A data exchange approach to integrating autonomous manufacturing databases

This item was submitted to Loughborough University’s Institutional Repository by the/an author.

Additional Information:


Metadata Record: [https://dspace.lboro.ac.uk/2134/27164](https://dspace.lboro.ac.uk/2134/27164)

Publisher: © Feng Zheng

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 2.5 Generic (CC BY-NC-ND 2.5) licence. Full details of this licence are available at: [http://creativecommons.org/licenses/by-nc-nd/2.5/](http://creativecommons.org/licenses/by-nc-nd/2.5/)

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
<table>
<thead>
<tr>
<th>Vol. No.</th>
<th>Class Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Loan copy</td>
</tr>
<tr>
<td>1 2</td>
<td>Jun 2000</td>
</tr>
<tr>
<td>27</td>
<td>Jun 1997</td>
</tr>
<tr>
<td>2 6</td>
<td>Jun 1996</td>
</tr>
<tr>
<td>2 5</td>
<td>Jun 1999</td>
</tr>
<tr>
<td>1 4</td>
<td>Jan 2000</td>
</tr>
</tbody>
</table>
A Data Exchange Approach to Integrating Autonomous Manufacturing Databases

by

Feng Zheng

A Doctoral Thesis
Submitted in partial fulfilment of the requirements
for the award of
Doctor of Philosophy of Loughborough University

September 1996

© by Feng Zheng 1996
For my mother

In memory of my father
ACKNOWLEDGEMENT

I wish to express my sincere thanks to my supervisor, Professor R. H. Weston, for his supervision, encouragement and extraordinary patience.

I am grateful to the staff of the MSI Research Institute for their generous assistance and helpful discussions. In particular, I would like to thank Mr. Paul Clements for making time to read the thesis and kindly offering his opinions. I would also like to thank Mr. Jak D. Gascoigne, Mr. Ian A. Coutts, Mr. I. Shaun Murgatroyd, Mr. Paul J. Gilders, and previous members of the MSI Research Institute Mr. Mike Leech, Dr. Ah Soon Goh, and Dr. Binglu Zhang. My thanks are also made to Mrs. M. E. Carden for her generous help and her kindness.

I would like to thank Mr. Chris Sumpter and Mr. Rick Standish at D2D for providing useful information for the case study of the research.

The financial support from the MSI Research Institute and the ORS (Overseas Research Student) Award granted by the CVCP (Committee of Vice-Chancellors and Principals of the Universities of the United Kingdom) are gratefully acknowledged.

My thanks are also made to all my friends at Loughborough, in China, and in other parts of the world for their friendship and support throughout the period of this study.

Finally, I would like to thank my sister for her love and support, and Yuzhao for his love and encouragement.
SYNOPSIS

It has become evident in recent years that due to the complexity and diversity of computer applications in manufacturing environments, not only will previously established 'islands of automation' continue to exist, but new islands will emerge as a result of system expansion and technical renovation. Therefore, it is vitally important that systems integration methods are capable of supporting pre-existing manufacturing application systems as well as incremental system growth.

The objective of this research is to devise a systematic method which supports the integration of manufacturing information systems with an emphasis on preserving local system autonomy. The approach proposed by this research is called a 'global data exchange' (GDE) system. The function of the GDE system is to support information sharing between heterogeneous and autonomous manufacturing databases. The approach is characterised by a combination of an off-line data exchange strategy usually adopted by file data exchange methods, with a distributed database access capability commonly found in multidatabase management systems.

The research is centred on the main issues involved in software realisation of the proposed GDE system. The research activities include a software system design process, a prototype implementation process, and a case study in which the prototype system is applied to support information sharing requirements in an industrial multidatabase environment.

The main contributions of the research are:

- The research identified major system requirements for realising the global data exchange approach and consequently suggested a software architecture for a GDE system.
- A working prototype of a GDE system was developed.
- The prototype GDE system was tested in a pseudo industrial case study.
- Based on the understanding gained from the study, the capability of the global data exchange system was evaluated with respect to other information systems integration methods, and further improvements were suggested.
CONTENTS

Acknowledgement
Synopsis
Contents
List of Figures

Chapter 1 Introduction ................................................................................... 1

Chapter 2 Literature Survey ........................................................................... 4

2.1 Introduction ........................................................................................................... 4
2.2 Computer Applications in Manufacturing Enterprises .......................................... 4
2.3 Enterprise Information System Environments ....................................................... 6
  2.3.1 Enterprise Information ................................................................................ 6
  2.3.2 Electronic Information Storage ................................................................... 7
  2.3.3 Types of Database System ........................................................................... 8
  2.3.4 Information Systems Environments .......................................................... 11
2.4 IT Approaches to Integrating Computer Information Systems ............................ 12
  2.4.1 Computer Interconnection ......................................................................... 13
  2.4.2 Computer Interoperation ........................................................................... 14
  2.4.3 Distributed Database Management Methods ............................................ 16
  2.4.4 Standard Data Format methods ................................................................. 19
2.5 Summary .............................................................................................................. 20

Chapter 3 Scope of the Research .................................................................. 21

3.1 Research Requirements .................................................................................. 21
Chapter 4  System Design ................................................................. 28
4.1 Introduction .................................................................................. 28
4.2 Understanding the Problem Area ................................................... 28
4.2.1 The Environment ...................................................................... 28
4.2.2 Shared Data ............................................................................. 29
4.2.3 Data Representation Heterogeneity .......................................... 30
   4.2.3.1 Database Model Heterogeneity ........................................... 30
   4.2.3.2 Database Schema Heterogeneity ....................................... 32
4.2.4 Data Sharing Relationships ....................................................... 33
   4.2.4.1 Source Data Objects and Target Data Objects .................... 33
   4.2.4.2 Master Copy Assumption ................................................. 33
   4.2.4.3 Data Sharing Situations .................................................. 34
4.3 System Design ............................................................................... 35
   4.3.1 Initial Design Decisions ......................................................... 36
   4.3.2 Top Level Decomposition ....................................................... 38
      4.3.2.1 The User Interface ......................................................... 39
      4.3.2.2 The Common Database Access Interface ....................... 41
      4.3.2.3 Interrelationships between the Functional Components ..... 42
4.3.3 The Exchange Service Module and its Information Requirements .... 43
4.3.4 Requests for Exchange ............................................................. 48
4.3.5 Conceptual Representation of the Meta Data ................................ 49
4.3.6 The Software Architecture ...................................................... 49
4.4 Summary ....................................................................................... 50

Chapter 5  A Prototype GDE System .................................................. 52
5.1 Introduction ................................................................................... 52
5.2 Selection of a Common Data Model ............................................... 53
5.3 Software Support for a Common Database Access Interface ........... 55
5.4 An Overview of the Prototype GDE System ............................................. 57
5.5 The Meta Data .......................................................................................... 59
  5.5.1 A Virtual Database Environment ...................................................... 59
  5.5.2 Shared Relations .............................................................................. 61
  5.5.3 Meta Data for Source Data Retrieval ................................................. 63
  5.5.4 Meta Data for Target Data Update ..................................................... 64
  5.5.5 A Conceptual Model of the Meta Data .............................................. 69
  5.5.6 Schema of the Data Dictionary .......................................................... 71
5.6 The User Interface .................................................................................. 75
5.7 The Exchange Service Module ................................................................. 76
5.8 Summary ................................................................................................. 79

Chapter 6 A Case Study .............................................................................. 81
6.1 Introduction ............................................................................................. 81
6.2 An Industrial Multidatabase Environment .............................................. 81
  6.2.1 Computer Application Systems at D2D ............................................ 82
  6.2.2 Data Sharing Relationships Between the D2D Databases .............. 84
6.3 Building an Experimental Environment ................................................. 86
  6.3.1 The Database Systems ..................................................................... 87
  6.3.2 The Database Schemas .................................................................... 87
6.4 The GDE System Set Up ....................................................................... 90
  6.4.1 System Configuration ....................................................................... 90
  6.4.2 Data Analysis .................................................................................. 92
  6.4.3 Populating the Data Dictionary ......................................................... 96
  6.4.4 User Support .................................................................................. 97
6.5 Supporting Systems Evolution ................................................................. 99
6.6 Summary ................................................................................................. 103

