

UNIVERSITY OF SOUTHAMPTON

An Architecture for Management of Large, Distributed, Scientific Data

Volume 1 of 1

Mark Papiani

Doctor of Philosophy

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science

This thesis was submitted in May 2000.

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE

ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

An Architecture for Management of Large, Distributed, Scientific Data

Mark Papiani

This thesis describes research into Web-based management of non-traditional data. Three prototype systems are discussed, *GBIS*, *DBbrowse* and *EASIA*, each of which provided examples of new ideas in this area.

In 1994/1995, when most Web pages consisted of static HTML files, *GBIS* (the *Graphical Benchmark Information Service*) [181] [117] demonstrated the benefits of interactive, dynamic Web pages for visualisation of scientific data. *GBIS* also highlighted problems with storing the underlying data in a filesystem, which initiated an investigation into the use of databases as the underlying source for dynamic Web pages.

In 1996/1997 this research investigated *automatic* generation of generic Web interfaces to databases to facilitate rapid deployment of interactive Web-based applications by developers with little Web development experience. A prototype system, *DBbrowse*, demonstrates the results [180] [75]. *DBbrowse* can generate Web interfaces to object-relational databases with intuitive query capabilities. *DBbrowse* also demonstrates a method for *browsing* databases to further support users with little database experience.

In 1999 concepts from *GBIS* and *DBbrowse* were used as the starting point for examining new architectures for archiving scientific datasets. Data from numerical simulations generated by the UK Turbulence Consortium was used as a case study. Due to the large datasets produced, new Web-based mechanisms were required for storage, searching, retrieval and manipulation of simulation results in the hundreds of gigabytes range. A prototype architecture and user interface, *EASIA* (*Extensible Architecture for Scientific Data Archives*) [182] [183] is described. *EASIA* demonstrates several new concepts for active digital libraries of scientific data. Result files are archived in-place thereby avoiding costs associated with transmitting results to a centralised site. The method used shows that a database can meet the apparently divergent requirements of storing both the relatively small simulation result metadata, and the large, distributed result files, in a unified, secure way. *EASIA* also shows that separation of user interface specification from user interface processing can simplify the extensibility of such systems. *EASIA* archives not only data in a distributed fashion, but also applications. These are loosely coupled to the archived datasets via a user interface specification file that uses a vocabulary defined by a markup language. Archived applications can provide reusable dynamic server-side post-processing operations. This can reduce bandwidth requirements for requested data through server-side data reduction. The architecture allows post-processing to be performed directly without the cost of having to rematerialise to files, and it also reduces access bottlenecks and processor loading at individual sites.

Table of Contents

| | |
|--|-----------|
| TABLE OF CONTENTS..... | 3 |
| LIST OF TABLES | 7 |
| LIST OF FIGURES..... | 8 |
| ACKNOWLEDGEMENTS | 10 |
| AUTHOR'S DECLARATION..... | 11 |
| 1 INTRODUCTION..... | 12 |
| 1.1 Outline of Research Areas | 12 |
| 1.2 Structure of this Thesis | 15 |
| 2 DATABASE AND WEB DEVELOPMENTS..... | 17 |
| 2.1 Database Developments..... | 17 |
| 2.1.1 Object-Relational and Object-Oriented Database Technology..... | 17 |
| 2.1.2 SQL:1999..... | 23 |
| 2.1.3 Parallel Databases..... | 26 |
| 2.1.4 Java Database Access | 29 |
| 2.1.5 Microsoft's Data Access Strategy..... | 34 |
| 2.2 Web Developments | 35 |
| 2.2.1 The Common Gateway Interface..... | 36 |
| 2.2.2 Web Server Extensions..... | 38 |
| 2.2.3 FastCGI..... | 39 |
| 2.2.4 Java and Java Applets | 39 |
| 2.2.5 Java Servlets | 43 |
| 2.2.6 Java Server Pages and Active server Pages | 43 |
| 2.2.7 Distributed Object Technologies | 45 |
| 2.2.8 XML and Dynamic HTML..... | 55 |

| | |
|--|-----------|
| 2.3 Multi-tier Web/Database Connectivity | 64 |
| 2.4 Summary | 68 |
| 3 THE GRAPHICAL BENCHMARK INFORMATION SERVICE..... | 69 |
| 3.1 Introduction..... | 69 |
| 3.2 GBIS Overview..... | 69 |
| 3.3 GBIS Implementation | 72 |
| 3.4 GBIS Result File Structure | 75 |
| 3.5 Updating the Results Database to include additional Machines and Manufacturers | 77 |
| 3.6 Conclusions | 78 |
| 4 AUTOMATICALLY GENERATING WEB INTERFACES TO RELATIONAL DATABASES | 80 |
| 4.1 Introduction..... | 80 |
| 4.2 Providing Web Access to the Database..... | 82 |
| 4.3 Automatically Generating the User Interface and SQL Queries..... | 83 |
| 4.4 Providing Database Browsing via Dynamic Hypertext Links Derived from Referential Integrity Constraints..... | 84 |
| 4.5 Example of a Database Browsing Session..... | 87 |
| 4.6 Conclusions | 93 |
| 5 AN ARCHITECTURE FOR MANAGEMENT OF LARGE, DISTRIBUTED, SCIENTIFIC DATA..... | 95 |
| 5.1 Introduction..... | 95 |
| 5.2 System Architecture and User Interface..... | 99 |
| 5.2.1 System Architecture..... | 99 |

| | |
|---|------------|
| 5.2.2 XML Specification of the User Interface..... | 101 |
| 5.2.3 Searching and Browsing Data..... | 101 |
| 5.2.4 Interface Customisation through XUIS Modification..... | 107 |
| 5.2.5 Suitable Processing of Data Files Prior to Retrieval: ‘Operations’..... | 109 |
| 5.2.6 Code Upload for Server-side Execution..... | 117 |
| 5.2.7 Administration Features..... | 119 |
| 5.3 Implementation and Design Decisions | 119 |
| 5.3.1 Experimental Bandwidth Measurements..... | 119 |
| 5.3.2 SQL Management of External Data: The New DATALINK Type..... | 121 |
| 5.3.3 Java Servlets and JavaScript..... | 123 |
| 5.4 Conclusions | 127 |
| 6 RELATED WORK | 129 |
| 6.1 Related Work on User Interfaces to Databases | 129 |
| 6.1.1 Introduction..... | 129 |
| 6.1.2 Stand-alone Graphical Query Interfaces to Databases..... | 130 |
| 6.1.3 Web-based User Interfaces to Databases..... | 137 |
| 6.2 Related Work on Web-based Management of Scientific Data | 140 |
| 6.3 Discussion | 147 |
| 7 SUMMARY | 149 |
| 7.1 Contributions to the Field..... | 149 |
| 7.1.1 GBIS..... | 149 |
| 7.1.2 DBbrowse..... | 150 |
| 7.1.3 EASIA..... | 150 |
| 7.2 Future Work | 153 |
| 7.2.1 Gathering Operation Statistics and Caching Results..... | 153 |
| 7.2.2 Providing a Multidatabase Capability..... | 154 |
| 7.2.3 Can Codes other than Java be Uploaded for Execution?..... | 155 |
| 7.2.4 Runtime Monitoring of Post-Processing Operations..... | 156 |
| 7.2.5 XML as a Scientific Data Standard..... | 156 |
| 7.2.6 Other Enhancements to the EASIA Architecture..... | 158 |

| | |
|--|------------|
| 7.3 Concluding Remarks | 160 |
| APPENDIX A : PUBLICATIONS AND PRESENTATIONS | 161 |
| APPENDIX B : CLIENT/SERVER 'PING' BENCHMARK RESULTS | 163 |
| REFERENCES | 167 |

List of Tables

Table 1: Experimental bandwidth measurements for file transfer between two UK universities... 120

List of Figures

| | |
|--|-----|
| Figure 1: A 2-tier architecture using a Java Applet and JDBC for database access. | 64 |
| Figure 2: A 3-tier architecture using a Java Applet, CORBA and JDBC | 65 |
| Figure 3: A 3-tier architecture using HTML/HTTP, Java Servlets and JDBC | 67 |
| Figure 4: Graph showing results of the Multigrid Benchmark. | 71 |
| Figure 5: Graph showing results of the LU Simulated CFD Application Benchmark. | 72 |
| Figure 6: GBIS manufacturer list page. | 73 |
| Figure 7: GBIS machine list page. | 74 |
| Figure 8: GBIS change defaults page. | 74 |
| Figure 9 Example contents of a GBIS result data file. | 76 |
| Figure 10: Interconnection strategy for providing Web accesses to a database. | 83 |
| Figure 11: Employee Activity database schema and relationships between entities. | 85 |
| Figure 12: Selecting tables of interest. | 87 |
| Figure 13: Selecting columns and specifying conditions. | 88 |
| Figure 14: Results from querying the DEPARTMENT table. | 89 |
| Figure 15: Browsing to find all employees in department number 'D11'. | 89 |
| Figure 16: Browsing to show project activities for each employee. | 90 |
| Figure 17: Browsing to inline full project details. | 91 |
| Figure 18: Refining a query during the browsing stage. | 92 |
| Figure 19: Query results after refinement. | 92 |
| Figure 20: Displaying the SQL that generated the result. | 93 |
| Figure 21: System architecture. | 99 |
| Figure 22: Login screen. | 100 |
| Figure 23: Table selection screen. | 102 |
| Figure 24: Searching the archive. | 103 |
| Figure 25: Result from querying the SIMULATION table. | 104 |
| Figure 26: Sample database schema for UK Turbulence Consortium. | 105 |
| Figure 27: CLOB browsing. | 105 |
| Figure 28: DATALINK browsing. | 106 |
| Figure 29: Customised display of results from a query on the SIMULATION table. | 108 |
| Figure 30: Result table showing 'operations' available for post-processing datasets. | 112 |
| Figure 31: 'Operation' description and parameter input form. | 113 |
| Figure 32: Output from 'operation' execution. | 114 |
| Figure 33: NCSA's SDB [243] has been specified as an 'operation' in the XUIS and invoked on a dataset managed within the EASIA architecture. | 116 |
| Figure 34: User administration screen. | 119 |
| Figure 35: Security mechanism employed for uploaded post-processing codes. | 125 |

| | |
|--|-----|
| Figure 36: The client/server ping benchmark..... | 163 |
| Figure 37: Client/server ping benchmark results..... | 164 |

Acknowledgements

The UK Turbulence Consortium provided data for the EASIA research prototype. IBM's DB2 Scholars programme provided DB2 licenses.

Thanks to Tony Hey for employing me as a research assistant for 5 years. Thanks to Ed Zaluska for putting me in touch with Tony after reading my initial speculative employment enquiry to the University. Thanks to Denis Nicole for putting me in touch with the UK Turbulence Consortium. Thanks to David Walker and Kirk Martinez for acting as external and internal examiner for my viva.

I would like to thank my parents Rolando and Jenny for all their support during my on/off 34-year reign as a student! Thanks to my sisters Sandra and Lisa for their support and for helping me to buy birthday presents. Thanks to the Lads in Bournemouth (Brett Colley, Dayle Colley, Andy Foote and Paul Brady) for dragging me out at weekends and accepting partial responsibility for this thesis taking so long.

Thanks to Dave and Fleur (and Callum) who were with me at the start of my University days back in 1987. Thanks for your friendship over the years, and please be patient - I promise to ring soon.

My colleague at Southampton University (and partner in crime/gym), Alistair Dunlop, made the 5 years at Southampton the most fun I have ever had in a job. Thanks to Jasmin Wason for working with me after Alistair had left the University for the lure of industry. Thanks also to Jasmin for helping get my viva together.

Finally, special love and thanks to the special people who had to put up with me during this project - Sara Gibbs and Tanya Smith.

1 Introduction

This thesis describes research (during the period 1994 to 2000) into Web-based management of non-traditional data. In this thesis traditional data is defined as simple datatypes including integers, floating-point types, characters, dates, times and timestamps (effectively datatypes that are associated with the traditional relational data model (see Chapter 2)). Non-traditional data is characterised by complex multimedia datatypes including text, audio, image and video, as well as binary files used for other purposes such as multidimensional scientific data (effectively datatypes that are associated with newer object-oriented and object-relational data models (see Chapter 2)).

The Internet (particularly the Web) is having a dramatic effect on all walks of life, from commerce to education and research to leisure. Over the last few years the nature of the Web has been changing from a file based, textual, static, insecure environment with dumb browsers to a database based, multimedia, dynamic, secure, environment with smart browsers. Three prototype systems are discussed in this thesis, *GBIS* (the *Graphical Benchmark Information Service*) [181] [117], *DBbrowse* [180] [75] and *EASIA* (*Extensible Architecture for Scientific Data Archives*) [182] [183], each of which has provided exemplars of new ideas for non-traditional data management in the fast evolving Web environment.

1.1 Outline of Research Areas

This thesis describes research into Web-based management of non-traditional data concentrating on the following areas.

- *The importance of dynamic, interactive Web-based visualisation for scientific data repositories.*

GBIS was an early system (1994/95) employing CGI (Common Gateway Interface) scripting [48] combined with standard application programs, to provide Web-based management of scientific data. *GBIS* was designed to manage non-traditional data in the form of textual output files from multiprocessor benchmark results. At a time when most Web pages consisted of

static HTML (Hypertext Markup Language) [122] files, GBIS demonstrated dynamic Web pages for visualisation of scientific data. GBIS employed some of the first technologies available for dynamic Web pages with user interaction (such as the CGI and associated scripting using the Bourne Shell and PERL).

- *User interfaces that integrate the Web and object-relational databases.*

At the ACM SIGMOD Conference in 1996, Manber suggested that one of the main lessons to be gained from the success of the Web was the importance of browsing [147]. He went on to say that an important step would be to find a way to browse even relational databases. The early part of this research involved a survey of database user interfaces. Existing techniques for browsing databases were studied. Existing methods for Web/database connectivity were studied in detail. At the time, most existing systems required programming effort. One aim was to find an automated technique for connecting databases to the Web and for searching and browsing the data via a Web-based user interface.

DBbrowse (1996/1997) was the result of research into *automatic* generation of generic Web interfaces to object-relational databases, to facilitate rapid deployment of interactive Web-based applications by developers with little Web development experience. DBbrowse demonstrated a novel method for browsing databases using the Web.

- *Architectures for active digital archives that can manage large, distributed scientific data.*

The Internet allows for fast, effective scientific collaboration on a scale that has previously been impossible. It is now possible to transfer research results, in the form of scientific papers, result files or metadata describing experiments, in seconds or minutes to worldwide locations. Advances in computing technology, such as larger, cheaper storage and faster processing, have affected the type of

data that can be manipulated, allowing, for example, *much larger* raw result data to be generated and exchanged.

Additional motivation for this research came from the Caltech Workshop on Interfaces to Scientific Data Archives [235], which identified an urgent need for infrastructures that could manage and federate *active* libraries of scientific data. Hawick and Coddington [112] define active data archives as follows: “An *active* data archive can be defined as one where much of the data is generated on-demand, as value-added data products or services, derived from existing data holdings”. They also state that the information explosion has led to a very real and practical need for systems to manage and interface to scientific archives.

Treinish [217] [218] presents ideas for interactive archives for scientific data. He believes that the capability to produce data is growing much faster than the ability to manage data. Typical storage and communications protocols are not suitable for current archives and there must be a fundamental change from providing static archives that provide bulk data access to dynamic, interactive systems. Data volumes are too large for practical examination and the starting point for locating relevant data should consist of searching metadata that provides abstractions of the archive. Treinish believes that browsing is extremely important for data selection. He suggests that large datasets can be represented by much smaller visual representations that allow browsing to identify features of interest. He concludes that future research should include experimentation with compression techniques to aid visual browsing and secondly the design of architectures for interactive archives. These architectures could include the integration of metadata, data servers, visual browsing and existing data analysis tools.

The final architecture and prototype implementation described in this thesis, EASIA, is an architecture for an active digital archive that can reduce bandwidth requirements for Web-based management of large, scientific datasets. The first aim for this architecture was to extend the automated Web/database connectivity techniques, developed in the DBbrowse, to the types of scientific data management problems described above. Browsing techniques

from DBbrowse could be used to facilitate navigation through metadata associated with scientific datasets to identify potential datasets of interest. Beyond this, EASIA provides features that meet Treinish's requirements for the architecture of future interactive scientific archives. Namely, integration of metadata, data servers, and existing data analysis applications.

Scientific data management introduces new problems associated with storage and retrieval of large, often unformatted, files in an environment where bandwidth is limited. EASIA provides new mechanisms for Web-based storage, searching, retrieval and manipulation of scientific datasets in the hundreds of gigabytes range. EASIA demonstrates several new concepts for active digital libraries of scientific data. EASIA archives data in a distributed fashion so that large datasets can be archived at (or close to) the sites where they are generated in order to eliminate the costs associated with transfer to a centralised repository. EASIA also archives applications. Archived applications can provide reusable dynamic server-side post-processing operations that can reduce bandwidth requirements for requested data. Post-processing can also be achieved by allowing users to upload code to be run securely on the file servers hosting the datasets.

Research and development of the GBIS, DBbrowse and EASIA prototypes has required comprehensive knowledge of Web and database technologies, and of related work. This thesis therefore contains significant critical review in these areas. Finally, it is worth noting that although successive prototypes use different technologies due to emergence of improved solutions in this fast changing field, wherever possible the prototypes have been implemented using commodity components, technologies and open standards.

1.2 Structure of this Thesis

The rest of this thesis is structured as follows:

- Chapter 2: Database and Web Developments – This chapter describes the state-of-the-art in database and Web technologies. The purpose of this research was to provide a critical survey of available technologies and to assess the best ways to implement the architectures developed during this research.
- Chapter 3: The Graphical Benchmark Information Service – This chapter provides a detailed description of the research surrounding GBIS.
- Chapter 4: Automatically Generating Web Interfaces to Relational Databases – This chapter provides a detailed description of the research surrounding DBbrowse.
- Chapter 5: An Architecture for Management of Large, Distributed, Scientific Data – This chapter provides a detailed description of the research surrounding EASIA.
- Chapter 6: Related Work – This chapter reviews related work on user interfaces to databases. Web based interfaces to databases are compared to DBbrowse, particularly in terms of techniques for data browsing. The second part of this chapter discusses related research in the area of Web-based management of scientific data, providing comparisons with the EASIA architecture.
- Chapter 7: Summary – This chapter provides a summary, ideas for future related research and closing remarks.

2 Database and Web Developments

This chapter provides an overview of developments in database and Web technology over the last few years. This survey was carried out to understand the current state-of-the-art in these fields and then to assess the suitability of different technologies for implementation of the systems to be developed during this research. The chapter is split into three main sections covering database developments, Web developments, and multi-tier Web/database connectivity.

2.1 Database Developments

This section describes why object-relational database technology was preferred to object-oriented database technology. Sections on SQL:1999, parallel databases and Java database access follow. Object-relational databases and SQL:1999 were used in the DBbrowse and EASIA architectures, and Java database access mechanisms were used in the implementation of EASIA. A separate section on Microsoft's data access strategy is included for completeness since Microsoft promotes different technologies to those being used by most of the other database vendors.

2.1.1 Object-Relational and Object-Oriented Database Technology

2.1.1.1 Reasons for Choosing Object-Relational Technology for this Research

A primary requirement for the database technology selected for this research was that it could support both traditional data types and non-traditional data, such as the large binary data files often used for scientific datasets. Object-oriented database management systems (OODBs) and object-relational database management systems (ORDBs) both provide this capability. ORDBs were chosen for this research for a number of reasons that are described below.

Firstly, ORDBs support the standardised Structured Query Language (SQL) [59] as well as providing metadata, which defines amongst other things, the database schema. These two features make it possible to build generic, schema driven, dynamic interfaces to databases. This was a requirement for both the DBbrowse (chapter 4) and EASIA prototypes (chapter 5). Second, ORDBs support a security

model as a fundamental function. This feature is lacking in OODBs. Third, parallel versions exist for all the major ORDBs, allowing migration to high performance parallel architectures if necessary. Fourth, despite around ten years of marketing and attempts to standardise their features, OODBs remain a niche technology with a corresponding increased risk associated with their usage. All of the market leaders associated with relational database management systems (RDBs) now offer ORDB products. These include IBM DB2 Universal Database [65] [42], Oracle8 [174], Informix Dynamic Server [128], Sybase Adaptive Server [214] and Microsoft SQL Server 7 [206]. With all the major database vendors supporting object-relational (OR) technology it seems likely that this will remain the dominant database technology.

2.1.1.2 The Progression of Object-Oriented Database Standardisation

Carey [30] discusses the progress that OODBs have made and why their impact has not lived up to expectations. He concludes that we are on the verge of an era where ORDBs will begin taking over the enterprise. Carey gives some of the reasons for the early success of RDBs (the foundation of all ORDBs) and contrasts these with the progression of OODBs. In the early days of RDBs there was a single, clearly defined data model based on sets of tuples with simple attributes. Similarly, SQL, emerged early on as the query language for RDBs. Development of OODBs has been very different, with no initial agreement on the details of the data model and no query model or language.

In the early 1990's a consortium of OODB vendors formed the Object Data Management Group (ODMG, formerly known as the Object *Database* Management Group) [168] to address these problems. An Object Database Standard, ODMG-93, was released in 1996 [37]. This contains several chapters which define; the ODMG object model (which is an extension of the Object Management Group (OMG) [170] data model); an object definition language (ODL); an object query language (OQL); a binding to C++ and Smalltalk for all functionality, including object definition, manipulation, and query. Release 2.0 of the standard, ODMG-97, also included a Java language binding [38] (refer to Section 2.2.4 for information on Java). (Release 3.0 of the Standard was published this year [39]. Unfortunately, as with the previous

version, this standard is available for purchase in book format only and is not available for free download. This probably hinders widespread knowledge of the standard.) Although the standard has been out in some form for about 6 years, there are still differences between many of the OODB products in terms of their programming interfaces and query support. Many vendors conform to parts of the specification corresponding to individual chapters in the standard. Variations in the level of support for standards across different OODBs made this technology unsuitable for underpinning the standards-based, vendor independent prototype scientific data archiving systems that were investigated during this research.

The lack of sufficient standardisation between OODBs is not solely an implementation issue. The standard itself is still not comprehensive in some areas. For example, the Java binding does not yet support certain features of the ODMG Data Model such as, extents, keys, relationships and access to metadata (see for example [43]). A further barrier to portability amongst Java bindings from different vendors is the fact that the mechanism for identifying *persistence-capable* classes is not specified. A Java OODB application can contain both persistent and transient objects of the same class (this is known as orthogonal persistence – where persistence is independent of class). The standard states that a transient object belonging to a persistence-capable class can be made persistent if it is bound to a name within the database or if it is referenced by another persistent object. This is known as *persistence by reachability*. As an example of the divergence in this area, the OODB from POET [188] currently uses a configuration file to indicate persistence-capable classes. This works in conjunction with a pre-processor for the Java source files, which extracts required information from the files and stores it in a dictionary, and then calls the standard Java compiler. The Objectivity/DB OODB [169], on the other hand, determines persistence-capability by requiring that such classes inherit from a specified superclass.

OODBs are designed to add persistence to objects within an object-oriented (OO) programming language. Despite an OODB standard, language differences make it very difficult to port OODB applications between languages. Even for non-database applications, written in the same language, portability between compilers from different vendors is sometimes an issue, which gets magnified once the OODB

environment is added. Architectural differences associated with *page servers* ('dumb server' with intelligent 'fat' clients) versus *object servers* ('thin' clients with an intelligent server) also complicate portability issues amongst OODBs [43].

2.1.1.3 Object-Relational: Combining the Benefits of Object-Oriented and Relational Technology

Kim [139] is an advocate of OR technology. He reviews the promises of OODBs, examines the reality of these systems, and concludes by discussing how their promises may be fulfilled through unification with relational technology. Kim begins by highlighting the following advantages that OODBs have over RDBs.

Within the relational model, the responsibility for defining and displaying the structure of the data typically lies with the application. The application must impose the 'object structure' on the data from the flat generalised relational structure. OODBs, on the other hand, can directly model data as structured objects. For example, hierarchical data (or complex nested data) must be represented as tuples in multiple relations in an RDB. OODBs allow the data type of an attribute to be a primitive type or an arbitrary user defined type (UDT). Row (or composite types) and multi-valued collection attributes (sets, bags, arrays and lists) are also available. This nested object representation allows hierarchical data to be naturally represented. This could for example, be used for an engineering bill of materials where an item list for an assembly may contain several sub-assemblies. The ability to represent this structure as nested objects avoids the need for tuples in multiple tables, which involves expensive joins for retrieval. Reference types can allow simpler query paths that avoid complex value-based joins, using instead, pointer-based navigation.

RDBs offer a set of primitive, built-in data types with no means of adding UDTs. OODBs allow complex unstructured data to be stored as UDTs. Furthermore, new data types may be created as new classes, possibly even as subclasses of existing classes, inheriting their attributes and methods.

Although RDBs offer stored procedures (a program written in a procedural language and stored in the database for later loading and execution) these are not

encapsulated with data. Further, since RDBs do not have the inheritance mechanism, the stored procedures cannot automatically be reused.

OODBs overcome these problems and have the potential to reduce the difficulty of designing large complex databases and applications. Inheritance and encapsulation make database design and application program reuse possible. However, most OODBs still lack basic database features that the users of RDBs have come to expect. These features include a full non-procedural query language, views, dynamic schema changes and parameterised performance tuning. Added to this, the robustness, scalability and fault tolerance of OODBs does not meet that of more mature RDBs. Kim recommends combining features from the OO and relational models in support of the OR model. This combination makes it possible to support UDTs, dynamic schema changes, SQL, triggers, constraints, automatic query optimisation, and views as a unit of authorisation.

Current OR products support features such UDTs, user-defined functions (UDFs), triggers and enhanced integrity constraints. UDTs allow the set of built-in types to be extended with new data types such as text, image, audio, video, time series, line, point, polygon, etc. There are variations in the capabilities of UDTs amongst the different vendors. The most basic form of UDT is a renamed or distinct base type, which can be used to add stronger typing. Beyond this UDTs can represent row types or full *abstract data types* (ADTs) that encapsulate arbitrarily complex structures and attributes. UDFs can apply to both base types and UDTs to define methods by which applications create, access and manipulate data. Vendors are beginning to market UDTs in type extension packages, for example, Informix's Datablades and IBM's Relational Extenders provide packages for spatial and time-series data.

The features and advantages of the OR model made it the natural choice for the EASIA scientific data archive (Chapter 5). Ferreira et al. [92] also support OR technology for scientific data management. They state that an important subset of scientific applications fall into the complex data with queries category (as defined in Stonebraker's classification matrix for DBMS applications [211]) and can therefore be supported by object-relational database management systems.

2.1.1.4 Support for Object-Oriented Databases

Celko and Celko [40] and Bloom [20] provide an alternative viewpoint to that given so far. Both are supportive of OODB technology. Bloom believes that the trend towards increasingly complex OO applications, particularly multi-tier distributed object systems, will require more database functionality and that RDBs will increasingly give way to OODBs. Celko and Celko believe that both models have a place. They state that rather than trying to fit data to a database model, a database model should be chosen according to the type of data and expected access patterns. For simple data, RDBs provide a proven high performance solution for both simple queries (such as those associated with transaction processing systems) and complex queries (such as those associated with data mining). For complex data, or data involving complex relationships, OODBs provide a better solution. Internet based multimedia solutions tend to fall into this second category, suggesting an increasing demand for OODBs. OODB vendors have been particularly quick at enhancing their existing products, or producing new product ranges with XML (Extensible Markup Language, see Section 2.2.8) capabilities. They argue that OODBs are a natural match for structured, nested, richly linked information and are therefore capable of storing XML data in native form as objects rather than having to disassemble the data into tabular data [120] [205]. (In fact, the market leading OODB vendor, until now known as Object Design, has recently rebranded itself, and is now known as Excelon [88] after one of its XML products, and is concentrating on dynamic XML-based business-to-business (B2B) commerce.)

Despite these claims, during this research ORDBs proved entirely adequate for representing complex data from numerical simulations. This data was stored in native binary formats in non-traditional data types. Advantages of the OO data model might have been more apparent if the scientific datasets needed to be stored at a lower-level of granularity, for example, in order to make individual elements of a multi-dimensional array directly accessible through a standard database query. However, a complex representation of relationships between data items, using for example, nested objects, carries with it its own set of disadvantages. For example, once an object is nested inside another object, then to maintain efficient delivery a mechanism of separating these objects may be required when the whole object is not

required. Also, if an existing nested class is needed in a new class definition, then the nested class definition needs to be repeated in the new class. Often, the solution to both of these problems is to separate objects, and to store links between them in the form of references. However, this leads increasingly to a one-to-one correspondence between classes in OODBs and tables in ORDBs. Furthermore, Ensor and Stevenson [85] report that they have rarely (if ever) experienced performance problems with the use of foreign key (see Section 4.4) links to navigate a parent child-relationship. Indeed, if the actual value of the foreign key is required then a reference based model exhibits the disadvantage that the reference must be navigated to obtain that value.

2.1.1.5 Additional Resources

The book by Ullman and Widom [220] covers the latest database standards (for OO and OR technology) including OQL, ODL, SQL2, and SQL3 (now SQL:1999, see Section 2.1.2) with explanations of how to design databases for both models using ODL and entity-relationship modelling [44].

Currently ORDB products do not provide complete object capabilities in terms of encapsulation, inheritance, polymorphism, object IDs and pointer based navigation. However, the future direction of ORDBs is to achieve many of the benefits of the object model. The emerging SQL:1999 standard, described in the next section, incorporates these features. For ORDBs that do support the majority of the new object features, such as row types, inheritance, references, path expressions and UDTs the *BUCKY* ('Benchmark of Universal or Complex Kwery Ynterfaces') *Object-Relational Benchmark* [31] was designed to test the performance of these new features. The benchmark provides an OR version of BUCKY and a semantically equivalent relational schema and queries so that the performance of the OO features can be compared with traditional RDB functionality.

2.1.2 SQL:1999

The prototype systems created during this research benefited in particular from several new OR features that form part of the emerging SQL:1999 Standard (formerly known as SQL3, see for example [79]). For example, DBbrowse and EASIA both made extensive use of non-traditional, Large Object (LOB) data types

described in Part 2 of the Standard. EASIA also uses JDBC which is discussed in Part 10 of the Standard (being the technology upon which SQLJ is layered, see Section 2.1.4.2). EASIA also used SQL/MED, described in Part 9 of the Standard, at the foundation of its architecture (see Section 5.3.2).

SQL:1999 enhances SQL2 (also known as SQL-92) [59] into a computationally complete language for the definition and management of persistent, complex objects. SQL:1999 includes generalisation and specialisation hierarchies, multiple inheritance, user defined data types, triggers and assertions, support for knowledge based systems, recursive query expressions, and additional data administration tools. It also includes the specification of ADTs, object identifiers, methods, inheritance, polymorphism, encapsulation, and all of the other facilities normally associated with object data management.

In 1993, the ANSI and ISO development committees decided to split future SQL development into a multi-part standard. Currently there are 9 parts:

- Part 1: SQL/Framework (ANSI/ISO/IEC 9075-1-1999) - A non-technical description of how the document is structured.
- Part 2: SQL/Foundation (ANSI/ISO/IEC 9075-2-1999) - The core specification, including all of the new ADT features.
- Part 3: SQL/CLI (Call Level Interface) (ANSI/ISO/IEC 9075-3-1999)
- Part 4: SQL/PSM (Persistent Stored Modules) (ANSI/ISO/IEC 9075-4-1999) - The stored procedures specification, including computational completeness.
- Part 5: SQL/Bindings (ANSI/ISO/IEC 9075-5-1999) - The Dynamic SQL and Embedded SQL bindings taken from SQL-92.
- Part 6: SQL/Transaction - An SQL specialisation of the popular XA Interface developed by X/Open.
- Part 7: SQL/Temporal - Adds time related capabilities to the SQL standards.
- Part 9: SQL/MED - Management of External Data (see Section 5.3.2).
- Part10: SQL/OLB - Object Language Bindings (see Section 2.1.4.2).

Part 8 existed at one time under the informal name SQL/Object, but its material got incorporated into Part 2. ISO also accepted a recommendation to cancel the project under which Part 6 was being developed. The rationale for the cancellation was that the working draft had not been changed since about 1995 and nobody seemed to be interested in publication of the material in question any more.¹

In the USA, the entirety of SQL:1999 is being processed as both an ANSI domestic project (the X3H2 committee covers Database and includes SQL) and as an ISO project (ISO/IEC JTC1/SC 21/WG3 DBL). The ISO standards lifecycle requires that every proposal for a standard starts life as a Working Draft (WD), progresses to Committee Draft (CD), then to Final Committee Draft (FCD), followed by Draft International Standard (DIS), and finally International Standard. Eisenberg and Melton [81] report on the status of each part of SQL:1999 as at March 2000. Parts 1 to 5 are International Standards with Part 10 expected to reach International Standard in 2000 (currently at DIS ballot stage), Part 9 in 2001 (currently at FCD ballot stage) and Part 7 in 2003. For the next generation of the SQL standard (after SQL:1999), Part 5 has been eliminated by merging its contents into SQL/Foundation, and a new Part 11, SQL/Schemata, has been created to hold the Information and Definition Schema specifications that were removed from SQL/Foundation [81].

In addition to the SQL:1999 work, a number of additional related projects are being pursued, including, SQL/MM. Approved in early 1993, this is a new ISO/IEC international standardisation project (within WG3) for development of an SQL class library for multimedia applications. This multi-part standard will specify packages of SQL ADT definitions using the facilities for ADT specification and invocation provided in the emerging SQL:1999 specification. SQL/MM intends to standardise class libraries for science and engineering, full-text and document

¹ Private communication to the author from Hugh Darwen, Database Specialist, IBM United Kingdom Limited, 15 Jul 1999.

processing, and methods for the management of multimedia objects such as image, sound, animation, music, and video.

The object management features of SQL:1999 incorporate many of the features of OODBs. In view of this a merger group [150] was formed with participation from ANSI X3H2 and the ODMG, with intent to merge the ODMG's OQL query language with SQL:1999. OQL would then form a read-only subset of SQL:1999, since OQL does not include *INSERT*, *UPDATE* and *DELETE*, preferring to implement these through method invocation.

Obtaining status information on the SQL Standards, and copies of the Standards themselves can be difficult. In the past the standards were available for purchase only, with SQL-92 costing around \$295. Now the first 5 Parts of SQL:1999 (that have reached the level of International Standard) are available for electronic download from both the *ANSI Electronic Standards Store* [12] and the *NCITS Standards Store* [161], at a price of \$20 per part. There is however, no official Web site that details the status of the emerging Parts of the SQL Standard. However, a document repository for ISO/IEC JTC1/SC21/WG3 is available at [130], although it is very difficult to navigate. Books are also beginning to emerge. Gulutzan and Pelzer [106] provide coverage of the first 5 parts of the standard and Fortier [97] provides information on SQL/Foundation. The Web site at [207] is outdated but still contains some useful information.

2.1.3 Parallel Databases

The availability of powerful, relatively inexpensive commodity CPU chips and other computer components now means that multi-processor computing offers mainframe or better than mainframe performance at a much lower cost than traditional hardware. As such, few large-scale data-processing projects are undertaken without first evaluating parallel technology. For database vendors, it is essential that their database management systems exploit multi-processor hardware platforms if they are to survive in the commercial database market place. Indeed, all the major vendors have parallel *relational* database products including, Oracle, IBM, Informix, Sybase, Tandem and Teradata. A comprehensive features based

comparison of parallel database management systems and hardware platforms for these systems is available in a report by Bloor Research carried out in 1995 [166]².

Parallel database systems are often categorised by the way they share hardware resources such as memory or disk. Three categories can be distinguished; shared-memory (SM), shared-disk (SD) and shared-nothing (SN). In SM systems all processors share all disks and all memory. In SD systems, each processor has its own private memory but all processors have access to all disks. Oracle 7 is an example DBMS that uses a SD environment. In SN systems each processor has its own private memory and disks. The majority of commercial parallel databases fit into this category.

There have been many debates as to which architecture is most suited to parallel databases. In 1992 DeWitt and Gray [67] asserted that the shared-nothing architecture had emerged as the consensus for parallel and distributed system architecture. In an earlier paper Stonebraker [210] concluded that SN systems would have no apparent disadvantages when compared to alternative systems. Baru *et al.* [17] also expound the virtues of the SN architecture, perhaps not surprisingly, since they were the team responsible for IBM's DB2 Parallel Edition (now DB2 Universal Database Extended Enterprise Edition). Arguments in favour of the SN architecture include a theoretically lower hardware cost due to commodity components and the ability to scale-up to higher numbers of processors. Disadvantages include data skew where data is not balanced across disks (and processors), the need for distributed deadlock detection and a multiphase commit protocol. Complex software is required to split SQL statements into many subtasks to be executed on different processors and then to merge the results. Where possible this approach uses function shipping, that is, operations are performed where the data reside to reduce inter-processor communications. This architecture has an availability problem in the case of a disk or processor failure. In practice, multiply attached disks and replication, are used, much the same as in a SD environment.

² This report is not freely available but can be purchased from Bloor Research.

Rahm [191] and Valduriez [224] [223] have more recently advocated the benefits of a shared-something and shared-disk respectively. The software required to provide parallel database processing is considerably less complex for SM due to the global memory address space. SM provides easy load balancing. Fast inter-processor communications are possible since this is carried out in memory. Potential disadvantages include a memory bottleneck, especially as the speed of the processors increases and the number of processors is increased. Maintaining availability is also more of a problem than with SN and SD, in the case of a memory fault. Hardware cost is potentially higher due to the need to link each processor to each memory module. SD provides the possibility of easier load balancing, less communications overhead than SN, and nodes can more easily be partitioned for different functions e.g. for complex query or transaction processing. Software is more complex than SM due to the need for coordinated global locking and two-phase commit. Access to shared disk can become a bottleneck due to limited bus capacity.

Norman and Thanisch [165] propose that developments in technology now mean that distinctions based on hardware architecture are no longer so relevant when comparing performance of parallel architectures. Important factors include the way processes and threads are organised to cooperate in transaction and complex parallel query organisation and the sophistication of the optimiser. Nearly all of the commercial parallel database products now run on both Symmetric Multiprocessor (SMP) platforms in which processors share memory or Massively-Parallel Processor (MPP) platforms in which processors have private memory. This is a necessity since the trend in hardware design is to combine the two architectures so that an MPP platform can include multiple SMP nodes. Both MPP and SMP parallel relational database products are based on three simple techniques:

- *Table partitioning* - this involves distributing the rows of a table across multiple disk drives. Three basic methods are used; hash, range and round robin.
- *Pipelining* - Operators are overlapped so that the results of one operator are incrementally sent to the next operator in the execution plan.

- *Partitioned execution* - relational operators are replicated to increase I/O bandwidth available through partitioned tables.

In a 1996 Object-Relational summit presentation, Dewitt [66] described extending parallelisation of RDBs to include ORDBs as a challenging area for current database research. The techniques used to parallelise RDBs are not adequate for parallelising ORDBs. Problem areas include row valued attributes and collection attributes which can lead to skewed data distributions and storing and retrieving multimedia data e.g. partitioning individual images. It remains to be seen how commercial DBMSs will meet these challenges.

Parallel databases provide transparent parallelism from the user's point of view. Applications that run on sequential DBMSs can run unmodified on parallel versions of the products. For efficient performance database administrator effort is needed to adjust performance tuning parameters (such as memory allocation) and for deciding on the best data partitioning strategy.

Whilst a parallel database was used as the underlying database for the DBbrowse prototype (described in Section 4.2), the capabilities of parallel databases were not exploited for any significant part of this research. Both the DBbrowse and EASIA prototypes were designed to support relatively simple queries that might be associated with access to *metadata* relevant to archived scientific data. EASIA stores the actual scientific datasets external to the database, which frees the database of the resource intensive post-processing of scientific data (refer to Section 5.2.5).

2.1.4 Java Database Access

This section focuses on JDBC and SQLJ. Respectively, these provide dynamic and static SQL interfaces to relational databases from within the Java programming language. JDBC is used in the EASIA prototype due to the ad-hoc (i.e. dynamic) nature of the queries, posed by the scientific users, aimed at locating datasets of interest. EASIA also requires JDBC to discover schema information about the database at runtime.

