000	5.000
	Average:	5.756	2.529	37.378	2.058
	Sample SD:	0.543	0.216	6.459	0.103
	t =	-10.503	-6.401	-8.989	3.741

J.2.9 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	7.908	0.795	15.654	1.429
	Sample SD:	1.902	0.068	2.431	0.098
DBMS-3	N:	5.000	5.000	5.000	5.000
	Average:	5.756	2.529	37.378	2.058
	Sample SD:	0.543	0.216	6.459	0.103
	t =	2.433	-17.115	-7.038	-9.868

J.3 Large Database

J.3.1 DBMS-1 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	1.230	2.090	2.110	1.450	1.270	1.370	2.450	2.080	1.580	1.300
	Traversal	50.740	65.011	56.620	46.781	58.510	61.900	56.140	61.580	56.310	56.419
2	Lookup	2.080	1.530	1.370	1.480	1.250	1.540	1.560	1.190	1.130	1.050
	Traversal	52.109	46.800	45.880	47.741	45.250	45.050	45.530	48.650	44.360	45.620
3	Lookup	1.410	1.440	1.280	1.800	1.230	1.520	1.430	1.370	1.050	1.020
	Traversal	53.800	64.060	54.031	51.040	51.260	49.750	48.730	54.200	51.000	54.420
4	Lookup	1.990	1.610	1.770	1.900	1.600	2.170	2.400	2.000	1.660	1.450
	Traversal	53.951	64.520	55.911	64.150	65.700	70.479	78.651	75.850	56.830	49.350
5	Lookup	1.480	1.100	1.290	1.430	1.180	1.420	1.600	1.350	1.750	1.330
	Traversal	48.920	67.120	70.461	58.100	64.760	53.760	49.070	48.180	44.620	58.340

J.3.2 DBMS-2 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	9.170	2.140	1.840	2.510	7.180	2.240	2.530	2.520	1.260	1.390
	Traversal	197.710	108.910	139.479	146.091	131.030	113.290	124.941	149.910	114.229	141.131
2	Lookup	6.290	1.660	1.580	1.470	1.500	1.450	1.670	1.510	1.110	1.220
	Traversal	209.826	143.441	103.642	130.701	92.900	116.111	126.710	107.359	142.530	174.341
3	Lookup	6.320	2.090	1.610	1.410	1.950	6.400	1.390	1.500	1.060	1.070
	Traversal	198.679	134.613	147.014	113.492	114.110	133.244	111.161	107.572	113.129	115.680
4	Lookup	8.270	1.900	1.700	1.680	1.640	1.570	1.860	1.600	1.160	1.190
	Traversal	247.071	145.850	205.719	222.060	249.211	144.510	106.450	99.290	134.232	188.347
5	Lookup	6.770	1.930	1.910	1.540	1.520	1.350	1.990	8.590	1.670	1.090
	Traversal	208.982	130.809	136.939	114.999	139.202	140.111	139.701	99.651	107.979	149.032

J.3.3 DBMS-3 Raw Results

Benchmark	Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	Lookup	4.710	2.240	2.450	2.550	2.510	2.750	3.330	3.810	3.150	3.270
	Traversal	16305.694	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
2	Lookup	5.680	2.530	2.580	2.440	2.530	2.830	3.020	3.000	3.120	3.030
	Traversal	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
3	Lookup	5.130	2.490	2.590	2.460	2.550	2.770	3.070	3.040	3.020	3.140
	Traversal	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4	Lookup	4.470	2.210	2.380	2.600	2.510	3.170	3.040	2.860	3.070	2.910
	Traversal	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5	Lookup	6.940	6.450	6.530	4.190	7.650	3.350	3.710	3.600	3.730	3.320
	Traversal	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

DBMS-3 failed most of the **Traversal** tests.

J.3.4 DBMS-1 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
1	1.230	1.744	50.740	57.697
2	2.080	1.344	52.109	46.098
3	1.410	1.349	53.800	53.166
4	1.990	1.840	53.951	64.605
5	1.480	1.383	48.920	57.157
Average:	1.638	1.532	51.904	55.745
Sample SD:	0.375	0.240	2.126	6.783

J.3.5 DBMS-2 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
1	9.170	2.623	197.710	129.890
2	6.290	1.463	209.826	126.415
3	6.320	2.053	198.679	121.113
4	8.270	1.589	247.071	166.185
5	6.770	2.399	208.982	128.714
Average:	7.364	2.025	212.454	134.463
Sample SD:	1.292	0.501	20.152	18.050

J.3.6 DBMS-3 Average and Sample Standard Deviation

Benchmark	Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
1	4.710	2.895	16305.694	N/A
2	5.680	2.787	N/A	N/A
3	5.130	2.792	N/A	N/A
4	4.470	2.750	N/A	N/A
5	6.940	4.726	N/A	N/A
Average:	5.386	3.190	16305.694	N/A
Sample SD:	0.983	0.860	N/A	N/A

DBMS-3 failed most of the **Traversal** tests.

J.3.7 DBMS-1 vs. DBMS-2 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	1.638	1.532	51.904	55.745
	Sample SD:	0.375	0.240	2.126	6.783
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	7.364	2.025	212.454	134.463
	Sample SD:	1.292	0.501	20.152	18.050
	t =	-9.516	-1.986	-17.716	-9.129

J.3.8 DBMS-1 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
DBMS-1	N:	5.000	5.000	5.000	5.000
	Average:	1.638	1.532	51.904	55.745
	Sample SD:	0.375	0.240	2.126	6.783
DBMS-3	N:	5.000	5.000	1.000	N/A
	Average:	5.386	3.190	16305.694	N/A
	Sample SD:	0.983	0.860	N/A	N/A
	t =	-7.968	-4.150	N/A	N/A

DBMS-3 failed most of the **Traversal** tests, so the value of *t* cannot be calculated.

J.3.9 DBMS-2 vs. DBMS-3 Small Sample Test

DBMS		Lookup Cold	Lookup Warm	Traversal Cold	Traversal Warm
DBMS-2	N:	5.000	5.000	5.000	5.000
	Average:	7.364	2.025	212.454	134.463
	Sample SD:	1.292	0.501	20.152	18.050
DBMS-3	N:	5.000	5.000	1.000	N/A
	Average:	5.386	3.190	16305.694	N/A
	Sample SD:	0.983	0.860	N/A	N/A
	t =	2.725	-2.616	N/A	N/A

DBMS-3 failed most of the **Traversal** tests, so the value of *t* cannot be calculated.

References

[Adiga93a] S. Adiga (1993) Object-oriented software: relevance to manufacturing. In: *Object-Oriented Software for Manufacturing Systems*, S. Adiga (ed.) (London: Chapman & Hall)

[Adiga93b] S. Adiga and P. Cogez (1993) Towards an object-oriented architecture for CIM systems. In: *Object-Oriented Software for Manufacturing Systems*, S. Adiga (ed.) (London: Chapman & Hall)

[Adiga93c] S. Adiga and M. Gadre (1993) Prototyping object systems and reusable object libraries. In: *Object-Oriented Software for Manufacturing Systems*, S. Adiga (ed.) (London: Chapman & Hall)

[Adiga93d] S. Adiga and J. Kolyer (1993) Object-oriented databases. In: *Object-Oriented Software for Manufacturing Systems*, S. Adiga (ed.) (London: Chapman & Hall)

[Ahad88] R. Ahad (1988) The object shell: an extensible system to define an object-oriented view of an existing database. *Proceedings of the Second International Workshop on Object-Oriented Database Systems.* Lecture Notes in Computer Science, 334, pp. 174-192 (Berlin: Springer-Verlag)

[Ahad92] R. Ahad and D. Dedo (1992) OpenODB from Hewlett-Packard: a commercial object-oriented database management system. *Journal of Object-Oriented Programming*. 4 (9):31-35.