Chapter 7 Discussion .................................................................................. 105
7.1 Introduction ............................................................................................ 105
7.2 The Global Data Exchange Approach versus Other Approaches .......... 105
  7.2.1 Comparison with Multidatabase Management Systems .................. 105
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2 Comparison with Other Data Exchange Methods</td>
<td>107</td>
</tr>
<tr>
<td>7.3 The Data-Driven Exchange Services</td>
<td>109</td>
</tr>
<tr>
<td>7.4 The Common Database Access Interface</td>
<td>109</td>
</tr>
<tr>
<td>7.5 The User Support</td>
<td>111</td>
</tr>
<tr>
<td>Chapter 8 Conclusions</td>
<td>112</td>
</tr>
<tr>
<td>8.1 Summary of the Research</td>
<td>112</td>
</tr>
<tr>
<td>8.2 Conclusions</td>
<td>113</td>
</tr>
<tr>
<td>8.3 Future Work</td>
<td>115</td>
</tr>
<tr>
<td>8.3 Conclusion</td>
<td>116</td>
</tr>
<tr>
<td>References</td>
<td>117</td>
</tr>
<tr>
<td>Appendix A The BOM Schemas in D2D Databases</td>
<td>128</td>
</tr>
<tr>
<td>A.1 Introduction</td>
<td>128</td>
</tr>
<tr>
<td>A.2 The BOM Schema in ENDB</td>
<td>130</td>
</tr>
<tr>
<td>A.3 The BOM Schema in OMAC</td>
<td>131</td>
</tr>
<tr>
<td>A.4 The BOM Schema in SPEAR</td>
<td>134</td>
</tr>
<tr>
<td>A.5 The BOM Schema in EM3</td>
<td>136</td>
</tr>
<tr>
<td>Appendix B The Data Dictionary: the Schema Definitions and the Meta Data</td>
<td>138</td>
</tr>
<tr>
<td>B.1 The DD Schema Definitions</td>
<td>138</td>
</tr>
<tr>
<td>B.1.1 SQL Statements for DD Relational Table Creation</td>
<td>138</td>
</tr>
<tr>
<td>B.1.2 SQL Statements for DD View Creation</td>
<td>139</td>
</tr>
<tr>
<td>B.2 A Populated Data Dictionary for the Experimental Environment</td>
<td>140</td>
</tr>
<tr>
<td>B.2.1 A View of the Relation-Database Relationships</td>
<td>140</td>
</tr>
<tr>
<td>B.2.2 A View of the Source Relations</td>
<td>141</td>
</tr>
<tr>
<td>B.2.3 A View of the Target Relations</td>
<td>141</td>
</tr>
<tr>
<td>B.2.4 A View of the Source-Target Relationships</td>
<td>142</td>
</tr>
<tr>
<td>B.2.5 A View of the Database-Relation-Attribute Relationships</td>
<td>142</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 3.1 Application migration required by a multidatabase systems .................. 24
Figure 3.2 The concept of a GDE system ................................................................ 26
Figure 4.1 An environment concerned by the system design .................................. 29
Figure 4.2 An example of database model heterogeneity ........................................ 31
Figure 4.3 Possible Data Sharing Requirements ..................................................... 35
Figure 4.4 Initial decomposition of a GDE system .................................................. 39
Figure 4.5 Functionality of the common database access interface .......................... 41
Figure 4.6 The interrelationships between the three functional components ............ 42
Figure 4.7 A basic data exchange situation ........................................................... 44
Figure 4.8 Software functions required for A→B exchange .................................... 44
Figure 4.9 The exchange service module and its information support ....................... 48
Figure 4.10 The software architecture of a GDE system ....................................... 50
Figure 5.1 A functional view of CIM-BIOSYS .................................................... 57
Figure 5.2 An overview of the prototype GDE system .......................................... 58
Figure 5.3 A virtual database environment ............................................................ 60
Figure 5.4 The meta data model represented using EXPRESS-G ............................. 72
Figure 5.5 The meta data model represented in the EXPRESS language .................. 73
Figure 5.6 Relational tables in the data dictionary ................................................ 75
Figure 5.7 A conceptual representation to the internal functions of the exchange service module ........................................................................................................... 77
Figure 5.8 Logical steps involved in querying the data dictionary ............................. 78
Figure 6.1 Computer application systems at D2D ................................................ 83
Figure 6.2 Data duplications in the D2D databases ................................................ 86
Figure 6.3 Data exchange relationships between the databases .............................. 86
Figure 6.4 Database products and information stored in the databases ................... 88
Figure 6.5 Database schema definitions ............................................................... 89
Figure 6.6 Physical setup of the experimental environment .................................... 91
List of Figures

Figure 6.7  Update requirements ................................................................. 96
Figure 6.8  User inputs and corresponding exchange operations ......... 99
Figure 6.9  A situation of systems evolution ........................................... 101
Chapter 1

INTRODUCTION

Many manufacturing companies have deployed multiple stand-alone computer application systems to support a wide range of product realisation activities such as product design, engineering, production planning and control, etc. These systems have been successful in improving the performance of individual tasks that they support. However, as the responsiveness to dynamically changing markets becomes increasingly important for companies to retain their competitive advantages, the heterogeneous and autonomous nature of such systems has created barriers which constrain inter-systems coordination and cooperation. This problem has been widely recognised within the manufacturing community [De Meyer and Ferdows, 1985] [Hunt, 1989] [Warnecke and Steinhilper, 1989] [Deasley, 1994] [Underwood, 1994]. Subjects which address the problem, such as 'computer integrated manufacturing (CIM)' and 'enterprise integration', have attracted considerable research attention in recent years; and the underlying issues have been explored from many different perspectives [Scholz-Reiter, 1992] [Senehi et al, 1992] [Weston, 1993] [Petrie, 1992].

The issue that this research has chosen to investigate is the integration of manufacturing information systems. In particular, this research is interested in those application systems which use databases. This is based on the observation that there is a growing demand for sharing data across database boundaries as the use of computer databases to support manufacturing activities is increasing [Hsiao and Kamel, 1989] [Ram, 1991]. Since most databases are individually designed to support individual applications, enabling data sharing becomes a challenge due to the
heterogeneity involved in computer operating systems, database models, database management systems, and database schemas.

Currently most companies cope with the problem by using point-to-point interface programs to support off-line data exchange between application systems. Such an approach would lead to the use of large numbers of interface programs when the number of systems to be connected increases. In addition, the software reusability of these programs is usually poor, which makes their maintenance difficult and costly when the application systems evolve.

The software system methodologies which aim to improve the situation can be classified into two categories. One is multidatabase management methods. The other is neutral data format methods.

A multidatabase management system typically supports on-line transactional style data sharing in a heterogeneous database environment. The nature of the approach requires relatively tight-coupling between the concerned application systems. Although such methods claim that pre-existing databases can be integrated without modification, they usually demand migration of pre-existing local applications to the global level. Multidatabase management systems particularly suit environments where on-line transactional data sharing is critically important, such as banking systems or air ticket reservation systems. So far they have not been widely adopted in manufacturing domains.

Neutral data format methods are most commonly used for supporting off-line data exchange between different file systems. The idea is to reduce the number of interface programs when the number of file systems is large. In the manufacturing industry neutral data format methods have been successfully used in the exchange of product design data between different CAD systems. However, neutral formats are usually designed for efficient file data exchange rather than for database data exchange.
This research has decided to explore a database biased solution with an emphasis on the support of pre-existing autonomous manufacturing application systems, and of future application systems which are autonomous.