2.1.4.1 Java Database Connectivity

The Java Database Connectivity (referred to as JDBC, although according to Sun this is a trademarked name not an acronym [137]) specification [232] is supported by all the major database vendors and allows open database connectivity directly from within Java. JDBC was added to Java version 1.02 in 1996. It consists of an API (found in the *java.sql* package of the standard Java API [132]) that contains a few implemented classes and many database neutral interface classes that specify behaviour without any implementation. Database vendors or other third parties provide the actual implementation of these interfaces in the form of *JDBC drivers*. Usually two initial statements in a JDBC application are used to firstly, register a JDBC Driver and secondly, to open up a database connection (using the *DriverManager* class from the JDBC API) using the previously registered driver. These two statements are often the only two that need to be changed to run the application against a DBMS from a different vendor.

The JDBC API is similar in concept to Microsoft's Open Database Connectivity (ODBC) [172]. The JDBC standard is based on the X/Open SQL CLI (Call Level Interface) [62], the same basis for ODBC. Applications talking ODBC to relational servers have reduced the need for writing embedded SQL. A CLI application does not require precompilation or binding but instead uses a standard set of functions to execute SQL statements and related services at runtime. Traditionally, precompilers have been specific to a particular database product. This requires source code to be written and compiled for each database product. Also embedded SQL applications have to be bound to a specific database before use. The CLI allows for portable applications that are independent of the database product and can be distributed in binary form.

ODBC is however not appropriate for direct use from Java since it is a C interface. Indirect usage of ODBC from Java, using calls from Java to native C code, has many disadvantages in the areas of portability, security, implementation and robustness. JDBC is designed to reduce these problems, and will not only allow applications which are independent of the database product but will also allow machine independent applications to be written. Whilst ODBC and JDBC are

designed to provide database independence, true portability still resides with the application designer. A driver must support at least ANSI SQL-92 (also known as SQL2) Entry Level to be called 'JDBC Compliant'. This gives applications that want wide portability a lowest common denominator. However, JDBC allows *any* query string to be passed to the underlying database, so that an application can use any database specific commands available, at the expense of reduced portability.

Currently, the JDBC specification also requires that selected semantics from the ANSI SQL-92 Transitional Level must be supported by drivers written for databases that support the SQL-92 Transitional Level. In view of this, JDBC supports a DBMS independent escape syntax for stored-procedures, scalar functions, dates, times and outer joins. A driver must convert the escape syntax into a DBMS specific syntax. The escape syntax is generally different to the SQL-92 syntax for the same functionality. In cases where all of the targets DBMSs for an application support SQL-92 syntax, the application designer can use this syntax. Finally, on the subject of portability, there are a number of JDBC metadata interfaces that provide information on the functionality of the target database. The application designer can use this metadata to provide different execution paths for databases that support different levels of SQL compliance. For example, the application designer can query the metadata to find out if the target database supports some form of outer join, and implement this in a different way if the database does not.

JDBC drivers may also support the JDBC Standard Extension API [233]. These include support for the Java Naming and Directory Interface (JNDI), connection pooling, distributed transaction support and *rowsets*. The JNDI can be used in addition to the JDBC driver manager to manage data sources and connections, which allows the application to be independent of a particular JDBC driver and JDBC URL. A *rowset* encapsulates a set of rows and may or may not keep an open database connection. The specification discusses several different types of rowsets. A *disconnected* rowset allows off-line updates to be performed and propagated to the underlying database using an optimistic concurrency control algorithm. Rowsets also add support for the Java Beans component model. A rowset object is a Java Bean and may be serialised. It is therefore a suitable container for

tabular data that can be passed between different components of a distributed application.

JDBC can be used for any database system for which a driver exists. This is not restricted to RDBs but includes ORDBs and even non-relational technology such as IBM's IMS. At the present time there are different ways to implement drivers that fit into one of four categories [136]:

1. *The JDBC-ODBC Bridge* provides JDBC access via most ODBC drivers. Note that some ODBC binary code and in many cases database client code must be loaded on each client machine that uses this driver, so this kind of driver is most appropriate on a corporate network, or for application server code written in Java in a 3-tier architecture.
2. *A native-API partly-Java driver* converts JDBC calls into calls on the proprietary client API. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.
3. *A net-protocol all-Java driver* translates JDBC calls into a DBMS-independent net protocol, which is then translated, to a DBMS protocol by a server. This net server middleware is able to connect its all Java clients to many different databases. The specific protocol used depends on the vendor. In general, this is the most flexible JDBC alternative. It is likely that all vendors of this solution will provide products suitable for Intranet use. In order for these products to also support Internet access they must handle the additional requirements for security, access through firewalls, etc., which the Web imposes.
4. *A native-protocol all-Java driver* converts JDBC calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Since many of these protocols are proprietary the database vendors themselves will be the primary source for this style of driver. Several database vendors have these in progress.

IBM's DB2 JDBC driver was used in the development of the EASIA prototype. IBM provides both type 2 and type 3 JDBC drivers for DB2. For Java *applications*

the DB2 Client software must be installed on the client making this a category 2 implementation (this driver was used in EASIA). For Java *Applets*, no DB2 code is installed on the client and a category 3 implementation is applicable. A DB2 server (or client) must be installed on the Web server machine along with DB2's JDBC Applet Server. When an Applet is downloaded, additional class files are downloaded associated with DB2's JDBC Driver. The Applet calls the JDBC API to connect to DB2, and the driver establishes communications with the database via the DB2 JDBC Applet Server on the Web server machine. (Although advertised as a type 3 driver this is not strictly the case since the Applet Server can only connect to IBM's DB2 database. Type 4 classification is also not strictly applicable either, since JDBC calls are not passed directly to the database server but to the Applet server.)

2.1.4.2 SQLJ

JDBC is becoming ubiquitous for relational and object-relational database access from Java. JDBC is primarily a *dynamic SQL* interface and does not require pre-compilation or binding to a particular database in advance. Dynamic SQL query plans are determined at run-time. This can be advantageous for DBMSs subject to frequent updates since the latest database statistics can be used for query plan optimisation. However, for DBMSs in which databases statistics do not change significantly, *static SQL* can provide a performance advantage because query plans are determined ahead of execution, during pre-compilation.

JDBC can, however, offer some of the potential performance advantages of static SQL through prepared SQL statements. An SQL statement containing host variables can be prepared before execution and can then be executed multiple times, with different values for the host variables. This can give a performance advantage because the database will compute the execution plan for the query only once, when the statement is prepared and subsequent execution of the query will use the same plan. JDBC method calls to prepared SQL statements can only be executed at run-time. Therefore, although the execution plan will only be computed once this computation will still occur at run-time. True static SQL allows the embedded SQL statements to be pre-compiled and the program can be bound to a particular database

ahead of execution. The optimised query plan can therefore be determined at this time. Static SQL offers several other advantages to the code developer:

- *Syntax checking* - of SQL statements.
- *Type checking* - to ensure that data exchanged between the host language and SQL have compatible types.
- *Schema checking* - to ensure that the SQL statements are compatible with the target database schema.

ANSI and ISO standards exist for Embedded SQL within the C, COBOL, FORTRAN and ADA languages amongst others, and in April 1997, Oracle, IBM and Tandem jointly proposed, SQLJ - Embedded SQL for Java (known at the time as JSQL). SQLJ Part 0 (Embedded SQL for Java) has been accepted as an ANSI standard and will form Part 10 of SQL:1999 known as SQL/OLB [61] (see also Section 2.1.2). Although SQLJ is aimed at providing a static SQL binding from Java, the standard layers SQLJ upon JDBC such that an SQL/OLB compliant implementation also provides access to JDBC features.

SQLJ Part1 (Java Stored Routines) and Part2 (Java Data Types) are also undergoing standardisation, though not as part of the SQL standard. For further details, see for example [208] [80] and [81].

2.1.5 Microsoft's Data Access Strategy

The previous section mentioned that most commercial database vendors provide JDBC drivers for their products. One notable exception is Microsoft. (However, some third parties provide JDBC drivers for Microsoft databases, and the JDBC-ODBC Bridge can be used to access a Microsoft DBMS). Microsoft has a strategy, known as *Universal Data Access* (UDA) [222], for providing access to database and non-database information across the enterprise. UDA provides data access services to Windows *Distributed interNet Application* (DNA) Architecture [236], which is Microsoft's overall strategy for building scalable, distributed, multi-tier Internet based, client/server applications.

UDA provides access to a variety of information sources, including relational and non-relational, and a programming interface that can be used with many (Windows based) languages and tools since it is based on Microsoft's COM (Component Object Model) component technology [56] [50]. UDA is implemented through the following technologies: OLE DB [173] (a system level interface), *ActiveX Data Objects (ADO)* [5] (an application level interface that is easier to use than to OLE DB and which can be used by any language or tool that can use COM) and ODBC [172]. Microsoft uses yet another acronym, MDAC (*Microsoft Data Access Components*), to describe the packaged release of these technologies.

Whilst ODBC has been very successful for Microsoft, it was designed for relational databases. OLE DB, on the other hand, defines a collection of COM interfaces for accessing relational data, ISAM/VSAM mainframe data, hierarchical databases, email, file system stores, and more. ADO will eventually replace ODBC. However, currently an OLE DB/ODBC bridge is available for databases that do not have a native OLE DB driver.

Microsoft has produced a paper comparing ADO with JDBC [7]. The paper (not surprisingly) criticises JDBC in a number of areas including the fact that JDBC is a low-level API really only suitable for relational data sources. OLE DB, on the other hand, can access many different data sources. Also OLE DB can be used in many different languages and tools since it is based on COM components. Whilst other vendors are building universal databases with new datatypes to centralise non-traditional datatypes, Microsoft is building OLE DB components to interface to the data in its original form. A useful overview of Microsoft's data access strategy is available in [144]. OLE DB was rejected as an implementation technology for this research as it is vendor and platform specific (requiring an underlying COM-based architecture). Also, although JDBC is a low-level API it is very easy to use. During the course of this research JDBC proved to be extremely versatile for connecting to different vendor's databases on different platforms.

2.2 Web Developments

The three prototype systems described in this research all provide Web-based management of non-traditional data. Each system demonstrated new mechanisms

and architectures. These 3 systems chart a progression in the usage of increasingly sophisticated technologies for creating dynamic/interactive Web pages and for providing Web/database integration. The previous section discussed the variety of available database technologies. This section discusses the Web technologies that were available, and describes why different technologies were used to implement the prototypes.

Initially, data for Web pages was stored in conventional data files containing static links to other files. There are still many Web sites that are constructed in this way. However, increasingly Web sites now produce dynamic Web pages in response to users' requests. There is huge effort both from industrial and research institutions devoted to developing tools and techniques for dynamic Web-based client/server systems. Often Web pages are no longer based on conventional files but are built from data extracted from database management systems. This section discusses the technologies that have enabled the transition from the static Web to the new dynamic, interactive Web.

2.2.1 The Common Gateway Interface

Initially the Web consisted of pre-written HTML pages, containing text, images and fixed links to additional pages. The CGI transformed the static Web by providing one of the earliest techniques for generating dynamic Web page content. With CGI, a Web server passes certain Hypertext Transfer Protocol (HTTP) [93] requests to an external program residing on the Web server. The output of this program is then returned as an HTML page to the client's browser. In addition to providing dynamic content, CGI also allowed for user interaction via HTML forms.

CGI is still the most widely used mechanism for server-side processing [121]. This is because the CGI approach has a number of benefits including ease of implementation, portability of server software, the use of standard Web browsers as clients and the existence of a wealth of existing tools and sample code. A CGI program receives input from the client's browser, via interaction with the Web server, by reading environment variables and or standard input, and provides HTML page output, via interaction with the Web server, by writing to standard output. This simplicity allows CGI programs to be written in any language, although PERL has

become the predominant choice [124]. The simplicity of CGI leads to a number of well-known limitations (see for example [124] [74]):

- *Sessions Problem*: In most client/server systems the client stays connected to the server through multiple transmissions. However, with CGI, once a request is handled the CGI program terminates, closing down any communications channel with the server. The underlying HTTP protocol is stateless and is not designed to maintain state between multiple requests from the same client. Crude mechanisms can be employed to maintain state, including hidden variables in HTML forms and Netscape's client state cookies [184].
- *Server Load/Scalability*: The CGI mechanism starts a new process every time a request is made that accesses a CGI program. For CGI programs written in PERL, this performance degradation was magnified in initial CGI implementations since each request also had to start a new PERL interpreter. A further consequence of server-side processing is that work that might usefully be done by the client (such as form validation) has to be emulated by CGI programs on the server, further increasing the workload on the server.
- *Slow*: Web browsers cannot send requests to the server asynchronously, performing other work while the request is processed. Clients wait for the server response which includes the time for the server to start the CGI process.
- *Limited Presentation Capabilities*: The presentation of the user interface and results from queries are restricted by the limitations of HTML.

Both the GBIS and DBbrowse prototypes were implemented using CGI technology largely because this was the emerging mechanism for implementing interactive HTML pages at the time. GBIS used UNIX shell scripting for implementing the CGI programs, whilst DBbrowse was written using PERL CGI programs. DBbrowse performed a new database connection for each query, even for subsequent queries submitted by the same user. This leads to performance degradation. Connecting to a database is typically very slow, often requiring several seconds to login a user [175]. Repeated database login also limits the client/server functionality, precluding

session-oriented database-applications. DBbrowse did not allow the use of an SQL cursor to retrieve a subset of the rows in the answer table, with user interaction to request further rows from the database. This problem is inherited from the statelessness of the HTTP protocol. Lack of state also lead to repeated queries for metadata in DBbrowse.

The EASIA prototype does not use the CGI mechanism. Instead EASIA uses Java Servlets (see Section 2.2.5) to remove the process per request overhead associated with CGI, to maintain state between requests and to maintain database connections for a complete client session. EASIA also uses some basic *Dynamic HTML* (DHTML) features to allow some server side-processing and more advanced presentation. Before discussing the reasons behind choosing Java Servlets for implementing EASIA, a few other alternatives to the CGI mechanism are discussed. Many technologies have been developed to overcome the limitations of CGI, but in turn these can exhibit other disadvantages.

2.2.2 Web Server Extensions

To overcome the performance problems associated with process creation using CGI, Web server vendors provided APIs to allow extensions to the Web server itself. Microsoft's Web server extension API is known as ISAPI [129] (Internet Server Application Programming Interface), Netscape's is known as NSAPI [167], whilst Apache provides the Apache API [13]. Applications built using these APIs run in the same process as the Web server so that communication between the application and the Web server is very fast. Additionally, applications remain in memory once loaded. However, there are a number of problems associated with Web server extension APIs. Firstly they are proprietary, tying the application to a particular Web server vendor. Also, development with these technologies is complex. ISAPI, for example, is only accessible from C++. Wizards are available to help create the framework for ISAPI code, however building on the framework is not simple [189].

To simplify development, whilst improving the performance of CGI programs, a number of products have been created on top of the Web server APIs. A module called *mod_perl* [186] is available for the Apache Web Server, which embeds a PERL interpreter into memory so that this only has to be done once at

initialisation time. In addition, each PERL CGI program is only compiled once and then kept in memory to be used each time the program is run. This module is only available for the Apache Web Server.

ActiveState's PerlEx [4] also improves the performance of PERL CGI programs running on several popular Web servers (including those from Microsoft, Netscape and O'Reilly). It uses the Web server's native API to accomplish this. PerlEx is, however, only available for Web servers running on the Windows NT platform.

Due to Web-server and language portability issues, Web server extensions were not considered a suitable technology for implementation of the prototype systems created during this research.

2.2.3 FastCGI

Open Market's *FastCGI* [90] is another attempt to deal with the performance limitations of CGI. It consists of a specification [26] along with freely available source and object code for extending Web server products. FastCGI uses persistent CGI processes that are reused to handle multiple requests to remove the overhead of creating a new process for each CGI request. Although processes are reused there is still at least one process for each FastCGI program, and in order to handle multiple concurrent requests for the same program, a pool of processes is required. Another problem with FastCGI is that it is not implemented for some of the most popular Web Servers, including Microsoft's Internet Information Server [124].

2.2.4 Java and Java Applets

In 1995 Sun Microsystems launched the Java programming language. Described as "write once, run anywhere" Java portability relies on compiling source code to byte codes for a virtual machine (specified by Sun Microsystems). The byte code is then interpreted on any platform using a platform specific Java Runtime Environment (JRE). Before discussing the implications of this for Web development a brief note on the ownership of Java follows.

Java is owned by Sun Microsystems and is licensed to third parties. In April 1999 [135], Sun Microsystems proposed to standardise formally Java technology

through ECMA [77], an internationally recognised standards developing body with strong ties to ISO. The proposed submission would consist of the Java 2 platform, Standard Edition (J2SE), Version 1.2.2 specifications. These consist of the following technology specifications:

1. Java Language Specification (with Clarifications and Amendments) [105].
2. Java Virtual Machine Specification [145].
3. Java 2 Platform API Specification [132].

However, on December 7, 1999 Sun issued a press release announcing the withdrawal of the proposal from ECMA [213] which stated, “Sun is withdrawing from the process in order to protect the integrity of the Java technology and the investment made in it by the worldwide community using Java technology”. The article goes on to state that Sun is committed to maintaining compatibility across implementations of the Java platforms and that they encourage the community to compete on implementation, not on standards. The press release also states that Sun noted that ECMA has formal rules governing patent protections but that there are currently no formal protections for copyrights or other intellectual property. Currently, therefore, Java remains a de facto standard.

Two fundamental categories of Java programs are Java applications and Java Applets. Java applications are standalone programs that can be run using a JRE and which have similar properties to programs written in other languages. Java Applets on the other hand, are designed to be downloaded and run inside a Web browser using an embedded JRE. Java Applets introduced a fundamentally new concept to dynamic Web pages. Instead of generating the pages server-side it is now possible to download executable content, in the form of Applets, to be run within the user’s Web browser. Java Applets therefore provide an alternative philosophy to the proprietary and non-standard CGI workarounds discussed in the previous sections. A Java Applet can be downloaded and run on the client. A Java Applet can make its own network connections using Java sockets, or can employ technologies such as JDBC and/or distributed object technologies (such as CORBA, see Section 2.2.7) for communicating with server-side databases and application logic. These technologies

can eliminate the bottleneck imposed by the CGI on the server and provide session-oriented communications. Additionally, sophisticated graphics are available via Java's Abstract Window Toolkit (AWT) class library.

Due to the security implications of allowing downloaded executable content to be run on a user's machine, Java Applets do however, run in a sandboxed environment. Generally speaking, Web browsers restrict Java Applets by:

- Preventing an Applet from running any external executable program.
- Preventing an Applet from reading or writing to the local file system.
- Preventing an Applet from communicating with any server other than the host from which they were downloaded (the *originating host*).
- Ensuring that an Applet attaches an Applet warning message to any window that it creates (to prevent, for example, the user from inadvertently typing in a password in what appeared to be a window from a local standalone application, but which might, in fact, be sent to the originating host).
- Preventing an Applet from accessing any local information except for the Java and operating system version, and the characters used to separate files, paths and lines.

In addition, a bytecode verifier ensures that all class files obey the rules of the Java language, which enforces, for example, memory protection (this is to prevent damage from Java classes bytecodes constructed by hand or by a non-compliant compiler).

It is now possible to relax some of these restrictions through available options in current Web browsers. Also, digital signatures provide authentication of the provider of Java Applet classes thereby allowing a user to grant very specific extended privileges to individual classes or signers.

During the construction of the DBbrowse and EASIA prototypes Java Applets were evaluated as a suitable implementation technology. However, Java Applets were rejected for a number of reasons.

1. Applets have restrictions associated with network access to multiple hosts, and access to the local environment as discussed above. These restrictions were too limiting for the EASIA application, which needs to connect to multiple file server hosts, and which is required to allow users to save and results to the client machine. Paepke *et al.* review technical challenges faced during construction of the Stanford Digital Library [178]. They state that Java security managers and their interaction with browsers were a constant source of trouble. The speed at which Java is developing leads to constant revisions and incompatibilities.
2. It was difficult to build robust Java Applet software for an environment where users will have browsers from different vendors and at different release levels. Applets really need to be tested on all possible client platforms, and even then new versions of browsers may introduce new incompatibilities [178] [124].
3. The Java Abstract Windowing Toolkit (AWT), the basis for graphical user interfaces in Java, is, according to Hunter and Crawford [124], the most error-prone and inconsistently implemented portion of the Java language. Although 'SWING' now provides an all Java class library for user interface components, which may prove to be more robust, SWING support in users' browsers cannot be relied on.
4. Often the response times of JAVA GUIs proved to be slow in execution and slow to download. This is an important consideration when there is no control over the client machines of the users.

Although Java Applets were not used for the prototypes, Java was used server-side. The portability of Java code and the extensive APIs (or class libraries, for example JDBC) provided great benefits. Java *Servlets* were used to implement server-side logic in the EASIA prototype. Servlets are discussed next.

2.2.5 Java Servlets

Version 1.0 of the Java Servlet API was released in mid 1997. A Java Servlet [63] is a dynamically loaded Java class that extends the capabilities of a Web server. Once loaded a Servlet remains in memory, and is handled by separate threads within the Web server process. This provides a much better performance than CGI, which creates a new process for each request. Orfali and Harkey [175] found that Servlets performed over an order of magnitude better than CGI in a ‘ping’ benchmark.

Servlets are written in Java and, unlike the previously discussed non-standard CGI workarounds, are supported by all major Web servers. Servlets were unavailable when the GBIS and DBbrowse prototypes were implemented. However, Servlet technology was used to implement EASIA. The decision to use Servlets allowed the system to be portable across operating systems and Web servers, and allowed access to the full range of Java APIs such as JDBC and the security APIs (see the discussion of code upload for server-side execution Section 5.2.6).

Servlets also provided a mechanism to maintain state for the duration of a user’s login session and to invalidate a user’s session and log a user out after a period of inactivity (by storing a stateful last modified variable).

Since Servlets are a server-side technology they do not provide a means of enhancing the user interface presented in the client’s Web browser. EASIA still uses HTML forms as the main user interface technology, enhanced with some browser independent DHTML.

2.2.6 Java Server Pages and Active server Pages

Sun has also released a technology called JavaServer Pages (JSP) [134]. This is very similar to Microsoft’s Active Server Pages (ASP). One main difference is that JSP uses Java as the language that is embedded within the HTML pages. However, unlike Java Applets, which execute client side, the embedded Java code is executed server-side (as with ASP) and a standard Web page is returned to the client. Also JSP is designed to work with different Web servers and on different platforms.

Sun state that JSP technology was designed to try and provide an industry-wide solution for creating Web pages with dynamic content. JSP is designed to be simpler than Sun's Servlet technology, thereby reducing the level of expertise required to build applications. JSP, like ASP, can separate application logic from Web page content. So, for example, the appearance of a page can be changed without modification to the application logic. A Servlet application, on the other hand, requires that the entire Servlet be edited and recompiled for such a change.

Behind the scenes, JSP pages are converted to Servlets and compiled the first time that they are requested to improve performance for subsequent requests. Since JSP pages are compiled into Servlets they benefit from Java security and memory management facilities. Finally, JSP pages can interact with Java Beans Components to perform complex processing. JSP was not employed during the implementation of EASIA. This was partly due to the fact that EASIA was started before JSP became widely available and partly because it proved to be straightforward to implement EASIA using Servlets directly.

ASP [3] technology was developed to run transparently on top of ISAPI. ASP can provides memory resident, multi-threaded server-side applications whilst offering simpler development than ISAPI applications. ASP is a similar technology to JSP, allowing Web pages to include embedded scripting instructions (using for example, Microsoft's JScript and VBScript languages) along with other HTML tags. When Microsoft's IIS (Internet Information Server) Web server first gets a request for a particular ASP page it compiles the script in the Web page and loads the compiled code into memory. The script then performs some server-side processing and writes a standard HTML page back to the client. Underneath the covers, ASP uses a Microsoft supplied ISAPI DLL that runs within the same memory space as the IIS Web server to process the Web pages.

Originally ASP simply provided server-side scripting. Now however, ASP is one of Microsoft's mainstream technologies for Web-based application development. It is now tightly integrated with other Microsoft technologies (see for example [189]) such as ADO, COM and MTS (Microsoft Transaction Server) [151]. Third parties have ported ASP technology to other platforms. As with other

Microsoft technologies, ASP was not used for any prototypes in this thesis as it generally restricts the architecture to the Microsoft Windows platform. Some attempts have been made to port ASP to other operating systems, for example, Chili!soft's *Chili!ASP* [58] provide a comprehensive commercial product range which allows ASP to operate on Web servers other than IIS and on alternative operating systems to Windows NT.

2.2.7 Distributed Object Technologies

At the same time that Applets were investigated as a possible implementation technology for client-side processing, distributed object technologies were evaluated as a means for the clients to communicate with server-based parts of the applications. Distributed objects have the potential to allow Java Applets to communicate with the server using a higher level of abstraction than alternative mechanisms such as a raw socket connection employing a user-defined protocol, or HTTP (and CGI) based communications. The result of the evaluation was to reject distributed object technologies as a possible implementation technology for EASIA, largely due to the previously discussed problems associated with the client-side Java Applets. However, this section provides a brief description of some of the issues presented by the distributed object technologies.

The need for better client/server performance in the Web environment lead to resurgence in the popularity of distributed object technologies, particularly in combination with Java. During the evaluation of these technologies a simple client/server 'ping' benchmark was run which showed distributed object technologies to perform two orders of magnitude better than the CGI mechanism with a performance level within the same order of magnitude as raw socket programming. The details of the experiment are given in Appendix B.

The major database and middleware vendors have all been active in the distributed object arena. The principal distributed object technologies are CORBA (Common Object Request Broker Architecture) from the OMG [53], DCOM (Distributed Component Object Model) from Microsoft [27] and RMI (Remote Method Invocation) from Sun [133]. Their promise is to provide an infrastructure for distributed computing which allows the invocation of methods on objects just as

if they were part of the local application, whereas the objects can actually be located anywhere on a network. That is, they are intended to provide local/remote transparency so that the developers do not have to worry about factors such as transports, server locations, object activation, target operating systems, etc. A brief overview of the features of these technologies follows (concentrating on CORBA as this formed the bulk of the evaluation) along with a general discussion of why they have not succeeded in the Web environment and a mention of the new direction these technologies are taking in middleware component architectures.

2.2.7.1 CORBA

CORBA is an open standard overseen by the OMG, a consortium of over 700 companies within the computer industry. CORBA's architecture is built around three key building blocks:

- OMG Interface Definition Language (IDL)
- The Object Request Broker (ORB)
- The standard Internet Inter-Orb protocol (IIOP)

The key to CORBA is that CORBA objects have well defined interfaces that can be expressed in *OMG Interface Definition Language* (IDL). IDL is a simple declarative language used to define object types by specifying their interfaces. IDL syntax is similar to C++ or Java, but it does not contain any programming constructs. An interface definition consists of:

- *operations* - method signatures
- *parameters* - arguments of operations - **in** (client to server), **out** (server to client), **inout** (both ways)
- *attributes* - instance or class variables - 'get' and 'set' methods must be supplied for each attribute (get only for read-only attributes)
- *exceptions* - exceptions that operations may *raise*

An IDL interface can also specify inheritance from parent interfaces, and typed events that it emits. An important feature of CORBA is that it is language neutral.

Clients and servers can be written in a number of different languages, including Java, C, C++, Smalltalk and ADA, and bindings for other languages such as COBOL are in the process of being specified. Clients written in one language are able to communicate with servers written in a different language. It is possible to implement the same interface in multiple objects.

IDL is used to map CORBA objects into particular programming languages through the use of IDL compilers. An IDL compiler for the Java language automatically produces a *stub* class for the client, a *skeleton* class for the server and a *Java interface* class that corresponds to the IDL. The stub and skeleton classes glue the actual client and server code to the ORB. The ORB is the middleware that handles the client/server interaction. An API is defined to allow client/server object interaction with the ORB.

The stub class provides local proxy objects that the client can invoke methods on. The methods in the stub proxy object in turn invoke operations on the real object implementation, via the skeleton on the server. Once the client stub has been instantiated on the client using an object reference to a CORBA server object, standard Java code is used to invoke methods on that object.

During a client request the stub automatically builds a block of information that identifies the object and method to be used, and which contains the parameters to be sent. This block of information is packaged in a device-independent manner. This is known as *parameter marshalling*. The skeleton class at the server unmarshalls the parameters. It directs the operation request to the appropriate method of the correct object implementation. The skeleton class then captures the return value or exception and sends this, in marshalled form, back to the stub on the client.

Two classes usually have to be written for the server side of the request. These are referred to as the *server* and *servant*. A *servant* object is an instance of the object implementation, i.e. code that implements the methods specified in the IDL interface. The programmer will add the body of the operations defined in the IDL (and Java interface) and also constructors for the object.

A *server* object can be started manually (at the command line) and it then instantiates a servant object (or multiple servant objects, possibly of different types). The server code also initialises the ORB environment and registers the available servant objects with the ORB.

The CORBA1.1 specification (introduced in 1991) concentrated on producing portable object applications by defining CORBA IDL and the CORBA API. However CORBA implementations from different vendors were not interoperable. CORBA2.0 [53] (first adopted in 1994) includes the specification of inter-ORB interoperability, known as the General Inter-ORB Protocol (GIOP). This protocol defines the message format for invoking operations on CORBA objects. One mandatory Inter-ORB protocol (IOP) that must be implemented by CORBA2.0 compliant ORBs is the *Internet Inter-ORB Protocol* (IIOP) which uses TCP/IP as its transport protocol. CORBA2.0 also allows an object reference to be used by a client using any compliant ORB, through the use of *Interoperable Object References* (IORs).

In the Web environment CORBA clients are subject to security restrictions. Applets that invoke operations on CORBA objects are limited to opening network connections back to the host from which they were downloaded. Another restriction occurs in the case of firewalls which do not permit TCP/IP based IIOP communications to cross them. A method often used to overcome these problems is known as *HTTP Tunneling*. IIOP messages are placed in a HTTP wrapper which enables them to pass through firewalls. The messages are sent back to the originating host of the applet and a daemon process on the host machine forwards the CORBA request to the machine that hosts the required object.

This completes a brief overview of some of the important features of CORBA. This overview provides details that allow comparisons with DCOM and RMI in the next two sections, followed by a discussion of why these technologies have not superseded HTTP as the protocol for Web communications, and how they might, instead, succeed in middleware component architectures. For more detailed comparisons of these distributed object technologies, see for example, [45] [175]

[187] [192] [215] and for excellent links to information on CORBA in general, refer to [41].

2.2.7.2 DCOM

DCOM is Microsoft's distributed object technology to compete with CORBA. DCOM extends Microsoft's COM from the desktop to the network allowing objects to communicate over the Internet. It is best to consider COM and DCOM as a single technology that provides a range of services for component interaction, from services promoting component integration on a single platform, to component interaction across networks. In fact, COM and its DCOM extensions are merged into a single runtime. This single runtime provides both local and remote access. COM is both a specification and an implementation that specifies a binary standard for implementing objects. The implementation part is a dynamic link library that includes API function calls. These can be used to instantiate an object and give it a unique ID. COM specifies how the objects can be instantiated and how they can communicate locally using the predefined interfaces that they implement. DCOM uses object-oriented remote procedural calls to extend COM, and is built on top of the Open Software Foundation (OSF) Distributed Computing Environment (DCE) Remote Procedural Call (RPC) [177].

DCOM, like CORBA, uses an interface definition language. However, these are not the same. A DCOM IDL file contains interface definitions, which are divided into an interface header and an interface body. The interface header contains details applicable to the whole interface. The interface body contains items such as function prototypes and pre-processor directives. CORBA IDL is far more succinct and self explanatory than DCOM IDL. A DCOM Interface is a group of related functions held in an array of function pointers known as a *virtual table* or *vtable*. The table points to the implementations of the interface functions. A client receives a pointer to the interface, but cannot instantiate an instance or create a unique DCOM object and hence cannot maintain state between connections. A client can only reconnect to an interface pointer of the same class. Hence DCOM has transient stateless objects compared with CORBA's persistent objects and object references.

DCOM is primarily a Windows based technology. Language bindings are available for all Microsoft development environments including Visual C++, Visual Basic and Visual J++. In contrast to CORBA, which is an open standard overseen by the Object Management Group (a consortium over 700 companies from within the Computer Industry), DCOM is proprietary. In 1996 Microsoft announced that it would hand over its object technology specification to the *Active Group* [2]. The Active Group consisted of software vendors and system vendors and was to be directed by a steering committee including. The aim of the group was to determine a process to transfer ActiveX specifications and appropriate technology to a standards body. However, subsequently Microsoft has quietly abandoned its 1996 commitment to hand over ActiveX to an independent body and the Active Group has also vanished along with this promise.

COM/DCOM has, however, been ported to a number of other operating systems by Microsoft and other organisations (see for example [51] [52] [86]) although support for many of the technologies built on top of COM/DCOM is very limited on these other platforms. A report by the OMG states that Microsoft Windows is always likely to be the reference platform for DCOM and that “non-Windows versions will always have second-class status” [54]. The report quotes Bob Muglia, the Vice-President of Microsoft's Developer Tools Division, as saying, “Microsoft unapologetically will make sure that ActiveX works best on Windows”. (ActiveX is a core Microsoft Internet technology based on COM/DCOM.)

2.2.7.3 RMI

RMI is a set of Java Classes that Sun first included in JDK1.1. When introduced, RMI was a non-CORBA compliant ORB restricted to use only from within the Java Language. This allows RMI to be optimised for Java usage. However, it prevents RMI from being directly incorporated into existing software written in other languages. It may also be a problem in applications where the execution time of interpreted Java byte code does not provide sufficient performance.

The main ways in which RMI differed from CORBA when it was first introduced are as follows:

- *No IDL* -- RMI uses Java *interfaces*, directly, to specify the interfaces of remote objects. These interface definitions contain the method signatures of all remote methods. On the server side a class is created to implement this interface. An RMI compiler is used to generate client stub and server skeleton classes using the byte code of this interface implementation class. This class can then be instantiated by a server program to create remote objects.
- *Objects can be passed by value* -- Since RMI uses Java interfaces and not IDL to define interfaces it is possible to include method invocations that reference local objects. Clients and servers can therefore pass one another objects for which no stub and skeleton classes exist. Additionally, it is not possible to simply pass an object reference, as would be the case for an object passed as a parameter to a local method, because object references are memory locations of objects which are valid only within the local Java virtual machine. RMI solves this problem by passing the actual value of the object instead of just a reference. There is therefore no further connection to the original object.
- *URL based object names* -- RMI allows remote object references to be stored using an URL-based naming scheme which can be very useful in the Internet environment.
- *Dynamic stub downloading* -- Clients can download stub classes from the server, dynamically as required, if they have a reference to a remote object and do not have the stub locally.

Over the last couple of years the differences between RMI and CORBA have become fewer. RMI can now be implemented on top of IIOP thereby allowing a level of interaction between RMI and CORBA objects. Also, the CORBA specification now supports objects by value.

2.2.7.4 Distributed Objects Have Not Succeeded in the Web Environment

Despite the promise of distributed objects they have not succeeded in the Web environment. HTTP is the protocol of the Web, not IIOP. There are a number of

reasons for this lack of penetration on the Web. Firstly, these technologies are complex to use and the idea that applications can be built without regard to whether objects are local or remote is currently false (see for example [227]). Second, they require significant runtime support to operate properly. Third, these technologies were not designed explicitly for the Web, with consequences such as their inability to traverse firewalls without specific workarounds. Fourth, the mechanism by which they tightly couple applications to interfaces is too rigid to cope with change. For example, CORBA IDL is used to produce binary programs for use as stubs and skeletons at the client and server. A change to an IDL defined interface requires regeneration of the stub and skeleton and consequential changes (code update, recompilation, redeployment) to the programs that use them. If these technologies are being used to bridge applications between different organisations, then exact agreement is required on the interfaces being used. The rest of this section reviews a number of books and papers that add weight to the above argument.

In 1997, Orfali and Harkey [175] asserted that the Web was on the verge of a distributed object revolution, with ubiquitous deployment of Java and CORBA. In their ‘Object Web’ environment, clients would dynamically discover and use objects made available through CORBA’s trading service. Clearly this *Object Web* has not materialised. The authors also quoted Marc Andreessen, the cofounder of Netscape, making the following prediction in 1996:

“The next shift catalyzed by the web will be the adoption of enterprise systems based on distributed objects and IIOP (Internet Inter-ORB protocol)”.

...

“We expect to distribute 20 million IIOP clients over the next 12 months and millions of IIOP-based servers over the next couple of years”.

If these *distribution* targets have been achieved then it is solely due to CORBA software being bundled with Netscape products. However, in terms of actual deployment, the numbers are orders of magnitude out.

Tallman and Kain [215] quote Gary Voth, Microsoft Group Manager for Marketing as saying in 1998, “There are only 50,000 or 60,000 deployments of CORBA around the world”. What Voth did not mention, is that there are even fewer

enterprise solutions using COM. Tallman and Kain state that they tried to find references for COM projects but were unable to identify much comparable experience. Whilst significant COM development occurs at the desktop and departmental level, it is currently an unproven enterprise technology.

Box [22] compares the suitability of Java, CORBA, COM/DCOM and XML as technologies for enabling component software that can provide collaboration and cooperation amongst software development organisations. He believes that it is unlikely that Java, CORBA or DCOM will dominate the Internet. Box states “Ironically, while Microsoft and the Object Management Group (OMG) were arguing over whether the Internet would be run on DCOM or CORBA, the Hypertext Transfer Protocol (HTTP) took over as the dominant Internet protocol”.

A news report in *Computing* [91] mentioned the heavy bias towards XML in an early preview of Microsoft’s *Developer Studio 7* and made the following statement concerning DCOM: “In essence, Microsoft is replacing the DCOM RPC messaging technology with an XML/HTTP technology that allows for remote method invocation”.

CommerceNet’s eCo System initiative [103] originally used CORBA to define business services. However, this was abandoned in favour of XML. The authors suggest that while CORBA appears workable within organisations that control APIs, it is not practical for inter-enterprise integration. Business documents, defined in XML provide a simpler, human readable, intuitive way for businesses to interoperate. Businesses already exchange information via documents on which they largely agree, whereas programming APIs for business system interfaces almost certainly differ. The role that XML has assumed in the Web environment is discussed in more detail in Section 2.2.8, but first a brief description of emerging middleware component architectures follows as these could be the environment in which distributed objects will succeed.

2.2.7.5 Server-side Component Architectures

Before leaving the discussion of distributed object technologies it is worth noting that new middleware component architectures have arisen in the last couple of years that may provide a lifeline for these technologies. Therefore, although distributed

objects have not proliferated in the Web environment, they may still succeed as the communication mechanism of choice within organisations. Sun's Enterprise Java Beans (EJB) architecture [148] (based on CORBA and RMI) and the Microsoft Transaction Server architecture [151] [216] (based on COM/DCOM) promise 'plug and play' *enterprise* computing (which effectively translates as *distributed* computing) features. (Note that EJB is an entirely different concept to JavaBeans technology [110] which allows visual program composition without the need to access source code. Programs built from JavaBeans are designed to be run in a single java virtual machine.) The idea is to provide a level of abstraction, which can virtually free the developer of any middleware expertise when building scalable, transactional, distributed applications. Rather than writing to middleware APIs (as was previously the case with CORBA and DCOM applications) components gain middleware services implicitly and transparently. These transparent services include transactions, persistence, security, state management, component lifecycle, threading, resource-sharing and more.

EJB and MTS also fit within wider enterprise computing strategies from Sun and Microsoft, embodied in Java Enterprise Edition [202] and Windows *Distributed interNet Application* (DNA) architecture [236] respectively. Java Enterprise Edition incorporates EJB, RMI, JDBC, Java Naming and Directory Interface (JNDI), Java Transaction API (JTA), Java Transaction Service (JTS), Java Messaging Service (JMS), Servlets, JSP, Java IDL, JavaMail, Connectors and XML. The Windows DNA architecture incorporates many Microsoft technologies, centred on its COM, for building scalable, distributed, multi-tier Internet based client/server applications. The DNA approach involves 'plugging together' COM components for developing all tiers of distributed applications, including, user interface and navigation, business logic, and storage. Developers can combine many different COM aware languages (C++, Java, Visual Basic, JScript, VBScript), tools and services to create applications. DNA services include, amongst other things, transactions (MTS), component management, Dynamic HTML, Web browser (Internet Explorer (IE)) and server (Internet Information Server (IIS)), scripting (JScript, VBScript), message queuing (Microsoft Message Queue Server (MSMQ)), security, directory services (Active Directory), data access (UDA, SQLServer), systems management,

and user interfaces. This list appears to be fairly flexible in different Microsoft publications, but in general it is fair to say that it incorporates most of Microsoft's core technologies.