[Ahmed92] S. Ahmed, A. Wong, D. Sriram and R. Logcher (1992) Object-oriented database management systems for engineering: a comparison. *Journal of Object-Oriented Programming*. 5 (3):27-44.

[Alagi97a] S. Alagic (1997) O_2 and the ODMG standard: do they match? (Position Paper). *OOPSLA '97 Workshop on Experiences Using Object Data Management in the Real-World*, Atlanta, Georgia, October 1997.

[Alagi97b] S. Alagic (1997) The ODMG object model: does it make sense? *Proceedings of the ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '97)*, Atlanta, Georgia, 1997, pp. 253-270.

[Alagi98] S. Alagic (1998) Flight-simulator database: object-oriented design and implementation. In: *Object Databases in Practice*, M.E.S. Loomis and A.B. Chaudhri (eds.) (Upper Saddle River, New Jersey: Prentice-Hall)

[Alban86] A. Albano, G. Ghelli, M.E. Occhiuto and R. Orsini (1986) A strongly typed, interactive object-oriented database programming language. *Proceedings of the International Workshop on Object-Oriented Database Systems,* Asilomar, California, 1986, pp. 94-103.

[Alfre94] C. Alfred (1994) Maximizing leverage from an object database. *IBM Systems Journal*. 33 (2):280-299.

[Ander90] T.L. Anderson, A.J. Berre, M. Mallison, H.H. Porter and B. Schneider (1990) The HyperModel benchmark. *Proceedings of the Second International Conference on Extending Database Technology (EDBT '90)*, Lecture Notes in Computer Science, 416, pp. 317-331 (Berlin: Springer-Verlag)

[Andre90] T. Andrews (1990) The Vbase object database environment. In: *Research Foundations in Object-Oriented and Semantic Database Systems*, A.F. Cárdenas and D. McLeod (eds.) (Englewood Cliffs, New Jersey: Prentice-Hall)

[Anon85] Anon et al. (1985) A measure of transaction processing power. *Datamation*. 31 (7):112-118.

[Arctu95] D. Arctur, E. Anwar, J. Alexander, S. Chakravarthy, M. Chung, M. Cobb and K. Shaw (1995) Comparison and benchmarks for import of VPF geographic data from object-oriented and relational database files. *Proceedings of the Fourth International Symposium on Large Spatial Databases (SSD '95)*, Lecture Notes in Computer Science, 951, pp. 368-384 (Berlin: Springer-Verlag)

[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.

[Atkin83] M. P. Atkinson, P.J. Bailey, K.J. Chisholm, W.P. Cockshott and R. Morrison (1983) An approach to persistent programming. *Computer Journal*. 26 (4):360-365.

[Atkin89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik (1989) The object-oriented database system manifesto. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989, pp. 223-240.

[Atwoo90] T. Atwood (1990) Applying the object paradigm to databases. *Computer Language*. 7 (9):36-42.

[Atwoo91] T.M. Atwood (1991) The case for object-oriented databases. *IEEE SPECTRUM*. 28 (2):44-47.

[Atwoo92] T. Atwood (1992) An introduction to ODBMSs. In: *JOOP Focus on ODBMS* (New York: SIGS Publications)

[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.

[Banci91] F. Bancilhon, C. Delobel and P. Kanellakis (eds.) (1991) Building an object-oriented database system: the story of O_2 (San Mateo, California: Morgan-Kaufmann)

[Barry91] D.K. Barry (1991) Perspectives on changes for ODBMSs. *Journal of Object-Oriented Programming*. 4 (4):19-20.

[Barry94] D. Barry (1994) Should you take the plunge? *Object Magazine*. 3 (6):24-27.

[Barry97] D.K. Barry (1997) Just the facts, please. *Distributed Object Computing*. 1 (1):56-57, 59.

[Bator86] D.S. Batory (1986) GENESIS: a project to develop an extensible database management system. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 207-208.

[Beech88] D. Beech (1988) A foundation for evolution from relational to object databases. *Proceedings of the First International Conference on Extending Database Technology (EDBT '88)*, Lecture Notes in Computer Science, 303, pp. 251-270 (Berlin: Springer-Verlag)

[Beech90] D. Beech and Ç. Özbütün (1990) Object databases as generalizations of relational databases. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 119-135.

[Bell84] D.A. Bell (ed.) (1984) Database performance (Maidenhead: Pergamon)

[Benba87] I. Benbasat, D.K. Goldstein and M. Mead (1987) The case research strategy in studies of information systems. *MIS Quarterly*. 11 (3):369-386.

[Berg96] C.A. van den Berg and A. van der Hoeven (1996) Monet meets OO7. *Object-Oriented Database Systems Symposium*, Montpellier, France, July 1996.

[Berre88] A.J. Berre, T.L. Anderson and M. Mallison (1988) The HyperModel benchmark. Technical Report No. CS/E 88-031, Oregon Graduate Center, Beaverton, Oregon, 1988.

[Berre91] A.J. Berre and T.L. Anderson (1991) The HyperModel benchmark for evaluating object-oriented databases. In: *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*, R. Gupta and E. Horowitz (eds.) (Englewood Cliffs, New Jersey: Prentice-Hall)

[Bitto83] D. Bitton, D. DeWitt and C. Turbyfill (1983) Benchmarking database systems: a systematic approach. *Proceedings of the Ninth Very Large Data Bases Conference*, Florence, Italy, 1983, pp. 8-19.

[Bjørn97] D. Bjørner (1997) Domains as a prerequisite for requirements and software: domain perspectives & facets, requirements aspects and software views (Position Paper). *Workshop on Requirements Targeted Software and Systems Engineering*, Bernried, Bavaria, Germany, October 1997.

[Bodne94] D.A. Bodner, S. Narayanan, U. Sreekanth, T. Govindaraj, L.F. McGinnis and C.M. Mitchell (1994) Analysis of discrete manufacturing systems for developing object-oriented simulation models. *Proceedings of the Third Industrial Engineering Research Conference*, Atlanta, Georgia, May 1994, pp. 154-159.

[Boncz96a] P.A. Boncz, F. Kwakkel and M.L. Kersten (1996) High performance support for oo traversals in Monet. *Proceedings of the Fourteenth British National Conference on Databases (BNCOD '96)*, Lecture Notes in Computer Science, 1094, pp. 152-169 (Berlin: Springer-Verlag)

[Boncz96b] P.A. Boncz, W. Quak and M.L. Kersten (1996) Monet and its geographic extension: a novel approach to high performance GIS processing. *Proceedings of the International Conference on Extending Database Technology (EDBT '96)*, Lecture Notes in Computer Science, 1057, pp. 147-166 (Berlin: Springer-Verlag)

[Bonne95a] A.J. Bonner, A. Shrufi and S. Rozen (1995) Benchmarking object-oriented DBMSs for workflow management (Position Paper). *OOPSLA '95 Workshop on Object Database Behavior, Benchmarks, and Performance*, Austin, Texas, October 1995.

[Bonne95b] A.J. Bonner, A. Shrufi and S. Rozen (1995) LabFlow-1: a database benchmark for high-throughput workflow management. Technical Report, CSRI, University of Toronto, 28 August 1995.

[Bonne96a] A.J. Bonner, A. Shrufi and S. Rozen (1996) Database requirements for workflow management in a high-throughput genome laboratory (Position Paper). *NSF Workshop on Workflow and Process Automation in Information Systems*, Athens, Georgia, May 1996.

[Bonne96b] A.J. Bonner, A. Shrufi and S. Rozen (1996) LabFlow-1: a database benchmark for high-throughput workflow management. *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT '96)*, Lecture Notes in Computer Science, 1057, pp. 463-478 (Berlin: Springer-Verlag)

[Boral84] H. Boral and D.J. DeWitt (1984) A methodology for database system performance evaluation. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Boston, Massachusetts, 1984, pp. 176-185.