The importance of preserving the autonomy of application systems while seeking further systems integration has been recognised in recent years [Brodie and Ceri, 1992] [Hsu et al, 1994] [Norrie et al, 1995]. It is based on the perception that manufacturing applications are extremely complex and very diverse in nature, and that the technical support required by different applications varies considerably. In addition, past investments, local performance, and reliability are also important factors that manufacturing companies have to take into account when considering systems integration. It has been anticipated that autonomous application development will continue to be the preference in manufacturing companies for the foreseeable future even though it introduces difficulties for integration [Stark, 1992].

The approach that this research proposes is a compromise between multidatabase management methods and data exchange methods. It combines a uniform database access capability which is usually found in multidatabase management systems, with an off-line data exchange service. The software system developed with this principle is referred to as a Global Data Exchange (GDE) system.

The research work carried out in this study includes:

- A literature survey on computer information management in manufacturing enterprises, and on information technology methods used to support information systems integration;
- A system design of the proposed GDE system;
- A prototype implementation of such a design;
- A case study which demonstrates the use of a GDE system in supporting a range of data sharing situations modelled from a genuine industrial environment;
- A discussion of the merits and limitations of the GDE approach.
Chapter 2

LITERATURE SURVEY

2.1 Introduction

This chapter reviews the major progress in computer applications and computer information management in manufacturing enterprises, and the state-of-the-art information technology methods for integrating heterogeneous and autonomous information systems.

2.2 Computer Applications in Manufacturing Enterprises

Advances in information technology (IT) over the last four decades have brought about significant changes to manufacturing enterprises. The deployment of computers in companies has been increasing and the range of activities that are supported by computers has been expanding—from physical production, to business administration, engineering, and production management. Many of today’s companies simply could not function effectively without computers [Martin and Powell, 1992].

Introducing IT into manufacturing environments inevitably involves huge investment. However, it is believed that such investment is justified since it can bring about the following competitive advantages to companies [Deasley, 1986] [Meredith, 1989] [Vail, 1988]:

4
• Speed of response both to market changes and to changing customer demands.
• Quality of the product.
• Consistency of both product performance and on-time delivery.
• Flexibility for enhanced variety, innovation and customisation.

Research shows that CAD (Computer Aided Design) generally saves 27 per cent of the total pre-manufacturing time [Meredith, 1989] [Voss, 1989]. CAPP (Computer Aided Process Planning) can typically reduce the process planning time by almost half if it uses a CAD database [Meredith, 1989]. The use of FMS (Flexible Manufacturing Systems) generates higher return of investment, higher machine utilisation, lower labour costs, and improved lead time, product variety, quality, delivery and cost [Browne et al, 1988] [Voss, 1989]. A successful MRP II (Manufacturing Resource Planning, i.e. closed loop Material Requirement Planning [Sartori, 1988]) installation can result in a wealth of improvement throughout the firm, ranging from 15 to 50 per cent improvements in on-time delivery, inventory levels, lead times, productivity, scrap rates, etc. [Meredith, 1989] [Browne et al, 1988].

As the number of computer aided systems increased, it was recognised that a lack of coordination and cooperation between the systems became the major barrier for companies seeking further improvement to their performance. People began to realise that having an optimal solution in individual areas did not necessarily coincide with an overall optimal solution for the entire enterprise. In the early 1980s, Computer Integrated Manufacturing (CIM) concepts started to emerge, which attracted considerable attention in business as well as in academics [Scholz-Reiter, 1992].

There is no consensus on the definition of CIM. However, it has been considered to be an attempt to achieve synergical effects of company computer systems so that they can be used to greater advantage to help decision-making in all different stages of manufacturing [Hunt, 1989] [Moir, 1989]. CIM is regarded as 'one of the major manufacturing strategies for competing as a world-class manufacturing firm' [Hitomi, 1990].
Angell and Smithson [Angell and Smithson, 1991] believe that the trends of IT in the future will be:

- Increase in computing power and storage capacity;
- Increasing diversity of computers;
- Falling prices;
- Increasing functionality and flexibility;
- Increasing user-friendliness;
- Improved connectivity.

There is no doubt that in the future more manufacturing activities will be supported by IT. At the same time, as new technologies are introduced, new 'islands of automation' will appear. Stark [Stark, 1992] believes that for the foreseeable future companies will have to cope with problems resulting from the co-existence of computerised functions and non-computerised ones, and of isolated IT systems and partially integrated systems.

2.3 Enterprise Information System Environments

2.3.1 Enterprise Information

In this thesis, the terms information and data will be used interchangeably. In information systems research, the term ‘information’ is regarded as ‘knowledge derived from data’, while the term ‘data’ is ‘recorded facts and figures’ [Kroenke, 1992]. The differentiation is necessary for the design of information systems because the concern is to produce useful information out of a collection of data. However, if we take the whole enterprise into account, the border between information and data may not be absolute. As stated by Martin and Powell [Martin and Powell, 1992], ‘one manager’s information could be another manager’s data’.

With widespread deployment of computers in manufacturing enterprise, large
quantities of data originally stored in paper files and maintained manually, has now been stored in electronic forms processible by computer.

The following classification by Jorysz and Vernadat [Jorysz and Vernadat, 1990] illustrates the nature of the information generated and used within a typical manufacturing enterprise.

- **Product information** includes product drawings, parts lists, and product versions.
- **Process information** describes work plans, processes, etc.
- **Production information** specifies quantities ordered, quantities produced, operation status and work-in-progress.
- **Planning information** includes long-term, medium-term and short-term plans, inventories, make or buy decisions.
- **Resource information** describes enterprise facilities.
- **Organisation information** specifies who is responsible for what and who is reporting to whom.
- **Administrative information** includes personnel data, suppliers and customers data, orders, and other management information.

### 2.3.2 Electronic Information Storage

In the early stages of the computer data processing era, data was stored in structured data files. In a file processing environment, an application and its data are closely coupled. Each application had to have the knowledge of the physical data structure of its own data file(s) and the access technique to the file(s). Consequently, data sharing between applications was difficult, since for any application which needs to access files of other applications, knowledge of data structure and access technique of other file systems had to be built into the application code. Moreover, software maintenance was extremely complex when the format of a file was changed [Bowers, 1993] [Page, 1990].
Data files are still widely used today whenever the speed of data access is important. Most of engineering applications such as CAD are file based [Ohr, 1990] [Urban et al, 1994]. Early MRP systems were also file based, while later ones use database systems [Browne et al, 1988].

The first commercial database management systems were introduced in the late 1960s [Date, 1990] [Helman, 1994], aiming to improve data sharing between applications. A database is a collection of data files which can be accessed via a single interface, i.e. the database management system (DBMS). A DBMS is a set of mechanisms and tools for logical definition, manipulation and control of the data in a database. With the support of DBMS software, people can build a single integrated database to serve a number of applications which need to share data with each other [Gunton, 1990].

A widely quoted database architecture is the so-called ANSI/SPARC (American National Standards Institute/Standards Planning And Requirements Committee) architecture. Proposed by the ANSI/SPARC Study Group in the late 1970s, the architecture divides a database system into three levels. The internal level is the one concerned with the way data is physically stored. The external level is the one concerned with the way data is viewed by individual users. The conceptual level is a level between the other two, which is concerned with a complete description of the entire information content of the database. Consequently, there are so-called internal schema, external schema, and conceptual schema associated with the three levels respectively [Bowers, 1993] [Occardi, 1992].

2.3.3 Types of Database System

Database systems are usually classified by the data models that they support. A data model is a set of concepts that can be used to describe the structure of and operations on a database. It provides building blocks or modelling constructs with which the structure of a database can be described [Navathe, 1992].
Hierarchical and Network Databases

The most popular databases in the 1970s were hierarchical databases and network databases. A hierarchical database consists of a hierarchic arrangement of record types, known as 'trees'. The most well-known hierarchical database system is IBM’s IMS (Information Management System) for IBM’s mainframe computers. A network database organises records in the form of directed graph. Network databases are also generally known as 'CODASYL systems', because they are based on the proposals of the Conference on Data Systems Language [Silberschatz et al, 1991] [Bontempo and Saracco, 1995].