Further comparison of EJB and MTS is available in [193] and Microsoft's view of the advantages they see in MTS over EJB is available in [55]. It is too early to say if either of these strategies will be successful, but the usual 'Microsoft Windows only' caveats apply to MTS/DNA. Chang and Harkey [43] provide words of scepticism aimed at the transparency that middleware component architectures may provide (from very complex underlying technologies and interactions), encapsulated in their statement, "It will be quite a feat to figure out the reason when something goes wrong. It will likely be a feat to explain when something goes right in terms of expected results and performance".

2.2.8 XML and Dynamic HTML

Currently XML (Extensible Markup Language) is an important and rapidly expanding technology area associated with the development of the Web. XML is discussed in this section along with DHTML, the DOM (*Document Object Model*) and JavaScript since there is significant overlap between these technologies. For example, two of the components of DHTML include the DOM and JavaScript. Also the DOM specification defines a JavaScript binding that can be used to manipulate XML (and HTML) documents.

In this research XML was used in the EASIA architecture, firstly as a means of specifying the content of the user-interface without specifying its representation, and second, to define interfaces to external server-side applications that can be incorporated into the EASIA scientific dataset post-processing capabilities. JavaScript is used in the EASIA user interface to provide a more dynamic feel than can be achieved with HTML forms alone.

2.2.8.1 XML

Until now HTML has been *the* language of the Web. HTML is a fixed markup language that is, in fact, an application of the *Standard Generalized Markup Language* (SGML) [127] [57]. However, HTML has a number of limitations. XML was designed to provide a new markup language for the Web without the limitations

exhibited by HTML. XML arrived in late 1996 and finally reached maturity as a *World Wide Web Consortium* (W3C) [238] Recommendation in February 1998 [23]. (The W3C is an international industry consortium founded in October 1994 to guide the development of the Web, by providing a repository of information, specifications, reference code implementations, prototypes and sample applications, amongst other things A *Recommendation* is the highest level a specification can attain within the W3C.)

XML is a *subset* of SGML. It is a *metalanguage* for defining other markup languages, which can be used for describing and exchanging structured data in the Internet environment. XML provides most of the functionality of SGML but without the complexity. Essentially XML documents are *text-based* and resemble HTML documents with user-defined elements. An element consists of an opening and closing tag (indicated by angular brackets similar to HTML's standard tags) which surrounds content. XML removes the limitations of HTML by providing:

- *Extensibility* – The ability to define custom tags and attributes. XML tags can use meaningful names to describe their content, thereby providing self-describing documents. Text-based XML documents with meaningful tags can facilitate data reuse across different platforms and applications.
- *Structure* – XML can describe data in a nested structure.
- *Description* – XML supports a metadata description (a schema) for the structured data. Currently the mechanism for this is to associate a *Document Type Definition* (DTD) with the XML document.
- *Validation* – If a DTD is included, XML supports verification to ensure the data is *valid* (according to the supplied DTD) and *well-formed*.

An XML document is *well-formed* if:

1. It contains one or more *elements*.

2. It has exactly one root element that has a unique opening and closing tag that surrounds the whole document.
3. All other elements within the document are nested with no overlap between elements.

A well-formed XML document may additionally be described as *valid* if it conforms to a DTD. These are expressed in *Extended Backus-Naur* (EBNF) notation (see for example, the XML Specification [23]). Although DTDs are currently the recognised way to associate a schema with an XML document (in line with the current XML specification), there are a number of efforts underway to develop alternative schemas for XML to overcome some of the problems associated with DTDs. DTDs can be difficult to write and are limited in their descriptive power. DTDs cannot specify data types, default element content, or relationships within the data. DTDs also require separate parsers to XML, and different authoring tools. The main contender is now *XML Schema* [240] [241], a two-part draft specification for a schema language, which provides a superset of the capabilities, found in DTDs. This working draft draws heavily on a number of other proposals.

Since XML consists of user-defined elements, an application usually needs to parse an XML document and specialised APIs have been created for this purpose. There are two major types of XML parsers: Tree-based and event-based. A tree-based XML parser compiles an XML document into an internal tree structure and then provides an API to navigate that tree. The DOM (Section 2.2.8.2) specifies such an API. Event-based parsers report parsing events (such as start and end tags) directly to an application via call backs without building an internal tree. Whilst this is a lower API it has the advantage that it is not as memory hungry as a tree-based API which builds an in-memory parse tree. SAX (the Simple API for XML) [203] provides a standard interface for event-based parsing. Most parsers will also validate an XML document against a specified DTD.

Having described the origins of XML and a brief overview of some of its features, the following two sections discuss two diverse applications of XML. Firstly, as a mechanism for presentation content management on the Web, and

second, as a data standard and enabling Technology for document-driven distributed computing.

2.2.8.1.1 XML for Presentation and Content management

XML was originally seen as a way to overcome the shortcomings in HTML by providing a markup language with user-defined tags that could separate presentation from content. One application of XML is therefore as a markup language for Web pages. However, XML does not provide any information about how data should be displayed. One mechanism to associate display information is to write a server-side application, which parses and transforms a requested XML document to an HTML format prior to it being served to the client. Such an application can be custom built or can be based on generic *Style Sheets* processors.

Style sheet processors are available that can be run-server-side or can be embedded within Web browsers such that they can be sent an XML document (which includes a link to a style sheet) directly. In this case the style sheet contains the rules that define how a document should appear and the Web browser can process the style sheet along with the XML document it is retrieving in order to display it. The two style sheet languages that are currently being used for this purpose are *Extensible Style Language (XSL)* [89] [14] and *Cascading Style Sheets* (currently a two level specification, CSS1 [33], CSS2 [34], with a third level in progress). Separation of content from presentation, in this way, has a number of advantages. Different style sheets can be used to display the same XML document in different ways to different users, or in different formats to different devices, for example, to provide simpler presentation for low-resolution screens, such as those found on hand-held computers. Also, the same style sheet can be used to display different XML documents in the same style. It is also beneficial for content management since fragments of XML content can be retrieved from multiple sources and rendered into a single document. Conversely, a single fragment of XML content can appear as a component of many different Web pages. XML can also enhance searching and indexing mechanisms used for Web pages. Pages containing XML markup facilitate metadata discovery through their use of machine recognisable tags (and possibly complete, standardised XML vocabularies used by particular industries or groups).

Despite this initial focus for XML, a major current focus is now on its role as a vendor, platform and application independent data format that can be used to connect autonomous, heterogeneous applications. This is the subject of the next section.

2.2.8.1.2 XML as a Data Standard and as an Enabling Technology for Document-Driven Distributed Computing

Phipps [185] discusses how XML can complete the picture for a paradigm shift in computing. Historically, computing solutions have consisted of complex systems containing mutually dependent hardware, operating systems, software packages, network software and data formatting amongst other things. However, it is now possible to provide a simpler framework, which breaks the traditional dependencies. Phipps suggests that there are four parts to a modern computer solution and also defines the technologies for each part:

1. Network - indisputably TCP/IP is the solution.
2. Desktop - a space to load solutions – probably browsers, but the key feature is that solutions can be instantiated without requiring additional installed software or proprietary operating system features.
3. Programs – In the Web environment Java is now established as the de-facto standard for code development.
4. Data – until now there has been no obvious generic data format.

XML fills the final gap by providing an open data formatting system. Whilst Java provides a platform independent application development language, XML provides an application independent data standard.

XML is often described as self-documenting because it consists of named tags and an optional schema that defines the language represented by these tags. Currently such schemas are constructed as DTDs. A DTD defines, amongst other things, valid elements, attributes and rules for their use. XML can specify new markup languages for a particular purpose, sometimes referred to as vocabularies.

Industry-specific XML vocabularies are beginning to proliferate (see for example, the repositories at OASIS [176] repository at XML.org [242]).

As touched on in Section 2.2.7.4 XML could become an enabling technology for *document centric* computing consisting of loosely coupling heterogeneous applications on the Web. One of the reasons for the success of the Web is that it is based on a simple stateless protocol. XML could be used as the syntax for request-response message exchange between applications. In this scenario XML DTDs are used to define interfaces between services. XML parsers are used to marshal data, which is sent to a server via an HTTP POST method, and an XML message is returned to the client. Services can be made available by exposing XML DTDs or vocabularies. Since the underlying protocol is HTTP, messages are not blocked by firewalls. Furthermore, this framework benefits from work has been dedicated to, and continues to be done to optimise the performance, scalability, and reliability of HTTP servers. The loose coupling of client and server also make it possible to complete requests even if a client uses an old version of a DTD.

XML is a mechanism for representing data and does not provide a transport protocol. As mentioned above, the simplest way to transport XML between services is to use the HTTP protocol. A number of proposals are being put forward for standardising XML remote procedural call mechanisms to remote processes or objects. These include the *Simple Object Access Protocol* (SOAP) [204] and *XML-RPC* [239]. These proposals each describe basic object invocation mechanisms with varying features, all of which use HTTP as the transport and XML for message syntax. They do not require the complex run-time support of distributed object technologies such as CORBA and DCOM, and they provide the major benefit of being *Web-native* and thus able to pass through firewalls that accept HTTP requests. Conversely, these protocols do not currently support such features as metadata discovery and objects-by-reference (the latter would require bi-directional HTTP and distributed garbage collection).

2.2.8.2 The Document Object Model

The DOM specification [71] from the W3C defines an interface that allows programs and scripts to dynamically access and update the content, structure and

style of HTML and XML documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them.

The DOM is designed to be language and implementation independent and as such the core of the specification consists of interfaces defined using OMG IDL [170]. However, the specification also contains language bindings for Java and JavaScript. Programmers can use the DOM to build documents, navigate document structure, and add, modify, or delete elements and content.

The W3C Document Object Model Working Group carries out Work on DOM. Work by this group covers:

- Modelling new parts of XML: the DOM is an API to an XML document. As new features are added to XML, the DOM API should model these. Namespaces is an example.
- CSS Object Model: an object model for modifying and attaching a CSS style sheet to a document.
- Event model: a model for allowing user and application events.
- Traversal interfaces: an interface for selectively processing parts of the document according to user-specified criteria.
- Content Models and Validation: an object model for modifying and attaching a Content Model to a document.
- Load and Save interfaces: loading XML source documents into a DOM representation and for saving a DOM representation as a XML document.
- Views and Formatting Object Model: physical characteristics and state of the presentation.

The DOM is a multi-level specification, with Level 1 [71] a full recommendation and Level 2 [72] currently a candidate recommendation of the W3C. A list of the

current and envisaged DOM specifications along with timescales is available at [70]. These include:

- Functionality equivalent to that evident in Netscape Navigator 3.0 and Microsoft Internet Explorer 3.0, which is referred to as Level 0. The model builds on this existing technology.
- Level 1. This concentrates on the actual core, HTML, and XML document models. It contains functionality for document navigation and manipulation.
- Level 2, which is at Candidate Recommendation stage, includes a style sheet object model, and defines functionality for manipulating the style information attached to a document. It also enables traversals on the document, defines an event model and provides support for XML namespaces [160].
- Level 3 will address document loading and saving, as well as content models (such as DTDs and schemas) with document validation support. In addition, it will also address document views and formatting, key events and event groups.
- Further levels. These may specify some interface with the possibly underlying window system, including some ways to prompt the user. They may also contain a query language interface, and address multithreading and synchronisation, security, and repository.

The DOM is used in XML parsers (see Section 2.2.8.1) and also in client-side JavaScript as discussed in the next section.

2.2.8.3 Javascript

JavaScript is an interpreted scripting language originally developed by Netscape. Despite its name JavaScript bears little resemblance to Java other than in some basic syntactic constructs such as selection and iteration statements (Flanagan [94] suggests that the name was simply chosen as a marketing ploy). Javascript is untyped and although object-oriented, it features *prototype-based* inheritance rather than the class-based inheritance of Java and C++ (refer to [94] for further details).

Javascript allows executable content to be embedded in Web browsers for client side execution. Unlike Java Applets, JavaScript cannot draw graphics, perform networking or file I/O, however it can control browser behaviour and content. Java Applets, by contrast, cannot control the browser as a whole, but can provide capabilities such as graphics, file I/O, networking and multithreading.

The core JavaScript language has been standardised in the ECMA-262 standard [78] and, to maintain vendor-neutrality, is officially known in the standard as ECMAScript. This is because the name JavaScript belongs to Netscape and Microsoft's version is officially known as JScript. In this document, however, the term JavaScript is used generically to refer to all these related technologies, as is common practice.

JavaScript interpreters are embedded in both the Netscape and Microsoft Web Browsers. JavaScript interpreters in these Browsers provide many features beyond those defined in the ECMA specification, which concentrates more on the core of the language syntax. Many of the additional features are incompatible between the two browsers but are being standardised in other places such as the Document Object Model Specifications (see Section 2.2.8.2). The incompatibilities between JavaScript and JScript, and the fact that they tend to include new features at a much faster rate than can be reflected in the ECMAScript and DOM specifications, vastly complicates writing HTML pages containing JavaScript that are intended to work in both Netscape and Microsoft Web browsers. Therefore, although Javascript was used in the EASIA user interface to provide a more dynamic feel, a small subset of cross-browser supported features was used. Sometimes, where JavaScript features were required that were inconsistently implemented in the two browsers, the version of the browser was first detected in the code, and different execution paths were followed accordingly.

2.2.8.4 Dynamic HTML

According to the W3C, "*Dynamic HTML* is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated" [73]. Flanagan [94] states that Dynamic HTML (DHTML) is a loosely defined term which encompasses the technologies included in fourth generation

browsers that enable HTML documents to be more dynamic and interactive than before. He suggests that the major new technologies are the DOM, the event model (being standardised in DOM Level 2), CSS and absolute positioning of HTML elements (now part of the CSS2). Hence, DHTML is really an umbrella term covering many of the technologies discussed earlier in this section. Consequently, as mentioned during descriptions of the individual technologies, DHTML is useful but subject to many compatibility issues between the major Web browsers.

2.3 Multi-tier Web/Database Connectivity

The final section of this chapter ties together database and Web technologies to discuss how the Web enables *multi-tier* computing. The GBIS, DBbrowse and EASIA architectures all use 3-tier architectures consisting of a client or presentation tier, a middle tier hosting the bulk of the application logic, and finally a data tier. The rest of this section compares two and three tier architectures, explaining the advantages of such a 3-tier architecture.

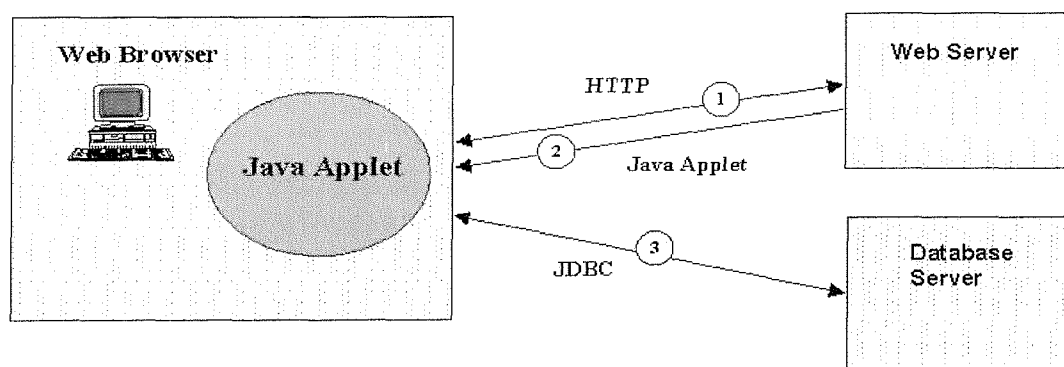


Figure 1: A 2-tier architecture using a Java Applet and JDBC for database access.

Figure 1 illustrates an example of a 2-tier architecture using Java Applets and JDBC. As stated in section 2.1.4.1 there are a number of different ways that JDBC drivers can be implemented with respect to their network architecture. The diagram shows a direct 2-tier connection between a Java Applet and a database server. Once the Applet has been downloaded JDBC is used to make a direct connection to a remote database server and SQL is shipped to the database. A client/server database application using such a 2-tier architecture is easy to implement. However, 2-tier

client/server systems can experience network performance problems. These stem from the fact that SQL can produce a large amount of network traffic if a client is issuing individual SQL select statements to a server with each query generating resulting data that has to be sent back across the network. SQL has been extended with stored-procedures to reduce this problem by allowing SQL to manage functions as well as data. These functions reside on the server and encapsulate a number of SQL commands, to avoid the need for sending intermediate results back to the client. However, procedural extensions to SQL are non-standard with no two vendors having an equivalent implementation.

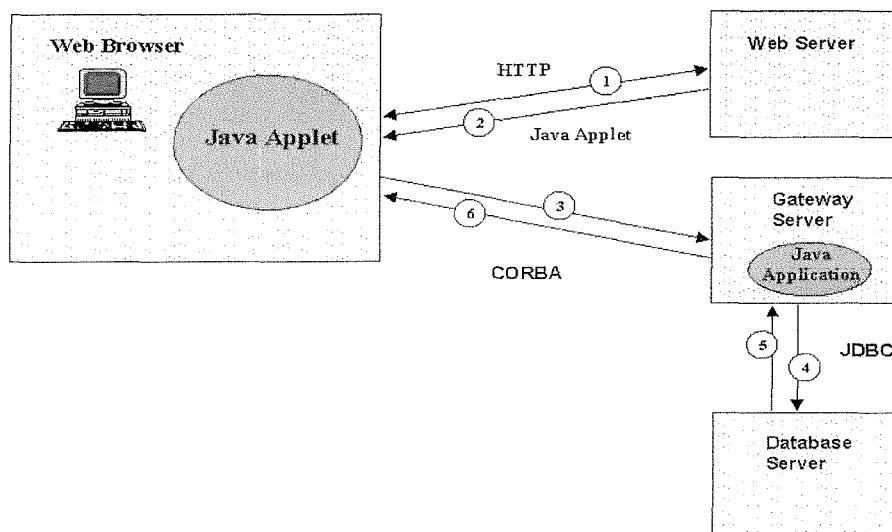


Figure 2: A 3-tier architecture using a Java Applet, CORBA and JDBC

Figure 2 illustrates an example of a 3-tier architecture. A Java Applet on the client tier communicates with a middle tier Java application using CORBA. The middle tier application communicates with the data tier using JDBC. A 3-tier client/server architectures such as this can offer the following advantages:

- *Reduced network traffic* - The volume of data passed between client and server can be reduced. The middle-tier logic can perform multiple SQL queries and additional data processing to return only the final result to the client. In contrast,

2-tier systems may need to perform multiple SQL queries to obtain a desired result, shipping data from server to client at each stage, with final data processing carried out at the client.

- *Superior load-balancing and scheduling* – Server programs can be duplicated and distributed across multiple server platforms and middle-tier logic can incorporate load balancing and scheduling. In addition, placement of expensive tasks on known high performance servers can be used to control server load. It is also possible for the client to communicate with different middle-tier servers for different tasks (whilst still using the same data tier).
- *Flexible distribution of client server logic* - client and server complexity can be controlled.
- *Database Connection Pools* – The middle tier can be used to pool database connections using prestarted processes that are already connected to the database.
- *Security advantages* – Data access is restricted to the middle-tier business logic. There is no direct access to the database from the client. This allows strict control over the types of data access operations that can be made. Even if a client has access to a valid database access password, there is no direct communication path to the database. In a 2-tier arrangement the client usually has control of a database access API, allowing the full range of SQL access. Additional security benefits are achieved between the client and middle-tier, where communication can be protected using existing Internet security mechanisms such as HTTPS - HTTP plus Secure Socket Layer (SSL)³.

³ SSL was introduced by Netscape and has become the de facto standard for secure online communications. It forms the basis of the Transport Layer Security (TLS) Protocol [125] under development by the Internet Engineering Task Force (IETF). HTTPS is not the same as S-HTTP [195], which is an extension to HTTP to support sending individual messages securely over the Web, whereas SSL is designed to establish a secure connection between two computers. SSL and S-HTTP have very different designs and goals so it is possible to use the two protocols together. Both have been submitted to the IETF for approval as standards. SSL is the far more prevalent technology, with support from all major Web servers and browsers.

- *Easier Maintenance* – In a 2-tier environment a precise SQL call can become ‘fragile’ - changes to the DBMS schema (such as renaming a column) will cause the client's SQL call to fail. One way to reduce this problem is to use ‘robust’ queries that contain more generic select statements and then to filter the retrieved data at the client, but this has a network performance overhead. If client code needs to be changed to accommodate schema changes then modified client code will have to be re-distributed to all the users. In a 3-tier environment such changes can be isolated from the client.
- *Individual tiers can be upgraded independently* – As user interface technology changes it is possible to upgrade the client tier without re-writing the business logic. It is also possible to support different user interfaces for different users simultaneously, without any change to the other tiers. Separation of the business logic from the data tier, via a standard database access API, makes it easier to upgrade or replace the underlying database.
- *Scalability* – Multi-tier Web-based systems usually comprise UNIX Workstations or Windows NT servers on the middle and data tiers. It is far easier, and cheaper to expand such systems, by increasing CPUs, memory, servers etc. compared with mainframe based solutions, which require more long-term planning.
- *More tolerant in the event of server failure* – A multi-tier system distributed across many middle-tier servers and possibly multiple-backend data sources can continue in the event of a server failure.

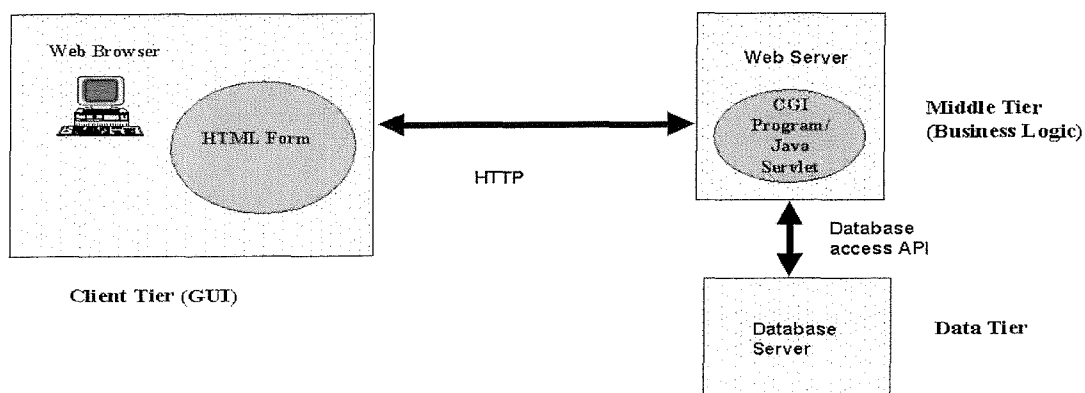


Figure 3: A 3-tier architecture using HTML/HTTP, Java Servlets and JDBC

Figure 3 illustrates another possible 3-tier architecture using HTML on the client tier communicating to a middle-tier Java Servlet using HTTP. The Java Servlet communicates with the data tier using JDBC. This is the basis of the architecture used in the EASIA prototype.

2.4 Summary

The three prototypes that were developed during this research used leading-edge database and Web technologies available at the time. This chapter has surveyed the technology options that were available and has described reasons for particular choices. Wherever possible the prototypes were implemented using open standards. The different choices of technologies used in the three prototypes reflect the ongoing emergence of new technologies, better suited to particular tasks.

- GBIS used the CGI with programs implemented using UNIX shell scripting and PERL.
- DBbrowse used CGI and PERL to connect object-relational databases to the Web. Database communication was achieved using system calls from PERL to a command line processor provided with the database.
- EASIA uses Java Servlets, Java Database Connectivity, XML and object-relational databases that support SQL/MED.

The client-side of each of these applications was implemented using HTML communicating with the server via HTTP. Some DHTML was used on the client to increase the dynamic feel of the user interface, however, this was restricted to features that would run in both of the two most popular Web browsers, Netscape Navigator and Microsoft Internet Explorer.

The next three chapters describe the GBIS, DBbrowse and EASIA prototypes that made use of the chosen technologies to illustrate new Web-based architectures for Web-based management of non-traditional data.

3 The Graphical Benchmark Information Service

3.1 Introduction

This chapter describes an investigation, carried out during 1994/1995, into the use of the Web for management and visualisation of scientific data. The Graphical Benchmark Information Service (GBIS) [181] [117] was developed to provide a graphical interface to display user-selected results from the PARKBENCH (PARAllel Kernels and BENCHmarks) [118] multiprocessor benchmark suite.

Two of the major factors that determine the success of a benchmark suite are the availability of the benchmark codes and the availability of the results obtained. The general availability of benchmark codes was easy to arrange using ftp sites and links to source code and documentation from Web pages. The main goal of GBIS was to target the second factor for success, by making the results from PARKBENCH as widely available as possible, in a format that was easy to assimilate. For this reason the method selected to implement the service was to use the Web, and to ensure that any software used was freely available in the public domain. GBIS provides graphs of benchmark results rather than tabular results because unlike single-processor benchmarks, multi-processor benchmarks can yield tens of numbers for each benchmark on each computer, as factors such as the number of processors and problem size are varied. A graphical display of performance surfaces therefore provides a useful way of comparing results. At a time when most Web pages consisted of static HTML pages, GBIS demonstrated the benefits of interactive dynamic Web pages for scientific data archives.

The rest of this chapter begins with an overview of GBIS. The implementation is then described. The structure of GBIS result files and the mechanisms for updating GBIS data are then covered. Finally, some conclusions are presented.

3.2 GBIS Overview

GBIS provides the following features:

- Interactive selection of benchmark and computers from which a performance graph is generated and displayed.
- Default graph options that can be changed: Available output formats; gif, xbm, postscript or tabular results.
- Selectable ranges or autoscaling.
- Choice of log/linear axes.
- Key to traces that can be positioned as required.
- Links to PARKBENCH and NAS Parallel benchmark information available on the Web (including documentation and source codes).
- Details of how to send in results for inclusion in the GBIS.

The PARKBENCH suite contains several categories of benchmark codes that test different aspects of computer performance. The suite is divided into four categories; low-level, kernel, compact application and HPF compiler benchmarks. The codes within these categories were developed specifically for PARKBENCH or adopted from existing benchmark suites. For example, the low-level codes were taken from the GENESIS [6] Benchmark Suite while several of the kernel benchmarks were taken from the NAS Parallel Benchmarks [15] [16].

The PARKBENCH suite contains single-processor codes and multi-processor codes that use the PVM (Parallel Virtual Machine) [101] and MPI (Message Passing Interface) [157] [158] message passing libraries. Both the single-processor and multi-processor codes produce tens of numbers as results and the most satisfactory way of displaying these numbers is in a graphical manner.

Many of the multi-processor codes are designed to be run numerous times, on the same computer, with the number of processors varied each time. A performance figure in Mflop/s is obtained for each run. By plotting a graph, called a trace, of performance against number of processors, the observed speedup can be seen i.e. the change in performance that results when more processors are used to solve the same sized problem. By incorporating performance traces for different

computers on the same graph, it is possible to compare the relative performance of different computers, as well as the actual speedup.

Another factor that can be varied for several of the benchmark codes, is the problem size. Traces of temporal performance in, for example, timesteps per second against number of processors, can be plotted for the same computer, with a different problem size associated with each trace. The observed scaleup can be seen from the relative position of the traces, i.e. the increase in the size of problem that can be solved in the same time period when more processors are used.

GBIS allows several other types of graphs to be plotted depending on the type of performance metrics that a particular benchmark produces. These performance metrics include benchmark performance, temporal performance, simulation performance, speedup and efficiency. These metrics are defined in Hockney [116] and the PARKBENCH Report [118].

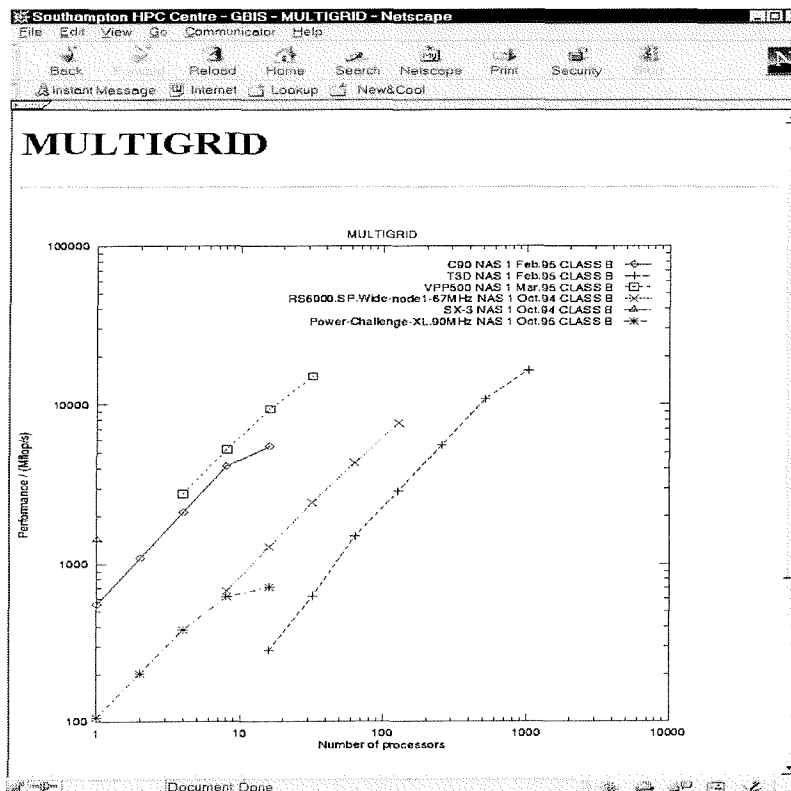


Figure 4: Graph showing results of the Multigrid Benchmark.

The low-level benchmarks in the PARKBENCH suite measure performance parameters that characterise the basic machine architecture. These parameters can be

used in performance formulae that predict the timing and performance of more complex codes. Hockney and Jesshope [119] describe the low-level parameters. These low-level benchmarks output vectors of numbers which can be satisfactorily displayed and analysed as graphs. For example, GBIS allows a choice of two graph types to be plotted for the COMMS1 benchmark from the PARKBENCH suite; message transfer *time* or transfer *rate* can be plotted against message length.

Figure 4 and Figure 5 provide examples of graphs generated by GBIS. They contain user-selected results for the Multigrid and LU Simulated CFD Application benchmarks [15], respectively. These graphs were generated using the default GIF output option, and the colour postscript output option, respectively. The data was loaded into the GBIS system from the NAS Parallel Benchmark (Version 1.0) results (NAS NPB1) [198] and the NAS Parallel Benchmarks (Version 2.1) results (NAS NPB2) [200].

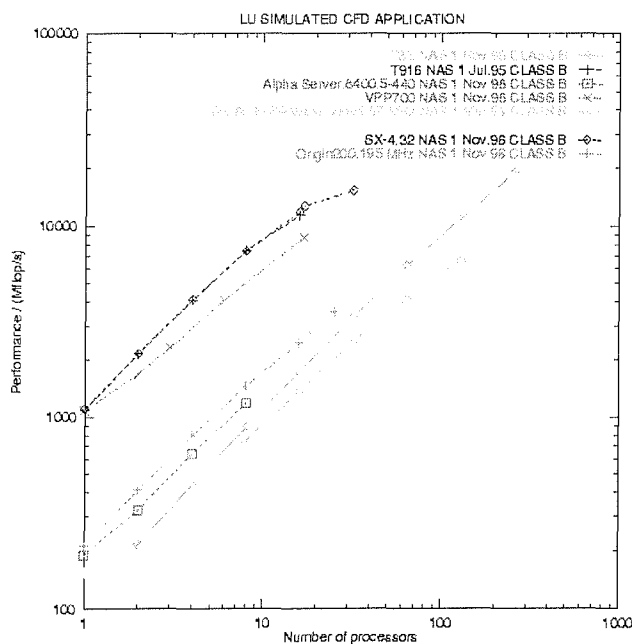


Figure 5: Graph showing results of the LU Simulated CFD Application Benchmark.

3.3 GBIS Implementation

The GBIS user interface was implemented using HTML forms and the CGI. UNIX Bourne shell scripts running on the Web server machine (at the University of

Southampton) processed the forms. The user navigates through a set of HTML forms, making selections along the way, to produce a graph of required results. As the user progresses from one HTML form to another, state is maintained using hidden variables, which point to unique temporary files that are stored on the server. The temporary files are automatically deleted when they are no longer required, with the exception of the final page showing the graph. These pages are deleted periodically via an automated shell script.

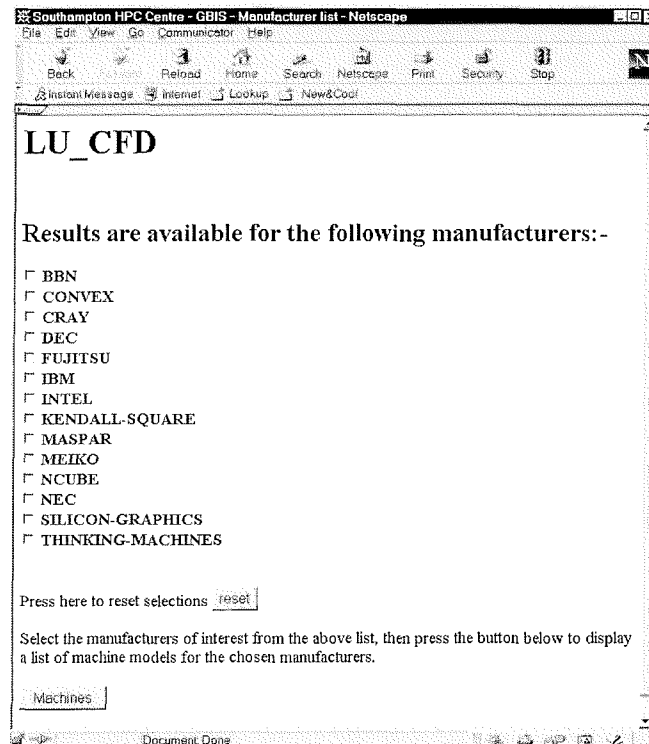


Figure 6: GBIS manufacturer list page.

The first HTML form displayed allows the user to select a benchmark. A Bourne Shell script processes this selection and returns another form consisting of a list of manufacturers, for which results exist (Figure 6). The user then selects a number of manufacturers which in turn produces another form displaying all machine results for the chosen benchmark plus manufacturers (Figure 7). Transparent to the user, each machine result corresponds to a data file of Cartesian coordinates stored on the Web server machine. These data files are used in conjunction with the public domain plotting package, GNUPLOT in order to plot the graphs.

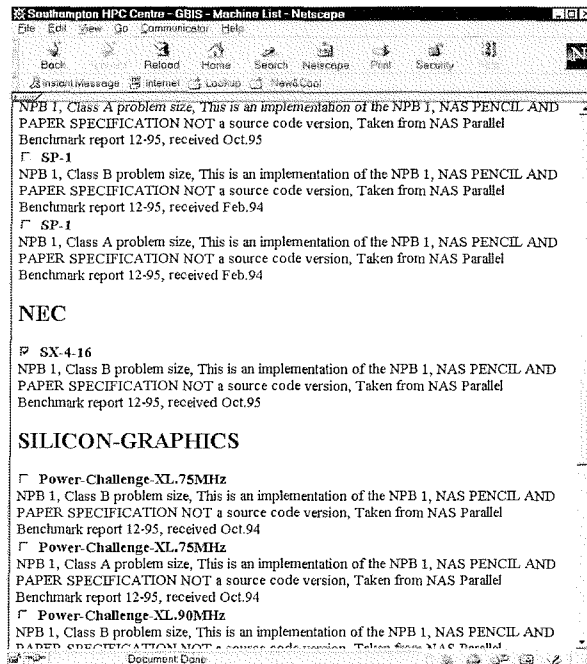


Figure 7: GBIS machine list page.

GBIS uses the information processed from the forms, to create an appropriate GNUPLOT batch file. This file contains a list of commands which provide a means of setting graph options (e.g. axes scaling, axes types, etc.) to default values or user selected values (Figure 8) and for naming the data files to be plotted.

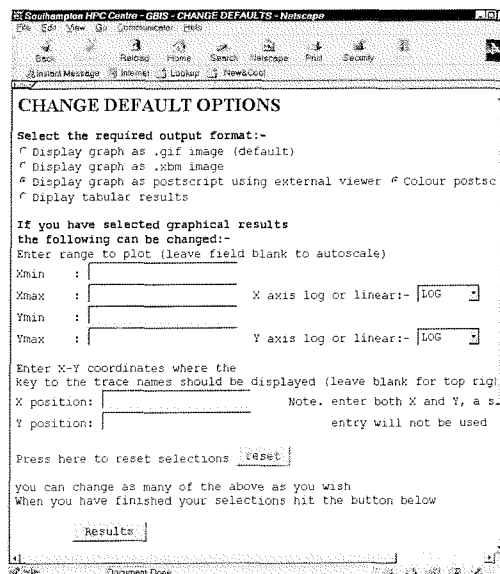


Figure 8: GBIS change defaults page.

The shell scripts then execute GNUPLOT using this batch file, on the Web server machine, to produce a postscript file of the graph as output. Depending on the

format the user selected for the results, this postscript file is made available for viewing by a hypertext link from a Web page, or, is converted to another format (e.g. GIF) which can be displayed directly on a Web page.

3.4 GBIS Result File Structure

The result data files associated with each machine are stored as a hierarchy of UNIX text files on the Web server machine. This method was chosen due to the diversity of different vectors of results that need to be stored for the different benchmarks. The convention used for the directory structure is:

benchmark name/machine manufacturer/result file name

Where, result file name is in the form:

(machine model)_(date benchmarked)_(problem size)_(benchmark metric)

For example,

MULTIGRID/IBM/SP-2_10.AUG.94_CLASSB_PERF

The machine model, date benchmarked and problem size, from the result file name, are used to label traces on the graph; while the benchmark metric is used in choosing labels for the graph axes and in selecting the choice of log or linear axes for the default graph format. For some benchmarks the result of a single run (on a single machine) corresponds to several results files in the GBIS database. This is to allow the results to be plotted in different ways. For example, if the benchmark produces results which can be plotted as any one of, benchmark performance, temporal performance or simulation performance, against number of processors, then three different files of coordinates must be stored. These are distinguished by using a different benchmark metric in the filename, so that the appropriate file can be found, for the type of graph requested.

The actual contents of results data files are in a format that GNUPLOT can use directly. This consists of lines that contain coordinates and comment lines, beginning with the # character. The availability of comment lines was used to incorporate additional information that was parsed by the GBIS system. This is explained with reference to the example file contents shown in Figure 9.

```

#!CS-2 at the University of Southampton.
##Genesis3.0, PVM, FORTRAN 77,
##26 May 94
#
#Message Length / ByteTransfer      Time / us
0                                     1.983E-04
1                                     2.001E-04
2                                     2.006E-04
3                                     2.007E-04
#
#RINF=9.810 MByte/s
#NHALF=1791.949 Byte
#STARTUP=195.200 us

```

Figure 9 Example contents of a GBIS result data file.

The path and filename for this file are:

COMMS1/MEIKO/CS-2_26.MAY.94_TIME

The file contents are plotted if the user selects time against message length. For the COMMS1 benchmark, the transfer rate can also be plotted against message length. Therefore, a file containing different coordinate values also exists, with a name ending in `_RATE` (indicating the metric).