[Bradl94] G.A. Bradley (1994) Personal communication, 18 March 1994.

[Bretl89] R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E.H. Williams and M. Williams (1989) The GemStone data management system. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Brodi89] M.L. Brodie, F. Bancilhon, C. Harris, M. Kifer, Y. Masunaga, E.D. Sacerdoti and K. Tanaka (1989) Next generation database management systems technology. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989, pp. 335-346.

[Brodi93] M.L. Brodie and M. Stonebraker (1993) DARWIN: on the incremental migration of legacy information systems. Technical Report No. TR-0222-10-92-165, GTE Laboratories, Inc., Waltham, Massachusetts, March 1993.

[Brown91] A.W. Brown (1991) Object-oriented databases: their applications to software engineering (New York: McGraw-Hill)

[Brown96] P. Brown (1996) Personal communication, 15 May 1996.

[Burle93] D. K. Burleson (1993) Practical applications of object-oriented techniques for relational databases (New York: John Wiley & Sons/QED)

[Butte91a] P. Butterworth (1991) ODBMSs as database managers, part 3. *Journal of Object-Oriented Programming*. 4 (4):55-57.

[Butte91b] P. Butterworth, A. Otis and J. Stein (1991) The GemStone object database management system. *Communications of the ACM*. 34 (10):64-77.

[Cairn91] T. Cairns, H. Timimi, M. Thick and G. Gold (1991) A generic model of clinical practice: the COSMOS project. *Proceedings of Medical Informatics Europe (MIE '91)*, Lecture Notes in Medical Informatics, 45, pp. 706-710 (Berlin: Springer-Verlag)

[Cairn92a] T. Cairns and M. Fowler (1992) A case study of domain analysis: health care (Tutorial). *ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '92)*, Vancouver, Canada, October 1992.

[Cairn92b] T. Cairns, A. Casey, M. Fowler, M. Thursz and H. Timimi (1992) The cosmos clinical process model. Information Management Centre, National Health Service, Birmingham, UK, 1 December 1992.

[Carey86] M.J. Carey, D.J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J.E. Richardson and E.J. Shekita (1986) The architecture of the EXODUS extensible DBMS. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 52-65.

[Carey89] M.J. Carey, D.J. DeWitt, J.E. Richardson and E.J. Shekita (1989) Storage management for objects in EXODUS. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Carey93] M.J. Carey, D.J. DeWitt and J.F. Naughton (1993) The OO7 benchmark. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington DC, 1993, pp. 12-21.

[Carey94] M.J. Carey, D.J. DeWitt, C. Kant and J.F. Naughton (1994) A status report on the OO7 OODBMS benchmarking effort. *Proceedings of the ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '94)*, Portland, Oregon, 1994, pp. 414-426.

[Catte88] R.G.G. Cattell (1988) Object-oriented DBMS performance measurement. *Proceedings of the Second International Workshop on Object-Oriented Database Systems*. Lecture Notes in Computer Science, 334, pp. 364-367 (Berlin: Springer-Verlag)

[Catte91a] 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)

[Catte91b] R.G.G. Cattell (1991) What are next-generation database systems? *Communications of the ACM*. 34 (10):31-33.

[Catte92] R.G.G. Cattell and J. Skeen (1992) Object operations benchmark. *ACM Transactions on Database Systems*. 17 (1):1-31.

[Catte94a] R.G.G. Cattell (1994) Object data management: object-oriented and extended relational database systems (Reading, Massachusetts: Addison-Wesley)

[Catte94b] R.G.G. Cattell (1994) Personal communication, 10 March 1994.

[CERN96] CERN (1996) Object databases and mass storage systems: the prognosis. Report No. CERN/LHCC 96-17 (LCRB/RD45), 8 February 1996.

[Chand93a] R. Chandra and A. Segev (1993) Active databases for financial applications. Technical Report No. 34422, Information and Computing Sciences Division, Lawrence Berkeley Laboratory, California, 1993.

[Chand93b] R. Chandra and A. Segev (1993) Performance optimization of financial database applications. *Proceedings of the Third Workshop on Information Technologies and Systems (WITS '93)*, December 1993.

[Chand94] R. Chandra and A. Segev (1994) Using next generation databases to develop financial applications. *Proceedings of the First International Conference on Applications of Databases (ADB '94)*, Lecture Notes in Computer Science, 819, pp. 190-203 (Berlin: Springer-Verlag)

[Chaud93] A.B. Chaudhri (1993) Object database management systems: an overview. *BCS OOPS Newsletter*. No. 18 (Summer 93), pp. 6-15.

[Chaud94a] A.B. Chaudhri and N. Revell (1994) Application-specific benchmarks for object databases: a multiple-case study approach. *Proceedings of the 1994 International Conference on Object-Oriented Information Systems (OOIS '94)*, London, UK, 1994, pp. 500-503.

[Chaud94b] A.B. Chaudhri and N. Revell (1994) Object database benchmarks: past, present & future. *Proceedings of the Seminar on Object-Oriented Databases: Realising their Potential and Interoperability with RDBMS*, London, UK, 1994, pp. 166-183.

[Chaud95a] A.B. Chaudhri (1995) An annotated bibliography of benchmarks for object databases. *SIGMOD Record*. 24 (1):50-57.

[Chaud95b] A.B. Chaudhri (1995) How to evaluate ODBMSs using benchmarks. *Proceedings of the Seminar on Object Databases*, London, UK, 1995, pp. 12-29.

[Chaud96a] A.B. Chaudhri and P. Osmon (1996) Databases for a new generation. *Object Expert*. 3 (1):26-31.

[Chaud96b] A.B. Chaudhri (1996) Object DBMSs: to benchmark or not to benchmark? *Object Magazine*. 6 (4):76-80.

[Chaud97] A.B. Chaudhri (1997) Tuning object database applications. *Object Expert*. 2 (2):50-53.

[Chaud98a] A.B. Chaudhri (1998) Workshop report on experiences using object data management in the real-world. *SIGMOD Record*. 27 (1):5-10.

[Chaud98b] A.B. Chaudhri (1998) Workshop report on experiences using object data management in the real-world (Expanded Version). Expanded version of [Chaud98a].

[Cheun92] N.T. Cheung (1992) Laying the foundations for a comprehensive computerized medical record system. MSc Thesis, Department of Computing, Imperial College of Science, Technology and Medicine, London, UK, September 1992.

[Clark96] M. Clarke (1996) ODBMS evaluation: terms of reference. Prepared for Reuters by Admiral Management Services Ltd., Camberley, 23 September 1996.

[Clark97] M. Clarke (1997) ODBMS evaluation: report. Prepared for Reuters by Admiral Management Services Ltd., Camberley, 14 January 1997.

[Commi90] Committee for Advanced DBMS Function (1990) Third-generation database system manifesto. *SIGMOD Record.* 19 (3):31-44.

[Coope96] S. Cooper (1996) Personal communication, 9 May 1996.

[Coulo94] G. Coulouris, J. Dollimore and T. Kindberg (1994) Distributed systems: concepts and design (Wokingham, England: Addison-Wesley)

[Darni93] V. Darnis (1993) OO7: un banc d'essais réaliste pour les performances des SGBD objet. *Génie Logiciel & Systèmes Experts.* No. 31 (June 1993), pp. 32-34 (in French).

[Date90] C.J. Date (1990) An introduction to database systems, volume 1 (Reading, Massachusetts: Addison-Wesley)

[David93] B. David, L. Raynal, G. Schorter and V. Mansart (1993) GeO_2: why objects in a geographical DBMS? *Proceedings of the Third International Symposium on Large Spatial Databases (SSD '93)*, Lecture Notes in Computer Science, 692, pp. 264-276 (Berlin: Springer-Verlag)

[Dawso89] J. Dawson (1989) A family of models. *BYTE.* 14 (9):277-286.