Both types of databases can be searched and accessed efficiently. However, applications built on the systems are not fully independent of the physical data structures of the databases; and as a result, they are extremely complex to build and very costly to use [Gunton, 1990].

Relational Databases

Relational database products began to be available in the early 1980s [Date, 1990]. They were based on the relational model pioneered by E. F. Codd in a series of papers in 1970-1972 [Silberschatz et al, 1991]. In a relational database, data is perceived by the user as two-dimensional tables called 'relations'. An important feature of relational databases is data independence. 'Data independence' means that applications developed to use the database are not dependent on physical data structure. This feature makes application development and maintenance much easier than that in file systems and in traditional database approaches (i.e. hierarchical and network approaches). In addition, the inherent simplicity of the relational model permitted development of powerful non-procedural\(^1\) query languages. One such

\(^1\) A non-procedural data query language allows users to specify what data they want, without giving a procedure for getting the data. Database system software will automatically navigate around to retrieve the data. By contrast, a procedural query language (such as those used by hierarchical and
language is SQL (Structure Query Language), now an ISO (International Standards Organisation) [ISO, 1992].

Relational databases have been very successful in commercial terms. They are now available on almost any hardware platform from mainframes to personal computers [Silberschatz et al, 1991]. Relational databases are widely used to support business and administration applications since they fulfil the demand for managing large amounts of simply-structured data and provide some intuitive data query and data manipulation. However, it was soon realised that the relational model was insufficient when representing complex data types. Furthermore, the mismatch of procedural programming languages used by applications and non-procedural relational query languages causes difficulties in application development [Kappel and Vieweg, 1995] [Loomis, 1994].

**Object Oriented Databases and Object Relational Databases**

Research prototypes of object oriented databases (OODBs) began to emerge in the late 1980s [Brown, 1991]. Now a number of them have become commercial products. Most of the OODBs are the extensions to OO programming languages [Loomis, 1994]. The major advantages of OODBs over relational databases lie in their rich modelling power which allows representation of complex data, the integration of data operation with conventional programming languages, and features like encapsulation and inheritance. However, so far there is not a generally agreed ‘OO data model’. With a few exceptions, most OODB products do not support non-procedural data query languages like SQL [Bontempo and Saracco, 1995].

Another new category of database system is called object relational database systems (ORDBs) [Stonebraker, 1995]. ORDBs are an extension of relational database systems with added OO features. Such systems support the SQL language, and network systems) demands tremendous speciality from users, for step by step instructions have to be written to get the data required [Date, 1990].
allow data in existing relational databases to be migrated to an ORDB.

2.3.4 Information Systems Environments

Today's enterprise information systems environments can be characterised as being distributed and heterogeneous [Cardenas, 1987] [Dwyer and Larson, 1987] [Pitoura et al, 1995] [Woelk et al, 1992].

In the early years of database technology development, databases were regarded as the means for systems integration [Ceri and Pelagatti, 1985] [Gunton, 1990]. Consequently, the concept of management information systems (MISs) was introduced [Kroenke, 1992] [Lucey, 1991]. An MIS typically used a single centralised database to store all the data required to support management activities at all levels of an enterprise. Such a system was usually extremely complex, very expensive to build and maintain, and did not always achieve designed performance [Angell and Smithson, 1991] [Martin and Powell, 1992]. It was later realised that data processing could be more efficient, flexible and manageable if the scope of any single database was narrowed down to a particular department or to a particular function [Gunton, 1990]. Today, many databases exist, especially in large companies. These databases are usually geographically distributed and used by different departments within a company.

Distribution of data and applications is generally believed to be advantageous for the following reasons [Angell and Smithson, 1991] [Ceri and Pelagatti, 1985] [Herbert et al, 1988] [Tanenbaum, 1995].

- Human organisations are distributed in nature. To locate computers close to users fits the operating and management structure of an organisation.
- Distributed systems are easier to accommodate changes in an organisation.
- Adoption of a distributed scheme allows applications to be tailored to meet diverse requirements of different functions in an enterprise.
- Distributed data processing is more reliable than centralised one, because the
failure of one system will not affect other systems.

- It is more economic to have distributed data processing because a collection of smaller computers have better price/performance ratio than a single large computer.

In fact, due to the wide diversity of manufacturing application requirements, it is almost impossible for a single vendor to offer computer support for all the applications in a company [Gunton, 1989] [Herbert et al, 1988]. After a period of incremental growth with uncontrolled purchasing policies, most manufacturing companies ended up using multiple computer information systems which run on different computer hardware and operating systems, use different database products, and are probably linked by different networks [Gray, 1993] [Stonebraker, 1989] [Vail, 1988].

While distribution and heterogeneity provide the flexibility and diversity required by manufacturing companies, they also cause substantial problems for information exchange across system boundaries. Even where the physical means of exchange (such as a network) is available, more often than not data is in the wrong format for the receiving software, or data items have been defined in a slightly different way. In order to improve the availability of data, there usually exists high and uncontrolled degree of data duplication in different systems. This presents a major challenge to global data consistency management [Kappel and Vieweg, 1995].

Bridging heterogeneous information systems and enabling data sharing between them has been a growing concern in manufacturing companies. IT methods which can contribute to the solutions are reviewed in the next section.

2.4 IT Approaches to Integrating Computer Information Systems

This section reviews advances in four areas of information technology. They are computer interconnection technology, computer interoperation technology, distributed database management technology, and standard data format technology.
2.4.1 Computer Interconnection

To enable information flow between computers, it is necessary to establish physical connection between the computers. The physical media may be coaxial cables, twisted-pair cables, or telephone lines [Adams, 1993] [Black, 1993]. Furthermore, the computers involved must agree on the form of the information that is to be exchanged. This requires some form of communication software to be installed in each computer which facilitates communication using appropriate communication protocols [Tanenbaum, 1995].

The history of data communication networks can be traced back to the late 1950s. Twenty years later, the first Local Area Network (LAN) model—Ethernet—was introduced [Shoch et al, 1985] [Cerutti, 1993a]. In the 1980s network technologies progressed at a rapid pace, fuelled by the widespread use of personal computers. Consequently there was a major increase in demand for sharing resources via the network [Lorin, 1993].

In response to the requirement, large computer vendors developed their proprietary standards to allow their own computer products to communicate with each other. Examples of such standards are IBM’s SNA and DEC’s DECnet, both introduced in the 1970s. The problems with proprietary standards are that the owners are free to change the standards at their discretion, and the standards may have been designed to provide optimal communications for the owners’ computer hardware and software systems [Stokes, 1993].

Since most organisations had installed computers from multiple vendors, there was a strong requirement for ‘open’ communication protocols. ISO’s Open Systems Interconnection (OSI) standard was defined to meet such a requirement. The OSI is a set of international standards based on a seven layer reference model [ISO, 1994a]¹

which defines the ways in which open systems should communicate, together with
the rules for implementing those standards [Black, 1993].

Because the OSI standards were designed to cover a wide range of application needs,
it is left to groups of users to define the set of choices that fits with their own
particular requirements. These are known as ‘functional standards’ or ‘standard
profiles’ [Gray, 1993]. The most publicised functional standards in the manufactur­
ing arena are Manufacturing Automation Protocol (MAP) specified by General
Motor and Technical and Office Protocol (TOP) by Boeing [Weston et al, 1987]
[Beale, 1988] [Beauchamp and Poo, 1995].

Although interconnection standards such as OSI, MAP and TOP have been endorsed
by many major vendors and were regarded as the strategic direction in the 1980s,
wide-scale implementations are still scarce [Cerutti, 1993b]. It is partly because the
standards are still evolving in some critical areas, and partly because the complexity
and the cost for full implementation of the standards are enormous [Gunton, 1990].