As well as lines containing coordinates, there are lines beginning with the characters `#!`, `##` and `#`. These lines are treated as comments by GNUPLOT because they begin with at least one `#` character. However, the GBIS system uses these lines in the following way. Lines beginning `#!` and `##` are parsed and then used in the construction of the 'machine list' HTML form (see for example, Figure 7). The string after the `#!` is used to label a checkbox identifying a particular machine, and strings following `##`, are placed below the checkbox to describe the particular result further, by providing, for example, details of the environment and for the particular benchmark run. Lines beginning with a single `#` are not parsed for display purposes,

but can be used to store additional comment information. This is accessible if the user requests text results as the complete contents of the file are returned in this case.

Bourne shell scripts and PERL programs were developed to convert automatically, or semi-automatically, the results files generated by the benchmarks into the format required by GBIS. For example, a PERL program was written to take the Latex version of the NAS Parallel Benchmark Results and creates the required directories and results data files for GBIS.

3.5 Updating the Results Database to include additional Machines and Manufacturers

GBIS provided an ftp site onto which users could upload new benchmark results. Users were instructed to supply results in the GBIS-specific format, or to supply results files in the raw output format used by the benchmarks. The latter could be converted using the scripts mentioned above. Once result files were in the required format, updating the GBIS database involved three steps:

1. Add the new result data file to the correct benchmark/manufacture directory.
2. If a new manufacturer subdirectory was created in step 1 then run a shell script which updates the list of manufacturers available for each benchmark.
3. Run a shell script to update the list of available machines available for each manufacturer.

The scripts mentioned above produce the HTML files that are used in the creation of the forms containing the manufacturer list for a given benchmark and the machine list for each manufacturer. The script in step 2 uses the names of all subdirectories of a given benchmark directory and hides the HTML file created in the benchmark directory for later use. The script in step 3 works through each of the manufacturer subdirectories and hides an HTML file in each one. This file is constructed by parsing the lines, describing the machine benchmarked, from each result data file.

The HTML files, used in constructing the forms, are updated periodically in this manner in order to improve the response time for GBIS results. Initially, the forms listing manufacturers, and then machines for selected manufacturers, were

generated dynamically for every user request, however the response times proved unacceptable, necessitating the batch update mechanism described above.

3.6 Conclusions

GBIS was an early system employing CGI scripting combined with standard application programs, to provide Web-based management of scientific data. GBIS successfully demonstrated the benefits of dynamic Web-based graphical results for displaying multiprocessor benchmark results. GBIS has been cited (in the NAS Parallel Benchmarks 2.0 specification [16], and Hockney's book [117]) and reused (mirrored at the University of Tennessee at Knoxville [100], and adopted by the *Real Applications on Parallel Systems* Consortium to form the basis of their Online Benchmark Information System [194]) by other members of the multiprocessor benchmarking community.

On-going use of GBIS also proved useful in highlighting a number of areas where improvements could be made to the approach used for scientific data management. Firstly, it was not possible to issue complex searches, to find, say, all the machines that exceeded a particular performance level with a specified number of processors. Instead, the primary method of locating results was browsing. Limited searching was available via a feature that generated an HTML page containing summary information from all the result files. This could then be searched using a 'find' option of the Web browser.

Secondly, GBIS was difficult to update with new results. The syntax for the content of the GBIS result files did not follow a well-known standard and very few results were submitted in the GBIS format. Although GBIS could also process results supplied as raw output files from the benchmarks, on a number of occasions, users took it upon themselves to 'tidy up' these files (which contained significant human readable commentary) prior to submission, with the result that the automated conversion scripts could no longer parse them.

The next two chapters describe two subsequent architectures that were developed to manage non-traditional data on the Web. Whilst these architectures are not directly related to GBIS – they provide *generic* rather than application specific architectures for management of non-traditional data – they do incorporate features

that overcome the problems described above. The first system, DBbrowse (Chapter 4), automates the integration of databases with the Web. In so doing, DBbrowse supports significantly improved search capabilities. Furthermore, by storing the underlying data in an object-relational database, DBbrowse capitalises on database update mechanisms (such as import of comma delimited text files). The EASIA architecture (Chapter 5) uses not only object-relational database technology for Web-based management of scientific data, but also XML to provide a well-defined language for defining the interface between data and applications. As discussed in Chapter 2, XML provides significant advantages for data exchange between applications.

4 Automatically Generating Web Interfaces to Relational Databases

4.1 Introduction

Whilst the GBIS prototype (from the previous chapter) described an *application specific* system for scientific data visualisation, this chapter describes a *generic* mechanism for Web-based management of non-traditional data that is not targeted to a particular domain. The catalyst for this new approach was the availability of digitised data from the Sir Winston Churchill Archive. The History Department at the University of Southampton had begun to archive digitised data using a simple relational database. The existing schema used traditional datatypes to store information describing a document (for example, a letter written by Sir Winston Churchill) and also the name of a local file containing a digitised image of the document. The archivists from the history department had the necessary skills to build the database and images, however they wanted to make the data accessible to a wider audience via the Web and sought advice on this might be accomplished.

As with GBIS, Web-based management of non-traditional data was at the heart of the problem. Like GBIS, the digitised archive contained some data in the form of text files. However, instead of containing scientific result data, the text files contained transcripts of some of the documents from the archive. In addition the data consisted of binary multimedia files containing digitised images (and perhaps audio and video files). Furthermore, as already mentioned, this complex data was complemented by simple relational data. In view of the mixture of datatypes and the work that had already been done to store the archive using a database, an initial prototype system was built to interface a database to the Web *directly*, and to make use of the database's sophisticated query capability and well-known update mechanism. A description of this work is published in [75]. (The described system replaced the initial relational database with an object-relational database so that the images could be stored directly within the database.) Like GBIS, this system was application specific and was written specifically for the database schema used by the archive. The report in [75] did however, briefly mention foreign keys (available in

some relational databases) as a possible generic mechanism for browsing from one table to another in a relational database, in order to focus on specific data. This idea underpinned the DBbrowse prototype [180] [179] [76] that is described in detail in this chapter.

The DBbrowse prototype was the culmination of research carried out during 1996/1997 into *automatic* generation of *generic* Web interfaces to object-relational databases. Experience with the Sir Winston Churchill Archive had highlighted the need for systems that could facilitate rapid deployment of interactive Web-based applications by developers with little Web development experience. DBbrowse provided such a system by generating Web interfaces, incorporating intuitive query capabilities, to object-relational databases. DBbrowse differed from commercial Web/database integration offerings that were emerging during this time, since they all required programming effort (refer to Chapter 6). DBbrowse also demonstrated a novel method for *browsing* databases on the Web to further facilitate users with little database experience. A summary of the features exhibited by DBbrowse follows:

- Completely automatic, schema driven creation of Web interfaces to relational and object-relational databases, which require no programming effort to use, and no additional database administrator effort.
- Because HTML is not stored, no separate maintenance of the HTML pages and database source data is required. Any changes made to the database are reflected the next time that the data is accessed.
- Data and *schema* changes are handled automatically.
- By connecting to the database in real-time the sophisticated search engine provided by the database management system (DBMS) is made use of.
- The interface is generic rather than application domain specific.
- The ability to *browse*, in a hypertext-like fashion, from any data to all related data held in the database. Links to related data are derived from *inferred* relationships and do not need to be specified.

- Sets of tuples can be operated on at the same time.
- Query refinement during the browsing stage.

In the remainder of this chapter the configuration used for providing Web access to the DBMS is first described. A section on automatically generating the user interface and SQL queries follows. Finally, a feature is described which provides hypertext browsing of the database based on schema relationships extracted from the database catalogue, and an example of this is described in detail.

4.2 Providing Web Access to the Database

To interface the database to the Web, an access path must exist between the machine nominated to be the Web server and the machine on which the database resides. One option is to install the Web server and database server on the same machine. An alternative approach that allows the Web server machine and database machine to be physically separate, requires the Web server machine to be a database client. The implementation described here uses this approach. IBM's DB2 Object-Relational Database (see for example [42]) was installed on a single node of an IBM RS/6000 SP2 parallel machine. The DB2 Client Application Enabler (CAE) was installed on a separate machine that also contains the Web server. The CAE provides run-time support to allow local applications access to remote database servers. The configuration is illustrated in Figure 10.

Users of this system interact with their local Web browser, which sends and receives information to/from the Web server. A user's request for information is sent via the Web server to the database and the result is shipped back to the user as an HTML page. The first step in this operation is for the Web server to construct an SQL query from the information provided by the user and to issue this query to the database. To do this, the Web server uses the CGI mechanism to invoke an external program (written in PERL) that has been developed. This program, *Query Generator*, decodes the information forwarded by the Web server and constructs a query in SQL that is sent to the database client. The database client in turn forwards

the query to the remote database server on the SP2. The result from the query is returned to the *Query Generator* program via the database client.

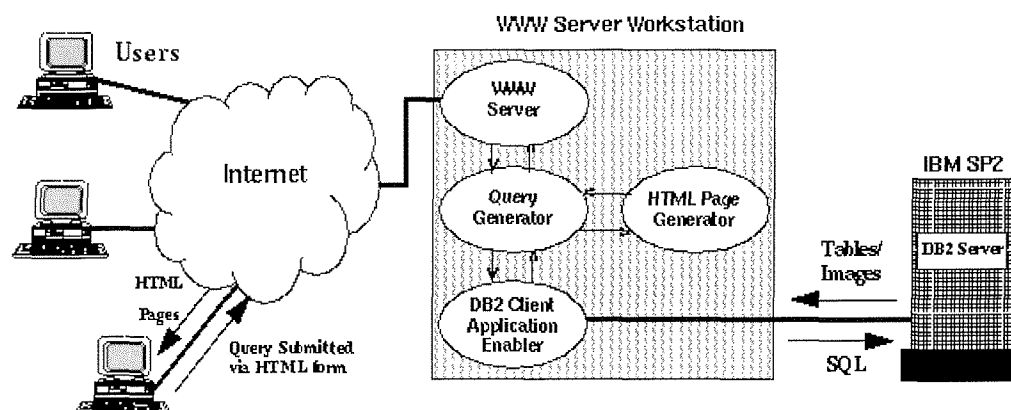


Figure 10: Interconnection strategy for providing Web accesses to a database.

The second step in returning the required information to the user is to construct an HTML page with the results returned from the database. After *Query Generator* receives the results from the database, it invokes the *HTML Page Generator* program. This program creates an HTML page for display to the user, based on a combination of static information and the results returned from the query.

4.3 Automatically Generating the User Interface and SQL Queries

The automatic Web interface to the database is constructed as follows. When the user accesses the home page a link is available to display all the available databases. This link invokes a CGI program (written in PERL) which queries the database manager instance to find out the names of all the individual databases being managed. The user next selects a link to the particular database of interest. This invokes a further CGI program that accesses the database system catalogue to obtain metadata describing the schema of the required database. Details of table names, column names, column data types and primary and foreign keys are extracted at various stages of interaction with the user, as required by the underlying system. The metadata is used to present the user with an HTML form, similar to Figure 12, to select the tables of interest from the chosen database. The columns in the selected table are then displayed in a query form based interface akin to *Query By Example* (QBE) [244]. To use the query form, users are required to select the fields to be

returned by the query, and optionally, they can restrict the data returned by placing selection conditions on one or more fields. Operators such as =, <, >, >=, <=, <>, *like* and *not like*, are included for use in specifying these constraints. The conditions are entered in a text box next to the field name. The text can contain wild cards, with an underscore replacing a single character and a percent sign replacing an arbitrary number of characters. For each field, the data type is also displayed to the user. For a UDT, both the name of the UDT and the underlying base-type name are supplied. Depending on the base-type of a field, any condition entered will be used in an appropriate way. For example, if the field is a CLOB, then a string in the condition field, will be used within a DB2 *LOCATE* function in the generated SQL. Once the query form is complete the user presses the ‘Submit Query’ button. The query result is a table consisting of the requested columns, and rows that meet the selection conditions.

4.4 Providing Database Browsing via Dynamic Hypertext Links Derived from Referential Integrity Constraints

The interface described above provides the user with a query access capability on the data, but does not provide the user with any browsing ability. To add a browsing facility on the data the existence of *foreign keys* in the database are exploited. A foreign key (FK) in a relation R2 is a subset of attributes in R2, such that FK is a primary key in relation R1. Most databases used in industry contain foreign keys because of the need to maintain *referential integrity*. (Informally, a database that uses referential integrity ensures that each value of a foreign key in R2 refers to an existing row in R1. For example, referential integrity may be used to ensure that the department number entered in a record of an employee table is valid, by ensuring that a corresponding department number entry exists in a department table). DBbrowse creates dynamic hypertext links based on the relationships implied by foreign keys.

The database catalogue tables identify the primary and foreign keys in all relations. When a user requests data from the database, the catalogue is examined to see if the requested table columns participate in referential integrity constraints. The system tables are also queried to identify any columns containing LOB data. For

both these cases the data returned from the database is modified so that the attributes within these columns are 'selectable'. The resulting HTML page thus contains a table with Hypertext links.

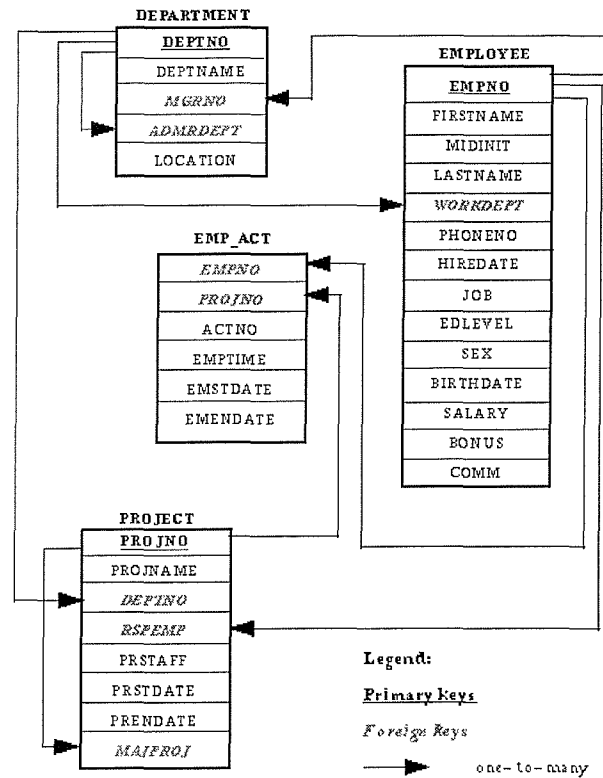


Figure 11: Employee Activity database schema and relationships between entities.

DBbrowse automatically generates 5 different link types. These are described below with examples based on the sample 'employee activity' database schema of Figure 11.

- *Link type 1. A single foreign key value will link to a row of a table in which the value appears as a primary key.*

This is a specialising link and will always return a single row. For example, referring to Figure 11, a row from the EMPLOYEE table displays a link on the WORKDEPT field. Selecting this link in the EMPLOYEE table will retrieve full details of the employee's department by displaying the appropriate row from the DEPARTMENT table.

- *Link type 2. A single primary key value may link to rows in a table, which contain the value as a foreign key.*

This is a generalising link. The user may have a choice of tables to link to, and for the selected table, zero or more rows may be returned. For example, referring to Figure 11, a row from the EMPLOYEE table will display links from the EMPNO (employee number) field to the PROJECT, EMP_ACT (employee activity) or DEPARTMENT tables. Selecting the link to, say, the EMP_ACT table will display any employee activity rows that contain that employee number.

Links of type 3 and 4 are special cases of types 1 and 2 above:

- *Link type 3. A complete foreign key column in the result table can be selected to perform a link of type 1 above, for each row present.*

For example, a result table consisting of rows from the EMPLOYEE table would display a link from the WORKDEPT column to the DEPARTMENT table. Selecting this link will perform a join to inline full details of each employee's department.

- *Link type 4. A complete primary key column in the result table can be selected to perform a link of type 2 above, for each row present.*

For example, a result table consisting of rows from the EMPLOYEE table would display a link from the EMPNO column to the PROJECT, EMP_ACT or DEPARTMENT table. Selecting the link to, say, the EMP_ACT table will, for every row present, perform a join to inline full details of any employee activity rows that contain the employee number.

- *Link type 5. Any fields that contain a LOB type will display the size of the LOB in bytes. This can be selected to retrieve the actual LOB from the database.*

A LOB field could, for example, contain image, audio, video or character data. DBbrowse relies on the user's browser to display different types of non-traditional data according to helper applications specified within the user's browser, for the various MIME types (see for example [98] [159]) associated with incoming content.

The links described above provide the capability of browsing the database for information without the need for the user to explicitly submit further database queries. When the user selects a dynamic link a CGI program is invoked to generate the new query based on the selection. The new query is then supplied to the *HTML Page Generator* program for execution and for displaying results to the user. The process then repeats.

4.5 Example of a Database Browsing Session

This section shows the screen shots from an example query submitted to a sample database ('*IBMSAMPL*' that was supplied with IBM's DB2/6000 version 2). The database contains employee records and details of their project activities. The schema and relationships between entities are shown in Figure 11.

The following queries will be answered by completing an initial query form and subsequent browsing:

Find all the employees in the 'MANUFACTURING SYSTEM' department. For each employee show which projects they are working on, and their activities within those projects. Finally, list the employees who have worked on activity '6' within the 'W L ROBOT DESIGN' project within the 'MANUFACTURING SYSTEMS' department.

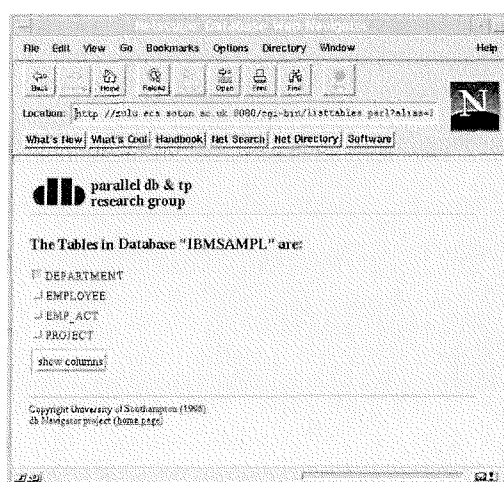
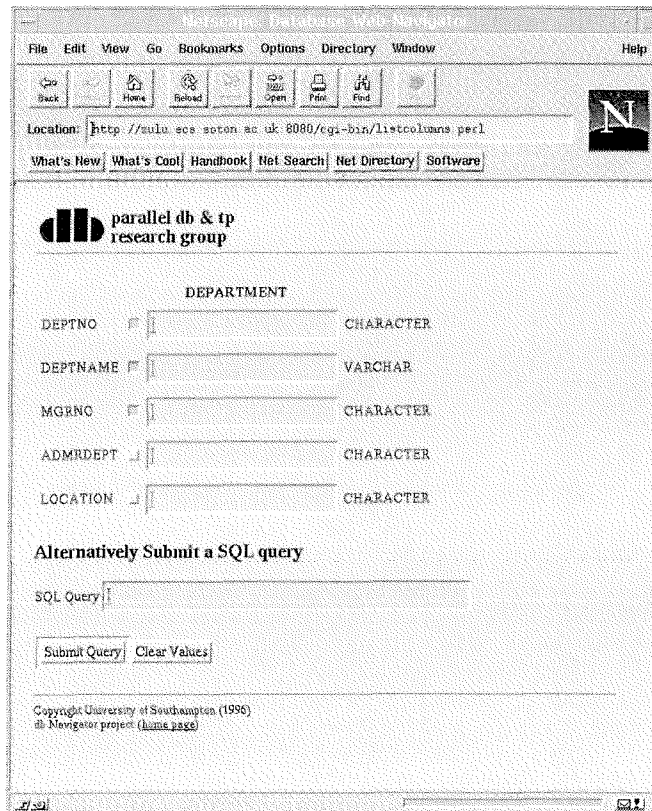


Figure 12: Selecting tables of interest.

The homepage displayed by DBbrowse contains a link that will list all the existing databases. This list is created dynamically, so that if a new database is created, it will automatically appear. Figure 12 shows the screen that results from selecting the

'IBMSAMPL' database. This screen displays the list of tables from the chosen database. The user can select a table. To answer the initial query, the DEPARTMENT table is selected and the 'show columns' button is pressed.

Figure 13 displays a list of columns from within the DEPARTMENT table. This is a QBE-like form. The user selects the columns of interest, enters any conditions and then presses the 'Submit Query' button.



parallel db & tp
research group

DEPARTMENT

| | | |
|----------|----------------------|-----------|
| DEPTNO | <input type="text"/> | CHARACTER |
| DEPTNAME | <input type="text"/> | VARCHAR |
| MGRNO | <input type="text"/> | CHARACTER |
| ADMRDEPT | <input type="text"/> | CHARACTER |
| LOCATION | <input type="text"/> | CHARACTER |

Alternatively Submit a SQL query

SQL Query:

Copyright University of Southampton (1996)
db Navigator project ([home page](#))

Figure 13: Selecting columns and specifying conditions.

Figure 14 shows the result table containing the requested columns of the DEPARTMENT table. To display all the employees within the 'MANUFACTURING SYSTEMS' department the user selects a link from the DEPTNO column with value 'D11'. Three links are available in this column from DEPTNO 'D11'. The second link, labelled 'T2', is a link to the EMPLOYEE table, as shown in the 'link to' text under the DEPTNO column heading (this is a link of type 2).

parallel db & tp
research group

Show Executed SQL Query

| MGRNO <small>(link to EMPLOYEE)</small> | DEPTNAME | DEPTNO <small>(link to DEPARTMENT, EMPLOYEE, PROJECT)</small> |
|--|-----------------------------|--|
| 000001 | SPIFFY COMPUTER SERVICE DIV | A00(T1, T2, T3) |
| 000002 | PLANNING | B01(T1, T2, T3) |
| 000003 | INFORMATION CENTER | C01(T1, T2, T3) |
| 000004 | DEVELOPMENT CENTER | D01(T1, T2, T3) |
| 000005 | MANUFACTURING SYSTEMS | D11(T1, T2, T3) |
| 000006 | ADMINISTRATION SYSTEMS | D21(T1, T2, T3) |
| 000007 | SUPPORT SERVICES | E01(T1, T2, T3) |
| 000008 | OPERATIONS | E11(T1, T2, T3) |
| 000009 | SOFTWARE SUPPORT | E21(T1, T2, T3) |

9 record(s) selected.

Refine Query

Copyright University of Southampton (1996)
db Navigator project (home page)

Figure 14: Results from querying the DEPARTMENT table.

Figure 15 shows the result table after selecting this link. The next step is to find out which projects and activities each employee works on. The table shows that a link is available from the EMPNO column to the EMP_ACT (employee activity) table. The user selects the EMP_ACT link available below the EMPNO column heading (this is a link of type 4).

parallel db & tp
research group

Show Executed SQL Query

| EMPNO <small>(link to EMPLOYEE)</small> | EMP_ACT <small>(link to EMPLOYEE, PROJECT)</small> | FIRSTNAME | MIDINIT | LASTNAME | WORKDEPT <small>(link to DEPARTMENT)</small> | PHONENO | HIREDAT |
|--|---|-----------|---------|-----------|---|---------|------------|
| 000001 | 000001 | IRVING | F | STERN | A00 | 4423 | 14/09/1973 |
| 000002 | 000002 | BRUCE | | ADAMSON | B01 | 4310 | 12/02/1972 |
| 000003 | 000003 | ELIZABETH | R | PIANEA | C01 | 3782 | 11/10/1977 |
| 000004 | 000004 | MASATOSHI | J | YOSHIMURA | D01 | 2890 | 15/05/1978 |
| 000005 | 000005 | MARILYN | S | SCOUTTEN | D11 | 3652 | 07/07/1973 |
| 000006 | 000006 | JAMES | H | WALKER | D21 | 3956 | 26/07/1974 |
| 000007 | 000007 | DAVID | | BROWN | E01 | 4501 | 03/03/1966 |
| 000008 | 000008 | WILLIAM | T | JONES | E11 | 0943 | 11/04/1979 |
| 000009 | 000009 | JENNIFER | K | LUTZ | E21 | 0672 | 29/08/1968 |

9 record(s) selected.

Refine Query

Figure 15: Browsing to find all employees in department number 'D11'.

Figure 16 shows the new result table. The employee activity data has been inlined (joined) with the employee fields. The number of rows has increased since some employees work on more than one activity within a project.

| EMP_ACT.PROJNO <i>(link to PROJECT)</i> | EMPLOYEE.WORKDEPT <i>(link to DEPARTMENT)</i> | EMPLOYEE.LASTNAME | EMP_ACT.ACTNO |
|--|--|-------------------|---------------|
| MA2112 | D11 | ADAMSON | 60 |
| MA2112 | D11 | ADAMSON | 180 |
| MA2112 | D11 | PIANKA | 60 |
| MA2112 | D11 | YOSHIMURA | 60 |
| MA2112 | D11 | YOSHIMURA | 70 |
| MA2112 | D11 | YOSHIMURA | 80 |
| MA2112 | D11 | SCOUTTEN | 70 |
| MA2112 | D11 | WALKER | 70 |
| MA2112 | D11 | WALKER | 80 |
| MA2111 | D11 | BROWN | 60 |
| MA2111 | D11 | BROWN | 60 |
| MA2111 | D11 | JONES | 80 |
| MA2111 | D11 | JONES | 180 |
| MA2111 | D11 | LUTZ | 40 |

Figure 16: Browsing to show project activities for each employee.

The user now has details about all employees and their activities within the 'MANUFACTURING SYSTEMS' department. Project details are limited to just a display of the project number at this stage. However, the column labelled EMP_ACT.PROJNO shows a link to the PROJECT table. The PROJECT link is selected to expand the project number to full project details (this is a link of type 3). The result is shown in Figure 17.

As an aside, a button labelled 'Refine Query' follows each result. Pressing this button allows the user to remove columns from the displayed table, and/or allows the user to add further conditions to reduce the number of rows. Figure 16 and subsequent screens have been refined to remove columns which are not of interest, to reduce the data for the screen shots. An example of refining a query is given below.

| PROJECT.RESPEMP (link to EMPLOYEE) | EMPLOYEE.LASTNAME | PROJECT.PROJNAME | EMP_ACT.ACTNO |
|---|-------------------|---------------------|---------------|
| 00000 | BROWN | W L PROGRAM DESIGN | 50 |
| 00000 | BROWN | W L PROGRAM DESIGN | 60 |
| 00001 | LUTZ | W L PROGRAM DESIGN | 40 |
| 00010 | ADAMSON | W L ROBOT DESIGN | 60 |
| 00010 | ADAMSON | W L ROBOT DESIGN | 180 |
| 00010 | YOSHIMURA | W L ROBOT DESIGN | 60 |
| 00010 | YOSHIMURA | W L ROBOT DESIGN | 70 |
| 00010 | WALKER | W L ROBOT DESIGN | 70 |
| 00010 | WALKER | W L ROBOT DESIGN | 90 |
| 00010 | PIANKA | W L PROD CONT PROGS | 60 |
| 00010 | YOSHIMURA | W L PROD CONT PROGS | 80 |
| 00010 | SCOUTTEN | W L PROD CONT PROGS | 70 |
| 00010 | JONES | W L PROD CONT PROGS | 80 |
| 00010 | JONES | W L PROD CONT PROGS | 180 |

Figure 17: Browsing to inline full project details.

By browsing through the database the first part of the query has been answered. That is:

*Find all the employees in the 'MANUFACTURING SYSTEMS' department.
For each employee show which projects they are working on, and their activities within those projects.*

In order to complete the final part of the query it is necessary to refine the result. This is achieved by pressing the 'Refine Query' button. Figure 18 shows how to refine a query. A query form is displayed which shows all columns in the current result table. Any existing conditions are also displayed. The EMPLOYEE.WORKDEPT field shows a condition 'D11' which has been added as a result of previous browsing which requested this department. Note that the button to the right of the label is not depressed for this column. This is because, although a condition exists on this field, the column is not currently displayed.

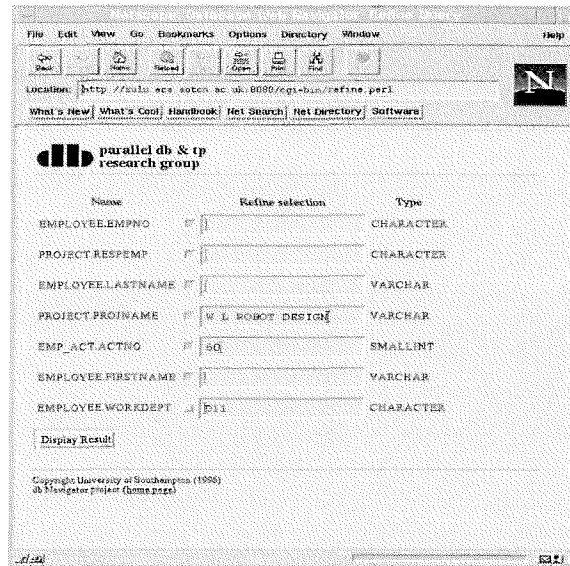


Figure 18: Refining a query during the browsing stage.

The user now has to add two further conditions. The PROJECT.PROJNAME must match 'W L ROBOT DESIGN' and the EMP_ACT.ACTNO must match '60'. Figure 19 shows the result of this refined query. Through an initial query and subsequent browsing the results to all the initial questions have now been answered.

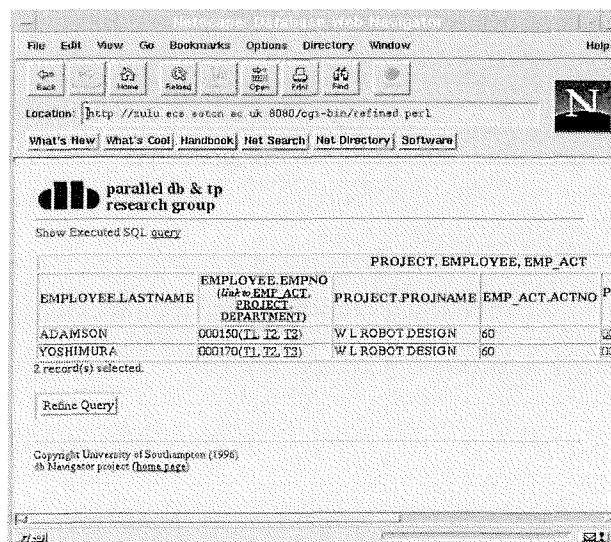


Figure 19: Query results after refinement.

The option to display the generated SQL is available for every result table. Figure 20 shows the SQL that has been generated automatically to obtain the result of Figure 19.

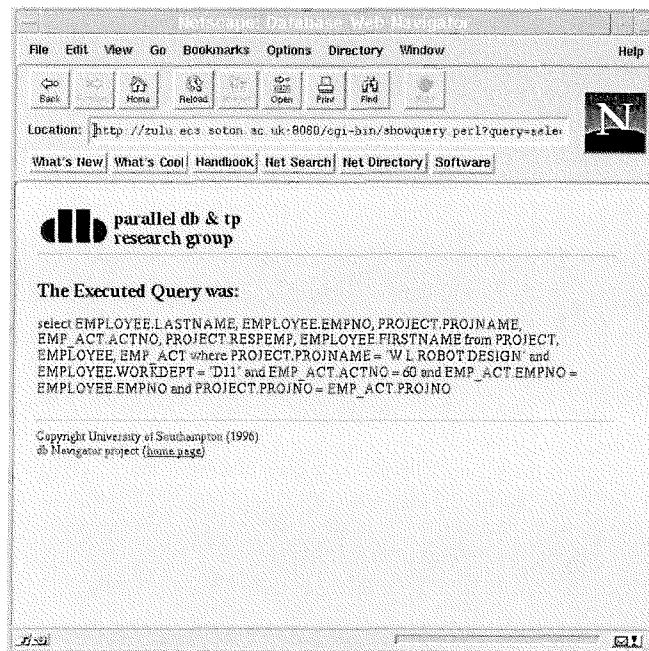


Figure 20: Displaying the SQL that generated the result.

This example did not provide an example of how LOB data is retrieved. An example of this is deferred until Section 5.2.3.2 of the description of the EASIA prototype.

4.6 Conclusions

The DBbrowse prototype was an early system that demonstrated a dynamic Web interface to an underlying object-relational database. Unlike GBIS, which was application domain specific, DBbrowse provides automatic generation of generic Web interfaces to facilitate rapid deployment of interactive Web-based applications by developers with little Web development experience. The dynamic approach, based on an underlying database, provides a number of advantages over static Web pages. In particular, no HTML page maintenance is required when the data changes and hypertext links do not become ‘stale’. Any changes made to the database are reflected the next time that the data is accessed. Schema changes are also handled automatically. By connecting to the database in real-time the sophisticated search engine provided by the database management system is made use of.

DBbrowse can generate Web interfaces to object-relational databases with intuitive query capabilities. DBbrowse automatically handles non-traditional BLOB and CLOB datatypes by delivering them to the user’s Web browser with an

appropriately specified content type. DBbrowse also demonstrated a novel method for browsing databases to further facilitate users with little database experience. The data browsing technique does not rely on explicit relationships defined in the data model, but extracts implied relationships from the relational model to dynamically include hyperlinks in results. DBbrowse differed from commercial products offering Web/database integration at the time, which required programming effort, for example, to define page templates and predefined embedded queries and hypertext links.

An automated system such as DBbrowse also exhibits a number of possible disadvantages compared to custom built or programmable interfaces. For example, whilst the earlier application specific GBIS interface interpreted text files as containing coordinates for post-processing with GNUPLOT, DBbrowse relies on the user's browser to display different content according to helper applications specified within the user's browser (for various received content types). Furthermore DBbrowse used names extracted from the database catalogue to label the QBE-like interface, with no level of customisation.

The implementation of DBbrowse also had a number of possible weaknesses. Firstly, DBbrowse was based on CGI scripts written in PERL with the associated performance issues discussed in Chapter 2. Second, DBbrowse used embedded system calls in the PERL scripts to communicate with the database. This restricts portability of the system to different databases.

The next chapter describes the EASIA prototype. This system addresses the weaknesses described above by providing an architecture that incorporates customisation, extensible post-processing capabilities, improved performance and database independence. The EASIA architecture also goes beyond this level of improvement by providing a distributed architecture, which can securely manage very large non-traditional data.

5 An Architecture for Management of Large, Distributed, Scientific Data

5.1 Introduction

This chapter describes the final results of this research, which culminated in the implementation of the EASIA (Extensible Architecture for Scientific Information Archives) prototype. The motivation for EASIA was a requirement of the UK Turbulence Consortium [199] [219] to provide an architecture for archiving and manipulating the results of numerical simulations. The EASIA architecture picks up where DBbrowse left off by providing automated Web-based management of non-traditional data, with the addition of customisation, extensible post-processing capabilities, improved performance and database independence. Beyond these initial enhancements, EASIA incorporates new Web-based data management techniques that were necessitated by the nature of the data being generated by the Consortium. The non-traditional data to be managed consisted, in the main, of *large* (multi-gigabyte) multidimensional scientific datasets, often stored in unformatted FORTRAN output files. This chapter therefore describes research that provided solutions to the many new problems that arose whilst trying to manage large datasets, in the relatively low bandwidth environment provided by the Web.

One of the objectives of the UK Turbulence Consortium is to improve collaboration between groups working on turbulence by providing a mechanism for dissemination of data to members of the turbulence modelling community. The consortium is now running simulations on larger grid sizes than has previously been possible, using the United Kingdom's new national scientific supercomputing resource⁴. One complete simulation, comprising perhaps one hundred timesteps, requires a total storage capacity of some hundreds of gigabytes. This necessitated new Web-based mechanisms for storage, searching and retrieval of multi-gigabyte

⁴ A 576 processor Cray T3E-1200E situated at the University of Manchester, which forms part of the *Computer Services for Academic Research (CSAR)* service run on behalf of the UK research Councils. <http://www.csar.cfs.ac.uk/>

datasets that are generated for each timestep in a simulation. In particular, an architecture was required that could minimise bandwidth usage whilst performing these tasks.

The Caltech Workshop on Interfaces to Scientific Data Archives [235] identified an urgent need for infrastructures that could manage and federate active libraries of scientific data. The workshop found that whilst databases were much more effective than flat files for storage and management purposes, trade-offs existed as the granularity of the data objects increased in size. If a database is being created to manage metadata describing scientific results, then ideally the database should also be used to store the actual scientific result data in a unified way. However for large output files it becomes costly and inefficient to store the data as BLOBs within the database.

EASIA uses an implementation of the new SQL:1999 DATALINK type, defined in *SQL Management of External Data (SQL/MED)* [60], to provide database management of scientific metadata and large, distributed result files simultaneously with integrity. This technology is applied to the Web, by providing a user interface to securely manage large files in a distributed scientific archive, despite limited bandwidth.

A database table containing an attribute defined as a DATALINK type can store a URL that points to a file on a remote machine. Once a URL has been entered into the database, software running on the remote machine ensures that the file is treated as if it was actually stored in the database, in terms of security, integrity, recovery and transaction consistency. This mechanism is used to allow large result files to be *distributed* across the Web.

The EASIA architecture provides the following features for scientific data archiving:

1. *Users of the scientific archive, who may have little or no database or Web development expertise, can access the system.* EASIA uses techniques from DBbrowse as the basis for its automatically generated user interface. However, instead of querying the database catalogue directly to produce the interface, EASIA generates a user interface from *an XML user interface specification file*

(XUIS), to a database that supports DATALINKs. An XML Document Type Definition (DTD) has been created to define the structure of the XUIS. An initial XUIS is constructed automatically using metadata extracted from the database catalogue. This can then be customised to provide specialised features in the interface.

2. *The default interface specification provides a data browsing facility to maintain a Web-based feel.* This is similar to database browsing in DBbrowse with additional functionality (described in detail later in the chapter) associated with DATALINK columns.
3. *Large result files can be archived at (or close to) the point where they are generated.* For the UK Turbulence Consortium, this means that files can, for example, be archived at the Manchester site on a local machine that is connected via a high-speed link to the supercomputer. By entering the URLs of these files (using a Web-based interface) into a DATALINK column of a remote database (hosted at Southampton for example) database security and integrity features can then be applied to the files. An alternative to this, which achieves similar database security and integrity for result files, is to use a Web interface to upload a file across the Internet and then store it as a BLOB in a centralised archive at the new location. However, this alternative is not feasible for large files due to limited Internet bandwidth. Even if a file can be transferred to a centralised site, additional processing cost is incurred (which is not present with the DATALINK mechanism) when loading the file as a BLOB type into the database.
4. *Because simulation results are stored in unmodified files, existing post-processing applications, that use standard file I/O techniques, can be applied to the files without having to rewrite the applications.* An alternative would be to modify applications to first access result objects from a database, but this would be very undesirable for many scientific users who often apply post-processing codes written in FORTRAN.
5. The Caltech workshop [235] recommended ‘cheap supercomputers for archives’. The report suggests that research is necessary to establish whether high-performance computing resources, built from commodity components, are viable for data-intensive applications (as well as compute-intensive applications, as has

been shown to be the case in for example, the Beowulf project [228]). *The EASIA architecture is being used to build a large scientific archive from commodity components, with multiple distributed machines acting as file servers for a single database.* Security, backup and integrity of the file servers can be managed using SQL/MED. This arrangement can provide high performance in the following areas:

- Data can be distributed so that it is physically located closest to intensive usage.
- Data distribution can reduce access bottlenecks at individual sites.
- Each machine provides a *distributed processing capability* that allows multiple datasets to be post-processed simultaneously. Suitable user-directed post-processing, such as array slicing and visualisation, can significantly reduce the amount of data that needs to be shipped back to the user. EASIA can archive not only data in a distributed fashion, but also applications. Post-processing codes that have been archived by the system can be associated with remote data files using the XUIS. This allows dynamic server-side execution of the stored applications, with chosen datasets as input parameters. These applications are loosely coupled to the datasets (in a many-to-many relationship) via XML defined interfaces defined in the XUIS. This allows reuse of these server-side post-processing operations. This level of extensibility and post-processing capability offers a significant advantage over the DBbrowse system for scientific data archiving. EASIA is an *active data archive*, according to the definition by Hawick and Coddington [112] (see Section 1.1), since additional data can be derived on demand as value-added data products or services.

The rest of this chapter expands the description of the EASIA architecture and features given above. Section 5.2 describes the system architecture and user interface in detail. Sections 5.3 discusses the implementation of EASIA and design decisions that were made. Finally some conclusions are presented.