[DBMS94] DBMS (1994) A new direction in DBMS. *DBMS.* 7 (2):50-60.

[DBWor97] DBWorld (1997) Report and directory 1997/8 (London: Interactive Information Services Ltd.)

[DeSme94] P. De Smedt, J. Annevelink, T. Pham and P. Strong (1994) A physician's workstation as an application of object-oriented database technology in healthcare. *Proceedings of the First International Conference on Applications of Databases (ADB '94)*, Lecture Notes in Computer Science, 819, pp. 234-252 (Berlin: Springer-Verlag)

[Deux90] O. Deux et al. (1990) The story of O_2. *IEEE Transactions on Knowledge and Data Engineering.* 2 (1):91-108.

[Deux91] O. Deux et al. (1991) The O_2 system. *Communications of the ACM.* 34 (10):34-48.

[Dewal90] S. Dewal, H. Hormann, U. Kelter, D. Platz, M. Roschewski and L. Schöpe (1990) Evaluation of object management systems. Internal SWT Memo No. 44, University of Dortmund, Germany, September 1990.

[Dewal92] S. Dewal, W. Emmerich and K. Lichtinghagen (1992) A decision support method for the selection of OMSs. *Proceedings of the Second International Conference on Systems Integration*, Morristown, New Jersey, 1992, pp. 32-40.

[Dewan97] H. Dewan and S. Agarwal (1997) OO-Fin benchmark: a scalability and performance benchmark for object-relational trading systems. White Paper, Persistence Software, Inc., 2 April 1997.

[DeWit85] D. DeWitt (1985) Benchmarking database systems: past efforts and future directions. *IEEE Database Engineering*. 8 (1):2-9.

[DeWit90] D. DeWitt, P. Futtersack, D. Maier and F. Velez (1990) A study of three alternative workstation-server architectures for object oriented database systems. *Proceedings of the Sixteenth Very Large Data Bases Conference*, Brisbane, Australia, 1990, pp. 107-121.

[DeWit91] D.J. DeWitt (1991) The Wisconsin benchmark: past, present, and future. In: *The Benchmark Handbook for Database and Transaction Processing Systems*, J. Gray (ed.) (San Mateo, California: Morgan-Kaufmann)

[Dick95a] K. Dick and A. Swett (1995) Objectivity/DB. *Object Magazine*. 5 (5):82-84, 95.

[Dick95b] K. Dick and A.B. Chaudhri (1995) Selecting an ODBMS: practical advice for evaluators (Tutorial). *ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '95)*, Austin, Texas, October 1995.

[Dick96] K. Dick and A. Swett (1996) ObjectStore C++. *Object Magazine*. 5 (9):86-89.

[Dietr92] S.W. Dietrich, M. Brown, E. Cortes-Rello and S. Wunderlin (1992) A practitioner's introduction to database performance benchmarks and measurements. *Computer Journal*. 35 (4):322-331.

[Dittr86] K.R. Dittrich (1986) Object-oriented database systems: the notion and the issues. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Pacific Grove, California, 1986, pp. 2-4.

[Donga93] J.J. Dongarra and W. Gentzsch (eds.) (1993) Computer benchmarks (Amsterdam: North-Holland)

[Duhl88] J. Duhl and C. Damon (1988) A performance comparison of object and relational databases using the Sun benchmark. *Proceedings of the ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '88)*, San Diego, California, 1988, pp. 153-163.

[Edels91] H.A. Edelstein (1991) Database world targets next-generation problems. *Software Magazine*. 11 (6):79-86.

[Emmer92] W. Emmerich and M. Kampmann (1992) The Merlin OMS benchmark: definition, implementation and results. Internal Memo No. 65, Department of Computer Science, University of Dortmund, Germany, October 1992, revised 22 July 1993.

[Emmer93] W. Emmerich and W. Schäfer (1993) Dedicated object management system benchmarks for software engineering applications. *Proceedings of the IEEE Conference on Software Engineering Environments*, Reading, UK, 1993, pp. 130-142.

[Engli92] L.P. English (1992) Object databases at work. *DBMS*. 5 (11):44-58.

[EOS95a] EOS (1995) Multi-mission inventory system (MMIS). Architectural Design Document, Document No. EOS-95/122-ADD-001, EOS Ltd., Farnham, 24 May 1995.

[EOS95b] EOS (1995) Multi-mission inventory system (MMIS). Detailed Design Document, Document No. EOS-95/122-DDD-001, EOS Ltd., Farnham, May 1995.

[EOS95c] EOS (1995) Multi-mission inventory system (MMIS). Performance Tuning, Document No. EOS-95/122-TN-007, EOS Ltd., Farnham, 5 June 1995.

[EOS95d] EOS (1995) Multi-mission inventory system (MMIS). Software User Manual, Volume 3, MMIS Operations and Maintenance, Document No. EOS-95/122-SUM-003, EOS Ltd., Farnham, 16 August 1995.

[EOS95e] EOS (1995) Multi-mission inventory system (MMIS). Version 1 Development Final Report, Document No. EOS-95/122-RP-001, EOS Ltd., Farnham, 5 October 1995.

[EOS96] EOS (1996) OS database technology: evaluation of database options for Ordnance Survey. Document No. EOS-170-RP-003, Issue No. 1.1, EOS Ltd., Farnham, August 1996.

[Evere92] G.C. Everest and M.S. Hanna (1992) Survey of object-oriented database management systems. Technical Report, Carlson School of Management, University of Minnesota, 1992.

[Fafch91] D. Fafchamps (1991) Ethnographic workflow analysis: specifications for design. *Proceedings of the Fourth International Conference on Human-Computer Interaction*, Stuttgart, Germany, 1991, pp. 709-715.

[Finkl93] R. Finkelstein (1993) Breaking the mold. *Database Programming & Design*. 6 (2):49-53.

[Fishm89] D. H. Fishman, D. Beech, J. Annevelink, E. Chow, T. Connors, J.W. Davis, W. Hasan, C.G. Hoch, W. Kent, S. Leichner, P. Lyngbæk, B. Mahbod, M.A. Neimat, T. Risch, M.C. Shan and W.K. Wilkinson (1989) Overview of the Iris DBMS. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Fowle91] M. Fowler (1991) The use of object-oriented analysis in medical informatics for large integrated systems. *Proceedings of the Conference on the Technology of Object-Oriented Languages and Systems (TOOLS)*, Paris, France, 1991, pp. 203-214.

[Fowle93] M. Fowler, T. Cairns and M. Thursz (1993) Prototyping a health care enterprise model with an ODBMS. *Proceedings of the Seminar on Object-Oriented Data Management: Tools for Creating New Generations of IT Applications*, London, UK, 1993, pp. 156-165.

[Fowle94] M. Fowler (1994) Application views: another technique in the analysis and design armoury. *Journal of Object-Oriented Programming*. 7 (1):59-66.

[Fowle95] M. Fowler, T. Cairns and M. Thursz (1995) Observations and measurements. *Report on Object Analysis and Design*. 2 (3):20-24, 37.

[Frenc90a] J.C. French, A.K. Jones and J.L. Pfaltz (1990) Scientific database management (Final Report). Technical Report No. 90-21, Department of Computer Science, University of Virginia, August 1990.

[Frenc90b] J.C. French, A.K. Jones and J.L. Pfaltz (1990) Scientific database management (Panel Reports and Supporting Material). Technical Report No. 90-22, Department of Computer Science, University of Virginia, August 1990.