On the other hand, the Internet protocol suite TCP/IP (Transmission Control Proto­
col/Internet Protocol) which was developed in 1970s for the US Department of De­
fence, has consistently grown in popularity. It is mature, readily available, flexible
and extensible. It is the basis of electronic communication world-wide, and has
evolved into a de facto standard [Comer, 1991] [Coulouris et al, 1994].

Stokes [Stokes, 1993] predicts that OSI’s outlook is good for more prevalent deploy­
ment in the late 1990s. Proprietary protocols will remain, but are evolving into or
incorporating the OSI protocols in a manner that retains the current application base
and network hardware and software investment.

2.4.2 Computer Interoperation

Interconnection standards such as OSI or TCP/IP are only partial solutions for infor­
mation sharing in a distributed computing environment. In fact, applications which
use interconnection protocols will have to explicitly specify the location of the resource which they intend to access [Cerutti, 1993b]. Moreover, only simple applications such as file transfer and email are currently supported by these protocols [Hubley, 1991].

The term ‘interoperability’ has been frequently used in recent years to refer to the ability of computer systems to work together. It goes beyond interconnectivity to address transparent sharing of resources over the network [Cerutti, 1993b].

In the late 1980s, IBM announced its Systems Application Architecture (SAA). SAA defined a collection of software interfaces and protocols that can provide an environment for application interoperability and portability within IBM’s computing environments [Gray, 1993]. Other proprietary architectures include DEC’s Network Application Support (NAS), and Hewlett-Packard’s NewWave Computing Architecture (NWCA) [Snell, 1992].

The DCE (Distributed Computing Environment) by the Open Software Foundation (OSF) is a vendor independent architecture. Introduced in 1992, the DCE defined a set of network services including RPCs (Remote Procedure Calls), directory service, time service, and security service. With the support of these services, it will be cost-effective to develop robust network software that works in heterogeneous, multi-vendor environments. The DCE has been extremely well received by industry and has been regarded as the marketplace standard for distributed computing [Hurwicz, 1993] [Leikam, 1993] [Millikin, 1994].

Another well-known vendor neutral architecture is CORBA (Common Object Request Broker Architecture) by the Object Management Group (OMG). Also launched in 1992, CORBA incorporated a set of industry standards into a framework so as to promote portability, interoperability and reusability of object-oriented applications in a heterogeneous computing environment [Leikam, 1993].
2.4.3 Distributed Database Management Methods

Distributed database management technology is based on two other technologies, namely, computer network technology and database technology [Ceri and Pelagatti, 1985]. According to Date [Date, 1990], a full support for distributed databases implies that 'a single application should be able to operate transparently on data that is spread across a variety of different databases, managed by a variety of different DBMSs, running on a variety of different machines, supported by a variety of different operating systems and connected together by a variety of different communication networks'. He further explains that the term 'transparently' means that 'the application operates from a logical point of view as if the data were all managed by a single DBMS on a single machine'.

Research prototypes and commercial products of distributed database management systems started to emerge in the late 1970s [Ceri and Pelagatti, 1985]. The early systems were based on the assumption that a distributed database differs from a centralised one only through its physical implementation [Litwin and Abdellatif, 1987]. Such a system was typically designed 'top-down' and had a single conceptual schema. They did not support integration of pre-existing databases except by discarding those databases and placing their data into the new system [Bright et al, 1992] [Litwin et al, 1990].

For many companies, this approach is not feasible because replacing existing databases can be very costly [Ram, 1991]. This has led to research on another type of distributed database systems (termed 'composite systems' in [Heimbigner and McLeod, 1985]) which can combine pre-existing databases into a logically centralised system. Such a system typically maintains a global schema which is constructed by integrating the conceptual schemas of existing local databases. The process of integrating local schemas to form a global schema is called schema integration [Batini et al, 1986] [Thomas et al, 1990]. The data model used to define the global schema is called a global data model. The global schema defines all the data which are contained in the distributed database as if the database were not distrib-
uted at all—a feature known as *distribution transparency* [Özsu and Valduriez, 1991]. Users and applications access such a system by making queries using a *global data query language* [Kappel and Vieweg, 1995].

The major criticism to this type of distributed database system is that schema integration is potentially very difficult, especially when local databases are heterogeneous\(^1\) [Czejdo and Taylor, 1992] [Heimbigner and McLeod, 1985]. Moreover, rigid centralised control would be required in order to maintain the global schema, and this may not be appropriate for some environments where distributed systems are largely autonomous and loosely related [Litwin et al, 1990].

Another category of distributed database systems is the so-called *federated database system* (FDBS). The term was first coined by Hammer and McLeod [Hammer and McLeod, 1979], and has since been used by many researchers to describe several different approaches to distributed database management.

In general, the concept of *federated database system* implies a loose integration of pre-existing distributed databases. Sheth and Larson [Sheth and Larson, 1990] defines an FDBS as ‘a system which consists of component databases that are autonomous yet participate in a federation to allow partial and controlled sharing of their data’. It represents a compromise between no integration and total integration. A very loosely coupled FDBS does not support a global schema (such as those systems proposed by Heimbigner and McLeod [Heimbigner and McLeod, 1985] and

\(^1\) Traditionally, a distributed database system is regarded as 'heterogeneous' if local databases support different data models [Ceri and Pelagatti, 1985] [Heimbigner and McLeod, 1985]. In recent years, a wider range of heterogeneity is taken into account when discussing distributed database management. It includes not only data model heterogeneity, but also DBMS heterogeneity between databases using the same data model [Bright et al, 1992], semantic heterogeneity of local database schemas [Cardenas, 1987] [Litwin and Abdelatatif, 1987], and heterogeneity in respect of the computer hardware, operating systems and network protocols used by individual database systems [Sheth and Larson, 1990] [Thomas et al, 1990] [Özsu and Valduriez, 1991].
by Litwin et al [Litwin et al, 1990]). A more tightly coupled FDBS may support a single global schema which is the integration of local external schemas, or it may support multiple federated schemas each of which is a different integration of the external schemas and will be used by different users [Sheth and Larson, 1990].

Distributed database management systems which support pre-existing databases are also generally referred to as *multidatabase systems* [Litwin et al, 1990] [Bright et al, 1992]. Most of the early multidatabase systems use the relational model as the global data model and SQL as the global query language [Özsu and Valduriez, 1991]. More recent research has sought to apply object-oriented technology to the development of multidatabase systems [Czejdo and Taylor, 1992] [Pitoura et al, 1995]. Surveys of research projects and commercial products of multidatabase systems can be found in [Ceri and Pelagatti, 1985] [Thomas et al, 1990] [Pitoura et al, 1995].

The complexity involved in creating and managing a distributed database system is enormous, especially when local databases are heterogeneous, and when pre-existing databases and their applications need to be preserved. So far it is not yet possible to buy a system off the shelf which can integrate all types of databases and provide complete functionality of distributed database management [Thomas et al, 1990].

Standardisation in this area is extremely difficult. As pointed out by Elmagarmid and Pu [Elmagarmid and Pu, 1990], as long as there exist diverse application requirements, a free market of ideas and products, political decentralisation of power in companies and frequent merger and acquisition of different companies, heterogeneity will not disappear. Furthermore, the rapid rate of innovation in the computer field can make standards obsolete even before they are agreed upon. Nevertheless, the Remote Database Access (RDA) standard by ISO [ISO, 1993] is regarded as a relevant standard in this area [Thomas et al, 1990]. RDA defines representation formats for SQL requests, data and results, and diagnostic information. It currently supports the ISO OSI network environment [Date, 1995].
2.4.4 Standard Data Format methods

Data format translation is very commonly practised in manufacturing companies, which enables data to be shared between heterogeneous computer data storage systems [Hammer, 1991]. There are basically two approaches to data format translation. One is to use point-to-point translators which directly interface with the systems in question. The other is to define a neutral data format, so that the data to be exchanged will first be translated into the neutral format before it is translated into its destination format.