5.2 System Architecture and User Interface

This section starts with a high level view of the EASIA architecture. The use of XML to specify the functionality of the user interface via the XUIS is then described. The differences between searching and browsing in EASIA and DBbrowse, and the effect of the XUIS customisation on these features are covered next. The ability to archive applications as well as data is discussed with examples of XUIS modification to include post-processing *operations*. The next section then shows how post-processing can be achieved not only via these archived applications, but also by allowing users to upload code for secure server-side execution against datasets. Finally, a brief description is given of the user administration features within EASIA.

5.2.1 System Architecture

The EASIA architecture is illustrated in Figure 21. It consists of a database server host (located at Southampton University) and a number of file server hosts that may be located anywhere on the Internet. All of these hosts have an installed Web server to allow HTTP (or HTTPS) communications directly from a Web browser client.

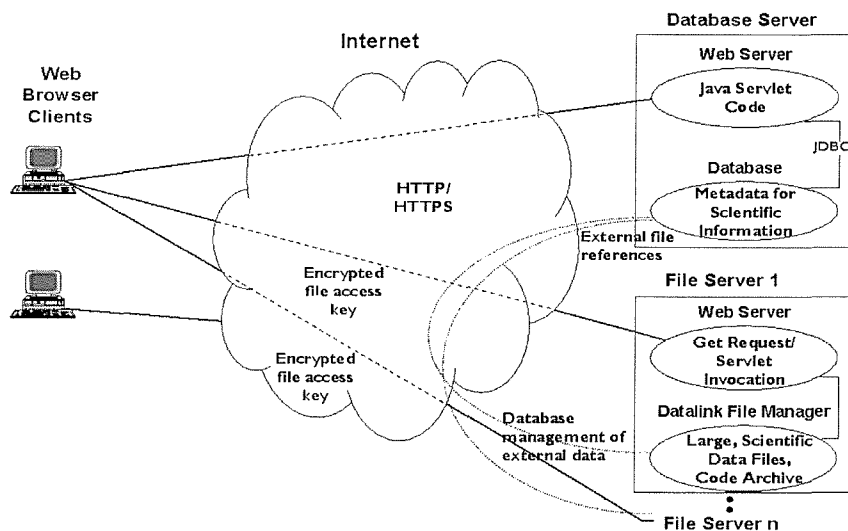


Figure 21: System architecture.

A user of the system initially connects to the Web server on the database server host. The URL of the system invokes a Java Servlet program. Separate threads within the

Servlet process handle requests from multiple Web browser clients. Each user is first presented with a login screen (Figure 22). Once a user has been verified, interaction with the database is possible via HTML pages that are dynamically generated by the Servlet code. These pages consist of HTML forms, JavaScript and hypertext links.

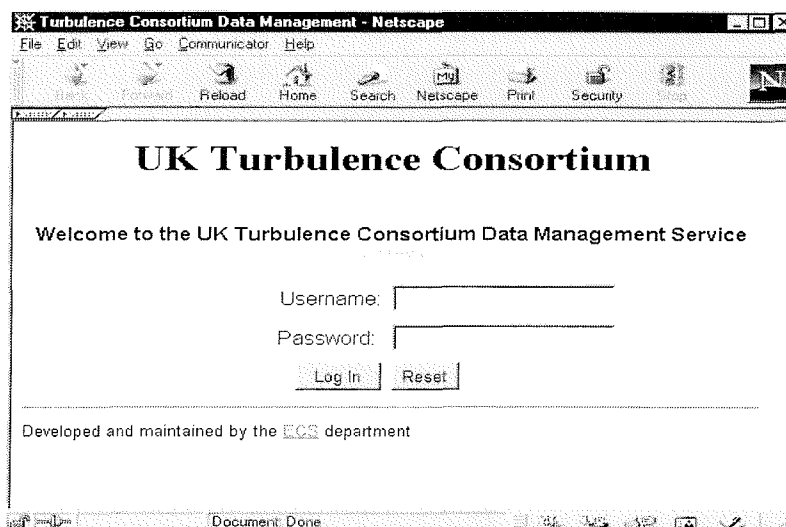


Figure 22: Login screen.

The database server stores metadata describing the scientific information such as, simulation titles, descriptions and authors. This data is stored locally in the database and is accessed by the Servlet code using JDBC. The data is represented by tables with attributes defined as standard SQL-types, BLOB types, or CLOB datatypes. The latter types are used in the system to store *small* image/video files, executable code binaries or *small* ASCII files containing source code or descriptive material for the turbulence simulations.

For scientific data archiving, an essential feature of the interface is the novel use of remote file servers that store files referenced by attributes defined as DATALINK SQL-types. These file servers manage the *large* files associated with simulations, which have been archived where they were generated. When the result of a database access yields a DATALINK value, the interface presents this to the user as a hypertext link that can be used to download the referenced file. The URL contains an encrypted key that is prefixed to the required file name. This key is verified by DATALINK file manager code (running on the file server host) which

intercepts attempts to access any files controlled by the remote database. Without this key, files cannot be accessed, either via the locally installed Web server or directly from the file system by a locally connected user. (The DATALINK mechanism is discussed further in Section 5.3.2.) As well as allowing a user to simply download a dataset, the interface also allows user-selected post-processing codes to execute on the remote file server host to reduce the volume of data returned.

5.2.2 XML Specification of the User Interface

The system is started by initialising the Java Servlet code with an XUIS. This initialisation can take several seconds but it is a one-off cost that is not repeated during subsequent requests from users. A default XUIS can be created prior to system initialisation using a tool that has been created. This tool, written in Java, uses JDBC to extract data and schema information from the database being used to archive simulation results. This default XUIS conforms to a DTD that has been defined for EASIA. The default XUIS can be customised prior to system initialisation. The XUIS contains table names, column names, column types, sample data values for each column, and details of primary keys and foreign keys that participate in referential integrity constraints. The XUIS also allows aliases to be defined for table and column names.

The following sections explain how this information is used to provide an interface with searching and browsing capabilities and how it facilitates customisation of the interface and dynamic execution of post-processing applications.

5.2.3 Searching and Browsing Data

After logging in users of the interface can begin to locate information in the scientific archive by searching or browsing data or by using a combination of both techniques. DBbrowse required the user to begin in searching mode and to submit at least one query form before browsing of results could commence. EASIA allows browsing from the start (a feature that was added in response to a request from a user). This is accomplished by allowing the user to request 'All Data for Table' via a single hypertext link (see for example, Figure 23), instead of initially requesting the

search form for that table. The effect is to return all the rows from the table with all available hypertext links marked up in the results. The two subsections that follow explain further enhancements to searching and browsing in EASfA.

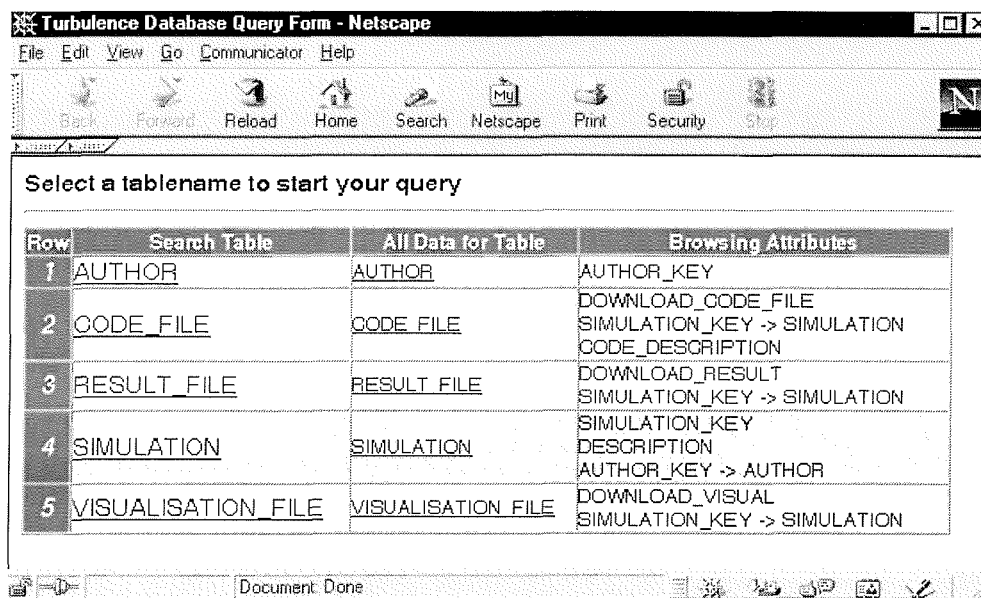


Figure 23: Table selection screen.

5.2.3.1 Searching

To search for data a user selects a link to a query form for a particular table (indicated by the hyperlinks in the 'Search Table' column of the example shown in Figure 23). This provides a query form similar to that in DBbrowse (see for example, Figure 24). The majority of users of the scientific data archives are frequent repetitive, simple queries. For this category of user, a form-based visual query system (VQS) represents a better alternative than an iconic VQSs or diagrammatic VQSs [35]. Form-based interfaces also facilitate non-expert users by capitalising on the natural tendency of people to organise data into tables [35]. An enhancement over DBbrowse is the availability of sample values for the text entry fields, available from drop down lists. Two of the major problems faced by users of text based query languages, such as SQL, are a semantic one in choosing the correct attributes and relationships between them, and a syntactic one in building the correct textual expression that corresponds to the chosen query. By offering the user choices

of column names, table names and operators, the instance of syntactic error can be reduced. Providing column names, datatypes and *sample values* can aid substantially in narrowing the semantic meaning of a domain [111]. For example, a column called ‘TITLE’ in a personnel database could be used to store values such as ‘Mr’, ‘Mrs’, ‘Dr’, etc., or it could store a person’s job title such as ‘Chief Engineer’. A sample value will quickly clarify the intended meaning. The interface randomly selects ten sample values to be displayed for each simple attribute type. The drop-down list of samples displays the SQL-type by default as this also provides useful information.

EASIA obtains the table names, column names, data types and sample values used in the query form from the XUIS used to initialise the system (rather than directly from the database catalogue as is the case with DBbrowse). This allows customisation as discussed in Section 5.2.4.

| Attribute | Restrictions | Sample Data | Sort | Count |
|--|---------------------------|---------------|-----------------------|-----------------------|
| <input checked="" type="checkbox"/> SIMULATION_KEY | = | VARCHAR(30) | <input type="radio"/> | <input type="radio"/> |
| <input checked="" type="checkbox"/> TITLE | = | VARCHAR(254) | <input type="radio"/> | <input type="radio"/> |
| <input checked="" type="checkbox"/> DESCRIPTION | like numerical simulation | CLOB(10MByte) | <input type="radio"/> | <input type="radio"/> |
| <input checked="" type="checkbox"/> AUTHOR_KEY | = | VARCHAR(30) | <input type="radio"/> | <input type="radio"/> |
| <input checked="" type="checkbox"/> DATE_RUN | = | VARCHAR(30) | <input type="radio"/> | <input type="radio"/> |
| <input checked="" type="checkbox"/> URL | = | VARCHAR(600) | <input type="radio"/> | <input type="radio"/> |

Figure 24: Searching the archive.

5.2.3.2 Browsing

Foreign key and primary key browsing in EASIA are similar to those in DBbrowse, except that the specification of these keys in the XUIS determines whether or not the links are displayed. Links are included in the XUIS by default if referential integrity constraints are available in the database metadata. If metadata describing referential

integrity is not available, the XUIS can still be customised to include these hypertext links. (Figure 25 illustrates that the display for primary key browsing has been changed in EASIA, for the case where there may be a choice of tables to browse to. Selecting one of these values will return all the rows that the key appears in from one of the referenced table, *as indicated in the currently checked radio button in the column header.*)

The description that follows explains *BLOB and CLOB browsing* that applies to both EASIA and DBbrowse. (During the description of DBbrowse in Chapter 4 an example of the ability to retrieve LOB data through browsing was deferred until this section, as it is similar in both prototypes.) The description of DATALINK browsing at the end of this section is, however, a new and essential feature, only available in EASIA.

| Row | SIMULATION_KEY links to <input type="radio"/> SIMULATION_KEY in CODE_FILE <input type="radio"/> SIMULATION_KEY in RESULT_FILE <input checked="" type="radio"/> SIMULATION_KEY in X3EASIASQL_FILE | TITLE | DESCRIPTION | AUTHOR_KEY links to AUTHOR | DATE_RUN/URL |
|-----|---|--------------------------------|-------------|-------------------------------|--------------|
| 1 | S19990110151042 | Channel Flow DNS | 504 Byte | A19990110151042 | |
| 2 | S19990209151042 | Laminar Separation Bubbles DNS | 704 Byte | A19990209151042 | 1997/1998 |

Figure 25: Result from querying the SIMULATION table.

Figure 25 shows a sample result screen that was generated after the query from Figure 24 (CLOB DESCRIPTION field containing the words ‘numerical simulation’) was run against the SIMULATION table described by the schema shown in Figure 26. This schema was used by the EASIA prototype to store data from the UK Turbulence Consortium. The SIMULATION result table (Figure 25) contains data corresponding to simple SQL-types, such as the title and date for a simulation. As well as simple types, BLOB and CLOB types are used to store small

files that can be uploaded over the Internet. Cells associated with these types display a *LOB link* to the object. Selecting one of these links causes the data associated with the cell to be returned to the client. The link displays the size of the object in bytes, which may help users decide whether they want to retrieve the object.

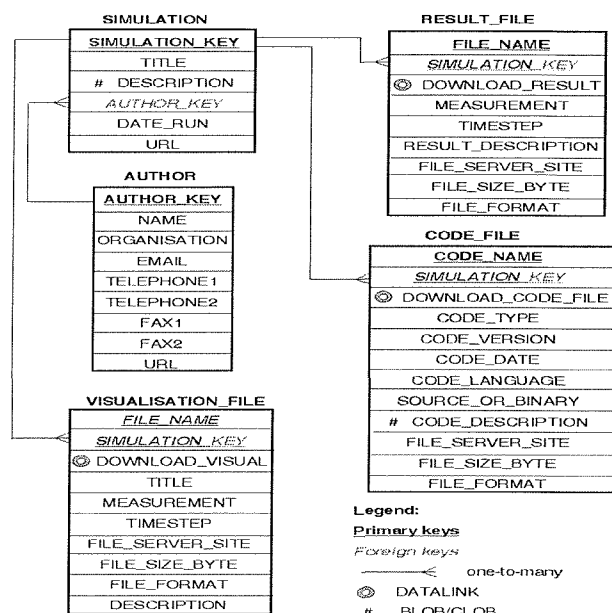


Figure 26: Sample database schema for UK Turbulence Consortium.

For example, Figure 26 indicates that the DESCRIPTION attribute in the SIMULATION table is a CLOB type. Selecting the hypertext link on the DESCRIPTION field labelled '704 Byte' in Figure 25, will retrieve the description (of the simulation entitled, 'Laminar Separation Bubbles DNS') and display it directly in the browser window since it contains character data. This is shown in Figure 27.

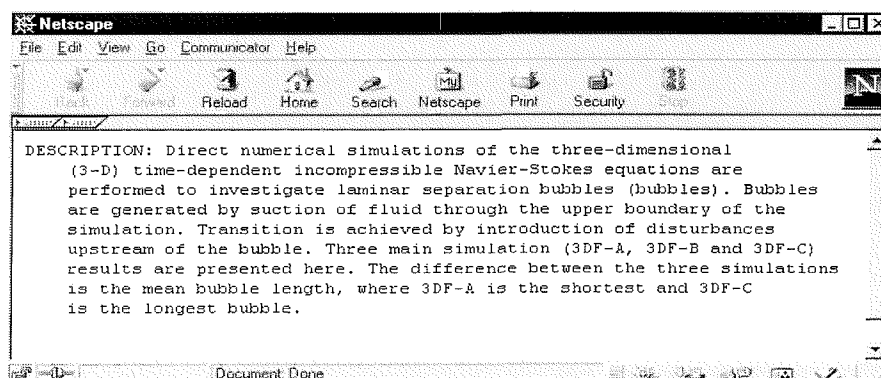


Figure 27: CLOB browsing.

For BLOB data (e.g. a stored image or executable) selecting the link will allow the user to save the data as a file, or process the retrieved data according to helper applications specified within the user's browser. EASIA attempts to identify the type of data being returned to the client and sets the content type in the HTTP header accordingly. For CLOB retrieval an attempt is made to identify the character data as either plain text or HTML content.

The sample schema of Figure 26 contains not only non-traditional LOB data, but also non-traditional data stored using attributes of the DATALINK SQL-type. These DATALINK attributes serve the main purpose of the architecture – to archive large scientific data via the Web. The attributes in the Turbulence Consortium schema are used for storing result files, code files, and visualisation files. When present in a result table, the value of a DATALINK attributes is displayed as a filename, with a hypertext link that contains an embedded encrypted key, required to access the file from the remote file server (see Section 5.3.2). If the link is selected EASIA uses the key to retrieve the file from the appropriate distributed remote file server and sets the content type accordingly. Figure 28 shows the effect of selecting a hypertext link on a DATALINK attribute in the VISUALISATION_FILE table of Figure 26. An MPEG movie of a channel flow simulation is retrieved and displayed.

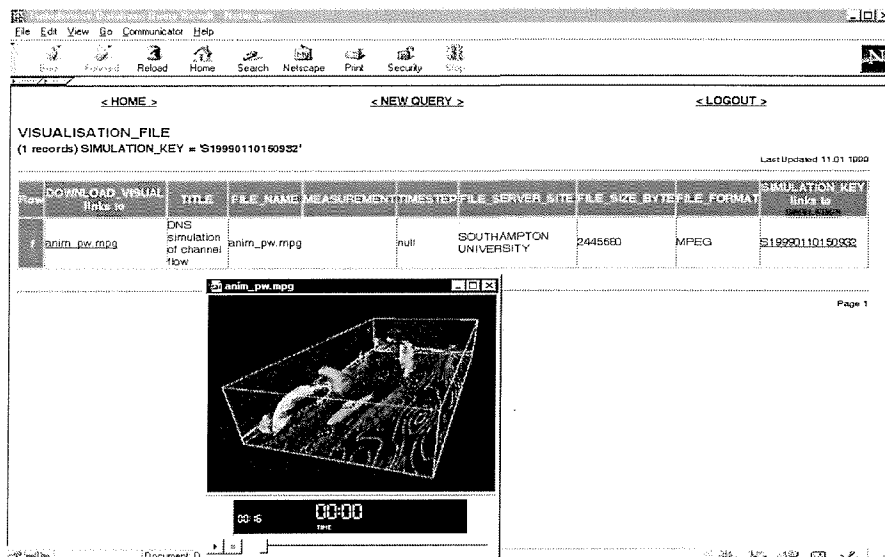


Figure 28: DATALINK browsing.

Hyperlinks on BLOB/CLOB and DATALINK types are included in the EASIA interface if the XUIS specifies a column as consisting of one of these types. The EASIA system also makes use of DATALINK columns for its post-processing services. This is an extremely important part of the architecture and is discussed separately in Sections 5.2.5 and 5.2.6. But first, the mechanism for *customising* the searching and browsing aspects of the user interface via XUIS modification is described.

5.2.4 Interface Customisation through XUIS Modification

The XUIS can be customised to provide aliases for table names and column names as well as user-defined sample values. It is also possible to prevent tables or columns being displayed in the query forms and results, by removing them from the XUIS. Hypertext links for navigation between tables can also be added or removed by modification of the XUIS. A fragment of the XML for the XUIS is shown below:

```
<table name = "AUTHOR" primaryKey = "AUTHOR.AUTHOR_KEY">
  <tablealias> Author </tablealias>
  <column name = "AUTHOR_KEY" colid = "AUTHOR.AUTHOR_KEY">
    <type>
      <VARCHAR/>
      <size>30</size>
    </type>
    <!-- Primary key links for this column defined next -->
    <pk>
      <refby tablecolumn = "SIMULATION.AUTHOR_KEY"/>
    </pk>
    <samples>
      <sample>A19990110151042</sample>
      <sample>A19990209151042</sample>
    </samples>
  </column>
</table>
```

Initial feedback from users indicated that they would like the option to replace the keys displayed in the results with more meaningful data. The XUIS can be modified to specify a 'substitute column' to be displayed in the results in place of foreign keys. The substitute column is a user-specified column from the table that the

foreign key references. For example, the AUTHOR_KEY attributes in the SIMULATION result table of Figure 25 can be replaced by, say, the NAME attribute from the referenced AUTHOR table (see the schema in Figure 26). These values still provide a browsing link to the full author details. The XUIS can also be customised to replace primary keys with the names of linked tables. Both of these customisations are illustrated in Figure 29, which shows a similar query result to Figure 25 but with a modified XUIS. Note that the foreign key links to the author table now list the author's name, which is far more intuitive than the long character string (key value) that was previously displayed. Also, the long character strings values of SIMULATION_KEY (primary key values) have been replaced instead with hypertext links showing the table names, 'CODE_FILE', 'RESULT_FILE' and 'VISUALISATION_FILE'. Again this is more intuitive for the user, as it is now possible, for example, to obtain all the result files for the 'Channel Flow Simulation' in the first row of Figure 29 by selecting the 'RESULT_FILE' link.

| Row | SIMULATION_KEY links to SIMULATION_KEY in CODE_FILE | SIMULATION_KEY links to SIMULATION_KEY in RESULT_FILE | SIMULATION_KEY links to SIMULATION_KEY in VISUALISATION_FILE | TITLE | DESCRIPTION | AUTHOR_KEY links to Name in AUTHOR | DATE_RUN/URL |
|-----|--|--|---|--------------------------------|-------------|--|--------------|
| 1 | CODE_FILE | RESULT_FILE | VISUALISATION_FILE | Channel Flow DNS | 604 Byte | Prof Neil D Sandham | |
| 2 | CODE_FILE | RESULT_FILE | VISUALISATION_FILE | Laminar Separation Bubbles DNS | 704 Byte | Mahbubul Alam | 1997/1998 |

Figure 29: Customised display of results from a query on the SIMULATION table.

An example of the XML used to specify a substitute column for a foreign key is as follows:

```

<table name = "SIMULATION" primaryKey = "SIMULATION.SIMULATION_KEY">
...
  <column name = "AUTHOR_KEY" colid = "SIMULATION.AUTHOR_KEY">
...
    <!-- Foreign key link defined here, with possible substitute columns -->
    <fk tablecolumn = "AUTHOR.AUTHOR_KEY" substcolumn =
        "AUTHOR.NAME"/>

```

5.2.5 Suitable Processing of Data Files Prior to Retrieval: ‘Operations’

McGrath [149] asserts that users of scientific data servers usually only require a selected part of a dataset that is typically stored in a large complex file, and that it is critical therefore, to provide a means to identify data of interest, and to be able to retrieve the selected data in a useable format. Ferreira *et al.* [92] also discuss multi-dimensional datasets that typify scientific results, stating that applications that post-process such data, do not typically use all possible data in the dataset. Hawick and Coddington [112] describe the need for *active* data archives, where data is generated on-demand from existing data because many of the data items required by users are actually obtained by processing *passive* data. This section explains how EASIA can be extended to provide an active archive that facilitates server-side post processing of stored datasets prior to delivery to the client.

Referring back to the system architecture, Figure 21 shows that the Web servers on the remote file servers can process a standard HTTP ‘Get request’ to return a complete result file to the client (if the encrypted file access key is correct). This is the functionality that has been described so far. However, Figure 21 also shows that a Java Servlet can be invoked on the file server to handle the incoming request.

A *fundamental* feature of the EASIA architecture is that it allows the XUIS to be modified to allow *post-processing* applications that have been archived using DATALINK values, to be executed dynamically server-side, to reduce the data volume returned to the user. These applications can consist of Java classes or any other executable format, suitable for the file server host on which the data resides, including C, FORTRAN and scripting languages. These applications do not have to

be specially written for the architecture (in fact, *operations* stored as DATALINKs can be downloaded separately for standalone execution elsewhere) and they can be packaged in a number of different formats including various compressed archive formats (such as tar.Z, gz, zip, tar etc.). The only restriction is that the initial executable file accepts a filename as a command line parameter and that any file output uses relative path names. The filename accepted as a command line parameter corresponds to the name of a dataset that is to be processed. Archived applications are associated with a number of archived datasets using a markup syntax that has been defined for 'operations' in the XUIS. If the application allows other user-specified parameters, the syntax for operations has been defined so that an HTML form will be created to request these parameters at invocation time. The XML syntax defined for operation parameters is similar to that used for HTML forms. Applications that correspond to operations do not have to be stored as DATALINKs. They can also be specified in the XUIS in the form of URLs that refer to CGI programs or Servlets that run on the file server host to provide a post-processing service. A fragment of the XUIS definition for an operation is shown below:

```
<table name = "RESULT_FILE"
    primaryKey = "RESULT_FILE.FILE_NAME RESULT_FILE.SIMULATION_KEY">
    ...
    <column name = "DOWNLOAD_RESULT" colid =
        "RESULT_FILE.DOWNLOAD_RESULT">

        <type><DATALINK/></type>
    <operation name="GetImage" type="JAVA" filename="GetImage.class"
        format="jar" guest.access="true" column="false">
        <!-- Only allow this operation on attributes in this column that meet the
        following conditions- -->
        <if>
            <condition colid="RESULT_FILE.SIMULATION_KEY">
                <eq>'S19990110150932'</eq>
            </condition>
            <condition colid="RESULT_FILE.MEASUREMENT">
                <eq>'u,v,w,p'</eq>
```

```

        </condition>
    </if>

<!-- The location of the operation can be a URL or code that is archived using
another DATALINK column, retrieved with the following query -->

<location>
    <database.result colid="CODE_FILE.DOWNLOAD_CODE_FILE">
        <condition colid="CODE_FILE.CODE_NAME">
            <eq>'GetImage.jar'</eq>
        </condition>
    </database.result>
</location>
<description>
    <!-- Place a description of the operation here or use a database.result
to select the description text from the database -->
</description>

<!-- Define any parameters that the above operation uses. These can be
constant, selected from database results, or prompted for from the user via
an HTML form -->
<parameters>
    <param>
        <variable>
            <description>Select the slice you wish to visualise:
</description>
            <select name="slice" size="4">
                <option value="x0">x0=0.0</option>
                <option value="x1">x1=0.1015625</option>
                ...
                <!-- alternatively a select can use a
database.result element instead of options -->
            </select>
        </variable>
    </param>
</parameters>

```



```

<variable>
  <description>Select velocity component or pressure:
</description>
  <input type="radio" name="type"
        value="u">u speed</input>
  <input type="radio" name="type"
        value="v">v speed</input>
  ...
</variable>
</param>
</parameters>
</operation>
</column>

```

An example of the processing associated with an operation stored as a DATALINK follows. This corresponds to the *GetImage* operation in the XUIS fragment above. Figure 30 shows the result of a query on the RESULT_FILE table from the schema of Figure 26. The columns containing a cog icon in the column header present the user with operations to execute against the preceding DATALINK column containing dataset result files. Not that the *GetImage* (and *GetChunk*) operation only appears against the DATALINK values that are contained in rows that meet the other conditions defined in the XUIS (for example, a specified SIMULATION_KEY and MEASUREMENT type).

The screenshot shows a web browser window titled "Turbulence Database - Netscape". The page displays a table of simulation results. The table has columns for Row, DOWNLOAD RESULT (with a cog icon), MEASUREMENT, FILE FORMAT, RESULT DESCRIPTION, and SIMULATION (with a cog icon). The rows contain data for various simulation runs, including data files (data_01.dat to data_24.dat), hmap00.hdf, and hmap01.hdf. The operations available for each row are indicated by cog icons in the DOWNLOAD RESULT and SIMULATION columns. The table is titled "Result File (93 records) No Restrictions" and was last updated on 11/01/1999. The browser interface includes a menu bar (File, Edit, View, Go, Communicator, Help) and a status bar at the bottom.

| Row | DOWNLOAD RESULT links to | MEASUREMENT | FILE FORMAT | RESULT DESCRIPTION | SIMULATION links to |
|-----|-----------------------------|-------------|-------------|--|-----------------------------------|
| 1 | data_01.dat | u,v,w,p | IEEE binary | | Channel Flow I |
| 2 | data_02.dat | u,v,w,p | IEEE binary | | Channel Flow I |
| 3 | data_20.dat | u,v,w,p | IEEE binary | | Channel Flow I |
| 4 | data_24.dat | u,v,w,p | IEEE binary | | Channel Flow I |
| 5 | hmap00.hdf | | HDF | 2 raster image plus nice annotation | Binary Simulati HDF data files |
| 6 | hmap01.hdf | | HDF | HDF SDS and Vdata example | Binary Simulati HDF data files |
| 7 | hmap02.hdf | | HDF | 8-bit raster images for layer feature demo | Binary Simulati HDF data files |

Figure 30: Result table showing 'operations' available for post-processing datasets.

Selecting the 'GetImage' hyperlink corresponding to the DATALINK value containing 'data_01.dat' produces the screen shown in Figure 31. This describes what the operation does, and requests that the user specifies any input parameters that were defined for the operation in the XUIS. The information contained in this form was specified in the XUIS and the operation itself is stored as a DATALINK in the CODE_FILE table of the schema shown in Figure 26. The example *GetImage* operation extracts a user-specified slice from a dataset corresponding to a simulation of turbulence in a 3-D channel and returns a GIF image of a user-selected velocity component or pressure from the flow field. The result (applied to 'data_01.dat') is shown in Figure 32. When the user submits the request in Figure 31 this initiates a sequence of processing. The location of the DATALINK file corresponding to the dataset to be processed and the location of the DATALINK file corresponding to the *GetImage* operation along with the selected parameter values, are passed to a Servlet running on the file server host. This Servlet also receives other information (stored as hidden fields in the HTML form of Figure 31) that was initially specified in the XUIS, such as the executable type for the operation (e.g. FORTRAN, C or Java) and whether the application is contained in a compressed archive format.

GetImage

This program reads a data file which contains the flow field from the channel flow simulation by Prof Neil Sandham and outputs a GIF image of user-specified slice from the input data.

2 files are output. image.gif contains the requested speed or pressure image. scale.gif contains the colour map. The minimum and maximum value represented by the colour map is written to standard out along with the time value for the dataset.

Select the slice you wish to visualise:

x0=0.0
x1=0 1015625
x2=0 203125
x3=0 3046875

Select velocity component or pressure:

u speed
 v speed
 w speed
 pressure

Submit Query Reset Form

Figure 31: 'Operation' description and parameter input form.

The Servlet then makes external system calls to unpack the application in the temporary directory created for the user and to execute the application with the

given dataset filename and other command line arguments (obtained from the parameters). The Servlet redirects standard output and standard error from the executable to files in the temporary directory. When the application EASIA displays the name of the operation and the parameters that were chosen as its arguments, then any standard output and error messages are displayed. This is followed by a checklist of result files produced by the application from which the user selects files to download as illustrated in Figure 32. Since the *GetImage* operation produces GIF images, which the browser understands, these are also displayed directly on the Web page.

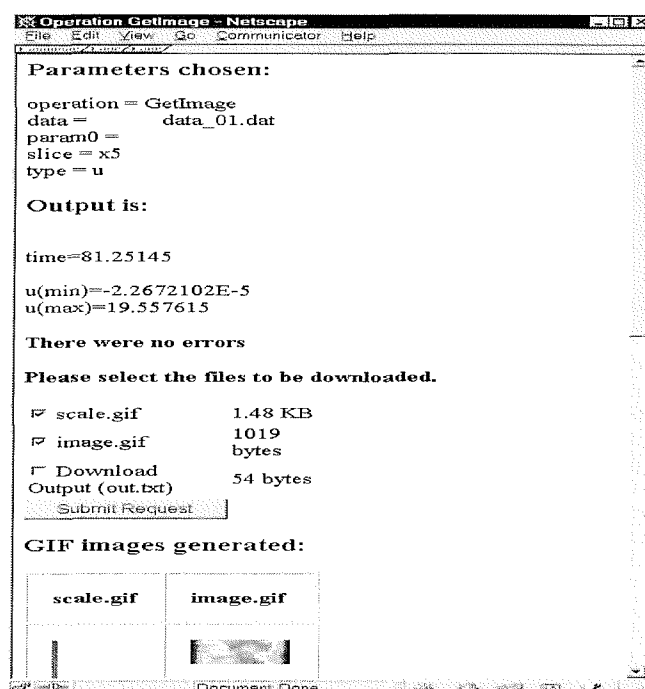


Figure 32: Output from 'operation' execution.

When the user hits 'submit request' in the results page of Figure 32 a multipart HTTP response is used to return all the files that were selected on the check box list. The multipart HTTP response uses a technique called Server Push where a sequence of response pages, not necessarily of the same content type, are sent to the client. The socket connection between the client and server remains open until the last page has been sent. In this case a series of 'Save As' dialogue boxes are presented to the user in turn. A default filename appears in the 'Save As' dialogue box corresponding to the actual file currently available for download.

After downloading the files the temporary directory and all of its contents are deleted. If the user does not download the files in this way the temporary directories are removed periodically when they are no longer being used. This is an important feature in multi-user system such as EASIA, which could otherwise produce large number of post-processing results on each file server.

The above example demonstrated a post-processing service consisting of an application that was archived by the database using DATALINKs, and loosely coupled to the dataset using the XUIS. The application is also available for use by other datasets by appropriate specification in the XUIS. Furthermore, since applications do not require any special coding specific to EASIA, the *GetImage* code used by the operation above can also be downloaded from the archive. This is achieved by locating it in the CODE_FILE table and selecting the hypertext link that accompanies the filename since it is stored as a DATALINK. Users with appropriate privileges are thus free to download archived datasets *and* operations for execution in other environments, if they choose not to post-process using the built in operation mechanism.

Operations in EASIA can consist not only of code archived in DATALINK types, but also of post-processing services that are identified by URLs specified in the XUIS. A URL-based service is hosted on the same machine as the dataset and consists of a CGI program or Servlet that can accept an HTTP POST request along with associated parameters. An example of this type of operation follows.

NCSA's *Scientific Data Browser* (SDB) [243] [149] consists of a CGI program for browsing data in a number of scientific formats such as the Hierarchical Data Format (HDF)⁵ [115]. This CGI program is written in C and provides functionality such as visualisation and extraction of subsets of data specifically written for the supported scientific data formats. However SDB does not provide

⁵ HDF, created at the National Center for Supercomputing Applications, is a multiobject self-defining file format for sharing scientific data in a distributed environment. The HDF library contains interfaces for storing and retrieving compressed or uncompressed raster images with palettes, and an interface for storing and retrieving n-Dimensional scientific datasets together with information about the data, such as labels, units, formats, and scales for all dimensions.

sophisticated storage and search capabilities for archiving and selecting the data files that it can process. It assumes that these files are simply placed in a directory structure belonging to the Web server. It is possible however, to incorporate SDB as an operation in EASIA by the inclusion of simple markup in the XUIS, which specifies the SDB URL (for post-processing appropriate result files). This allows SDB to benefit from secure data management architecture and search capabilities provided by EASIA. The XML fragment to include the *SDB operation* in EASIA is shown below:

```
<operation name="SDB" type="" filename="" format="" guest.access="true"
                                column="false">
  <if>
    <condition colid="RESULT_FILE.FILE_FORMAT">
      <eq>'HDF'</eq>
    </condition>
  </if>
  <location><URL> http://dns.ecs.soton.ac.uk/cgi-bin/SDB </URL></location>
  <description>
    NCSA Scientific Data Browser (SDB) obtained from
  </description>
</operation>
```

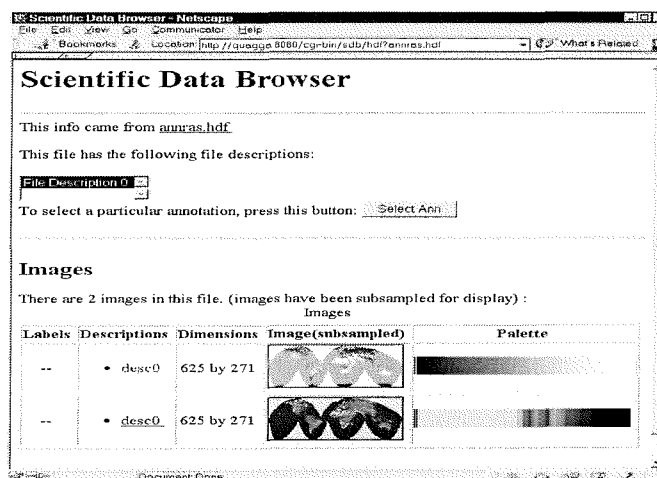


Figure 33: NCSA's SDB [243] has been specified as an 'operation' in the XUIS and invoked on a dataset managed within the EASIA architecture.

An example of the functionality that this can provide is illustrated in Figure 30 and Figure 33. Figure 30 shows a hyperlink labelled ‘SDB’ in an operation associated with the DATALINK file ‘annras.hdf’. This link initialises SDB with this file as shown in Figure 33. The user can now use SDB to post-process the dataset.

5.2.6 Code Upload for Server-side Execution

Moore et al. [152] discuss mechanisms for data-intensive computing applications (i.e. applications where execution time is dominated by movement of data) on emerging ‘GRID’ based computer systems. They believe that such applications will be difficult to support in GRID environments due to insufficient bandwidth for access to remote data repositories. They suggest, therefore, that one challenge is to support distributed processing, in which an application is moved to the site where the data resides.

As well as predefined operations, EASIA also provides a mechanism to allow users to upload Java code for secure server-side execution to post-process datasets stored on the file server hosts. In Figure 30 this facility is indicated by the arrow icon. The intention to allow authorised users to post-process datasets stored in EASIA with uploaded code is specified in the XUIS as shown in the following XML fragment:

```
<table name = "RESULT_FILE"
  primaryKey = "RESULT_FILE.FILE_NAME RESULT_FILE.SIMULATION_KEY">
  ...
  <column name="DOWNLOAD_RESULT" colid=
    "RESULT_FILE.DOWNLOAD_RESULT">
    <type><DATALINK/></type>
    <!-- Code upload is allowed against this DATALINK attribute, but not for
      guest users. A Java jar file can be run against the data- -->
    <upload type="JAVA" format="jar" guest.access="false" column="false">
      <!-- Only allow this operation on attributes in this column that meet
        the following conditions- -->
      <if>
        <condition colid="RESULT_FILE.SIMULATION_KEY">
```

```

        <eq>'S19990110150932'</eq>

        </condition>
        <condition colid="RESULT_FILE.MEASUREMENT">
            <eq>'u,v,w,p'</eq>
        </condition>
    </if>
</upload>
</column>

```

The user selects the arrow icon next to the dataset of interest and is then prompted for the name of a Java jar archive to upload, and the name of the initial Java class file to run. Processing then follows a similar pattern to predefined operations. The initial Java class file that is run must accept a filename as its first parameter since the path to the selected dataset will automatically be inserted here. It must also use relative path names for any file output. The uploaded archive is unpacked in a temporary directory and the code is run against the dataset on the file server host where the dataset is resident (and being controlled by the DATALINK mechanism). Any output files from running the code, including console output to standard output and standard error, are returned to the user.

A major difference between uploaded post-processing codes and predefined post-processing operations, is that uploaded codes are restricted to the Java programming language. This is because EASIA makes use of Java security features to restrict the capabilities of the uploaded code. These restrictions effectively mimic those applied to downloaded Java Applets on the Web. The only relaxation to this 'sandboxing' is that the code is allowed to read the input dataset file and is allowed to write output files to the temporary directory that has been specially created. Another implication of the use of Java for uploaded codes is that the data files to be processed must be compatible with the Java language. That is, the byte arrangements for the datatypes stored in the files must match those of the Java language or the uploaded code must be capable of making any necessary conversions.

The code upload feature can be of great benefit where datasets are too large to download in their raw format, and where the dataset is hosted on a high-performance platform.

5.2.7 Administration Features

Users of the EASIA system are managed on the Web via the screen shown in Figure 34. This allows users to be added to the underlying database, removed or modified. This screen is only available to users with administrative privileges.