[Fritc90] B.L. Fritchman, R.L. Guck, D. Jagannathan, J.P. Thompson and D.M. Tolbert (1990) SIM: design and implementation of a semantic database system. In: *Research Foundations in Object-Oriented and Semantic Database Systems*, A.F. Cárdenas and D. McLeod (eds.) (Englewood Cliffs, New Jersey: Prentice-Hall)

[Galli91] R.D. Galliers (1991) Choosing appropriate information systems research approaches: a revised taxonomy. In: *Information Systems Research: Contemporary Approaches and Emergent Traditions*, H.-E. Nissen, H.K. Klein and R. Hirschheim (eds.) (Amsterdam: North-Holland)

[Geppe94] A. Geppert, S. Gatziu and K.R. Dittrich (1994) Performance evaluation of an active database management system: OO7 meets the BEAST. Technical Report No. IFI-94-18, Computer Science Department, University of Zurich, Switzerland, November 1994.

[Gerlh92] C. Gerlhof, A. Kemper, C. Kilger and G. Moerkotte (1992) Clustering in object bases. Technical Report No. 06/92, Faculty of Informatics, University of Karlsruhe, Germany, June 1992.

[Ghand93] S. Ghandeharizadeh, V. Choi and G. Bock (1993) Benchmarking object-based constructs. *Proceedings of the Eighth Brazilian Symposium on Databases*, Campina Grande, Brazil, 1993, pp. 207-221.

[Giffe94] R. Giffen (1994) Personal communication, 30 August 1994.

[GiST97] GiST (1997) Source code for the generalised search tree program. Available at http://gist.cs.berkeley.edu/.

[Gold92] G. Gold and M. Thick (1992) Systems evolution: the challenge of real time. *Proceedings of Systems Science in Health Care*, Prague, Czechoslovakia, 1992.

[Gray91] J. Gray (ed.) (1991) The benchmark handbook for database and transaction processing systems (San Mateo, California: Morgan-Kaufmann)

[Gray92] P.M.D. Gray, K.G. Kulkarni and N.W. Paton (1992) Object-oriented databases: a semantic data model approach (New York: Prentice-Hall)

[Haas90] L.M. Haas, W. Chang, G.M. Lohman, J. McPherson, P.F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M.J. Carey and E. Shekita (1990) Starburst mid-flight: as the dust clears. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):143-160.

[Haber95] J. Haberfield and K. Dittrich (1995) Object-oriented databases (London: UNICOM)

[Hales89] K. Hales and C. Guilfoyle (1989) The future of the database (London: Ovum)

[Hallo93a] T.J. Halloran (1993) Performance measurement of three commercial object-oriented database management systems. Master's Thesis, AFIT/GCS/ENG/93D-12, US Air Force Institute of Technology, December 1993.

[Hallo93b] T.J. Halloran (1993) Source code for the OO1 benchmark and the AFIT simulation benchmark. Technical Report No. AFIT/EN-TR-93-09, US Air Force Institute of Technology (AETC), 5 November 1993.

[Hallo93c] T.J. Halloran and M.A. Roth (1993) "Magic mirror on the wall, who's the fastest database of them all?": a survey of database benchmarks. Technical Report No. AFIT/EN-TR-93-05, US Air Force Institute of Technology (AETC), 21 June 1993.

[Hamme81] M. Hammer and D. McLeod (1981) Database description with SDM: a semantic data model. *ACM Transactions on Database Systems*. 6 (3):351-386.

[Harru91] G. Harrus, V. Benzaken and C. Delobel (1991) Measuring performance of clustering strategies: the CluB-0 benchmark. Technical Report No. 66-91, Altaïr-INRIA, Le Chesnay Cedex, France, January 1991.

[Hawth85] P. Hawthorn (1985) Variations on a benchmark. *IEEE Database Engineering*. 8 (1):19-28.

[Hazza90] A. Hazzah (1990) Objects are taking shape in flat relational world. *Software Magazine*. 10 (7):32-42.

[Hodge89] P. Hodges (1989) A relational successor? *Datamation*. 35 (21):47-50.

[Hohen97a] U. Hohenstein, V. Pleßer and R. Heller (1997) Evaluating the performance of object-oriented database systems by means of a concrete application. *DEXA '97 Workshop on Object-Oriented Database Systems*, Toulouse Cedex, France, September 1997.

[Hohen97b] U. Hohenstein, V. Pleßer and R. Heller (1997) Evaluating the performance of object-oriented database systems by means of a concrete application. Expanded version of [Hohen97a].

[Horma90] H. Hormann, D. Platz, M. Roschewski and L. Schöpe (1990) The HyperModel benchmark: description, execution and results. Internal SWT Memo No. 53, University of Dortmund, Germany, September 1990.

[Hughe91] J.G. Hughes (1991) Object-oriented databases (New York: Prentice-Hall)

[Hurso93] A.R. Hurson, S.H. Pakzad and J.-b. Cheng (1993) Object-oriented database management systems: evolution and performance issues. *IEEE Computer*. 26 (2):48-60.

[IDC94] IDC (1994) Object technologies: object-oriented database markets continue rapid growth in 1993. Report #8790, International Data Corporation, Framingham, Massachusetts, April 1994.

[Inmon89] W.H. Inmon (1989) Benchmarking the benchmarks. *Database Programming & Design*. 2 (8):54-59.

[Innoc96] V. Innocente (1996) A test of Objectivity to store L3 mini-DST (DVN). *Objectivity Workshop*, CERN, Switzerland, 26 February - 1 March 1996.

[Jain91] R. Jain (1991) The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling (New York: John Wiley & Sons)

[Jeffc89] J. Jeffcoate, K. Hales and V. Downes (1989) Object-oriented systems: the commercial benefits (London: Ovum)

[Jeffc91] J. Jeffcoate and C. Guilfoyle (1991) Databases for objects: the market opportunity (London: Ovum)

[Josep89] J. Joseph, S. Thatte, C. Thompson and D. Wells (1989) Report on the object-oriented database workshop. *SIGMOD Record.* 18 (3):78-101.

[Josep91] J.V. Joseph, S.M. Thatte, C.W. Thompson and D.L. Wells (1991) Object-oriented databases: design and implementation. *Proceedings of the IEEE.* 79 (1):42-64.

[Jun97] W. Jun and Le Gruenwald (1997) Experiences using the OO7 benchmark for concurrency control technique performance evaluations (Position Paper). *OOPSLA '97 Workshop on Experiences Using Object Data Management in the Real-World,* Atlanta, Georgia, October 1997.

[Kelle93] A.M. Keller, R. Jensen and S. Agarwal (1993) Persistence software: bridging object-oriented programming and relational databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data,* Washington DC, 1993, pp. 523-528.

[Kelte89] U. Kelter (1989) The Simple benchmark Technical Report No. UNIDO/UK/89/02, Department of Computer Science, University of Dortmund, Germany, 11 September 1989.

[Kempe90] J. Kemper and J. Chriesten (1990) A benchmark to scale behaviorally object-oriented database systems. *Proceedings of the Fifth Jerusalem Conference on Information Technology,* Jerusalem, 1990, pp. 677-687.

[Kempe95a] J. Kempe, W. Kowarschick, W. Kießling, R. Hitzelberger and F. Dutkowski (1995) Benchmarking object-oriented database systems for CAD. *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA '95),* Lecture Notes in Computer Science, 978, pp. 167-176 (Berlin: Springer-Verlag)

[Kempe95b] J. Kempe, W. Kowarschick, W. Kießling, R. Hitzelberger and F. Dutkowski (1995) The OCAD benchmark for object-oriented database systems. FORWISS Report FR-1995-004, Bayerisches Forschungszentrum für Wissensbasierte Systeme (FORWISS), Munich, Germany, 1995.

[Ketab90] M.A. Ketabchi, S. Mathur, T. Risch and J. Chen (1990) Comparative analysis of RDBMS and OODBMS: a case study. *Proceedings of COMPCON IEEE Computer Society International Conference,* San Francisco, California, 1990, pp. 528-537.