Direct translators can be very efficient since they are customised to the systems between which data is to be exchanged. However, as the number of systems increases, the number of translators required could be very high \((n(n-1))\) for \(n\) systems. Using an intermediate neutral format in data exchange can significantly reduce the figure especially when the number of systems is large \((2n\) for \(n\) systems) [Ohr, 1990].

Substantial effort has been made in the past twenty years to develop standard neutral formats for data exchange. IGES (Initial Graphics Exchange Specification) is an ANSI standard [ANSI, 1988a] for the exchange of mechanical engineering drawings and three dimensional models produced by CAD/CAM systems [McMahon and Browne, 1993]. Its version 1.0 was developed in 1981 [Chorafas and Legg, 1988], and the version 5.0 was released in 1990 [Stark, 1992]. EDIF (Electronic Design Interchange Format) is a standard neutral format for the exchange of electronic product design. Its version 1.00 was released by the EDIF steering Committee in 1985 [Felger, 1988], and its versions 2.00 and 3.00 became ANSI standards in 1988 and 1994 respectively [ANSI, 1988b] [ANSI, 1994]. STEP (STandard for the Exchange of Product Model Data), now an ISO standard [ISO, 1994b], has evolved from what is known as PDES (Product Data Exchange Specification) since 1984 [McMahon and Browne, 1993]. STEP is designed for the exchange of product data throughout the life cycle of a product. The standard covers not only geometric data, but also other data associated with a product such as tolerance, materials and features. It was hoped that the use of STEP would encourage wider scope integration of manufactur-
ing activities within a company and improve communications between manufacturing companies and their suppliers [Sargent, 1991] [Schenck and Wilson, 1994] [Shaw, 1990].

2.5 Summary

This chapter provided background knowledge on the features of computer information systems in manufacturing companies, and on the progress of information technology in supporting information systems integration. The understanding has provided a foundation on which this research is based.
Chapter 3

SCOPE OF THE RESEARCH

3.1 Research Requirements

As described in Chapter 2, most computerised manufacturing information is stored either in the form of computer data files, or in computer database systems. As the data sharing problems related to the two types of storage system are quite different, the author decided that the focus of this research would be on data sharing between database systems.

The deployment of database systems in manufacturing companies has been increasing in recent years owing to the falling prices of computer hardware and software as well as the improvement in user friendliness of database products [Kamel and Kamel, 1992]. On the other hand, the rise of new database technologies such as object oriented database technology provides new possibilities for using databases in areas where traditional database technology has had difficulties to support, for example, in product design and product engineering areas [Härder et al, 1987] [Spooner, 1994].

The main purpose of information systems integration efforts is to enable information sharing between different applications. The main difficulty of data sharing lies in the fact that most application systems and their databases were designed to be used in isolation. This leads to two consequences: 1) the same data which is shared between different application systems had to be duplicated in different databases for
availability; and 2) heterogeneity could be introduced in many different aspects, e.g. in operating systems, database models, database management systems, and database schemas.

By far the most common practice in manufacturing companies for resolving data sharing problems is to use proprietary point-to-point interface programs. As explained in section 2.4.4, a problem associated with such an approach is that the number of interface programs would be large as the number of databases increases. Using neutral data format methods can reduce the number of interface programs required. However, the nature of the methods is to support efficient file data exchange rather than data exchange between database systems [Sargent, 1991]. Moreover, neutral format methods are only concerned about the format of the data to be exchanged. It does not address issues such as how data should be retrieved from and updated in a database.

The most relevant methods that address issues of data sharing between database systems are the multidatabase management methods. However, so far little evidence has been found in literature of the use of multidatabase management systems in manufacturing companies. In the author's opinion, the inherent centralised nature of multidatabase management solutions presents a major obstacle to their acceptance in manufacturing domains.

Being an extension of traditional centralised database management technology, multidatabase management methods still require some degree of central control despite a current trend in multidatabase development towards preserving local system autonomy. The basic nature of multidatabase systems is to support global applications to have on-line and concurrent access to distributed databases. In order to do this, it requires a global schema to facilitate global level data access, and a global transaction management mechanism to synchronise data manipulation in local databases. However, this is very difficult to achieve when local database systems are heterogeneous, especially if they need to maintain control over their own data. It is obvious that if local applications exist, it will be almost impossible to maintain
transaction management at the global level since local transactions cannot be controlled by the global system. This is why most multidatabase management systems which are committed to preserve pre-existing application systems, do not support synchronous updates to multiple local databases [Thomas et al, 1990].

To resolve the conflicts, Sheth and Larson [Sheth and Larson, 1990] suggested that local applications be gradually migrated to become global applications (see Figure 3.1). However, from a user’s point of view, application migration may not be preferred (or even practical) for the following reasons:

- **Investment.** Considerable investment has been involved in software development and staff training for the existing application systems. The cost of system re-engineering and staff retraining may therefore be prohibitive.

- **Diversity.** Different applications may require the support of different types of database. While some applications may operate efficiently and effectively using the data model and query language adopted by the multidatabase management system, others may not.

- **Performance.** Due to the extra overhead introduced by multidatabase management functions, a global level application will generally have a poorer response time than a local one. Therefore, the run-time performance of a global application tends to be less satisfactory than that of its local version.

- **Reliability.** A global application is prone to network failure and remote computer site failure.

---

1 For example, engineering data is often complex in structure and requires complex data management capabilities (such as version control) when compared to data in other areas such as sales and marketing [Stark, 1992]. Therefore, an engineering application may prefer to use an object oriented database system, while a sales application may prefer to use a relational database system.
The above analysis shows that current IT methods have yet to provide satisfactory answers to data sharing problems in manufacturing database environments. This research will make an effort in seeking an alternative solution.

![Diagram]

(a) Original stand-alone systems

(b) The transitional state of multidatabase integration

(c) The final state of multidatabase integration

Figure 3.1 Application migration required by a multidatabase system
Chapter 3 Scope of the Research

3.2 A Proposed Solution

This research proposes an approach which combines off-line data exchange principle with a distributed database access capability. A software system based on the approach is referred to as a global data exchange (GDE) system. Figure 3.2 illustrates the concept of such an approach.

The basic task of a GDE system is to enable off-line data exchange between multiple heterogeneous and autonomous database systems. Being a data exchange based method, it will not support global level applications, and therefore there will be no need to carry out schema integration, nor will it be necessary to enforce global transaction control. The implication of this is that local system autonomy will not be sacrificed under this scheme in return for data sharing capabilities.

The approach also differs from other data exchange approaches (point-to-point or neutral format methods) in that its data exchange functions will be supported by a distributed database access platform which is responsible for establishing connections to heterogeneous database systems and providing a uniform interface for their access. Such an approach is commonly adopted by multidatabase management systems. With this capability, the GDE system will be able to directly interact with different local DBMSs without having to use an intermediate file to facilitate data exchange (which is what other data exchange methods usually do).

The fundamental idea of the approach is to overlay a minimum degree of integration onto a manufacturing information system environment so that local systems autonomy can be respected as much as possible.

The same thoughts have also been reflected in other research projects reported in literature, although the approaches suggested by these projects are different from the one proposed in this research. Hsu et al [Hsu et al, 1994] proposed a metadatabase approach which could support interactions between independent application systems according to predefined rules. Norrie et al [Norrie et al, 1995] reported a coordina-
tion repository system which provided a layer of control services on top of pre-existing application systems, coordinating tasks according to inter-system dependency information stored in a coordinate repository. None of the projects supported global level applications, global schemas and global transaction control. Instead, data sharing was achieved by enabling coordination between autonomous application systems.

![Diagram](image)

**Figure 3.2** The concept of a GDE system

### 3.3 Research Objectives

The objective of the research is to investigate the global data exchange approach. The investigation is carried out through a software development of a GDE system and a case study.