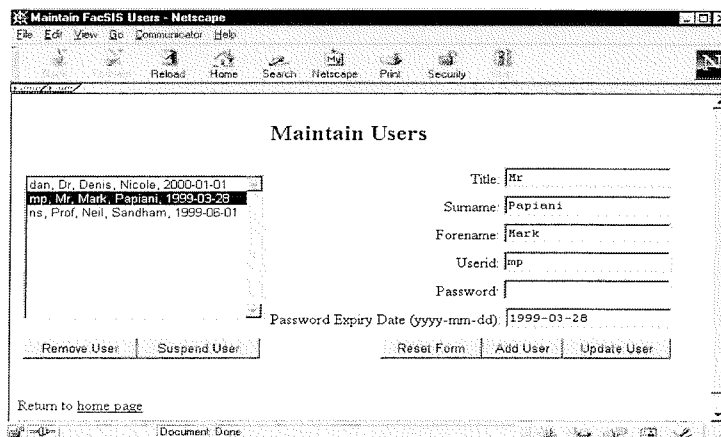


Figure 34: User administration screen.

5.3 Implementation and Design Decisions

This section discusses the implementation of the EASIA architecture. The results of initial experimental bandwidth measurements for electronic file transfers between UK universities are given. These results led to the rejection of the option of transferring large simulation files to a *centralised* archive. Next the DATALINK SQL-type is discussed as this presented an alternative *distributed* data management mechanism. Finally, the decision to use Java Servlets and JavaScript is discussed.

5.3.1 Experimental Bandwidth Measurements

From the outset limited bandwidth was likely to be the major factor influencing the design of the system. To assess what might be possible, some experimental results were obtained for the time required to transfer a file between two Universities using the UK's academic computing network. Before presenting the results, some background on the status of this network is provided.

Since the late 1970s the United Kingdom Universities and Research Councils have operated their own computer networks. In the early 1980s these networks were consolidated into a single system known as the Joint Academic Network (JANET). JANET is managed and developed by the United Kingdom Education and Research Networking Association (UKERNA) [221].

The latest phase of improvement to this network, SuperJANET III, recently replaced the backbone of the network. The new backbone connects a central ring of switching centres (at London, Bristol, Manchester and Leeds) via 155 Mbit/s ATM technology. From this core the network fans out at 155 or 34 Mbit/s to *backbone edge nodes*. These points connect to a number of regional networks called *Metropolitan Area Networks (MANs)* serving areas where several institutions are located closely. Some institutions are not connected to a MAN (for geographic or other considerations) and receive SuperJANET access via leased lines or British Telecoms's *Switched Multi-Megabit Data Service (SMDS)*. SuperJANET tries to ensure broad parity of provision for each institution. Nodes are not necessarily operated at full link capacity. Instead the actual bandwidth is related to the number of institutions connected to a node.

| Time | Direction of Transfer | Bandwidth (Mbit/s) | Estimated transfer time for small simulation data file | Estimated transfer time for large simulation data file |
|---------|-----------------------|--------------------|--|--|
| Day | To Southampton | 0.25 | 45m20s | 4h50m08s |
| Day | From Southampton | 0.37 | 30m38s | 3h16m02s |
| Evening | To Southampton | 0.58 | 19m32s | 2h05m03s |
| Evening | From Southampton | 1.94 | 5m51s | 37m23s |

Table 1: Experimental bandwidth measurements for file transfer between two UK universities.

The University of Southampton currently has a 10 Mbit/s SMDS connection to SuperJANET. Repeated measurements were made of the time required to transfer an

85 MByte file to/from Queen Mary & Westfield College, London (also connected to SuperJANET via a 10 Mbit/s link). These tests were performed using ftp, between two UNIX workstations, on different days and at different times to account for varying network loads. The average measured bandwidths from repeated transfers are shown in Table 1.

In the table 9am to 6pm is classified as daytime and 6.01pm to 8.59am as evening. Weekends (6.01pm Friday to 8.59am Monday) are included in the evening measurements. The table shows the measured bandwidths and also the estimated time required for transferring a file of a particular size, calculated from this bandwidth result. The two file sizes are 85 MByte for a small simulation and 544 MByte for a large simulation. These correspond to two current simulation resolutions being used by the UK Turbulence consortium. It should be noted that each timestep in a simulation generates four such files (three velocity components and a pressure component).

These results show, not surprisingly, that evening is the best time to transfer files. Less predictable is the fact that supplying result files *from* Southampton will achieve significantly better performance than trying to send results *to* Southampton. This suggests that currently most of the University's available bandwidth is being used to retrieve data from the Internet rather than supply it.

These results showed that using the Internet to transfer results from large simulations *to* Southampton for archiving was not a viable option (requiring around 8 hours for the four files per timestep in the evening) and hence that a centralised architecture, similar to that used for example in DBbrowse, was not viable. At this point a distributed architecture was considered and a mechanism was sought that could offer distributed storage of datasets in the EASIA architecture. The DATALINK feature of SQL:1999 provided a suitable mechanism and is described next.

5.3.2 SQL Management of External Data: The New DATALINK Type

Both ANSI and ISO have accepted the proposal for SQL Part 9: Management of External Data [60], which includes the specification of the DATALINK type. ISO

submitted SQL/MED for Final Committee Draft (FCD) ballot at the beginning of 2000 and it should be published as an International Standard in 2001 [81].

DATALINKs provide the following features for database management of external files: *Referential Integrity* (an external file referenced by the database cannot be renamed or deleted), *Transaction Consistency* (changes affecting both the database and external files are executed within a transaction to ensure consistency between a file and its metadata), *Security* (file access controls can be based on the database privileges) and *Coordinated Backup and Recovery* (the database management system can take responsibility for backup and recovery of external files in synchronisation with the internal data).

The EASIA architecture uses IBM's implementation of the DATALINK type that is available for DB2 [64]. This uses a DB2 database that stores the data associated with standard types internally, plus DATALINK specific software running on the remote file servers to manage external data. This software processes SQL *INSERT*, *UPDATE* and *DELETE* statements that affect DATALINK columns, to link and unlink files for the database. It manages information about linked files, and previous versions of linked files for recovery purposes. It also intercepts file system commands to ensure that registered files are not renamed, deleted and optionally, check the user's access authority. An example of the SQL syntax for creating a table with a column containing a DATALINK SQL-type is as follows (refer to the RESULT_FILE table in the schema of Figure 26):

```
CREATE TABLE RESULT_FILE (
    download_result DATALINK
    LINKTYPE URL
    FILE LINK CONTROL
    READ PERMISSION DB
    ...
```

The *FILE LINK CONTROL* parameter specifies that a check should be made to ensure the existence of the file during a database insert or update. *Read permission* can be set to *DB* (database) or *FS* (filesystem). If the database manages read permission then files can only be accessed using an encrypted file access token,

obtained from the database by users with the correct database privileges. Without a valid token, the DATALINK software denies the read request. Further details of these parameters, and additional parameters that can be specified in the DATALINK column definition, are available in [60].

A DATALINK value can be entered via a standard SQL *INSERT* or *UPDATE* statement. The value takes the form:

http://host/filesystem/directory/filename

The *filesystem* part of this URL is a specially mounted DATALINK File System (DLFS) in the AIX version of the software. The *directory* and *filename* are standard UNIX file systems. If read permission is managed by the database, an SQL *SELECT* statement retrieves the value in the form:

http://host/filesystem/directory/access_token;filename

The file can then be accessed from the filesystem in the normal way using the name:

access_token;filename

or, by using the full URL if the file is placed on a Web server (as in EASIA). The access tokens have a finite life determined by a database configuration parameter. This can be set to expire after an interval.

5.3.3 Java Servlets and JavaScript

The EASIA architecture was implemented using Java Servlets on the server side and HTML forms/JavaScript on the client side. Whilst Java Applets can produce sophisticated user interfaces the technology was rejected for the client side of EASIA for the reasons given in Chapter 2. Namely, the difficulty in building robust Applets, the restrictions posed on client's browsers, the requirement for the client to have hardware that could offer reasonable performance, and the security restrictions that apply to Applets. HTML forms were used instead, augmented with JavaScript to give the interface a more dynamic feel than can be achieved with HTML form alone. A subset of JavaScript is used which works with both the Netscape and Microsoft Web browser.

Servlets were chosen for the server side development for the reasons outlined in Chapter 2. Namely, improved performance over CGI, the ability to maintain state

and track users' sessions through the Servlet API, and the availability of many other useful Java APIs for example, security and database access APIs.

One of the most difficult problems encountered during the implementation of EASIA concerned the processing mechanism for predefined post-processing operations and code upload. Sun's Java Web Server (JWS) was used as the Servlet engine on the distributed file server hosts. The initial idea for predefined Java post-processing codes and uploaded codes was that a specially written operation *startup* Servlet running within the JWS would dynamically load the required Java class (using *Java Reflection*). Any output would be written to a temporary directory that had a unique name based on the user's Servlet session identifier (and time/date information). However, it proved extremely difficult to redirect any file output to the temporary directory using this mechanism. Although it is straightforward in Java to redirect any output directed to standard output or standard error to files in the temporary directory, it was not possible to get the *startup* Servlet to redirect any other file output from the user's code (which used relative path names as mentioned in Sections 5.2.5 and 5.2.6). Instead, it was found that all output was written to the directory that the *startup* Servlet was run from. According to *Bug Id 4307856* at Sun's Java Developer Connection⁶, the Java API does not allow the working directory to be changed as this complicates multithreaded code.

⁶ The following bug report is taken from the Sun's Java Developer Connection Bugs Database (<http://developer.javasoft.com/developer/jdchome.html>)

Bug Id 4307856 **Submit Date** Jan 27, 2000 **Workaround** None.

Description There should be a way for a Java application (not an Applet) to set the current directory. This feature has a number of uses but it's mainly necessary to run certain applications launched from the Java application. This has been sorely missing for years and it's causing a lot of problems for developer that are then forced to use ugly hacks to work around the problem.

Evaluation The Java platform API specifically does not allow the working directory to be changed, since having such mutable global state would vastly complicate writing multithreaded programs (4307856). For the purpose of creating a subprocess that runs in a different directory, however, a new variant of the `Runtime.exec` method was introduced in J2SDK 1.3 (Kestrel, see RFE 4156278).

The implemented solution uses the *startup* Servlet to invoke the operation class file via a system call to a dynamically created batch file. This batch file changes the directory to the appropriate temporary directory, unpacks the code if it is stored in an archive format (such as jar or zip) and then invokes another Java interpreter to run the operation class file for the user's requested post-processing code. Another advantage to this batch file approach is that it also supports post-processing codes written in other languages. The batch file is still dynamically created by the *startup* Servlet but in this case it contains appropriate commands to invoke the non-Java post-processing code.

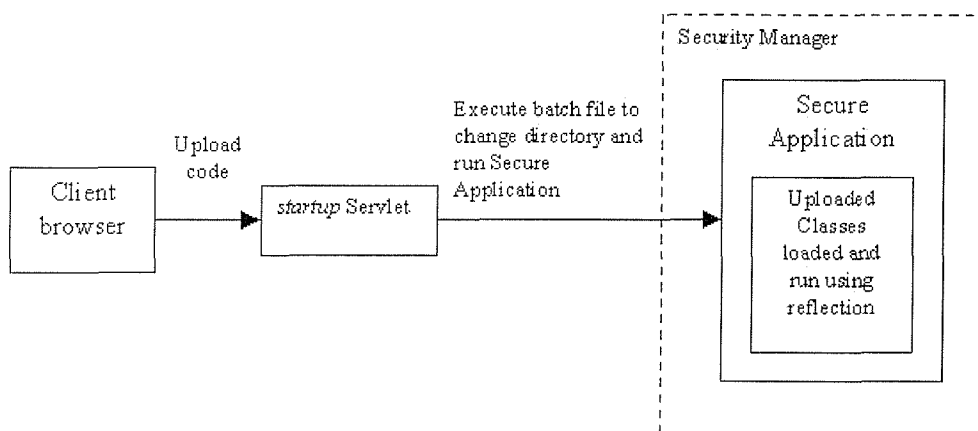


Figure 35: Security mechanism employed for uploaded post-processing codes.

The batch file mechanism is also used to run uploaded post-processing codes. This is required to not only fix the file output problem, but also to implement the 'sandboxing' restrictions required in this case. Firstly, a multipart HTTP request is used to send the file and name of the initial class to run, as input at the user's Web browser, to the file server host where the chosen data resides. The *startup* Servlet running at the file server host receives the request using its input stream. As illustrated in Figure 35 the *startup* Servlet then creates a script that changes to the required temporary directory, unpacks the uploaded jar archive, and then invokes another Java interpreter with another special *secure application* class that is used to implement the 'sandboxing' for uploaded code. This special class declares appropriate security restrictions and then dynamically loads and runs the user's

uploaded code using Java reflection (applied to the user's input corresponding to the name of the class to run).

To end this section on implementation of the Java Servlet code that comprises the majority of the EASIA system a few comments on the portability and performance of the system are warranted. To demonstrate the portability of the Servlet code used for post-processing on remote file servers, the code has been ported to both AIX and Windows NT. This was a simple task as the code uses a configuration class file to retrieve any system specific information. The only change to the configuration file was the location of the JDK. The configuration class can automatically derive all other system specific information such as file separators and the appropriate format for batch files. The Servlet code that runs on the database server host at Southampton (Figure 21) and which implements the main EASIA backend processing, is also portable to different platforms. Database independence depends on support for LOB datatypes and the new SQL:1999 DATALINK type.

Query and post-processing performance of the EASIA system has not been studied in detail, as it is reasonably independent of the architecture. As has already been described, EASIA uses multithreaded Servlet code to improve performance over the CGI mechanism used in the earlier GBIS and DBbrowse prototypes. Furthermore, the response times for database queries will depend on the number of concurrent users, but as explained in Chapter 2, most object-relational databases have parallel versions available if performance is insufficient on sequential platforms.

One area that should be re-implemented to improve performance is the batch file mechanism for executing post-processing operations. As stated above this was used primarily to change the working directory in Java and to sandbox uploaded post-processing codes. However, each concurrent post-processing operation running on a machine will result in a new Java interpreter being started and running for the duration of the operation. A more efficient mechanism would be to create a permanent server process in Java that simply received parameters from the batch files for particular operation requests and spawned multiple threads to service the operations.

It is not possible to say much more about the performance of post-processing codes in EASIA as the system can incorporate diverse code types through the XUIS specification. Additionally, the performance of post-processing codes will depend upon the file server platform that they run on. In general, the overhead created by EASIA's Java code is likely to be negligible compared to Web response times and the times taken for scientific codes to post-process large datasets.

5.4 Conclusions

A prototype system, EASIA, has been constructed to meet a requirement of the UK Turbulence Consortium to make available to authorised users, large result files from numerical simulations, with a total storage requirement in the hundreds of gigabytes range. The major limiting factor in trying to meet this requirement is the available bandwidth in the Web environment. EASIA greatly reduces bandwidth requirement by allowing datasets to be managed in a *distributed* fashion (with centralised metadata) thereby avoiding costly network transfers associated with uploading data files to a centralised site. Data distribution also reduces retrieval bottlenecks at individual sites. Bandwidth requirement is further reduced by the *active* nature of the EASIA architecture, which allows data reduction through post-processing. Post-processing can be achieved via archived applications or by allowing users to upload code to be run securely on the file servers hosting the datasets. Although EASIA is distributed in nature, it maintains integrity between metadata and the actual data files.

The interface presented by EASIA is specifically aimed at users with a scientific background who are not familiar with SQL. As such, EASIA aims to help users locate scientific data files of interest, using an intuitive searching and browsing mechanism that has been inherited from DBbrowse. Unlike DBbrowse, however, the user interface in EASIA is not only automated, but also customisable and extensible. Automated construction is achieved by allowing the user interface specification to be defined in an XML file used to initialise the system and by providing a tool that can generate a default XML specification. Separating the user interface specification from the user interface processing can provide a number of further advantages:

- The user interface, although schema driven can be customised to use aliases for table and column names and to present different sample values. Tables and attributes can also be hidden from view.
- Hypertext links to related data can be specified in the XML even if there are no referential integrity constraints defined for the database.
- Different Users (or classes of user) can have different XML files thereby providing them with different user interfaces to the same data.
- For scientific data archiving a major benefit, facilitated by the XML user interface specification, is the capability to associate ‘operations’ with database columns, so that a user can extend the interface by including standard post-processing codes. EASIA can archive these codes along with data. They can then be applied dynamically to the data. These applications are loosely coupled to the datasets via XML defined interfaces. They provide reusable server-side post-processing operations such as data reduction and visualisation.

The EASIA architecture uses distributed commodity computing and non-proprietary technologies such as the new SQL DATALINK type, defined in SQL/MED. EASIA combines leading edge Web-based technologies and techniques to provide a complete working architecture, including Web-based user interface for archiving large, distributed files, whilst maintaining database security, integrity and recovery.

6 Related Work

This chapter describes related work. It is split into two main sections. The first section discusses related work on user interfaces to databases and provides comparisons with DBbrowse. In particular, this section concentrates on user interfaces to databases that include browsing mechanisms, and on Web-based user interfaces to databases. The majority of this section focuses on the state-of-the-art at the time DBbrowse was published in 1997. The second section discusses related work on Web-based management of scientific data and provides comparisons with the current EASIA architecture.

6.1 Related Work on User Interfaces to Databases

6.1.1 Introduction

Graphical interfaces to databases are an important tool in the database world to provide easy access to data without the need for technical knowledge on database design or the need to learn declarative languages such as SQL. Accordingly, there has been a substantial body of research in this area. PC products such as Access and Paradox still show the influences of early work such as TIMBER [212] and FADS [197] and owe much to the seminal work on Query By Example (QBE) [244]. However, database interfaces have long since surpassed this early work due to advances in technology. This section describes some of the research that has led to the current generation of database interfaces. Firstly, stand-alone interfaces to databases are discussed as these have provided many of the techniques later adopted for Web-based interfaces to databases. Subsequently new Web interfaces to databases are discussed. Contrasts will be drawn between these systems and DBbrowse, which provides a Web interface to relational and object-relational databases that is generated completely automatically. Interfaces that provide database browsing will be highlighted so that the method of browsing can be compared to that in DBbrowse, which is based on generating automatically links to browse for related data, based on inferred relationships derived from referential integrity constraints.

6.1.2 Stand-alone Graphical Query Interfaces to Databases

6.1.2.1 Early Graphical Interfaces Incorporating Database Browsing

Early examples of database browsing consisted of low-level commands rather than visual techniques and were usually applied to extended relational database models. These systems incur significant cost in terms of defining the database schema and loading the database due to the extended data model. Motro's work on BAROQUE [153] describes natural-language-like browsing commands for exploring a view of a relational database that resembles a semantic network. Each real world entity is modelled with one database item. This data item could appear in different attributes e.g. the data item "Los Angeles" could appear in both the location attribute of a University table and the name attribute of a city table. An additional item directory table is constructed, with a row for every distinct data item, that contains attribute and data item pairs. BAROQUE is therefore amenable to access by value without knowledge of the schema, i.e. the user can use "Los Angeles" as a search value without knowing the name of the attribute in which this value occurs. The system uses commands such as 'what is' (to find out the details of the items position in the network hierarchy i.e. schema information) and 'what is known about' to extract all the relationships in which the item appears. Relationships are obtained between a data item and all other attributes in all the tuples that the value is found. One problem with browsing related items based on data item values alone, is that spurious links can be generated which have no real relationship. Browsing in BAROQUE occurs additional cost in the space required to store the item directory table and the computation required for its initialisation and update.

Stonebraker's database browser, TIMBER [212], was a specification and partial implementation of a graphics based browser for an extended relational database. The browser included a relation browser for fixed format relations, an editor for text data stored in relations and a map browser for geographic image data. Icon fields were also available to provide visual representation of database rows. Commands were specified for selection and projection of data and for browsing which consisted of finding the next icon or row that satisfied an expression.

Browsing in VAGUE [155] involved language based queries that had a ‘fuzzy’ specification of the data. The relational data model had to be extended with the concept of *data metrics* and the query language with a *similar-to* comparator. A special metric table was created to define the type of metric for each domain used in the database. A second additional table showed which domain corresponded to each database attribute. Data metrics could be computational. For example, for the domain money, the metric could calculate the absolute difference between two data items. Data metrics can also be tabular, whereby a new table was created to provide a value of the data metric for every possible pair of data item values. Once all these database extensions were in place (no small task presumably!) vague queries could incorporate the *similar-to* comparator in order to browse the database based on the distance between data items defined by the metrics.

Browsing is the principal method of navigation in Motro’s work on loosely structured databases [154]. Such databases are collections of facts that are defined as named binary relationships between data values, for example, (employee, earns, salary) and (employee, works-for, department). The data can be regarded as a network of values that can be browsed using a dedicated query language.

6.1.2.2 Data Browsing within Data Models Containing Explicit Relationships

Systems have been designed to provide *data* browsing for database models that have *explicit* relationships between objects in the database. Such database models include *entity-relationship* (E-R) [44] and *object-oriented* data models. Most of these systems are closely bound to the underlying database system and do not provide a generic interface to different DBMSs. Examples of these interfaces are Databrowse [196], LID [96], KIVIEW [156], OdeView [8], and PESTO [32].

Databrowse provided a user interface to an E-R database system. The user produced an E-R diagram with a tool called *Schemadesigner* and then expanded the entities into entity tables through a property sheet that defines fields, data types and keys. No direct query facility was provided. However, the user could begin by selecting a table to display a scrollable list of all the tuples. It was possible to expand any single tuple on display to show all related tuples in referencing tables that had been linked by arrows in *Schemadesigner*. Binary and text data types were

supported in this system so that links could be made to associated images and text files.

Fogg's Living in a Database (LID) graphical query interface uses similar ideas to Databrowse to provide browsing for an E-R database, but with improved presentation graphics and also the ability to enter queries by entering search conditions for the attributes of the currently selected entity. Both LID and Databrowse suffer from the weakness that sets of tuples cannot be operated on at the same time. For example, in a personnel database, the user may want to display the department details for all employees simultaneously, without having to browse the details for each employee one at a time, moving backwards and forwards between the employee and department entities.

KIVIEW provided an interface to a database characterised by a semantic network consisting of objects connected by binary relationships. The user sets up several views, linked in a tree-like structure. A view of an object consists of all the facts in which the object participates. A fact is a triplet containing two objects and the name of the relationship between them. KIVIEW introduced 'synchronous browsing' of related objects, a technique that is also used in OdeView and PESTO. When the information in the root view is modified, the content of the other views changes automatically. In KIVIEW commands were used to explicitly indicate the synchronising links.

OdeView provides a GUI for the Ode OODB. *OdeView* allows browsing of objects, following chains of references and displaying selected portions of objects. The display features of *OdeView* were implemented by requiring object class definers to provide certain display-oriented and query-oriented functions for their classes. Objects can be displayed in one or more formats depending on the semantics of the display function. *OdeView* also provides schema browsing where users can display object classes, select an object class, display subclasses, display superclasses and display metadata, e.g. the number of objects in the class. The system as implemented did not include selection and projection although these were discussed as extensions.

PESTO (Portable Explorer of STructured Objects) [32] is an on-going joint project between IBM Almaden and the University of Wisconsin to develop a user interface that supports browsing and querying of OODBs. The interface allows users to navigate the relationships that exist among objects. In addition, users can formulate complex object queries through an integrated query paradigm that presents querying as an extension of browsing. PESTO is schema-driven, and allows the display of multiple objects of a single collection at the same time and references or collections of references to be displayed in separate windows, which can be synchronised to change when the referencing object changes value. The user can browse the same collection independently in several windows. PESTO has two major differences to DBbrowse. Firstly, PESTO is a stand-alone application that is not integrated with the Web. Second, PESTO is designed for object databases with explicit relationships defined between objects. However, PESTO has been used to interface to DB2/6000, an ORDB, by using an object to relational mapping tool. Relational joins, which create new objects out of attribute data copied from pairs of other objects, are not supported and due to PESTO's OO interaction model, relational projection is not supported, though it can be simulated through attribute hiding.

6.1.2.3 Database Schema Browsing

Other systems have concentrated on browsing databases at the *schema* level rather than the data level. Schema browsing provides a useful tool for understanding the structure and relationships between database entities, particularly for large databases. These include, GUIDE [237], ISIS [104], SKI [140], SNAP [28] and OPOSSUM [108].

GUIDE used a graph-based view of the E-R schema with queries expressed as traversal paths on this network. ISIS and SKI provided design, browsing and querying at the schema level for semantic database models that are considerably richer than the E-R model. *SNAP* was a system for schema design, schema browsing and specification of graphical queries. Schema browsing in SNAP consisted of navigating the schema diagram using pan and zoom, by repositioning objects, and by reformatting hierarchies. SNAP allowed queries to be posed against the *schema* of a specific object-based semantic database model to return browsable data. Queries

were expressed using query graphs of entity sets and its attributes with results returned as tabular text.

OPOSSUM is a recent project that provides a direct schema editor for schemas from virtually any data model and allows schema exploration through a choice of visual representations. *OPOSSUM* does not include any query capabilities. Haber [107] provides a framework for developing graphical schema browsers. The framework is based on formal definitions of a *data model* for capturing schemas, a *visual model* for capturing visualisations, and a *visual metaphor* that defines a mapping between the two models. Visual metaphors for database schemas include directed-graphs, E-R diagrams and textual tables. Mixed metaphors allow different visual metaphors for different parts of the same schema.

6.1.2.4 Graphical Query Interfaces to Entity-Relationship and Object-Oriented Database Models

Campbell [29] defined an E-R algebra over a set of entity-set and relationship-set descriptors with an established set of operators. Graphical manipulation of E-R diagrams was used to formulate queries using this underlying E-R algebra. Operations such as delete, restrict, project, union, intersection and difference were available to manipulate the E-R diagram into the required query format. In order that the language was relationally complete (i.e. the language could answer any query that relational algebra can answer) additional operators such as duplicate (part of E-R diagram) and create-relationship were added.

Elmarsy and Larson [82] also provided query formulation through E-R diagram manipulation. The system involved several steps for query formulation. Firstly, the user chooses the entities and relationships of interest. Second, the user chooses one of the entities to be the root entity. The diagram is redrawn as a hierarchy with the root entity at the top. If the graph has cycles, these are removed by duplicating leaf nodes. This is done so that when the user selects attributes from different entities, only one connecting path is possible. For example, consider three entities department, student, and course. With department as the root entity, the student and course entities could appear directly below department. Additionally the student entity could appear again below the course entity. During the third step the user next selects attributes to enter restriction conditions. These are numbered so

that after simple conditions are completed, complex conditions can be specified using, AND, OR and NOT, on the numbered conditions. Finally, attributes are selected for projection. The query is then submitted.

Query by Diagram (QBD) [36] makes use of an E-R data model and a query language that uses diagrams. The user interacts with the conceptual schema its information and content, and with successive approximations, extracts the subschema of interest. A number of strategies are available for this. The user can simply delete unwanted entities and attributes. The user can select two objects and retain only objects associated with all the paths that connect them. The user can type in attribute names in Boolean expressions and retain only those entities that have attributes that satisfy the expression. It is also possible to expand the subschema by including the next level of connected objects. The user next transforms the subschema building a temporary view. This involves operations such as replacing an entity-relationship-entity path with a single entity. Finally the subschema is used to pose the query. The user selects a starting entity. The query path is built by selecting further symbols. Selection conditions can be placed on attributes. If two disconnected entities are chosen the user must apply a comparison condition between them. Once a path has been built for an elementary query, the user can build a second path and link it to the first via union, intersection or difference.

The Pasta-3 interface [143] uses a drag and drop method for forming and refining queries. Some of the features include query editing through handy values and automatic path completion. Pasta-3 is an interface to KB2; a knowledge base system embedded in Prolog. KB2 uses an E-R data model, extended with inheritance and deductive rules. Schema browsing is provided through an E-R diagram or an entity inheritance lattice. A tabular display window shows the result of querying or browsing. E-R items can be copied into a query workspace window or selected from menus. A popup menu can be used to choose properties associated with the entity for further work. The user can mark properties for projection and/or enter conditions. Operators and functions can be selected from menus. Handy values are available through menu selection. These are the values of attributes that have a domain of no more than 15 different possible values. Automatic path completion is available for E-R items in the query workspace that are connectable but not yet

connected. If evaluate is selected, the system will find missing items and offer to add them in, if exactly one path is possible, or propose a choice if several paths exist. Advanced features include recursive queries using duplicated entity icons, and subqueries, formulated by shrinking a query to an icon and using it as an operand in a new query condition.

Doan *et al.* [69] describe the concept of a *multi-paradigm* query interface to OODBs that includes textual, form-based and graph-based queries with automatic translation between these paradigms. The interface described in the experiments supports only two paradigms, textual and graph-based. Graph queries are formulated by direct manipulation using popup menus attached to boxes in a query graph window. The basic structure of the textual query involves iteration over classes of objects, and the application of functions to retrieve attribute values. Clicking on a translation button makes automatic translation between graphical and textual queries.

6.1.2.5 Systems for Database Graphical User Interface Development

There are a number of research projects building tools to help construct graphical interfaces to OO databases. *FaceKit* [142] [141] allows the user to define a user interface and an application simultaneously, rather than building a distinct user interface on top of an application. The system treats the interface design as an integrated unit with the database. The interface is built using *representational components* and *operational components*, both of which are stored in the database itself as methods. The representational component builds, maintains and invokes the methods used to produce the visual representation of objects and is also responsible for input and output associated with the interface. The operational component is responsible for processing user queries and sending the results to the representational component, it has access to the database management tools. The system is tied to a specific OODB ('Cactis') and is useful for building domain specific application interfaces rather than generic database exploration interfaces.

Flynn and Maier [95] take a similar approach in the Object Display Definition System, where display information for complex objects is a complex object itself that is stored in the database and managed by that system instead of the

individual applications. Advantages include the reuse of displays for different applications built on the same database objects and modularity since display definitions can be developed independently of applications.

O2 provided *O2Look* and *ToonMaker* [21]. *O2Look* is a graphical toolkit, built on top of Motif, to provide functions to manipulate database objects rather than widgets usually associated with such toolkits. *ToonMaker* is a user interface editor that allows the user to graphically specify how objects should be displayed and which then generates *O2Look* code.

6.1.3 Web-based User Interfaces to Databases

This section looks at different approaches to providing Web access to databases. One approach to providing Web access to database information is to maintain the original data in the database and additionally construct static HTML pages using data exported from the database. *BestWeb* [18] is an example of a product that helps users to construct HTML pages using data extracted from databases and provides basic query tools on these pages. Because database information on the Web is not linked to source data in real-time, this approach is only appropriate if the amount of data to be viewed on the Web is small and changes infrequently. In addition the facilities of the underlying database such as the search engine and access controls are lost.

IBM's *Net.Data* [162] (formerly, *DB2 Web Connection* [164]) uses the power of the underlying database by providing access to source data through a macro language. This includes SQL and HTML sections linked together via cross-language variable substitution. The HTML sections are used to express the format of the input and output reports, and the SQL sections to express the database commands. Because the forms and reports are built in advance, users are constrained with respect to the information they receive and the queries they are able to make. Similar projects included amongst others, the *GSQL Database Gateway* [84], *Sybase's web.sql* [230], *Illustra's Web DataBlade Module* [99] and *Allaire's Cold Fusion* (using *Cold Fusion Mark-up Language* embedded in HTML) [49].

Hadjiefthymiades et al. [109] describe a 'generic' framework for databases on the Web. Really this approach is similar to those above. The user provide a

hypertext link to a ‘query specification file’ (QSF) which is effectively a CGI script which has database and presentation components. The QSF sets up a QBE-like form. However, all field names must be defined in the QSF, and there is no automation. In fact, the only section of a select statement that users can effect is the *where* clause through restrictions placed on fields in the QBE. The fields for projection are fixed in the QSF, and fields that might be joined are explicitly stated in the QSF. The argument that their approach is generic because it uses dynamic SQL as defined by the X/Open standard does not really hold. A QSF written for one DBMS is unlikely to run on another system, since the parameters extracted from the QSF are passed to a compiled C program containing Dynamic SQL. This would have been pre-processed using a DBMS specific precompiler.

HyperSQL [163] allows users to select queries from menus and compose queries by filling out forms. The results of a query contain hypertext links, ‘querylinks’, to browse the database for related information. Although the interface is designed for users who are not computer experts, the HyperSQL language itself is designed for database administrators. A HyperSQL query interface requires a set of pre-defined text-files. These files contain HTML and additional HyperSQL descriptors. The HyperSQL descriptors provide the syntax for connecting to the target database, describing the layout of a query form, generating SQL from the completed form and describing the layout of query output. Querylinks are statically associated with pre-defined queries, which can however receive data fields as parameters. HyperSQL differs from the systems described above through the availability of Query Designer [190]. This tool allows users to create or customise query forms without having to use any of the scripting or query languages. Users can develop query forms and output screens graphically using a Web browser. This involves selecting a database then selecting input and output fields from the database. Two input fields will be joined automatically if they are selected from two different tables for which there is only one possible relationship. More complex joins are specified manually by selecting fields and keys that join them. The layout of the input form and result screen can be defined using relative coordinates, and different font types. A graphical schema display is available to show the database layout. This shows table names, and the keys that join them.

The UMass Information Navigator [123] automates generation of the initial Web interface to the database and subsequent links. A forms-based input screen is created from database metadata. This allows the user to graphically formulate a query. The system then supports browsing using hypertext links that perform an SQL select on another table. However, the system relies upon matching column names to find additional tables to link to. There is no guarantee that matching column names contain related information, consequently the resulting select statement may be completely unrelated to the original data.

Much of the related work described in this section has reflected the status of research related to DBbrowse up until 1997. Subsequently, commercial products have appeared offering similar functionality to DBbrowse. Impromptu Web Query from COGNOS [126] is a good example of such a product. It supports automated generation of QBE-like interfaces to relational databases. A user can start with an ad-hoc query and then navigate to related information using hypertext links that are embedded in the query results. Unlike DBbrowse, Impromptu Web Query also supports pre-defined queries. In addition, when performing predefined or ad hoc queries, users can; add, remove, and resequence columns; modify column fonts, colors, names, descriptions, and sort orders; incorporate totals and subtotals; specify filters using fixed values or prompts with picklists, and apply predefined tabular and chart templates. However, Impromptu Web Query does not support all the features that were present in DBbrowse in 1997. It does not support LOB columns and does not inline data from joined tables to existing results. Also it does not support query refinement during browsing.

Further information on user interfaces to databases is available in the survey by Catarci *et al.* [35]. A number of books are available with further information specific to Web/database integration techniques, for example [138]. Also, the Web site at [1] still contains some useful links in this area despite being out of date. DBbrowse has moved forward since 1997, having been used as the basis for the EASIA architecture. Research related to the EASIA architecture is discussed in the next section.

6.2 Related Work on Web-based Management of Scientific Data

EASIA used simulation data from the UK Turbulence Consortium for proof of concept. There are a number of existing Web-based interfaces to turbulence simulation databases such as the ‘European Research Community on Flow Turbulence and Combustion (ERCOFTAC) fluid dynamics database’ in the UK [87]. The system is implemented using static HTML pages with links to the data files. Secure access to the data files is provided by Access Control Lists (ACLs), which are a standard feature of most Web servers. The ‘DNS Database of Turbulence and Heat Transfer’ in Japan provides Web pages with links to data on an ftp server [68]. There is no search facility but each directory contains an index file containing brief details of all the files in that directory.

Moving outside of the turbulence domain there are a number of *active* scientific data archives. The Ocean Circulation and Climate Advanced Modelling Project (OCCAM) [171] provides a Web-based mechanism for obtaining slices of large scientific datasets from high-resolution models of the world ocean. The *OCCAM data selector* is a Java Applet that allows the user to request data extracts by clicking and dragging the mouse pointer on the area of interest on a map of the world. The user then enters an email address and the required format for the data that is prepared off-line and placed on an ftp server for the user to download. This system uses the notion of post-processing large datasets to reduce the volume of data that has to be returned to the client. The system is however a customised application for one specific application and is only partially automated.

NCSA’s *Scientific Data Browser* (SDB) [243] [149] consists of a CGI program for browsing data in a number of scientific formats such as HDF [115]. This system is not tied to a particular application, instead it is tied to the specified scientific data formats. The CGI program is written in C and provides functionality such as visualisation and extraction of subsets of data. However SDS does not provide sophisticated storage and search capabilities for archiving and selecting the data files that it can process. It assumes that these files are simply placed in a directory structure belonging to the Web server.

Ferreira *et al.* describe how scientific datasets usually consist of multi-dimensional data representing, for example, spatial coordinates, time, temperature or velocity. Several research and commercial systems have been developed for managing/visualising multi-dimensional data, however, they provide little support for analysing or processing the datasets in other ways, focusing instead on management and retrieval aspects. They state that an important subset of scientific applications fall into the complex data with queries category (as defined in Stonebraker's classification matrix for DBMS applications [211]) and can therefore be supported by *object-relational* database management systems. To this end they describe an infrastructure known as the *Active Data Repository* (ADR) that provides a framework, based on an ORDB, for building databases that support *storage*, *retrieval* and *processing* of multi-dimensional scientific datasets. These three features coupled with the use of an ORDB, match the main features of the EASIA architecture. However, ADR is not a Web-based architecture and the implementation and ease of use is very different to EASIA. ADR is not automated, but is a complex system requiring expert domain knowledge and extensive C++ programming for each customised application.

In addition to the main features mentioned above, ADR supports index generation, memory management and scheduling of processing across a parallel machine. ADR allows customised processing of datasets by applications that must follow a defined style. This is known as an ADR query and consists of:

- Retrieving input data items from selected by a range query (which probably uses a complex user-defined index and lookup method).
- Mapping the retrieved data items to output items (by projecting the input data points to points in the output attribute space and finding the output data items that correspond to the projected points).
- Computing the output items by some aggregation of all the retrieved input items mapped to the same output item.

Unlike EASIA which stores datasets and post-processing operations in external binary files tightly coupled to the database via DATALINKs, ADR stores datasets internally in an ORDB, and makes use of internally stored user-defined functions to process the data. The authors state that this usually requires data structures that are more complex than simple tables. In many cases, complex user-defined types consisting of nested relations are used. The complex types may also support attributes that are sets or references to other objects. Regardless of the data types that are defined to store datasets in ADR, datasets must be partitioned into *chunks*, a job that must be done by 'the domain engineer prior to loading the database'. Data chunks are required to support parallel processing in ADR by distributing them across a disk farm in a shared nothing architecture. The expertise required to load data into ADR is in contrast with the EASIA architecture. Also it does not look at the problems associated with initially shipping the data to an archive because ADR does not assume a Web-based environment.

To build a version of ADR for a particular application, *a domain engineer* with the necessary authorisation to customise the system, must provide functions to pre-process input datasets, access input data items, and provide the necessary processing to map input data items to output data items. These functions must be written in C++ and inherit from a set of base classes that ADR provides. A client program must also be implemented for each domain, for example, a GUI to allow end users to generate requests and display output. A number of successful customised ADR databases are described in the paper, running on an IBM SP2 supercomputer. Overall, ADR appears to be geared to expert users and administrators who augment the system with customised code for each application. It is also geared towards customised parallel processing algorithms on the back end. ADR is not easily extensible and does not allow code to be uploaded for server-side execution.

The University of Adelaide is involved in a number of projects associated with scientific data archives [46] [47] [112] [113] [114] [131]. Active digital archives for the satellite/geospatial domain are a common theme in all these papers. In particular, Web-based data archives have been built to access two dimensional image data associated with the Japanese GMS-5 geostationary meteorological

satellite. Images from this satellite amount to around 75 gigabytes a year, and are stored on a RAID disk array and tape silo, accessible from a high-speed (155 Mbit/s) network. The initial Web interface to the data, known as ERIC, constituted an active scientific archive, since it provided services for not only searching and downloading data of interest, but also processing of the data prior to download. ERIC allowed cropping of a selected region of an image, and the creation of downloadable MPEG animations of specified time slices for a given region. ERIC was initially implemented using HTML and CGI scripts in PERL and was functional, but restrictive and not easily extended. The ERIC architecture is similar in nature to GBIS. Both offered bespoke CGI-based Web interfaces to scientific data, which produced visual output. Both systems also used underlying data stored as files in a UNIX filesystem. James and Hawick [131] discussed the fact that the types of requests that ERIC could service were all hard coded and that a new idea would be to define a generalised service specification language. This could be used to generate automatically the sequence of interface forms, which would allow the possibility of dynamic addition of new services to a running system. These ideas are close to the ideas that have been implemented by the XUIS in EASIA.