[Khosh90] S. Khoshafian and R. Abnous (1990) Object orientation: concepts, languages, databases, user interfaces (New York: John Wiley & Sons)

[Khosh92a] S. Khoshafian, B. Baker, R. Abnous and K. Shepherd (1992) Intelligent offices: object-oriented multi-media information management in client/server architectures (New York: John Wiley & Sons)

[Khosh92b] S. Khoshafian, A. Chan, A. Wong and H.K.T. Wong (1992) A guide to developing client/server SQL applications (San Mateo, California: Morgan-Kaufmann)

[Kiese92] N. Kiesel, A. Schuerr and B. Westfechtel (1992) Design and evaluation of GRAS, a graph-oriented database system for engineering applications. Technical Report No. 92-44, Technical University of Aachen (RWTH Aachen), Germany, 1992.

[Kim89] W. Kim, N. Ballou, H.-T. Chou, J.F. Garza and D. Woelk (1989) Features of the ORION object-oriented database system. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Kim90a] W. Kim (1990) Introduction to object-oriented databases (Cambridge, Massachusetts: MIT Press)

[Kim90b] W. Kim (1990) Object-oriented databases: definition and research directions. *IEEE Transactions on Knowledge and Data Engineering*. 2 (3):327-341.

[Kim90c] W. Kim, J.F. Garza, N. Ballou and D. Woelk (1990) Architecture of the ORION next-generation database system. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):109-124.

[Kim91] W. Kim (1991) Object-oriented database systems: strengths and weaknesses. *Journal of Object-Oriented Programming*. 4 (4):21-29.

[Kim92] W. Kim (1992) On object-oriented database technology. White Paper, UniSQL, Inc., 1992.

[Kim94a] W. Kim (1994) Observations on the ODMG-93 proposal for an object-oriented database language. *SIGMOD Record*. 23 (1):4-9.

[Kim94b] W. Kim and J.F. Garza (1994) Requirements for a performance benchmark for object-oriented database systems. In: *Modern Database Systems: The Object Model, Interoperability, and Beyond*, W. Kim (ed.) (New York: ACM Press & Reading Massachusetts: Addison-Wesley)

[King89] R. King (1989) My cat is object-oriented. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Knigh94a] C. Knight (1994) NBI treasury dealing system. Technical Overview, 1st Draft, Nomura Research International, London, 2 August 1994.

[Knigh94b] C. Knight (1994) Nomura treasury dealing system. Design Document, Nomura Research International, London, 8 August 1994.

[Lagun89] Laguna Beach Participants (1989) Future directions in DBMS research. *SIGMOD Record*. 18 (1):17-26.

[Lai91] K.-W.L. Lai and L. Guzenda (1991) How to benchmark an OODBMS. *Journal of Object-Oriented Programming*. 4 (4):12-15.

[Lakey87] B. Lakey, D. Maier and J. Stein (1987) Benchmarking methodology for object-oriented data management systems. Unpublished manuscript.

[Lakey89] B. Lakey (1989) Developing benchmarks for comparing relational and object-oriented database systems. Master's Thesis, Oregon Graduate Center, Beaverton, Oregon, 31 July 1989.

[Lamb91] C. Lamb, G. Landis, J. Orenstein and D. Weinreb (1991) The ObjectStore database system. *Communications of the ACM*. 34 (10):50-63.

[Larse92] A.B. Larsen (1992) A test evaluation procedure for object oriented and relational database management systems. Master's Thesis, Institute of Informatics, University of Oslo, Norway, 5 February 1992.

[Lathr87] R.H. Lathrop, T.A. Webster and T.F. Smith (1987) ARIADNE: pattern-directed inference and hierarchical abstraction in protein structure recognition. *Communications of the ACM*. 30 (11):909-921.

[Lauch92] S. Lauchlan (1992) Oracle goes towards objects. *Computing*. 26 March 1992, p. 1.

[Leach95] E. Leach (1995) Object technology in the UK. *Introduction to CORBA*. London, UK, September 1995.

[Litwi92] W. Litwin and T. Risch (1992) Main memory oriented optimization of oo queries using typed datalog with foreign predicates. *IEEE Transactions on Knowledge and Data Engineering*. 4 (6):517-528.

[Locke92] P.C. Lockemann (1992) Object-oriented databases and deductive databases: Systems without market? Market without systems? *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA '92)*, Valencia, Spain, 1992, pp. 1-7.

[Lohma91] G.M. Lohman, B. Lindsay, H. Pirahesh and K.B. Schiefer (1991) Extensions to Starburst: objects, types, functions, and rules. *Communications of the ACM*. 34 (10):94-109.

[Longb93] R. Longbottom (1993) Database system benchmarking and performance testing. In: *Computer Benchmarks*, J.J. Dongarra and W. Gentzsch (eds.) (Amsterdam: North-Holland)

[Loomi90] M.E.S. Loomis (1990) Database transactions. *Journal of Object-Oriented Programming*. 3 (3):54-61.

[Loomi92] M.E.S. Loomis (1992) Integrating objects with relational technology. In: *JOOP Focus on ODBMS* (New York: SIGS Publications)

[Loomi95] M.E.S. Loomis (1995) Object databases: the essentials (Reading, Massachusetts: Addison-Wesley)

[Loomi96] M.E.S. Loomis and A.B. Chaudhri (1996) The a to z guide to object data management (Tutorial). *ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '96)*, San Jose, California, October 1996.

[Loomi98] M.E.S. Loomis and A.B. Chaudhri (eds.) (1998) Object databases in practice (Upper Saddle River, New Jersey: Prentice-Hall)

[Lozie93] B. Lozier (1993) FlowStream: an object-oriented plant floor management system. In: *Object-Oriented Software for Manufacturing Systems*, S. Adiga (ed.) (London: Chapman & Hall)

[Maier89] D. Maier (1989) Making database systems fast enough for CAD applications. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Maier90] D. Maier, J. Stein, A. Otis and A. Purdy (1990) Development of an object-oriented DBMS. In: *Research Foundations in Object-Oriented and Semantic Database Systems*, A.F. Cárdenas and D. McLeod (eds.) (Englewood Cliffs, New Jersey: Prentice-Hall)

[Manol86] F. Manola and U. Dayal (1986) PDM: an object-oriented data model. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 18-25.

[Manol89] F. Manola (1989) An evaluation of object-oriented DBMS developments. Technical Report No. TR-0066-10-89-165, GTE Laboratories, Inc., Waltham, Massachusetts, 31 October 1989.

[Manol94] F. Manola (1994) An evaluation of object-oriented DBMS developments. Technical Report No. TR-0263-08-94-165, GTE Laboratories, Inc., Waltham, Massachusetts, 31 August 1994.

[Marti92] J. Martin and J. Odell (1992) Object-oriented analysis and design (Englewood Cliffs, New Jersey: Prentice-Hall)

[McCan92] J.A. McCann (1992) A fine grained database system performance model. DPhil Thesis, Faculty of Informatics, University of Ulster at Jordanstown, Northern Ireland, 1992.

[McCla91] J.T. McClave and P.G. Benson (1991) Statistics for business and economics (San Francisco, California: Dellen Publishing Company)

[McClu92] S. McClure (1992) Object technology: a key software technology for the '90s. White Paper, International Data Corporation, 1992.

[McFar93] G. McFarland and A. Rudmik (1993) Object-oriented database management systems: a critical review/technology assessment (Indialantic, Florida: Software Productivity Solutions)

[Moore96] A. Moorley (1996) Initial results on OSST performance. Internal Paper, Cumulus Systems Ltd., London, 8 January 1996.

[Nag95] B. Nag and Y. Zhao (1995) Implementing the OO7 benchmark on Persistence. CS 764 Course Project, University of Wisconsin-Madison, 9 February 1995.