The objective of the software development is to explore major issues which are concerned by software realisation of the proposed approach. Apart from fulfilling the basic functionality of a GDE system (which is to support off-line data exchange between heterogeneous databases), this research will in particular seek to pursue an open and flexible system design which can support evolutionary change in a multi-database environment.
The objective of the case study is to demonstrate the use of the GDE system to support data exchange requirements in a multidatabase environment.

The above research activities will lead to a general assessment of the capabilities of the proposed approach, and to a discussion on the technical problems raised during the software realisation.

3.4 Research Approach

The software development of the GDE system was conducted in two steps, i.e. system design and prototype development.

- **System design.** The purpose of the system design is to identify the essential building blocks of a GDE system, and to define their interrelationships as well as interactions between the GDE system and its environment. The result of the system design was a software architecture which represents a basic understanding of the system at a high level of abstraction.

- **Prototype development.** Due to the high level of complexity involved, it was not practical within a single PhD research study to build a complete GDE system. Hence, it was decided that a prototype system should be developed. The prototype should implement major functions of a GDE system, although the scope of each function had to be limited.

The prototype GDE system was then used in the case study. In the case study, a real industrial environment was investigated, and its data exchange patterns were modelled with simplifications so that they could be handled by the prototype GDE system.
Chapter 4

SYSTEM DESIGN

4.1 Introduction

This chapter reports a system design of the proposed global data exchange (GDE) system. It starts with an explanation to the main issues which need to be considered when designing a GDE system. During the process, relevant concepts and terms are defined so that they can be used unambiguously in the rest of the thesis. The rest of the chapter explains the design decisions which lead to the definition of a software architecture for a GDE system.

4.2 Understanding the Problem Area

4.2.1 The Environment

The environment to which a GDE system will be concerned is the computing environment of a manufacturing company. In particular, the GDE system is interested in those application systems which are supported by databases. As shown in Figure 4.1, such an environment is perceived as being composed of multiple computer sites. At each site, there are application software, a local database used by the application, and a local end-user to which the application supports.

To provide data exchange services the GDE system will need to access local databases through their DBMSs as does the application software. Although the GDE
system may also need to interact with the applications or the end-users (as will be discussed in section 4.3.2.1), the major problem that the GDE system design will have to face is the linking of multiple heterogeneous and isolated\(^1\) database systems in the environment (as illustrated by the dotted box in Figure 4.1).

\[\text{(possible network connections)}\]

![Diagram](image)

Figure 4.1 An environment concerned by the system design

### 4.2.2 Shared Data

The majority of data in a multidatabase environment only has local significance, and therefore is not of concern to the study. What the study is concerned about is data which is commonly used by more than one applications, i.e. *shared data*. For example, a Bill of Materials (BOM) generated during the process of product design can be viewed as a piece of shared data because it is also required by process planning and production planning applications. Similarly, inventory information collected by a stock management application and used by a MRP system can also be regarded as a piece of shared data.

\(^1\) Although the computer sites may be linked by a network, the databases are regarded as isolated if the network system is only used to support simple data transfer and human email communications (as is often the case in most manufacturing companies).
As explained in section 3.1, one of the consequences of incremental system development is that shared data is duplicated in multiple databases. Therefore, shared data is also regarded as *duplicate data*. However, the term 'duplicate data' used in this study is different from the way in which it is used in distributed database studies [Ceri and Pelagatti, 1985] or centralised database studies [Date, 1990] where 'duplicate data' usually refers to data which has the same logical representations. In this study, 'shared data' or 'duplicate data' is defined as data which contains the same real world information yet may have different logical representations. Such a definition exemplifies the presence of data representation heterogeneity in a heterogeneous database environment, as will be explained in the next subsection.

### 4.2.3 Data Representation Heterogeneity

Heterogeneous data representations mainly result from two types of heterogeneity, namely: *database model heterogeneity* and *database schema heterogeneity*.

#### 4.2.3.1 Database Model Heterogeneity

As known from section 2.3.2, databases are classified by the data models they support. A data model determines how data is represented in a database. For example, the constructs used to describe data in relational databases are relation, attribute, tuple, domain, etc., whilst in hierarchical databases they are record type, record occurrence, root, subtree, etc. [Date, 1990].

Consequently, a BOM in a relational database will be represented as a two dimensional table (which is called a 'relation' in formal terms) which may have a table name 'BOM', and three attributes 'parent_part_no', 'child_part_no', and 'quantity' (Figure 4.2(a)). While in a hierarchical database, the BOM will be represented in two record types, a parent record type called 'part' which has a single attribute 'parent_part_no', and a child record type called 'part_structure' which has two attributes 'child_part_no' and 'quantity'. There is a link between the two types to represent the connection. Such a data structure is usually called a 'tree structure', in which the parent part type is called a 'root type', and the child part type is a 'subtree'
of the root\(^1\) (Figure 4.2(b)).

\[\text{BOM}\]

<table>
<thead>
<tr>
<th>Parent_part_no</th>
<th>Child_part_no</th>
<th>Quantity</th>
</tr>
</thead>
</table>

(a) A relational structure of BOM

\[\text{Part}\]

| Parent_part_no |

| Part_Structure |

| Child_part_no | Quantity |

(b) A hierarchical structure of BOM

Figure 4.2 An example of database model heterogeneity

The data model also determines the database query language and data management methods (such as concurrency control and recovery management) in a database\(^2\).

---

\(^1\) Here we follow the terminology used by Date [Date, 1990]. The hierarchical representation of the BOM presented here is conceptual. Its database implementation may vary according to different design requirements.

\(^2\) In fact, even databases supporting the same data model may use different query languages (e.g. both QUEL and SQL are database languages for relational databases) or different dialects of the same language (e.g. SQL has different dialects), and different control mechanisms [Sheth and Larson, 1990].
4.2.3.2 Database Schema Heterogeneity

A database schema is a description of the data stored in a database. Schema heterogeneity arises as the same data is represented differently due to different database designs. The reason behind this is that different applications may view the same data in different ways or may have different preferences on the logical representation of the data. This is also known as 'semantic heterogeneity' [Sheth and Larson, 1990] [Kapple and Vieweg, 1995].

Since schema heterogeneity is associated with database designs, it is very difficult to generalise the types of heterogeneity that might occur. However, Kim and Seo [Kim and Seo, 1991] made an effort to classify schematic conflicts in heterogeneous databases. They first assume that database schemas represented in different data models can be converted into equivalent schemas represented in the relational model. With the assumption, they based their classification on the relational model. The classification is briefly presented below in order to provide some understanding of the problem.

According to the paper, schema conflicts can be divided into three groups:

- Table (or relation) definition conflicts, which include table name conflicts (i.e. the same name for different tables, or different names for equivalent tables), missing attributes in tables, table constraint conflicts, and many-to-many table conflicts (e.g. the same information is represented with two tables in one database and with three tables in another).

- Attribute definition conflicts, which include attribute name conflicts (i.e. the same name for different attributes and different name for equivalent attributes), attribute constraint conflicts (e.g. data type conflicts, integrity constraint conflicts), and many-to-many attribute conflicts (e.g. the same information is represented with two attributes in one database and with three attributes in another).

- Table vs attribute conflict, which refers to the situation where the same informa-
tion is represented as a table in one database, and as an attribute in another.

4.2.4 Data Sharing Relationships

Two databases are regarded as interrelated if they need to share data with each other. Sometimes a database may have data sharing relationships with more than one database. The following paragraphs define the meaning of 'source data objects' and 'target data objects' as used in this study, and explains the notion of 'master copy assumption'. Then some typical data sharing situations are described.

4.2.4.1 Source Data Objects and Target Data Objects

The basic task of a data exchange process is to use a piece of data which has recently been updated in one database to update its duplicate copies in other databases, so that applications which share the same data can have a consistent view of the data. The term 'source data object'\(^1\) will be used to refer to the data which is used to update the duplicate copies in such a process, and its database will be referred to as the 'source database'. The data which is updated by the source data object will be referred to as the 'target data object', and the corresponding database(s) as the 'target database(s)'.