The initial ERIC interface has now been superseded by a more advanced interface that uses Java and CORBA, which the authors suggest, can more easily be customised for different users and applications. However, this interface does not yet appear to support the generalised service specification language and the automated generation of interface forms described above. The new architecture uses the Java/CORBA infrastructure to integrate Web-based clients with broadband networks, mass data stores, and high performance computers. The system conforms to a subset of the Geospatial Imagery Access Services (GIAS) specification from the US National Imagery and mapping Agency (NIMA) [102], which uses CORBA IDL to define an object-oriented API for remote access to an image server. The University of Adelaide GIAS prototype is implemented using a commercial product for management of multimedia objects. StudioCentral from Silicon Graphics is used at the heart of the system. This provides a C++ based API for user management, repository access, querying, hierarchical data modelling and asset versioning. StudioCentral uses a persistent data store (usually an Oracle or Informix database) to

store asset metadata and a file system for storing multimedia files (since this is usually faster and more scalable than using the database). The Adelaide GIAS system uses Java Native Methods (JNI) [132] to wrap the StudioCentral C++ API and match it to the interface required for GIAS. A test client has been implemented as a Java Applet, which uses CORBA to interact with the server side of the repository. Overall, the system supports remote invocation of methods on the server, and handling the transfer of data between the client and server. It also supports loading, accessing and querying of image data via the database, and routines for processing the data and metadata. In terms of customisation, the authors state [46] that a particular application would have its own specialised user interface, which might consist of a Java Applet that could use additional application specific classes along with the basic Applet. The system differs from the EASIA architecture in many ways. EASIA rejected the use of Java Applets/CORBA due to client side performance issues, firewall restrictions and general stability of Applet-based Java/CORBA systems. The Adelaide system also requires code to be written to customise the system for different applications. Also, it does not support allowing arbitrary post-processing code to be uploaded by the users for secure execution against the data.

Caltech are also carrying out research into active scientific data archives. A number of papers published by Caltech researchers discuss their *Synthetic Aperture Radar Atlas* (SARA) system [9] [10] [11] [234]. The SARA system provides a Web-based mechanism for obtaining synthetic aperture radar (SAR) data covering the Puglia region of Italy. The system first presents a user with a map of the earth. The user can zoom to a particular area by clicking on the map. Rectangular *tracks* are indicated on the map for regions that have an associated SAR image. The user can select a track to invoke a Java Applet that displays a thumbnail image of the area. The Applet includes GUI controls to allow a subset of the track to be chosen, along with an output format and colouring information. The user also selects a location from which the final data should be generated and returned (SARA data is mirrored in multiple locations around the world. The initial SARA system was based on stateless CGI scripting. Latest proposals for the architecture include an extensible, scalable digital library that allows complex, supervised processing and data mining.

This architecture shares a number of ideas that are common to those used in EASIA. The architecture uses Java Servlets and XML. An XML vocabulary is used to describe SAR data. This contains metadata about a track and a list of base URLs and associated files from which the binary SAR data can be obtained. In the new system a number of services are combined using XML requests and responses over HTTP. The XML response contains only textual data that includes URLs associated with any binary files generated by the request. The client can retrieve the binary files using these URLs. The architectures used in the Caltech projects are often integrated with supercomputing facilities. Their architectures are designed to allow on-demand back-end processing to post-process data in order to reduce data returned to clients with low bandwidth connections.

Whilst the proposals for the SARA system share similarities with EASIA features there are also a number of differences. The user interface is not automatically generated. Instead, the system appears to use customised GUI front-ends for the particular domain. The architecture does not concentrate on secure, distributed in-place storage of data for bandwidth reduction. They do not describe a facility for allowing arbitrary post-processing code to be uploaded by the users for secure execution against the data. The Caltech architecture uses XML as a means to transfer metadata and control information between services. The system is extended by adding new services. However, SARA does not appear to use XML as the mechanism for defining the user interface or for defining how new services are included in the system. Code has to be written to incorporate any changes. EASIA uses XML to define the user interface and to define how new post-processing logic is included, and simply requires changes to the markup to incorporate extensions.

The *WebWinds Interactive Data System* is also under development at Caltech by members of the Jet Propulsion Laboratory [83] [229]. This is a Java application that allows atmospheric scientists to visualise and analyse data. WebWinds is downloaded as a Java application and runs on the client machine. Data for analysis must exist local to the client machine or be accessible via a URL in which case the data must first be downloaded and saved. Alternatively, accessing data via a URL from a browser can launch WebWinds as a helper application for a Web browser with an appropriately configured MIME type. Data formats must be self-describing,

e.g. HDF or NetCDF, or raw binary/textual data can be handled by first using a tool to specify parameters for the data. For large datasets, too large to fit in memory or too large to view, WebWinds allows reduction in resolution or sub-setting to limit the viewed area.

WebWinds is designed to aid understanding of data through visualisation and scientific collaboration through transfer of information among computers and people. A user can construct a sophisticated interactive visualisation interface using simple 'point and click' actions without programming knowledge. Construction consists of interactively linking data, control and display components in windows on the client machine. Components consist of around 20 'application tools' split into 3 categories: displays, display filters and controls. Sophisticated visualisation is available for up to 3 dimensions. Two-dimensional slices are displayed, often with a slider controlling the third dimension. 3D displays are planned for the future.

A WebWinds session generates a file (a 'script') containing the set of commands that were issued. These can then be rerun by the user later to recreate the session, and possibly sent to a collaborator for their own use. The underlying scripting language also facilitates collaboration as follows. If two remote users both have WebWinds running, and they approve connection, then both users' WebWinds desktops will mimic each other. This activity requires that both users have a local copy of the data, or access the same data via a URL. In this way, only scripting commands need to be exchanged during the collaborative session.

Future research on WebWinds aims to enhance the architectural options by the end of 2000. Enhancements include allowing distribution of the components of a Java application to a server so that data can be accessed by the server through high-bandwidth connections to make selecting only the data that is required much more efficient. They also plan to provide server-side packages for subsetting data near the data source. The authors acknowledge that large datasets present a challenge due to limited Internet bandwidth and that a means of reducing bandwidth requirement is to subset data at source. Finally, they plan to build interfaces to other database systems to allow more sophisticated data queries, and to allow the results to be fed directly into the display and control applications.

Whilst WebWinds provides a mechanism for visualising/analysing large datasets along with a novel method of facilitating collaboration, it is the ideas for future work that bear closest resemblance to EASIA. EASIA already integrates database queries for initial discovery of data, with results fed directly into distributed server-side post-processing applications. EASIA provides a number of other advantages. First, EASIA manages distributed data in a secure fashion using access tokens associated with data stored as DATALINKs. Secondly, because EASIA stores data on distributed servers, it is possible to manage different types of raw binary files as required. WebWinds can manage raw binary files but, since these are read by Java applications on the client platform, they need to use the standard Java sizes to represent different data types and the raw files need to use *big-endian* byte ordering (or the Java applications need to make all the necessary conversions). Finally, EASIA is extensible, allowing new post-processing operations to be added. The final two advantages allow, for example, EASIA to manage datasets written as unformatted FORTRAN files on a platform that uses *little-endian* byte ordering, with associated post-processing applications associated with the datasets through customisation of the XUIS.

6.3 Discussion

This chapter has described research related to the DBbrowse and EASIA architectures. DBbrowse, which automatically generated Web interfaces to databases, differed from related systems in a number of ways. The majority of related systems were targeted at creating application domain specific interfaces rather than generic interfaces. These required programming effort to define page templates and pre-defined queries. At the time, none of these systems provide hypertext browsing via automatic links derived from referential integrity constraints extracted from the catalogue. Some stand-alone systems provided a degree of data or schema browsing, however these were usually tied to prototype database systems rather than commercial database systems. In addition, the systems that did include some form of database browsing were usually designed to work with object-oriented or entity-relationship databases and schemas, in which explicit relationships are

defined. DBbrowse derived inferred relationships and was targeted at the object-relational data model.

DBbrowse was used as the basis for the EASIA architecture. The EASIA architecture provides a generic architecture for management of large, distributed scientific data. Most existing Web-based interfaces to scientific data archives do not consider the problems associated with initially transferring large datasets to the archive. Nor do they provide mechanisms for maintaining integrity between metadata describing results and the actual data files. The EASIA architecture provides a Web-based interface to a distributed scientific data archive whilst maintaining database security, integrity and recovery features (implemented using the new SQL:1999 DATALINK SQL-type).

The interface presented by EASIA uses automation techniques from DBbrowse, requiring no HTML page maintenance. As such, EASIA does not rely on the scientific user-base having Web development experience. The extensibility mechanism used in EASIA also differs from related research. EASIA allows customisation through changes to XML markup. Additional post-processing services can be added to EASIA in this way. This provides an easy way to increase the capabilities of the system with the inclusion of post-processing codes written in any language. These codes do not have to be specially written to comply with any API defined by the system, as is the case with most other extensible architectures. The new post-processing services merely have to follow a simple design pattern, which requires them to read the data to be processed from a filename supplied as a command line argument, and to use relative path names for any file output. Since post-processing codes used in EASIA follow this generic pattern, and do not use any EASIA specific interfaces, they can be downloaded and used for other purposes if desired. Finally, EASIA differs from related architectures by allowing users to upload arbitrary post-processing code for secure server-side execution against the archived data.

7 Summary

This thesis describes research into Web-based management of non-traditional data. Traditional data was defined as simple datatypes including integers, floating-point types, characters, dates, times and timestamps (effectively datatypes that are associated with the traditional relational data model (see Chapter 2)). Non-traditional data is characterised by complex multimedia datatypes including text, audio, image and video, as well as binary files used for other purposes such as multidimensional scientific data (effectively datatypes that are associated with newer object-oriented and object-relational data models (see Chapter 2)). Three prototype architectures are discussed, *GBIS*, *DBbrowse* and *EASIA*, each of which provided exemplars of new ideas in this area of Web-based management of non-traditional data. Research and development of these prototype systems has required comprehensive knowledge of Web and database technologies. Therefore, this thesis also contains critical review of technologies and developments in these areas.

7.1 Contributions to the Field

Research into the *GBIS*, *DBbrowse* and *EASIA* architectures has been published in [181] [117] [180] [75] [182] [183] (and has appeared in conference poster sessions, tutorials and other invited talks, see Appendix A). Each of these prototype systems demonstrated new ideas for Web-based management of non-traditional data using commodity components, technologies and open standards.

7.1.1 *GBIS*

GBIS was an early system employing CGI scripting combined with standard application programs, to provide Web-based management of scientific data. *GBIS* successfully demonstrated the benefits of dynamic Web-based graphical results for displaying multiprocessor benchmark results. *GBIS* was well received by members of the multiprocessor benchmarking community. The system has been reported in Hockney's book [117], mirrored at the University of Tennessee at Knoxville [100], adopted by the RAPS (Real Applications on Parallel Systems) Consortium to form the basis of their Online Benchmark Information System [194], and cited in the NAS

Parallel Benchmarks 2.0 specification [16] as a site for accessing NPB results in graphical Web pages.

7.1.2 DBbrowse

The DBbrowse prototype demonstrated an automatically generated, generic Web interface to an underlying object-relational database. DBbrowse can generate Web interfaces with intuitive query capabilities. DBbrowse automatically handles non-traditional BLOB and CLOB datatypes by delivering them to the user's Web browser with an appropriately specified content type. At the time DBbrowse was created, Web/database integration was just beginning to emerge, as a way of overcoming the shortcomings associated with static Web pages or dynamic Web pages based on data in a file system (as was the case in GBIS for example). Web/database integration can reduce HTML page maintenance when the data changes, and reduce the instance of invalid hypertext links when static pages are removed and reordered. Web pages dynamically generated from databases can also benefit from the sophisticated search engine provided by the database management system. DBbrowse differed from commercial product offerings at the time, which required programming effort, for example, to define page templates and predefined embedded queries and hypertext links.

DBbrowse also demonstrated a method for browsing databases to further support users with little database experience. This technique provides one possible solution to Manber's suggestion that one of the main lessons to be gained from the success of the Web was the importance of browsing, and that an important step would be to find a way to browse even relational databases [147]. The data browsing technique used in DBbrowse does not rely on explicit relationships defined in the data model (as was the case with some of the related work), but extracts implied relationships from the relational model to dynamically include hyperlinks in results.

7.1.3 EASIA

The EASIA architecture was motivated by the need for research into scientific data archives as highlighted in a number of reports and papers. For example, the Caltech Workshop on Interfaces to Scientific Data Archives [235] identified an urgent need for infrastructures that could manage and federate active libraries of scientific data.

Hawick and Coddington [112] state that the information explosion has led to a very real and practical need for systems to manage and interface to scientific archives. Treinish's [217] [218] recommendations for future research into interactive archives for scientific data include the integration of metadata, data servers, visual browsing and existing data analysis tools. The features of EASIA provide several novel ideas for addressing these requirements.

The EASIA architecture demonstrates Web-based management of large, distributed, scientific data. EASIA uses sample data from numerical simulations run by the UK Turbulence Consortium to investigate Web-based mechanisms for management of large datasets, with a total storage requirement in the hundreds of gigabytes range, in the relatively low bandwidth environment exhibited by the Web.

EASIA combines leading edge Web-based technologies including an implementation of the new SQL:1999 DATALINK type, defined in *SQL Management of External Data (SQL/MED)* [60], to provide database management of scientific metadata and large, distributed result files simultaneously with integrity. This technology is applied to the Web, by providing a user interface to securely manage large files in a distributed scientific archive, despite limited bandwidth.

The EASIA architecture demonstrates a number of advantages for Web-based archiving of large scientific datasets:

- Unlike most existing Web-based scientific data archives, EASIA considers the problems associated with initially transferring large datasets to the archive. The EASIA solution is to allow datasets to be managed in a *distributed* fashion (with centralised metadata) thereby avoiding costly network transfers associated with uploading data files to a centralised site.
- Result files can be archived at (or close to) the point where they are generated.
- Proper data distribution reduces access bottlenecks at individual sites.
- Bandwidth requirement is further reduced by the *active* nature of the EASIA architecture. EASIA can archive applications as well as data. Post-processing of data can be achieved via these archived applications, and also by allowing users to *upload* code to be run securely on the file servers hosting the datasets.

Suitable user-directed post-processing, such as array slicing and visualisation, can significantly reduce the amount of data that needs to be shipped back to the user.

- Data can be distributed so that it is physically located closest to its intensive usage.

EASIA helps users to locate scientific data files of interest, using an intuitive interface that incorporates a searching and browsing mechanism that has been inherited from DBbrowse. Also, unlike many existing scientific data archives, EASIA is easily extended by allowing the user interface specification to be defined in an XML file. Separating the user interface specification from the user interface processing demonstrates a number of further advantages:

- The user interface, although schema driven can be customised and relationships between data can be specified even if these relationships do not exist, or are not apparent from the database schema.
- Different users (or classes of user) can have different XML files thereby providing them with different user interfaces to the same data.
- The EASIA architecture provides a generic interface that can be applied to many different applications.
- For scientific data archiving a major benefit, facilitated by the XML user interface specification, is the ease with which standard post-processing codes can be associated with data. Post-processing codes that have been archived by the system can be associated with remote data files *using simple XML markup* in the XUIS. This allows dynamic server-side execution of the stored applications, with chosen datasets as input parameters, to generate derived data on-demand. Applications are loosely coupled to the datasets (in a many-to-many relationship) via XML defined interfaces specified in the XUIS. This allows reuse of these server-side post-processing operations.

- Because simulation results are stored in unmodified files, existing post-processing applications, that use standard file I/O techniques, can be applied to the files without having to rewrite the applications. An alternative would be to modify applications to first access result objects from a database, but this would be very undesirable for many scientific users who often apply post-processing codes written in FORTRAN. This would also be more costly since the data would have to be loaded into the database as BLOBs and extracted from the database when required. Each file server host machine provides a distributed processing capability thereby allowing multiple datasets to be post-processed simultaneously.

7.2 Future Work

7.2.1 Gathering Operation Statistics and Caching Results

A useful extension to EASIA would be to store in the database the execution times for operations, which have already been run by users, along with the output type and the size of the output. The identity of the operation and the identity of the dataset, as well as any parameters would also be recorded. This would allow future users to scan existing results to get an estimate for the execution time of any post-processing that they require. An extension to this would be to allocate an amount of storage on the file servers as an ‘operation result cache’. The results from post-processing operations (i.e. derived data, images, etc.) could then be stored in the database using DATALINKs. The table of operation statistics would be updated to indicate that the operation result is available from cache. A replacement policy such as FIFO (first in first out) or LRU (least recently used) could be applied once the cache is full. Storing operation results as DATALINKs should be a reasonably easy task. Operation results are already written to a directory structure containing unique directory names. The Servlet code running on the database server host (Figure 21) already handles the invocation of operations and returns the results to the user. It could be amended to time the invocation and update the database.

The above scenario mirrors the belief of Hawick and Coddington [112] that it is not necessarily feasible or desirable to store all possible derived data combinations for a given archive. Instead, it is more practical to store the services

needed to create the derived data products and to cache results to amortise their cost of production. However, they also make the important point that trade-offs need to be considered before caching derived data. Factors such as user access patterns and the time and resources required to generate the data need to be evaluated against the resources required to store and manage the derived data. EASIA could include additional configuration parameters associated with this cache, such as a minimum execution time, below which it is not worth caching results.

7.2.2 Providing a Multidatabase Capability

Pepcke *et al.* [178] state that collections for digital libraries are often maintained by different organisations so that autonomous control over access to collections is required in many cases. EASIA could be modified to provide a single view of multiple autonomous databases in a loosely coupled *federated database system* (FDBS). The FDBS would use a *multidatabase language system* (in the taxonomy by Bright *et al.* [24]) in which heterogeneous databases can be accessed at the user interface level through a query language and tools that can integrate data from different sources without a global schema. This functionality could be implemented by extending the DTD for the XUIS to contain multiple database user interface definitions. The relationships, which currently allow browsing of related data in associated tables, would be extended to allow browsing to related data in tables in disjoint databases. Queries could be shipped to a remote database using 2-tier JDBC drivers that operate over networks or by using a distributed object technology in a multi-tier architecture, or by using an XML/HTTP combination (as described in Chapter 2). As well as FDB browsing, a new XUIS DTD could allow several tables to be grouped into a *supertable* that could present a single view of the grouped tables for query purposes. This FDB functionality, although limited, would serve a useful purpose of allowing not only distribution of result files, but also distribution and autonomous administration of the database containing the simulation metadata stored internally in the database at different sites. This would remove the potential single point of failure from the architecture – the database server host (Figure 21). A simpler way to remove this single point of failure initially is simply to install the core EASIA code on multiple database server hosts.

7.2.3 Can Codes other than Java be Uploaded for Execution?

Many scientific researchers use FORTRAN or C to post-process their results. It would therefore be extremely beneficial if EASIA could accept uploaded code consisting of binary codes generated from these languages (not just Java uploads). If security were not an issue this would be trivial as the code could simply be executed in a temporary directory as an external command similar to Java uploads. Whilst EASIA allows this to happen for pre-defined archived applications (with the assumption that an administrator is only providing proven code as part of the system), it is not an acceptable policy for uploaded code.

The Java Native Interface (JNI) allows native code (currently C and C++) to be integrated into Java applications. However, the security manager cannot control native methods as they bypass the low-level bytecode checks [231]. Java security is designed around the implicit assumption that the user's entire application is coded in Java. Research could be devoted to finding secure mechanisms for executing uploaded binary codes. Brown [25] states that one approach to ensuring secure execution of remotely sourced binary code on user's systems is "the use of 'software fault isolation technologies' [226] which augment the instruction stream, inserting additional checks to ensure safe execution [209]". An interim solution, that is not ideal, would be to allow the upload of C or FORTRAN binaries, by specially trusted users.

C and FORTRAN code would also be platform specific. Whilst it is easier to ensure that pre-defined archived applications can be compiled for the particular file server host that they will reside on, this becomes more of an issue for uploaded codes. Users may not have access to the correct platforms or compilers for the intended server-side host. Another line of enquiry might therefore be to look at the possibilities for uploading source codes. It might also prove to be easier to vet source code [25]. Another possibility along these lines would involve research into automatic translation of uploaded C and FORTRAN source code into secure Java code.

7.2.4 Runtime Monitoring of Post-Processing Operations

Another feature that may be a desirable extension to EASIA is the provision of feedback during the execution of post-processing codes (particularly where execution times are long (see for example [146])). The user could query the system during a long simulation and observe information such as current output to standard output or standard error, or other result files generated so far, or the average time to produce a result file. An option to stop the simulation from the Web could also be provided.

7.2.5 XML as a Scientific Data Standard

Scientific data archives and tools for processing scientific data can benefit from standardised data formats. Standardised data formats make it easier for scientists to exchange data, and they make it easier for tools and processes to exchange data. Self-describing data formats can additionally benefit from keeping all the information necessary to understand and process the data, bound to the data. Self-describing data can also benefit from tools and processes that require less user intervention to specify the details of the data representation.

Currently, EASIA can archive datasets that use any data standard or representation. It is then up to the user to either download that dataset to their client and post-process it locally, or to use archived operations that have been associated with the dataset, or to upload post-processing code written in Java that understands the data representation. If the user post-processes data using archived operations then details of the data format are less relevant to the user. Instead, the data format of any output derived from the post-processing is important to the user in this case. If the user uploads post-processing code, or downloads a dataset to the client then the exact format of the data representation needs to be known. This information can be stored in metadata in other attributes in the database that are linked to the relevant dataset. Alternatively, the dataset itself can be self-describing, for example, using a format such as HDF [115].

Operations are loosely coupled to data in EASIA, allowing them to be reused for many different datasets. This reuse is greatly enhanced if the operations are written in a generic fashion that reads and writes standardised data formats (refer to

[201] for an extensive listing of other commonly used scientific data formats). For example, Section 5.2.5 explained how the SDB operation was used in EASIA to post-process arbitrary HDF datasets.

Whilst HDF is a commonly used scientific data format, at the current time, XML is emerging as the standardised data format for just about any application. XML is therefore gaining an advantage over other data formats in terms of recognition throughout the software community. Also since XML is being incorporated into many emerging software tools, data stored as XML will benefit from the availability of generic *off-the-shelf* tools. Furthermore, XML is both human and machine-readable. These advantages do not make formats such as HDF redundant. Indeed, it is possible for XML to provide a wrapper around binary formats such as HDF, and probably necessary since XML is predominantly a text-based format. XML can incorporate binary data through text encoding such as base-64 encoding but this is likely to be too verbose for large scientific datasets. The more likely integration for XML and scientific data formats is for XML to store simple textual metadata internally and for the XML to link to external binary files containing the raw results. In this fashion, existing scientific formats can benefit from the advantages offered by XML. The Extensible Scientific Interchange Language (XSIL) [19] is an example of this.

XSIL is an XML defined language for representing collections of scientific data objects. This is not tied to a particular scientific domain but has support for common scientific data formats such as parameters, arrays and tables and binary streams which can consist of internal character data or links to external data in various formats, for example, binary files. The authors indicate a number of uses for XSIL including use as general transport format between disparate applications, as a documentation format for collections of scientific data, and as an “ultra-light” data format, whereby the XML contains all the required metadata for external files. This format is light since the metadata can be digested and then deleted once the description of the linked raw files has been obtained. Whilst the DTD for XSIL looks fairly superficial at this stage, and probably needs extending to be able to represent more scientific datasets, the concept is extremely important.

On-going research into scientific data archives should investigate the use of XML as a common data format. This research would look at defining new XML schemas for representing a wide variety of scientific data. Once these XML languages exist then there will be enormous benefits in terms of data and application reuse, for example for post-processing operations in an architecture like EASIA.

Beyond this, XML could also be used to define the interfaces between applications. Well known XML schemas could specify the type of data that an application can handle, and also the type of post-processing service and output that is offered. It may be possible in future to dynamically discover distributed services from the XML interfaces they present. It may then be possible to ship small XML defined datasets to the services for post-processing, or alternatively the code for the service could be shipped to the data for secure execution.

7.2.6 Other Enhancements to the EASIA Architecture

Brief descriptions of other enhancements that could be made to EASIA are listed below. The first list of enhancements is associated with extensions to the DTD for the XUIS, to allow a greater diversity of operations and parameters to be specified.

- Currently operations apply to a single dataset at a time. This mechanism could be extended to allow operation processing to be applied to all datasets present in a result, thereby allowing post-processing code that requires, say, average values over many timesteps in a simulation to be selected once, rather than repeatedly for many individual datasets.
- Currently, only one operation can be applied to a dataset before the output is returned to the user. Operations could be chained to allow multiple stages of post-processing in which the output of the first operation is used as the input of the next operation in the chain.
- Operations need not be restricted to post-processing files stored in DATALINK columns. They could also be applied to LOB columns, or indeed columns associated with traditional types (providing, for example, string processing, mathematical functions, or graphs for numeric columns).

- Operations are currently archived by the database using DATALINKs, or exist as hosted services invoked through a URL with associated parameters. This model could be extended to allow operations to be stored on the database server host in BLOB attributes. These operations would then need to be uploaded to the file server host for execution against a dataset. These codes would have to be Java to cope with platform heterogeneity. Alternatively binary versions of the code would need to be stored for different platforms, or a mechanism of source code distribution and automatic compilation would need to be implemented.
- EASIA could be extended with more interactive operations. These could take the form of Java Applets or Java applications that are downloaded to the user. This would be desirable for some applications where a complex GUI is more suitable to the particular post-processing method. For example, a Java image or movie viewer Applet on the client could display images/movies being generated server side and streamed to the client.
- The XUIS could be extended to allow user-defined presentation of the output of queries, particularly the presentation of post-processing operation results.

Additionally, EASIA could benefit from the following enhancements.

- EASIA could be extended to include version control for both data and code.
- EASIA could be extended to support scheduling of post-processing operations, particularly where the operations are resource intensive and where they require parallel computing facilities.
- EASIA could be extended to include off-line storage.
- Currently EASIA is updated using raw database update mechanisms such as importing data from comma delimited text files. A Web-based GUI needs to be implemented based on information in the XUIS in a similar fashion to the query interface.
- An investigation could be carried out into caching and replication extensions to the EASIA architecture. This could include the asynchronous migration (or

replication) of datasets to file server hosts closest to the location of most intensive client usage.

- EASIA could be extended with a facility to download all the files displayed in a query result or operation result, without multiple intervention from the user. For example, in the case where a query yielded say 100 separate files representing datasets from different timesteps in a simulation. Currently, the user would need to click on each of the displayed links in turn to save the files. (Note that commercial tools are available to extend Web browsers with such download facilities. Some can even restart downloads of individual files from the point of failure if they fail mid way through). In a similar vain EASIA could offer an in-built option to allow the user to retrieve large files in several smaller chunks. (Currently, this can only be achieved for operation output files, and only then by including a specific operation with this facility.)

7.3 Concluding Remarks

The final results of this thesis demonstrated a novel Web-based architecture for management of scientific data. Bandwidth availability is a limiting factor in this environment, and is of particular relevance to the management of the large scientific datasets that often result from numerical simulations run on high performance computing platforms. Although bandwidth availability is likely to increase dramatically in the next few years, advances in computing technology will also result in increases in the size of scientific datasets that are generated. Active digital libraries are therefore likely to remain an important area of research for the foreseeable future. In order that scientists can concentrate on their core competency, data management architectures need to be automated, where possible, and easily extensible to make use of new and existing data post-processing tools.

Appendix A : Publications and Presentations

◆ Refereed Journal Papers

- Papiani, M., Wason, J., L., Dunlop, A., N. and Nicole, D., A. A Distributed Scientific Data Archive Using the Web, XML and SQL/MED. *ACM SIGMOD Record*, Vol. 28(3), September, 1999, 56-62.
- Dunlop, A.N., Papiani, M. and Hey, A., J., G. Providing Access to a Multimedia Archive Using the World-Wide Web and an Object-Relational Database Management System. *IEE Computing & Control Engineering Journal*, Vol. 7(5), October, 1996, 221-6. ISSN 0956-3385.
- Papiani, M., Hey, A., J., G. and Hockney, R., W. The Graphical Benchmark Information Service. *Scientific Programming*, Vol. 4(4), 1995, 219-227. ISSN 1058-9244.
 - Also appears *In: Hockney, R., W. The Science of Computer Benchmarking*, Philadelphia, USA, SIAM, 1996, 104-15. ISBN 0-89871-363-3.

◆ Refereed Conference Proceedings

- Papiani, M., Wason, J., L. and Nicole, D., A. An Architecture for Management of Large, Distributed, Scientific Data Using SQL/MED and XML. *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, Konstanz, Germany, March, 2000.
(In: Zanolio, C., Lockermann, P., C., Scholl, M., H. and Grust, T., eds. Advances in Database Technology, EDBT 2000, Lecture Notes in Computer Science, Vol. 1777, Springer-Verlag, 2000. ISBN 3-540-67227-3)
- Papiani, M., Dunlop, A., N. and Hey, A., J., G. Automatic Web Interfaces and Browsing for Object-Relational Databases, *Advances in Databases: Proceedings of the 15th British National Conference on Databases, BNCOD15*, London, 7-9th July, 1997, 131-2.

(In: Embury, S., M., Fiddian, N., J., Gray, W., A. and Jones, C., J., eds. *Advances in Database*, Lecture Notes in Computer Science, Vol. 1271, Springer-Verlag, 1997. ISBN 3-540-63263-8.)

◆ Refereed Poster Sessions

- Papiani, M., Dunlop, A., N. and Hey, A., J., G. Automatic Web Interfaces and Hypertext Browsing for Object-Relational Databases, *The Eighth ACM International Conference on Hypertext*, Southampton, England, April 1997.

◆ Related International Conference Tutorial Presentations

- Dunlop, A., N. and Papiani, M. Java, the Web and Databases. *Tutorial session, Euro-par '98, 4th International Euro-Par Conference*. 1-4 September 1998, University of Southampton, UK.

◆ Related Invited Talks

- Papiani, M., Dunlop, A., N. and Hey, A., J., G. Automatically Generating World-Wide Web Interfaces to Relational Databases, *British Computer Society Seminar Series on New Directions in Systems Development: Intranets The Corporate Superhighway*, University of Wolverhampton, April 1997.
- Papiani, M. Parkbench (PARallel Kernels and BENCHmarks) Release 1.0 and the Graphical Benchmark Information Service (GBIS), In: *proceedings of the RAPS (Real Applications on Parallel Systems) Workshop*, Graz, Austria, 18 May, 1995.

Appendix B : Client/Server ‘Pi ng’ Benchmark Results

This appendix reports some benchmark results from 1997, which compared the performance of available technologies for Web based client/server interaction. The codes for the benchmarks were taken from Orfali and Harkey [175]. The benchmark measures the round trip time for a client to increment a variable on a server and receive a reply from the server (see Figure 36). First, the client program calls the server program to initialise the variable on the server to zero and then starts a timer. Secondly, the client calls the server repeatedly to increment the variable. Thirdly, the client program stops the timer and calls the server to find the final value of the variable. The average round-trip ping time is calculated and displayed.

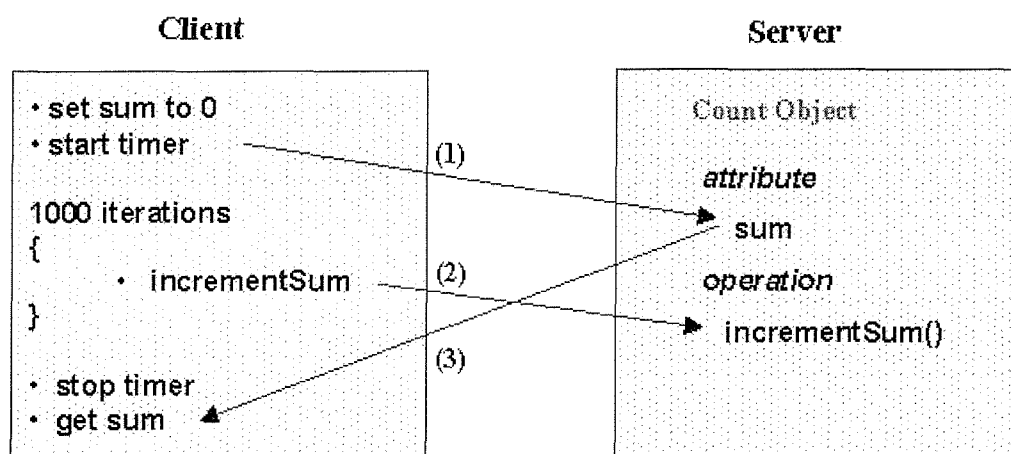


Figure 36: The client/server ping benchmark.

Four different client/server technologies/configurations were compared. Throughout, Java was used as the programming language. For each configuration the experiment was repeated between five and ten times. Each experiment measured the time for between ten and one thousand increments and recorded the average time for one increment (or ‘ping’). The results of the experiments are displayed in Figure 37. The error-bars on the graph show the minimum and maximum value recorded for the average ping for each configuration.

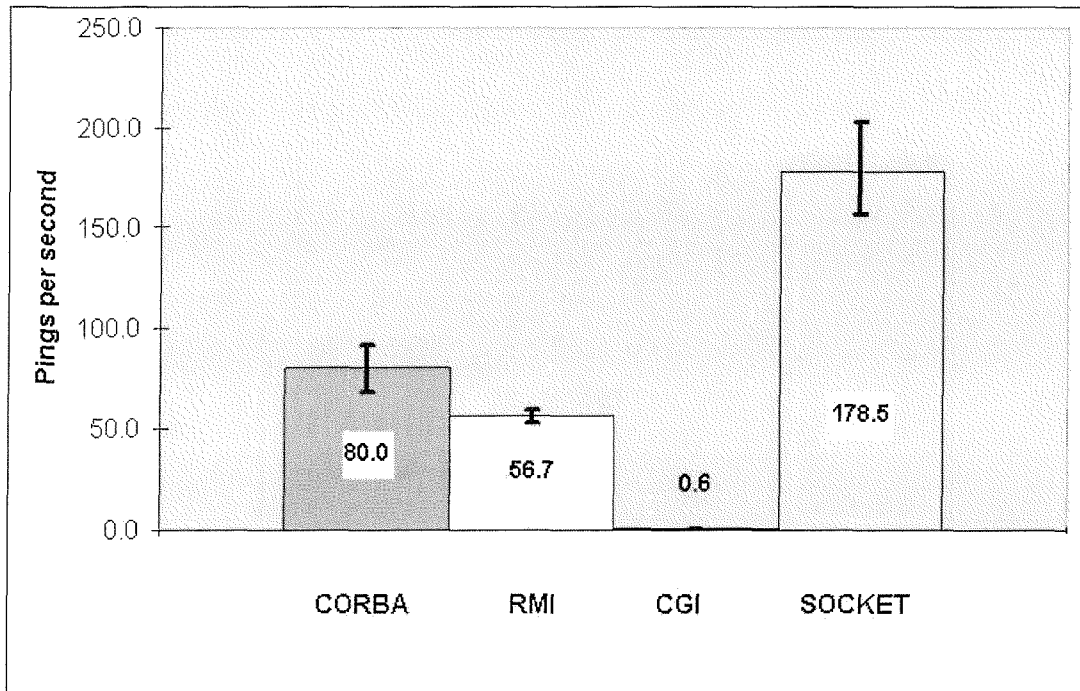


Figure 37: Client/server ping benchmark results.

The following notes apply generally to the experiment:

- The client ran on a workstation and the server on a single node IBM SP2 supercomputer, although little difference was noted if the locations of the client and server were transposed.
- The client platform consisted of an IBM Power Series 820, PowerPC workstation with a 100MHz RS6000 603e CPU, 32MByte of memory, running the AIX 4.1.4.0 operating system.
- The server Platform consisted of a single 66.5MHz thin-node 2 on an IBM SP2, with 256MByte memory, running the AIX 4.1.4.0 operating system.
- The network consisted of Southampton University's shared 10 Mbit/s Ethernet network. Results were recorded over several days on a typically loaded network.
- IBM's Java Development Kit (JDK), version 1.0.2D, was used for all the experiments because a just-in-time compiler was available for this version. This is with the exception of the RMI experiment that used IBM's JDK version 1.1.1

because the RMI classes were not available in the earlier release. There was no just-in-time compiler available with IBM's JDK1.1.1. However, in practice, the just-in-time compiler did not appear to have an effect on the performance of this benchmark. It appears that the network delays were far more significant than the computation times of the Java programs.

- The CORBA experiment used the VisiBroker for Java [225] (version 1.2.0, December 9, 1996) CORBA2.0 implementation.
- The socket experiment used the Java *BufferedOutputStream* class that requires data to first be converted to bytes. The bytes are then stored in a buffer and written to the output stream when the buffer is full, or explicitly via a method call.
- The CGI experiment used the *Netscape Fast Track* HTTP Server. A Java Applet running on the client machine made a socket connection to the server and used the 'POST' method of the HTTP protocol to initiate the CGI program on the server. The CGI program was also written in Java.

The following conclusions can be drawn from the results:

1. Sockets give the best performance of around 180 pings per second.
2. Distributed object technologies (CORBA and RMI) give the next best performance (approximately half that of the pure socket performance).
3. CGI performs two orders of magnitude worse than the distributed object technologies in this simple client/server ping benchmark.

Hence, whilst the CGI mechanism is widely used to provide HTML forms for user interaction with the Web, in its raw form this technology exhibits poor performance when compared with other Java based technologies that are available for Web-based client/server interaction. Java Servlets did not form part of this experiment as it was carried out before they became generally available. However, Orfali and Harkey

have shown Servlets to perform an order of magnitude better than CGI in the above benchmark [175]. Chapter 2 discusses the technologies and options in more detail.

References

- [1] Accessing a Database Server via the World Wide Web.
<http://kulichki-iso.rambler.ru/moshkow/WEBMASTER/dbcgigateways.txt>
- [2] The Active Group Homepage. <http://www.activex.org/>
- [3] Active Server Pages (ASP), Microsoft Corporation.
<http://msdn.microsoft.com/workshop/server/toc.htm>
- [4] ActiveState PerlEx, ActiveState Tool Corporation.
<http://www.activestate.com/plex>
- [5] ActiveX Data Objects (ADO), Microsoft Universal Data Access Web Site, Microsoft Corporation. <http://www.microsoft.com/data/ado/>
- [6] Addison, C., A., Getov, V., S., Hey, A., J., G., Hockney, R., W. and Wolton, I., C. The GENESIS Distributed memory Benchmarks. *In: Dongarra, J., J. and Gentsch, W., eds. Computer Benchmarks*, North Holland, 1993, 257-271.
- [7] ADO/WFC vs. JDBC, Microsoft Universal Data Access, Technical Materials, Microsoft Corporation, August, 1998.
<http://www.microsoft.com/data/techmat.htm>
<http://www.microsoft.com/data/ado/adotechinfo/adovsjdbc.htm>
- [8] Agrawal, R., Gehani, N., H. and Srinivasan, J. OdeView: the Graphical Interface to Ode. *Proc. ACM SIGMOD Conference*, May, 1990.
- [9] Aloiso, G., Cafaro, M., Messina, P. and Williams, R., D. A Distributed Web-based Metacomputing Environment. *Proceedings of the 5th Int. Conf. On High Performance Computing and Networking Europe*, Vienna, Austria, 1997.
- [10] Aloiso, G., Cafaro, M. and Williams, R. The Digital Puglia project: An Active Digital Library of remote Sensing Data. *Proceedings of the 7th Int. Conf. On High Performance Computing and Networking Europe*, Amsterdam, The Netherlands, 1999.