[Naray92a] S. Narayanan, D.A. Bodner, C.M. Mitchell, L.F. McGinnis, T. Govindaraj and L.K. Platzman (1992) Object-oriented simulation to support modeling and control of automated manufacturing systems. *Proceedings of the Society for Computer Simulation Western Multiconference*, Newport Beach, California, January 1992, pp. 59-63.

[Naray92b] S. Narayanan, D.A. Bodner, U. Sreekanth, S.J. Dilley, T. Govindaraj, L.F. McGinnis and C.M. Mitchell (1992) Object-oriented simulation to support operator decision making in semiconductor manufacturing. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Chicago, Illinois, October 1992, pp. 1510-1515.

[Nelso90] M.L. Nelson, J.M. Moshell and A. Orooji (1990) A relational object-oriented management system. *Proceedings of the IEEE Ninth Annual International Phoenix Conference on Computers and Communications*, Scottsdale, Arizona, 1990, pp. 319-323.

[Niers89] O.M. Nierstrasz and D.C. Tsichritzis (1989) Integrated office systems. In: *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky (eds.) (Reading, Massachusetts: Addison-Wesley)

[Objec93] Objectivity (1993) A guide to interpreting and applying the University of Wisconsin OO7 benchmark. White Paper, Objectivity, Inc., 23 August 1993.

[Objec95a] Objectivity (1995) Object-oriented database management systems: applications for finance. White Paper, Objectivity, Inc., 1995.

[Objec95b] Objectivity (1995) ODBMSs in the telecommunications industry. White Paper, Objectivity, Inc., 1995.

[Objec96] Objectivity (1996) Choosing an object database. White Paper, Objectivity, Inc., 1996.

[ODI93a] ODI (1993) Object Design's results on the OO7 benchmarks. Publicity Note, Object Design, Inc., 26 April 1993.

[ODI93b] ODI (1993) Understanding the OO7 research project. White Paper, Object Design, Inc., 4 April 1993.

[ODI96a] ODI (1996) Financial object data management. White Paper, Object Design, Inc., 1996.

[ODI96b] ODI (1996) Telecommunications object data management (TODM) into the 21st century. White Paper, Object Design, Inc., 1996.

[ODMG93] ODMG (1993) The object database standard: ODMG-93, R.G.G. Cattell (ed.) (San Mateo, California: Morgan-Kaufmann)

[Ohkaw93] H. Ohkawa (1993) Object-oriented database support for scientific data management: a system for experimentation. PhD Thesis, Department of Computer Science and Engineering, Oregon Graduate Institute, April 1993.

[OODBT91] OODBTG (1991) Object data management reference model. (ANSI/X3/SPARC/DBSSG/OODBTG). Final Technical Report. 17 September 1991.

[OOS92] OOS (1992) Object-oriented database management system products. *Object-Oriented Strategies*. February 1992.

[OOS96] OOS (1996) The object-oriented software development tools market. *Object-Oriented Strategies*. May 1996.

[Orens90] J. Orenstein and E. Bonte (1990) The need for a DML: why a library interface isn't enough. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 83-93.

[Otis96] A. Otis (1996) Personal communication, 23 May 1996.

[Oxbor91] E.A. Oxborrow, M.J. Davy, Z.P. Kemp, P.F. Linington and R. Thearle (1991) Object-oriented data management in specialized environments. *Information and Software Technology*. 33 (1):22-30.

[Paton91] N.W. Paton and O. Diaz (1991) Object-oriented databases and frame-based systems: comparison. *Information and Software Technology*. 33 (5):357-365.

[Peder93] M. Pedersen (1993) An analysis of introducing UQ LBEs to persistent storage environments. Working Paper No. 1, Dept. of Computer Science, University of Queensland, Australia, 6 August 1993.

[Peder94a] M. Pedersen (1994) Benchmarking the UQ1 editor. Working Paper No. 3, Dept. of Computer Science, University of Queensland, Australia, 31 March 1994.

[Peder94b] M. Pedersen (1994) Developing benchmarks for recognition editors in persistent storage environments. Working Paper No. 2, Dept. of Computer Science, University of Queensland, Australia, 31 March 1994.

[Peder94c] M. Pedersen (1994) Towards persistence for recognition editors. Working Paper No. 4, Dept. of Computer Science, University of Queensland, Australia, 8 April 1994.

[Perry96] S. Perryman (1996) Personal communication, 21 March 1996.

[Potte94] C. Potter (1994) Object databases: an evaluation and comparison (Milton Keynes: Bloor Research Group)

[Prabh92] C.S.R. Prabhu (1992) Semantic database systems: a functional introduction (London: Sangam Books)

[Preme90] W.J. Premerlani, M.R. Blaha, J.E. Rumbaugh and T.A. Varwig (1990) An object-oriented relational database. *Communications of the ACM*. 33 (11):99-109.

[Priet91] R. Prieto-Díaz and G. Arango (1991) Tutorial organization. In: *Domain Analysis and Software Systems Modeling*, R. Prieto-Díaz and G. Arango (eds.) (Washington: IEEE Computer Society Press)

[Rabit93] F. Rabitti, R.S. Sferrazza, M.G. Tori and P. Zezula (1993) Performance evaluation system for object stores. *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA '93)*, Lecture Notes in Computer Science, 720, pp. 289-300 (Berlin: Springer-Verlag)

[Rasmu92] D.W. Rasmus (1992) Relating to objects. *BYTE*. 17 (14):161-165.

[Reute96a] Reuters (1996) DCx high level design. Draft Document, Version 0.4, Reuters Ltd., London, 6 August 1996.

[Reute96b] Reuters (1996) Data collection architecture (DCA). Draft Document, Version 0.6, Reuters Ltd., London, 11 October 1996.

[Rotze90] K. Rotzell (1990) Transactions and versioning in an ODBMS. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 55-61.

[Rotze91] K. Rotzell and M.E.S. Loomis (1991) Benchmarking an ODBMS. *Journal of Object-Oriented Programming*. 4 (1):66-72.

[Rozen89] S. Rozen and D. Shasha (1989) Using a relational system on wall street: the good, the bad, the ugly, and the ideal. *Communications of the ACM*. 32 (8):988-994.

[Ruben87] W.B. Rubenstein, M.S. Kubicar and R.G.G. Cattell (1987) Benchmarking simple database operations. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Francisco, California, 1987, pp. 387-394.

[Salem90] K. Salem and H. Garcia-Molina (1990) System M: a transaction processing testbed for memory resident data. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):161-172.

[Schek90] H.-J. Schek, H.-B. Paul, M.H. Scholl and G. Weikum (1990) The DASDBS project: objectives, experiences, and future prospects. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):25-43.

[Schre94] H. Schreiber (1994) JUSTITIA: a generic benchmark for the OODBMS selection. *Proceedings of the Fourth International Conference on Data and Knowledge Systems in Manufacturing and Engineering*, Hong Kong, 1994, pp. 324-331.

[Schre95] H. Schreiber (1995) Benchmarking structure preservation and multi-user behavior of object-oriented database systems (Position Paper). *OOPSLA '95 Workshop on Object Database Behavior, Benchmarks, and Performance*, Austin, Texas, October 1995.

[Seque93] Sequent (1993) A benchmark for object storage. White Paper, Sequent Computer Systems, Inc. and Servio Corporation, April 1993.

[Sherl94] P. Sherlock (1994) A CORBA compliant population health summary system. *Proceedings of Object World UK '94*, London, UK, 1994.

[Shier98] J. Shiers (1998) Building a multi-petabyte database: the RD45 project at CERN. In: *Object Databases in Practice*, M.E.S. Loomis and A.B. Chaudhri (eds.) (Upper Saddle River, New Jersey: Prentice-Hall)

[Shipm81] D.W. Shipman (1981) The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*. 6 (1):140-173.