- [11] Aloisio, G., Milillo, G. and Williams, R. An XML Architecture for High Performance Web-based Analysis of Remote-Sensing Archives. Technical Report CACR-167, Center for Advanced Computing at Caltech, September 1988. (Submitted to Future Generation Computer Systems.)
- [12] American National Standards Institute (ANSI) Electronic Standards Store.
<http://webstore.ansi.org>
- [13] Apache API Notes, The Apache Server Group.
<http://www.apache.org/docs/misc/API.html>
- [14] Associating Style Sheets with XML documents, Version 1.0, W3C Recommendation, 29 June, 1999. <http://www.w3.org/TR/xml-stylesheet>
- [15] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrisnan, V. and Weeratunga, S. The NAS Parallel Benchmarks, Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA 94035, USA, March 1994. <http://www.nas.nasa.gov/Software/NPB/>
- [16] Bailey, D., Harris, T., Saphir, W., van der Winjgaart, R., Woo, A. and Yarrow, M. The NAS Parallel Benchmarks 2.0., Technical Report RNR-95-020, NASA Ames Research Center, Moffett Field, CA 94035-1000, USA, December, 1995. <http://www.nas.nasa.gov/Software/NPB/>
- [17] Baru, C., K., Fecteau, G., Goyal, A., Hsiao, H., Jhingram, A., Padmanabhan, S., Copeland, G., P. and Wilson, W., G. DB2 Parallel Edition. *IBM Systems Journal*, Vol. 34(2) 1995, 292-322.
- [18] BestWeb Intelligent Interface Builder, Version 1.0., Best-Seller Inc., 3300 Cote Vertu, Suite 303, Montreal, Quebec H4R 2B8.
- [19] Blackburn, K., Lazzarini, A., Prince, T. and Williams, R. XSIL: Extensible Scientific Interchange Language. *Proceedings of the HPCN 99 Conference*, Amsterdam, 1999. <http://www.cacr.caltech.edu/~roy/papers/xsil.pdf>
- [20] Bloom, I., P. Object Databases versus Universal Servers: Reality and Myth, May, Volpe Brown, Whelan and Co., San Francisco, USA, 1997.

<http://gate.vwco.com/>

- [21] Borrás, P., *et al.* Building User Interfaces for Database Applications: The O2 Experience. *SIGMOD Record*, Vol. 21(1), March 1992.
- [22] Box, D. Lessons from the Component Wars: An XML Manifesto, September, 1999.
<http://msdn.microsoft.com/workshop/xml/articles/xmlmanifesto.asp>
- [23] Bray, J., Paoli, J. and Sperberg-McQueen, C., M. *eds.* Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February, 1998.
<http://www.w3.org/TR/REC-xml>
- [24] Bright, M., W., Hurson, A., R., and Pakzad, S., H. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, Vol. 25(3), 1992, 50-60.
- [25] Brown, L. Mobile Code Security, *Proceedings of AUUG 96 and Asia Pacific World Wide Web*, World Congress Centre, Melbourne, Victoria, Australia, 1996.
<http://www.csu.edu.au/special/auugwww96/proceedings/brown/brown.html>
- [26] Brown, M., R. FastCGI Specification Version: 1.0, Open Market, Inc., 29 April 1996. <http://www.fastcgi.com/fcgi-devkit-2.1/doc/fcgi-spec.html>
- [27] Brown, N. and Kindel, C. Distributed Component Object Model Protocol - DCOM/1.0, Microsoft Corporation, January, 1998.
<http://www.microsoft.com/com/resources/specs.asp>
- [28] Bryce, D. and Hull, R. SNAP: A Graphics Based Schema Manager. *Proc. IEEE 2nd Int. Conf. Data Engineering*, Los Angeles, California, February, 1986, 151-64.
- [29] Campbell, D., M., Embley, D., W. and Czejdo, B. Graphical Query Formulation for an Entity-Relationship Model. *Data and Knowledge Engineering*, Vol. 2, 1986, 89-121.
- [30] Carey, M., J. and DeWitt, D., J. Of Objects and Databases: A Decade of Turmoil. *Proceedings of the 22nd International Conference on Very Large Databases*, Mumbai (Bombay), India, 3-6 September, 1996, 3-14.

- [31] Carey, M., J., DeWitt, D., J., Naughton, J., F., Asgarian, M., Brown, P., Gehrke, J., E., and Shah, D., N. The BUCKY Object-Relational Benchmark. *Proceedings of the SIGMOD Int. Conf.*, 1997.
- [32] Carey, M., J., Haas, L., M., Maganty, V. and Williams, J., H. PESTO: An Integrated Query/Browser for Object Databases. *Proceedings of the 22nd International Conference on Very Large Databases*, Mumbai (Bombay), India, 3-6 September, 1996, 203-14.
- [33] Cascading Style Sheets, level 1, W3C Recommendation, 17 December, 1996. <http://www.w3.org/pub/WWW/TR/REC-CSS1>
- [34] Cascading Style Sheets, level 2, CSS2 Specification, W3C Recommendation 12 May, 1998. <http://www.w3.org/TR/REC-CSS2>
- [35] Catarci, T., Costabile, M., F., Levialdi, S., and Batini, C. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8, 1997, 215-60.
- [36] Catarci, T. and Santucci, G. Query By Diagram: A Graphic Query System. *Proc. 7th Int. Conf. Entity-Relationship Approach*, Rome, Italy, 1988.
- [37] Cattell, R. ed. *The Object Database Standard: ODMG-93 (Release 1.2)*, Morgan Kaufman Publishers, 1996. <http://www.odmg.org>
- [38] Cattell, R., G., G. and Barry, D., K., eds. *The Object Database Standard: ODMG-97, Release 2.0*, Morgan Kaufman Publishers, 1997, 256pp. <http://www.odmg.org>
- [39] Cattell, R., G., G., Barry, D., K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T. and Fernando Velez, eds. *The Object Database Standard: ODMG 3.0*, Morgan Kaufman Publishers, 2000, 300pp. <http://www.odmg.org>
- [40] Celko, J. and Celko, J. Debunking Object-Database Myths, *Byte*, October, 1997, 101-106.
- [41] The Cetus Links. <http://www.cetus-links.org/>

- [42] Chamberlain, D., D. *A Complete Guide to DB2 Universal Database*, Academic Press/Morgan Kaufmann, 1998, 800pp.
- [43] Chang, D. and Harkey, D. *Client/Server Data Access with Java and XML*. John Wiley and Sons, Inc., 1998, 606pp.
- [44] Chen, P. An Entity-Relationship Model - Towards a Unified View of Data. *ACM Transactions on Database Systems*, Vol. 1(1), January, 1976.
- [45] Chung, P., E., Huang, Y. and Shalini, S. *DCOM and CORBA Side by Side, Step by Step, and Layer by Layer*, Bell Laboratories, 1997.
http://www.bell-labs.com/~emerald/dcom_corba/Paper.html
- [46] Coddington, P., D., Hawick, K., A. and James, H., A. Web-based Access to Distributed High-Performance Geographic Information Systems for Decision Support. *Proc. of the Hawaii Int. Conf. On System Sciences (HICSS) '99*, January, 1999.
- [47] Coddington, P., D., Hawick, K., A., Kerry, K., E., Mathew, J., A., Silis, A., J., Webb, D., L., Whitbread, P., J., Irving, C., G., Grigg, M., W., Jana, R. and Tang, K. Implementation of a Geospatial Digital Library Using Java and CORBA. *Proc. Of TOOLS Asia '98*, September, 1988.
- [48] The Common Gateway Interface, National Center for Supercomputer Applications, University of Illinois, Urbana-Champaign, 1995.
<http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>
- [49] Cold Fusion Homepage, Allaire Corporation.
<http://www.allaire.com/products/coldfusion/index.cfm>
- [50] COM: Delivering on the Promises of Component Technology, Microsoft Corporation. <http://www.microsoft.com/com/default.asp>
- [51] COM for Solaris 1.0, Microsoft Corporation.
<http://www.microsoft.com/com/resources/solaris.asp>
- [52] COM for Tru64 UNIX, Compaq Corporation.
<http://www.unix.digital.com/com/>

- [53] The Common Object Request Broker: Architecture and Specification, Revision 2.3.1, Object management Group, October, 1999. <http://www.omg.org>
- [54] Comparing ActiveX and CORBA/IIOP, Object Management Group, 1997.
<http://ftp.omg.org/library/activex.html>
- [55] Comparing Microsoft Transaction Server to Enterprise JavaBeans, Microsoft Corporation, 30 July, 1998. <http://www.microsoft.com/com/wpaper/mts-ejb.asp>
- [56] The Component Object Model Specification, Microsoft Corporation and Digital Equipment Corporation, Draft Version 0.9, October 24, 1995.
<http://www.microsoft.com/com/resources/specs.asp>
- [57] Cover, R. The SGML/XML Web Page. September, 1999.
<http://www.oasis-open.org/cover/sgml-xml.html>
- [58] Cross-Platform ASP Development Strategies, white paper, Chili!Soft, 1999.
<http://www.chilisoft.com/whitepeppers/default.asp>
- [59] Database Language SQL, ANSI Standard No. X3.135-1992. American National Standards Institute, 1992. (ISO/IEC 9075:1992.)
- [60] Database Language SQL - Part 9: SQL/MED, Version for ISO FCD Ballot, November 1999. ftp://jerry.ece.umassd.edu/SC32/WG3/Progression_Documents/FCD/fcd1-med-1999-11.pdf
- [61] Database Language SQL - Part 10: SQL/OLB, ISO Final Committee Draft, November, 1999. ftp://jerry.ece.umassd.edu/SC32/WG3/Progression_Documents/FCD/fcdi2-olb-1999-11.pdf
- [62] Data Management: SQL Call Level Interface (CLI), *X/Open CAE Specification*, March, 1995. ISBN 1-85912-081-4, C451.
- [63] Davidson, J., D., and Ahmed, S. Java Servlet API Specification, Version 2.1a, November, 1998. <http://java.sun.com/products/servlet/index.html>
- [64] Davis, J., R. DATALINKS: Managing External Data with DB2 Universal Database, White paper, IBM Corporation, February, 1999.

<http://www.software.ibm.com/data/pubs/papers>

[65] DB2 Universal Database, IBM Corporation.

<http://www.software.ibm.com/data/db2/udb>

[66] DeWitt, J., D. Combining Object Relational Parallel: Like Trying to Mix Oil and water. <http://www.cs.wisc.edu/~dewitt/vldbsum.ps>

[67] DeWitt, D., J. and Gray, J. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, Vol. 35(6), 1992, 85-98.

[68] The DNS Database of Turbulence and Heat Transfer, Turbulence and Heat Transfer Laboratory, University of Tokyo, Japan.

<http://www.thtlab.t.u-tokyo.ac.jp/>

[69] Doan, D., K., Norman, W., P. and Kilgour, A. Design and User Testing of a Multi-Paradigm Query Interface to an Object-Oriented Database. *SIGMOD Record*, Vol. 24(3), September 1995.

[70] Document Object Model (DOM) Activity Statement, W3C.

<http://www.w3.org/DOM/Activity>

[71] Document Object Model (DOM) Level 1 Specification, Version 1.0, W3C Recommendation, October, 1998. <http://www.w3.org/TR/REC-DOM-Level-1>

[72] Document Object Model (DOM) Level 2 Specification, Version 1.0, W3C Candidate Recommendation, May, 2000.

<http://www.w3.org/TR/WD-DOM-Level-2>

[73] Document Object Model (DOM) Web page maintained by the W3C DOM Working Group, 11 December, 1998. <http://www.w3.org/DOM/>

[74] Duan, N., N. Distributed Database Access in a Corporate Environment Using Java. *Fifth International World Wide Web Conference*, Paris, France, May 6-10, 1996.

[75] Dunlop, A., N., Papianni, M. and Hey. A.J.G. Providing Access to a Multimedia Archive Using the World-Wide Web and an Object-Relational

- Database Management System. *IEE Computing & Control Engineering Journal*, Vol. 7(5), October 1996, 221-6.
- [76] Dunlop, A., N., Papiani, M., Quinn, M., J. and Hey. A.J.G. User-Directed Database Searching and Browsing via the World Wide Web, Technical Report 96-80-3, Department of Computer Science, Oregon State university, USA, October, 1996. <http://www.cs.orst.edu/~quinn/www-db/Overview.html>
- [77] ECMA - European Association for Standardizing Information and Communication Systems. <http://www.ecma.ch/>
- [78] ECMAScript Language Specification, 3rd edition, ECMA Standard ECMA-262, December 1999. <ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf>
- [79] Eisenberg, A. and Melton, J. SQL:1999, formerly known as SQL3. *SIGMOD Record*, Vol. 28(1), March, 1999.
- [80] Eisenberg, A. and Melton, J. SQLJ Part 0, Now Known as SQL/OLB (Object-Language Bindings). *SIGMOD Record*, Vol. 27(4), December, 1998.
- [81] Eisenberg, A. and Melton, J. SQL Standardization: The Next Steps. *SIGMOD Record*, Vol. 29(1), March, 2000, 63-7.
- [82] Elmarsi, R. and Larson, J., A. A Graphical Query Facility for ER Databases. *Proc. 4th Int. Conf. Entity-Relationship Approach*, Chicago, October, 1985, 236-45.
- [83] Elson, L., Allen, M., Goldsmith, J., Orton, M. and Weibel, W. An Example of a Network-Based Approach to Data Access, Visualization, Interactive Analysis and Distribution, Draft Paper, Jet Propulsion Laboratory, Caltech, 1999. <http://webwinds.jpl.nasa.gov/papers/paper1.pdf>
- [84] Eng, J. *GSQL Database Gateway*, National Center for Supercomputer Applications, University of Illinois, Urbana-Champaign, 1994.
<http://hoohoo.ncsa.uiuc.edu/SDG/People/jason/pub/gsql/starthere.html>
- [85] Ensor, D., Stevenson, I. *Oracle 8 Design Tips*, O'Reilly & Associates Inc., 1997, 115pp.
- [86] EntireX Homepage, Software AG.

<http://www.softwareag.com/entirex/Default.htm>

- [87] European Research Community on Flow Turbulence and Combustion (ERCOFTAC) fluid dynamics database. <http://fluindigo.mech.surrey.ac.uk>
- [88] Excelon Corporation Homepage. <http://www.odi.com/>
- [89] Extensible Stylesheet Language (XSL) Specification, W3C Working Draft, 21 Apr 1999. <http://www.w3.org/TR/WD-xsl>
- [90] FastCGI: A High-Performance Web Server Interface, Technical White Paper, Open Market, Inc., April, 1996.
<http://www.fastcgi.com/fcgi-devkit-2.1/doc/fastcgi-whitepaper/fastcgi.htm>
- [91] Fawcett, N. Microsoft Ditches its Java Tool, *Computing*, 2 December, 1999.
<http://www.vnunet.com/News/104030>
- [92] Ferreira, R., Kurc, T., Beynon, M., Chang, C., Sussman, A. and Saltz, J. Object-relational Queries into Multidimensional Databases with the Active Data Repository. *International Journal of Supercomputer Applications and High Performance Computing*, 1999.
- [93] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, T. Hypertext Transfer Protocol - HTTP/1.1, IETF RFC 2616, June 1999. <http://www.ietf.org/rfc/rfc2616.txt>
- [94] Flanagan, D. *JavaScript, 3rd edn*, O'Reilly & Associates Inc., 1998, 776pp.
- [95] Flynn, B. and Maier, D. Supporting Display Generation for Complex Database Objects *SIGMOD Record*, 21(1), March 1992.
- [96] Fogg, D. Lessons from a "Living in a Database" Graphical Query Interface. *Proc. ACM SIGMOD Conf.*, June 1984.
- [97] Fortier, P., J. *SQL 3: Implementing the SQL Foundation Standard*, McGraw-Hill Book Company, 1999, 414pp.
- [98] Freed, N. and Borenstein, N. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, IETF RFC 2045, November, 1996. <http://www.ietf.org/rfc/rfc2045.txt>

- [99] Gaffney, J. Illustra's Web DataBlade Module. *SIGMOD Record*, Vol. 25(1), March, 1996.
- [100] GBIS Home Page, University of Tennessee at Knoxville.
<http://www.netlib.org/parkbench/gbis/html/>
- [101] Geist, A., Beguelin, A., Dongarra, J., J., Manchek, R., Jiang, W. and Sunderam, V. PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing, MIT Press, 1994, 176pp.
http://www.epm.ornl.gov/pvm/pvm_home.html
- [102] Geospatial Imagery Access Services (GIAS) specification, version 3.2, N0101-C, U.S. National Imagery and Mapping Association, 28 July, 1998.
<http://www.nima.mil/aig/>
- [103] Glushko, R., J., Tenerbaum, J., M. and Meltzer, B. An XML Framework for Agent-based E-commerce. *Communications of the ACM*, Vol.42(3), 1999.
- [104] Goldman, K., J., Kanellakis, P., C. and Zdonik, S., B. ISIS: Interface for a Semantic Information System. *Proceedings ACM SIGMOD Conference*, 1985, 328-42.
- [105] Gosling, J., Joy, W. and Steele, G., L. The Java Language Specification, Addison-Wesley, 1996, 825pp. <http://java.sun.com/doc/>
- [106] Gulutzan, P. and Pelzer, T. *SQL-99 Complete, Really*, R&D Publications, 1999, 1078pp.
- [107] Haber, E., Ioannidis, Y. and Livny, M. Foundations of Visual Metaphors for Schema Display. *Journal of Intelligent Information Systems*, 3(3/4), July, 1994.
- [108] Haber, E., Ioannidis, Y. and Livny, M. OPOSSUM: Desk-Top Schema Management through Customizable Visualization. *Proceedings of the 21st Very Large Database Conference (VLDB)*, September, 1995.
- [109] Hadjiefthymiades, S., P. and Martakos, D., I. A Generic Framework for the Deployment of Structured Databases on the World Wide Web. *Fifth International World Wide Web Conference*, Paris, France, May 6-10, 1996.

- [110] Hamilton, G. JavaBeans Specification, Version 1.01, Sun Microsystems, Inc., July, 1997. <http://www.javasoft.com/beans/index.html>
- [111] Haw D., Goble, C., A., and Rector, A., L. GUIDANCE: Making it easy for the user to be an expert. *Proc. 2nd Int. workshop on User Interfaces to Databases*, Ambleside, UK, 13-15th July, 1994, 19-44.
- [112] Hawick, K., A. and Coddington, P., D. Interfacing to Distributed Active Data Archives, *Journal on Future Generation Computer Systems*, to appear.
- [113] Hawick, K., A. and James, H., A. Distributed High-Performance Computation for Remote Sensing. *Proc. Of Supercomputing '97*, San Jose, USA, November, 1997.
- [114] Hawick, K., A. and James, H., A. Eric: A User and Applications Interface to a Distributed Satellite Data Repository. Technical Report DHPC-008, Computer Science Department, University of Adelaide, 1997.
- [115] The Hierarchical Data Format (HDF) Homepage. <http://hdf.ncsa.uiuc.edu>
- [116] Hockney, R., W. A Framework for Benchmark Performance Analysis. *Supercomputer 48*, Vol. IX-2, March, 1992, 9-22.
- [117] Hockney, R., W. *The Science of Computer Benchmarking*, Philadelphia, USA, SIAM, 1996, 104-115. ISBN 0-89871-363-3.
- [118] Hockney, R., W. and Berry, M., eds. Public International Benchmarks for Parallel Computers, PARKBENCH Committee: Report Number 1. *Scientific Programming*, 3(2), 1994, 101-46.
- [119] Hockney, R., W. and Jessope, C., R. *Parallel Computers 2.*, Bristol and Philadelphia, Adam-Hilger/IOP Publishing, 1988.
- [120] Hogan, M. Defining a Scalable Database for XML, *Documentation 98 West*, Santa Clara, USA, 10 March, 1998.
<http://www.poet.com/about/presentations/presentations.html>
- [121] Horstmann, C., S. and Cornell, G. *Core Java 2: Volume 1 – Fundamentals*, Sun Microsystems Press, 1999, 742pp.

- [122] HTML 4.0 Specification, W3C Recommendation, 24 April, 1998.
<http://www.w3.org/TR/REC-html40>
- [123] Hudson, R., L. UMass Information Navigator. *IDEA Conference*, New Orleans, USA, October 17-20, 1994. <http://home.oit.umass.edu/>
- [124] Hunter, J. and Crawford, W. *Java Servlet Programming*, O'Reilly & Associates, 1998, 510pp.
- [125] IETF TLS Working Group Home Page.
<http://www.ietf.org/html.charters/tls-charter.html>
- [126] Impromptu Web Query, COGNOS, Inc.
<http://www.cognos.com>
- [127] Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), ISO 8879:1996.
<http://www.iso.ch/cate/d16387.html>
- [128] Informix Dynamic Server, Informix Corporation.
<http://www.informix.com/informix/products/ids/overview.htm>
- [129] ISAPI Reference, Microsoft Corporation, 1999.
<http://msdn.microsoft.com/library/psdk/iisref/isre9llx.htm>
- [130] The ISO/IEC JTC1/SC21/WG3 document repository.
<ftp://jerry.ece.umassd.edu/>
- [131] James, H., A. and Hawick, K., A. A Web-based Interface for On-Demand Processing of Satellite Imagery Archives. *Proc. Of the Australian Computer Science Conference (ACSC) '98*, Perth, Australia, February, 1988.
- [132] Java 2 Platform, Standard Edition, v1.2.2 API Specification, Sun Microsystems, Inc. <http://java.sun.com/products/jdk/1.2/docs/api/index.html>
- [133] Java Remote Method Invocation (RMI) Specification, Sun Microsystems.
<http://java.sun.com/products/jdk/rmi/>

- [134] Java Server Pages, White paper, Sun Microsystems, Inc., June, 1999.
<http://www.javasoft.com/products/jsp/whitepaper.html>
- [135] Java Technology Standardization, Sun Microsystems Inc., June, 1999.
<http://java.sun.com/aboutJava/standardization/index.html>
- [136] JDBC Data Access API: Drivers, Sun Microsystems Inc., July, 1999.
<http://java.sun.com/products/jdbc/drivers.html>
- [137] JDBC Guide: Getting Started, Sun Microsystems, Inc., March 6, 1997.
<http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/getstart/introTOC.doc.html>
- [138] Keyes, J. *Datacasting: How to Stream Databases Over the Internet*, McGraw-Hill, 1998, 551pp.
- [139] Kim, W. Object-Oriented Database Systems: Promises, Reality, and Future. *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993, 676-87.
- [140] King, R. and Melville, S. Ski: A Semantics-Knowledgeable Interface. *Proc. 10th Int. Conf. Very Large Databases*, Singapore, Aug., 1984, 30-33.
- [141] King, R. and Novak, M. Building Reusable Data Representations with FaceKit. *SIGMOD Record*, 21(1), March 1992.
- [142] King, R. and Novak, M. FaceKit: A Database Interface Design Toolkit. *Proc. 15th Int. Conf. Very Large Databases*, Amsterdam, The Netherlands, August, 1989, 115-23.
- [143] Kuntz, M. and Melchert, R. Pasta-3's Graphical Query Language: Direct manipulation, Cooperative Queries, Full Expressive Power. *Proc. 15th Int. Conf. Very Large Databases*, Amsterdam, The Netherlands, August, 1989, 97-105.
- [144] Lazar, D. Microsoft Strategy for Universal Data Access, Microsoft Corporation, October, 1997 (Updated August 15, 1998).
<http://www.microsoft.com/data/udastra.htm>
- [145] Lindholm, T. and Yellin, F. *The Java Virtual Machine Specification, 2nd edn*, Addison-Wesley Pub Co., 1999, 496pp. <http://java.sun.com/docs/>

- [146] Long, J., Spencer, P. and Springmeyer, R. SimTracker – Using the Web to Track Computer Simulation Results. *Int. Conf. on Web-Based Modeling and Simulation*, San Francisco, USA, 1999. (Proceedings available in: *Simulation Series*, Vol. 31(3), from the Society for Computer Simulation.)
- [147] Manber, U. Future Directions and Research Problems in the World Wide Web. *Proceedings of the Fifteenth ACM SIGAT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Montreal, Canada, June 3-5, 1996, 213-15.
- [148] Matena, V. and Hapner, M. Enterprise JavaBeans Specification, Version 1.1, Sun Microsystems, Inc., December, 1999.
<http://www.javasoft.com/products/ejb/>
- [149] McGrath, R., E. A Scientific Data Server: The Conceptual Design. White Paper, National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign, January, 1997.
http://hdf.ncsa.uiuc.edu/horizon/DataServer/sds_design.html
- [150] Melton, J. *Accommodating SQL3 and ODMG*, ANSI X3H2-95-161/DBL:YOW-32, 15 April 1995.
<ftp://jerry.ece.umassd.edu/isowg3/dbl/YOWdocs/yow032.pdf.gz>
- [151] Microsoft Transaction Server (MTS), Microsoft Corporation.
<http://www.microsoft.com/com/tech/MTS.asp>
- [152] Moore, W., R., Baru, C., Marciano, R., Rajasekar, A. and Wan, M. Data-Intensive Computing. *In: Foster, I. and Kesselman, C., eds. The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, California, Morgan Kaufmann Publishers, Inc., 1999, 105-129.
- [153] Motro, A. BAROQUE: A Browser for Relational Databases. *ACM Transactions on Office Information Systems*, Vol. 4(2), April, 1986, 164-81.
- [154] Motro, A. Browsing in a loosely structured database. *Proc. ACM SIGMOD Conf. on Management of Data*, Boston, June, 1984, 197-207.

- [155] Motro, A. VAGUE: A User Interface to Relational Databases that Permits Vague Queries. *ACM Transactions on Office Information Systems*, 6, 1988, 187-214.
- [156] Motro, A., D'Atri, A. and Tarantino, L. The Design of KIVIEW: An Object-Oriented Browser. *Proc. 2nd Int. Conf. on Expert Data Systems*, April, 1988.
- [157] MPI: A Message Passing Interface Standard, Version 1.1, Message Passing Interface Forum, University of Tennessee, Knoxville, Tennessee, June, 1995.
<http://www.mpi-forum.org/docs/docs.html>
- [158] MPI-2: Extensions to the Message-Passing Interface, Message Passing Interface Forum, University of Tennessee, Knoxville, Tennessee, 18 July, 1997.
<http://www.mpi-forum.org/docs/docs.html>
- [159] Multipurpose Internet Mail Extensions (MIME).
<http://www.oac.uci.edu/indiv/ehood/MIME>
- [160] Namespaces in XML, W3C Recommendation, 14 January, 1999.
<http://www.w3.org/TR/REC-xml-names>
- [161] National Committee for Information Technology Standards (NCITS) Standards Store. <http://www.cssinfo.com/ncits.html>
- [162] Net.Data Homepage, IBM Corporation.
<http://www.software.ibm.com/data/net.data/>
- [163] Newsome, M., Pancake, C., Hanus, J. and Moore, L. HyperSQL User's Guide and Language Reference, Technical Report, Department of Computer Science, Oregon State University, February 1996.
- [164] Nguyen, T. and Srinivasan, V. Accessing Relational Databases from the World Wide Web. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 4-6, 1996, 529-40.
- [165] Norman, M. and Thanisch, P. Much Ado About Shared-Nothing. *SIGMOD Record*, Vol. 25(3), September 1996.

- [166] Norman, M. and Thanisch, P. *Parallel Database Technology: An Evaluation and Comparison of Scalable Systems*, The Bloor Research Group, UK, 1995. ISBN 1-874160-17-1. <http://www.bloor.co.uk/>
- [167] *NSAPI Programmer's Guide*, Netscape Communications Corporation, December, 1997.
<http://developer.netscape.com/docs/manuals/enterprise/nsapi/index.htm>
- [168] The Object Data Management Group. <http://www.odmg.org/>
- [169] Objectivity/DB, Objectivity Corporation. <http://www.objectivity.com>
- [170] The Object Management Group. <http://www.omg.org/>
- [171] Ocean Circulation and Climate Advanced Modelling (OCCAM) Project, The Southampton Oceanography Centre in collaboration with the Universities of East Anglia and Exeter. <http://www.soc.soton.ac.uk/JRD/OCCAM/welcome.html>
- [172] *ODBC 2.0 Programmer's Reference and SDK Guide*, ISBN 1-55615-658-8.
- [173] OLE DB, Microsoft Universal Data Access Web Site, Microsoft Corporation. <http://www.microsoft.com/data/oledb/>
- [174] Oracle8, Oracle Corporation. <http://www.oracle.com>
- [175] Orfali, R. and Harkey, D. *Client/Server Programming with JAVA and CORBA, 2nd edn*, John Wiley & Sons, Inc., 1998, 1022pp.
- [176] Organisation for the Advancement of Structured Information (OASIS).
<http://www.oasis-open.org>
- [177] The OSF Distributed Computing Environment, Open Software Foundation, 1996. <http://www.osf.org/dce/>
- [178] Paepcke, A., Baldonado, M., Chang, C-C., K., Cousins, S. and Garcia-Molina, H. *Building the InfoBus: A Review of Technical Choices in the Stanford Digital Library Project*. Working Paper SIDL-WP-1998-0096, Stanford Digital Library Project, Stanford University, 1998.
<http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1998-0096>

- [179] Papiani, M., Dunlop, A., N. and Hey, A.J.G. Automatically Generating World-Wide Web Interfaces to Relational Databases, British Computer Society Seminar Series on New Directions in Systems Development: Intranets - The Corporate Superhighway, University of Wolverhampton, 23 April, 1997.
<http://www.ecs.soton.ac.uk/~mp>
- [180] Papiani, M., Dunlop, A., N. and Hey, A., J., G. Automatic Web Interfaces and Browsing for Object-Relational Databases. *Advances in Databases: Proceedings of the 15th British National Conference on Databases, BNCOD15*, London, 7-9th July, 1997, 131-2.
(Lecture Notes in Computer Science, Vol. 1271, Springer-Verlag, 1997.)
- [181] Papiani, M., Hey, A.J.G. and Hockney, R.W. The Graphical Benchmark Information Service. *Scientific Programming*, Vol. 4(4), 1995, 219-227.
- [182] Papiani, M., Wason, J., L., Dunlop, A., N. and Nicole, D., A. A Distributed Scientific Data Archive Using the Web, XML and SQL/MED. *ACM SIGMOD Record*, Vol. 28(3), September, 1999, 56-62.
- [183] Papiani, M., Wason, J., L. and Nicole, D., A. An Architecture for Management of Large, Distributed, Scientific Data Using SQL/MED and XML. *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, Konstanz, Germany, March, 2000, 447-61.
(Lecture Notes in Computer Science, Vol. 1777, Springer-Verlag, 2000.)
- [184] Persistent Client State HTTP Cookies, Netscape Communications Corporation, 1996. http://www.netscape.com/newsref/std/cookie_spec.html
- [185] Phipps, S. IBM, e-business, & XML, Transcript of talk presented at *XML'98*, Chicago, Illinois, November, 1998.
<http://www-4.ibm.com/software/developer/speakers/hiphps/papers.html>
- [186] Pisoni, A. Popular Perl Complaints and Myths, Version 1.04. 5 August, 1999. http://perl.apache.org/perl_myth.html

- [187] Plasil, F. and Stal, M. An Architectural View of Distributed Objects and Components in CORBA, Java RMI and COM/DCOM, Software Concepts and Tools, Vol. 19(1), Springer, 1998.
- <http://nenya.ms.mff.cuni.cz/thegroup/COMP/FPLUSOBR.PS.gz>
- [188] POET Object Server Suite, POET Software Corporation.
- <http://www.poet.com>
- [189] Powers, S. *Developing ASP Components*, O'Reilly & Associates, Inc., April, 1999, 490pp.
- [190] Query Designer: Graphical Query-Building Tool, NACSE - Northwest Alliance for Computational Science and Engineering, 1996.
- <http://www.nacse.org/mgd/qd/>
- [191] Rahm, E. Parallel Query Processing in Shared Disk Database Systems. *Proc. 5th Int. Workshop on High Performance Transaction Systems (HPTS-5)*, Asilomar, September, 1993.
- [192] Raj, G., S. A Detailed Comparison of CORBA, DCOM and Java/RMI, September 21, 1998. <http://www.execpc.com/~gopalan/misc/compare.html>
- [193] Raj, G., S. A Detailed Comparison of Enterprise JavaBeans (EJB) and The Microsoft Transaction Server (MTS) Models, May, 1999.
- <http://members.tripod.com/gsraj/misc/ejbmts/ejbmtscomp.html>
- [194] RAPS (Real Applications on Parallel Systems) Online Benchmark Information System. <http://www.pallas.de/raps.html>
- [195] Rescorla, E. and Schiffman, A. The Secure HyperText Transfer Protocol, IETF RFC 2660, August, 1990. <http://www.ietf.org/rfc/rfc2660.txt>
- [196] Rogers, T. and Cattell, R. Entity-relationship Database User Interfaces. *In: Stonebraker, M. ed. Readings in Database Systems*, Morgan Kaufman, 1988.
- [197] Rowe, L. and Shoens, K. FADS - A Forms Application Development System. *Proceedings of the ACM SIGMOD Conference on Management of Data*, June 1982.

- [198] Saini, S. and Bailey, D., H., NAS Parallel Benchmark (Version 1.0) Results 11-96, Technical Report NAS-96-18, NASA Ames Research Center, Moffett Field, CA 94035-1000, USA, November 1996.
<http://www.nas.nasa.gov/Software/NPB/>
- [199] Sandham, N.D. and Howard, R.J.A. Direct Simulation of Turbulence Using Massively Parallel Computers. *In: A. Ecer et al., eds. Parallel Computational Fluid Dynamics '97*, Elsevier, 1997.
- [200] Saphir, W., Woo, A. and Yarrow, M. The NAS Parallel Benchmarks 2.1 Results, Technical Report NAS-96-010, NASA Ames Research Center, Moffett Field, CA 94035-1000, USA, August, 1996.
<http://www.nas.nasa.gov/Software/NPB/>
- [201] Scientific Data Format Information FAQ, 13 Oct 1995.
<http://www.cv.nrao.edu/fits/traffic/scidataformats/faq.html>
<ftp://rtfm.mit.edu/pub/usenet/news.answers/sci-data-formats>
- [202] Shannon, W. Java2 Platform Enterprise Edition Specification, v1.2, Sun Microsystems, Inc., December, 1999. <http://java.sun.com/j2ee/>
- [203] The Simple API for XML, SAX 2.0, May, 2000.
<http://www.megginson.com/SAX/sax.html>
- [204] Simple Object Access Protocol (SOAP) Specification, May, 2000.
<http://msdn.microsoft.com/xml/general/soapspec.asp>
- [205] Sprenger, P. Relational DBMSes Trail Objects in XML Race, Planet IT, 20 July, 1998.
<http://planetit.com/techcenters/docs/database/technology/PIT19980911S0058/1>
- [206] SQL Server 7, Microsoft Corporation. <http://www.microsoft.com/sql/>
- [207] SQL Standards Home Page. JCC Consulting, Inc.
http://www.jcc.com/SQLPages/jccs_sql.htm
- [208] SQLJ Home Page. <http://www.sqlj.org>

- [209] Stefen, J., L. Adding Run-Time Checking to the Portable C Compiler. *Software Practice and Experience*, Vol. 22(4), April, 1992, 305-16.
- [210] Stonebraker, M. The Case for Shared Nothing, *IEEE Database Engineering*, Vol. 9(1), 1986, 4-9.
- [211] Stonebraker, M. and Brown, P. Object-Relational DBMSs, Tracking the Next Great Wave. Morgan-Kaufman Publishers, Inc., 1998.
- [212] Stonebraker, M. and Kalash, J. TIMBER: A Sophisticated Relation Browser. *Proceedings of the 8th Int. Conf. on Very Large Databases*, September, 1982, 1-10.
- [213] Sun Microsystems Withdraws Java 2 Platform Submission from ECMA, Sun Microsystems Inc., 7 December, 1999.
<http://java.sun.com/pr/1999/12/pr991207-08.html>
- [214] Sybase Adaptive Server, Sybase, Inc.
<http://www.sybase.com/products/databaseservers/>
- [215] Tallman, O. and Kain, J., B. COM versus CORBA: A Decision Framework. *Distributed Computing*, September, 1998.
http://www.quoininc.com/quoininc/COM_CORBA.html
- [216] Transaction Server Overview, Microsoft Corporation, September 7, 1998.
http://www.microsoft.com/ntserver/appservice/exec/overview/trans_overview.asp
- [217] Treinish, L., A. Interactive Archives for Scientific Data, *Informatics and Telematics*, Vol. 11(4), November, 1994.
- [218] Treinish, L., A. Interactive Archives for Scientific Data, 1999.
<http://www.research.ibm.com/people/l/lloyd/IA/home.htm>
- [219] UK Turbulence Consortium Web Site, University of Southampton, 1999.
<http://www.hpcc.ecs.soton.ac.uk/~turbulence/>

- [220] Ullman, J., D. and Widom, J. A First Course in Database Systems, Prentice-Hall Inc., 1997, 470pp.
- [221] United Kingdom Education and Research Networking Association (UKERNA) Homepage. <http://www.ja.net>
- [222] Universal Data Access Web Site, Microsoft Corporation.
<http://www.microsoft.com/data/>
- [223] Valduriez, P. Parallel Database Systems - Open Problems and New Issues. *Distributed and Parallel Databases*, Vol. 1(2), 1993, 137-65.
- [224] Valduriez, P. Parallel Database Systems: The Case for Shared-Something. *IEEE 9th International Conference on Data Engineering*, Vienna, Austria, April 19-23, 1993.
- [225] VisiBroker Homepage, Corel Corporation (formerly Borland/Inprise).
<http://www.inprise.com/visibroker/>
- [226] Wahbe, R., Lucco, S., Anderson, T. and Graham, S., L. Efficient software-based fault isolation, *Operating Systems Review*, Vol.27(5), December, 1993, 203-16.
- [227] Waldo, J., Wyant, G., Wollrath, A. and Kendall, S. A Note on Distributed Computing, Sun Microsystems Laboratories, Inc., November 1994.
http://www.sunlabs.com/techrep/1994/abstract_29.html
- [228] Warren, M., S., Germann, T., C., Lomdahl, P., S., David M. Beazley, D., M. and Salmon, J., K. Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for \$150k. Gordon Bell Price/Performance Prize, Supercomputing 1998.
<http://cnls.lanl.gov/avalon/>
- [229] Web Winds Interactive Data System, Jet Propulsion Laboratory, Caltech, 1999. <http://webwinds.jpl.nasa.gov/>
- [230] web.sql, Sybase Corporation.
<http://www.sybase.com/products/internet/websql/>
- [231] Weiss, M., Johnson, A. and Kiniry, J. Security Features of Java and HotJava, Open Software Foundation Research Institute, 1996.

- <http://www.peru.edu/~mccaslin/java/security.htm>
- [232] White, S. and Hapner, M. *JDBC 2.0 API*. Sun Microsystems Inc., Version 1.0, June, 1998. <http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/index.html>
- [233] White, S. and Hapner, M. *JDBC 2.0 Standard Extension API*. Sun Microsystems, Inc., Version 1.0, December, 1998.
<http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/index.html>
- [234] Williams, R. and Sears, B. A High-Performance Active Digital Library. In: Herzberger, O. and Sloot, P., M., A. eds. *Proceedings of HPCN98*, Lecture Notes in Computer Science, Springer-Verlag, 1998.
- [235] Williams, R., Bunn, J., Reagan, M., and Pool, C., T. Workshop on Interfaces to Scientific Data Archives, California, USA, 25-27 March, 1998, *Technical Report CACR-160*, CALTECH, 42pp. <http://www.cacr.caltech.edu/isda>
- [236] Windows Distributed interNet Application (DNA) Architecture, Microsoft Corporation. <http://www.microsoft.com/dna/>
- [237] Wong, H., K., T. and Kuo, I. GUIDE: Graphical User Interface for Database Exploration. *Proceedings of the 8th Int. Conf. on Very Large Databases*, September, 1982, 22-32.
- [238] The World Wide Web Consortium (W3C) Home Page. <http://www.w3.org/>
- [239] XML-RPC Specification, XML-RPC.com, October, 1999.
<http://www.xml-rpc.com>
- [240] XML Schema Part 1: Structures, W3C Working Draft, 7 April, 2000.
<http://www.w3.org/TR/xmlschema-1/>
- [241] XML Schema Part 2: Datatypes, W3C Working Draft, 7 April, 2000.
<http://www.w3.org/TR/xmlschema-2/>
- [242] XML.org. <http://xml.org>
- [243] Yaeger, N. A Web Based Scientific Data Access Service: The Central Component of a Lightweight Data Archive, National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

<http://hdf.ncsa.uiuc.edu/apps/ogis/isprpaper.html>

<http://hdf.ncsa.uiuc.edu/horizon/>

- [244] Zloof M.M. Query By Example. *American Federation of Information Processing (AFIPS) Conference Proceedings*, Vol. 44, National Computer Conference, 1975, 431-8.