[Simme92] S.S. Simmel and I. Godard (1992) Objects of substance. *BYTE*. 17 (14):167-170.

[Skiad94] M. Skiadelli (1994) Object oriented database system evaluation for the DAQ system. Diploma Thesis, Patras Polytechnic School, Greece, March 1994.

[Soley92] R.M. Soley (1992) Object management architecture guide, volume 2 (Framingham, Massachusetts: OMG)

[Spier91] J. Spiers (1991) Object-oriented databases: evolution or revolution? In: *Hypermedia/Hypertext and Object-Oriented Databases*, H. Brown (ed.) (London: Chapman & Hall)

[Stefi86] M. Stefik and D.G. Bobrow (1986) Object-oriented programming: themes and variations. *AI Magazine*. 6 (4):40-62.

[Stein92] J. Stein (1992) Evaluating object database management systems. *Journal of Object-Oriented Programming*. 5 (6):71-73.

[Stone85] M. Stonebraker (1985) Tips on benchmarking data base systems. *IEEE Database Engineering*. 8 (1):10-18.

[Stone88] M. Stonebraker (1988) Future trends in database systems. *Proceedings of the Fourth International Conference on Data Engineering*, Los Angeles, California, 1988, pp. 222-231.

[Stone90a] C.M. Stone and D. Hentchel (1990) Database wars revisited. *BYTE*. 15 (10):233-242.

[Stone90b] M. Stonebraker, L.A. Rowe and M. Hirohama (1990) The implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):125-142.

[Stone91] M. Stonebraker and G. Kemnitz (1991) The POSTGRES next-generation database management system. *Communications of the ACM*. 34 (10):78-92.

[Stone93a] M. Stonebraker (1993) Object-relational database systems. White Paper, Illustra Information Technologies, 1993.

[Stone93b] M. Stonebraker, J. Frew, K. Gardels and J. Meredith (1993) The SEQUOIA 2000 storage benchmark. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington DC, 1993, pp. 2-11.

[Stone96] M. Stonebraker with D. Moore (1996) Object-relational databases: the next great wave (San Mateo, California: Morgan-Kaufmann)

[STR97] STR (1997) Java data management: comparing ODBMS and RDBMS implementations of quantum objects. White Paper, Strategic Technology Resources, 1997.

[SURFn95] SURFnet (1995) Introducing a directory service. Final report introduction phase SURFnet X.500 pilot project. SURFnet, The Netherlands, 1995.

[Talbo95] S. Ross-Talbot (1995) Stress testing ObjectStore R4.0. Technical Report, Version 1.1, Nomura Information Systems, London, 16 October 1995.

[Talbo96] S. Ross-Talbot (1996) ObjectStore stress testing. Technical Report, Version 0.1, Nomura Information Systems, London, 20 August 1996.

[Talbo98] S. Ross-Talbot (1998) Building a push-based information system using an active database. In: *Object Databases in Practice*, M.E.S. Loomis and A.B. Chaudhri (eds.) (Upper Saddle River, New Jersey: Prentice-Hall)

[Taylo91] D.A. Taylor (1991) Object-oriented technology: a manager's guide (Reading, Massachusetts: Addison-Wesley)

[Taylo92] D.A. Taylor (1992) Object-oriented information systems: planning and implementation (New York: John Wiley & Sons)

[Teeuw93a] W.B. Teeuw (1993) Parallel management of complex objects: the design and implementation of a complex object server for Amoeba. PhD Thesis, University of Twente, The Netherlands, September 1993.

[Teeuw93b] W.B. Teeuw, C. Rich, M.H. Scholl and H.M. Blanken (1993) An evaluation of physical disk I/Os for complex object processing. *Proceedings of the Ninth International Conference on Data Engineering*, Vienna, Austria, 1993, pp. 363-371.

[Thomp93] D. Thompson (1993) Interfacing objects with the relational DBMS. *Database Programming & Design*. August 1993, pp. 33-41.

[Thurs93] M. Thursz, M. Fowler, T. Cairns, M. Thick and G. Gold (1993) Clinical systems design. *Proceedings of the Sixth Annual IEEE Symposium on Computer-Based Medical Systems*, 1993, pp. 134-139.

[Tiwar95] A. Tiwary, V.R. Narasayya and H.M. Levy (1995) Evaluation of OO7 as a system and an application benchmark (Position Paper). *OOPSLA '95 Workshop on Object Database Behavior, Benchmarks, and Performance*, Austin, Texas, October 1995.

[Ullma87] J.D. Ullman (1987) Database theory: past and future. Keynote Speech. *Proceedings of the Sixth ACM Symposium on Principles of Database Systems*, 1987, pp. 1-10.

[Ullma88] J.D. Ullman (1988) Principles of database and knowledge-base systems, volume 1 (Rockville, Maryland: Computer Science Press)

[Unlan90] R. Unland and G. Schlageter (1990) Object-oriented database systems: concepts and perspectives. *Proceedings of the International Symposium on Database Systems of the 90s*. Lecture Notes in Computer Science, 466, pp. 154-197 (Berlin: Springer-Verlag)

[Wade96] A.E. Wade (1996) Personal communication, 8 May 1996.

[Wagne91] D.F. Wagner (1991) Development and proof-of-concept of a comprehensive performance evaluation methodology for geographic information systems. PhD Thesis, Department of Geography, The Ohio State University, 1991.

[Weick84] K.E. Weick (1984) Theoretical assumptions and research methodology selection. In: *The Information Systems Research Challenge*, F.W. McFarlan (ed.) (Boston, Massachusetts: Harvard Business School Press)

[Wells92] D.L. Wells, J.A. Blakeley and C.W. Thompson (1992) Architecture of an open object-oriented database management system. *IEEE Computer*. 25 (10):74-82.

[White89] D. White (1989) Is it time to abandon the poor relations? *Computing*. 2 February 1989, pp. 24-25.

[Wilco94] J. Wilcox (1994) Object databases. *Dr. Dobb's Journal.* November 1994, pp. 26-34.

[Wilki90] K. Wilkinson, P. Lyngbæk and W. Hasan (1990) The Iris architecture and implementation. *IEEE Transactions on Knowledge and Data Engineering.* 2 (1):63-75.

[Wilki93] G. Wilkie (1993) Object-oriented software engineering (Wokingham: Addison-Wesley)

[Winbl90] A.L. Winblad, S.D. Edwards and D.R. King (1990) Object-oriented software (Reading, Massachusetts: Addison-Wesley)

[Works91] Workshop (1991) Workshop on objects in data management. *Proceedings of the Third Joint Meeting*, Anaheim, California, 1991.

[Worlt93] J. Worlton (1993) Towards a taxonomy of performance metrics. In: *Computer Benchmarks*, J.J. Dongarra and W. Gentzsch (eds.) (Amsterdam: North-Holland)

[Yin89] R.K. Yin (1989) Case study research: design and methods (Newbury Park, California: Sage Publications)

[Youss93] M.W. Youssef (1993) Transaction behaviour in large database environments: a methodological approach. PhD Thesis, Department of Business Computing, The City University, London, UK, August 1993.

[Zdoni90] S.B. Zdonik and D. Maier (1990) Fundamentals of object-oriented databases. In: *Readings in Object-Oriented Database Systems*, S.B. Zdonik and D. Maier (eds.) (San Mateo, California: Morgan-Kaufmann)

[Zingl95] M. Zingler (1995) Implementing a distributed object technology environment: approach: rationale, benefits and choices. *Proceedings of Object World UK '95*, London, UK, 1995.

[Zorn95] B.G. Zorn and A.B. Chaudhri (1995) Object database behavior, benchmarks, and performance. *Addendum to the Proceedings of the ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '95)*, Austin, Texas, October 1995, pp. 159-163.