4.2.4.2 Master Copy Assumption

In an environment where data exchange methods are employed to support data sharing, there is a danger that duplicate data may become inconsistent as a result of uncontrolled update operations. For example, it could be possible that the source data object used in an exchange process was not the most up-to-date copy.

---

\(^1\) In the context of this study, the phrase 'data object' is used to refer to a data structure (it could be a relation in a relational database, or an object class in an OO database) and data records held by the structure. The statement 'a data object is retrieved' means that the data records in that particular data structure are retrieved.
In order to avoid such problems, this study assumes that among multiple copies of shared data, one copy must be designated as being the 'master copy'. The master copy is the only copy from which original updates should be made, and all other copies should be updated by using the master copy.

Although it may not suit some data sharing situations (in which the nature of the data determines that it has to be updated at multiple sites concurrently), the master copy principle is commonly applied in many manufacturing environments to enable consistent use of important data, ensuring that the availability of the data is properly controlled and that unauthorised updates do not occur. On the other hand, it is not unusual in manufacturing environments that a piece of shared data is generated by a single application, and is required only for read access by other applications. In these situations, the master copy assumption fits in naturally.

4.2.4.3 Data Sharing Situations

The simplest data exchange situation will involve only two databases and a single pair of source and target data objects (see Figure 4.3(a)). However, sometimes a source data object may have a number of matching target data objects in different databases (as illustrated by Figure 4.3(b)). An example of this is that BOMs may be duplicated in a design database, a process planning database and a production planning database. In this case, the master copy of BOMs will naturally be the one in the design database. The other two copies are target objects which need to be updated whenever a new product is issued for production.

Another typical situation is that a target database may need to use data from a number of source databases to update its own data (see Figure 4.3(c)). For example, an MRP database needs to have its job status records, inventory balance records, and customer order records updated by using the data from other databases before it starts to calculate material requirement for another week. A variation of this is that multiple source data objects from multiple databases may be used to update a single target object (Figure 4.3(d)). In such a situation, the source objects must be of the same data type. For example, job status information collected in different production
areas and stored in several shopfloor databases may need to be used to update job status records in an MRP database.

Figure 4.3 Possible Data Sharing Requirements

4.3 System Design

As stated in Chapter 3, a GDE system supports off-line data exchange between heterogeneous databases and directly accesses data via local DBMSs rather than via an intermediate file. The task of the system design is to further determine the behaviour of the system and subsequently to identify the basic functional modules required to build such a system. The system design should ensure that not only is the basic functionality of the system realised, but it is realised with flexibility which enables the system to be readily adapted to evolutionary change in an industrial
environment. The result of this design process will be the development of a software architecture which represents major design decisions made at this stage.

4.3.1 Initial Design Decisions

A software system with a particular functionality can be designed in many different ways if no restrictions or preferences are specified [Easteal and Davies, 1989]. In this study, the design of a GDE system was shaped by the following initial design decisions.

A Data-Driven Approach

It was decided that a data-driven approach should be adopted in the design of the GDE system. The decision was based on the observation that in a manufacturing environment, shared data and source-target relationships among established databases are relatively stable\(^1\) and usually well understood. This has led to the consideration that a GDE system could be built with generic data exchange functions driven by predefined data sharing knowledge which is specific to a particular environment.

By separating exchange functions from the knowledge of data sharing in any given environment, the following benefits can be expected:

- Such a system will be generic and therefore can be ported to different environments which have different data exchange requirements.

- Such a system will be more flexible than hard-coded direct interface programs. Less maintenance work would be required when data sharing requirements change in an environment as a result of system expansion or technical renovation, for the

\(^1\) This may not be true in other environments. For example, in an office information system environment, information sources may be changing and expanding all the time. A more relevant information sharing support for such an environment will probably be a software tool which allows people to search around for information they need.
GDE system can be adapted to the new situation by simply modifying some of the definitions of the data sharing requirements\(^1\). By contrast, for an environment where direct interface programs are used, such a change would usually result in redevelopment of the programs which are affected by the change.

In this study, the knowledge which specifies data sharing requirements in a given environment will be referred to as the *meta data* of that environment. The meta data will be stored in a data repository and be accessed at run-time by the functional part of a GDE system. The data repository will be referred to as a *data dictionary* (or DD in short) of the GDE system.

**A Common Database Access Interface**

The distributed database access capability of the GDE system will be realised by a *common database access interface* module. Such a module should be capable of linking distributed databases located at different computing sites and providing a uniform access method so that other software modules built on top of it can access databases as if the databases are local and homogeneous.

The common database interface approach is advantageous over a possible alternative—a direct local database access approach, in that it raises the level of abstraction for other functional modules which need to access multiple databases and therefore reduces the level of complexity with which the modules have to cope. Without a common database interface, these modules would have to directly access different local databases using different local query languages, and would have to be capable of converting data from any format into any other format during the process of data exchange. Obviously such an approach is inflexible and therefore was not adopted.

---

\(^1\) At this point, database heterogeneity is temporarily ignored. If database heterogeneity is taken into account, those functional modules which interface with local databases may need to be modified as well.
A common database interface approach is generally adopted by multidatabase management methods for accessing heterogeneous databases. However, the difference between a GDE system and a multidatabase system is that the higher level services of a multidatabase system are designed to support integrated database access capabilities, while those of a GDE system are designed to support off-line data exchange capabilities.

**Transparency to Users**

The author also decided that it would be preferable for the GDE system to be designed in such a way that users can use it without having a detailed knowledge about the underlying environment or of the technical means to realise exchange tasks (e.g. where to get the required data from, how to transfer the data, how to convert the data, etc.).

### 4.3.2 Top Level Decomposition

At the beginning of the system design a GDE system can be viewed as a black box whose external and internal behaviour is unknown (see Figure 4.4(a)). Having made the design decisions explained in section 4.3.1, such a system can subsequently be decomposed as shown in Figure 4.4(b).

The decomposition identifies three software building blocks, namely: a user interface module, a common database access interface module, and an exchange service module supported by a data dictionary.

The user interface and the common database access interface are required for supporting interactions between the GDE system and its environment (i.e. the user and the local databases). However, they will not be the main focus of the system design. The main focus of the design will be on the exchange service module and its data dictionary. A more detailed description of the internal function of the exchange service module and its information support will be presented in section 4.3.3. The rest of this section will be devoted to characterising the two interface modules. This
should provide an understanding of the environment with which the exchange service module works.

![Diagram of GDE system decomposition]

Figure 4.4 Initial decomposition of a GDE system

4.3.2.1 The User Interface

The role of the user interface module is to enable GDE users to interact with a GDE system in order to achieve certain exchange tasks. Such an interaction can be initially seen as a request-respond model, i.e. the user will activate the GDE system by sending a request to it, and the system will respond with messages indicating the success or failure of the request. The exact information that will be communicated between the user and the system will be described in subsection 4.3.4 after the behaviour of the exchange service module is defined.
Various options exist regarding the kind of user that the interface may support. Here we look at three possible ways in which an exchange request can be raised.

- A request may be generated by an application at a certain point in its execution. This can be viewed as an 'application-triggered' request. This type of request may be required when an exchange operation is linked with an application execution. For example, an application may always download data from its local database to other databases at the end of its execution.

- A request may be generated at certain time intervals or at a fixed time, and hence, can be called a 'time-triggered' request. For example, some exchange activities may need to be activated every hour, every day, or at a particular time of a day. This can be achieved by using a special software timer.

- A request may be generated by a human user at a time when he/she decides that an exchange operation should be done. Such a request can be called a 'human-triggered' request. This is the most casual way of raising a request. It may be preferred when the exchange operations are neither strongly related to specific events nor tied to a fixed time table.

In this context, application software, a software timer and a human user, are all users of the GDE system. This study will not pursue the mechanisms required to support the three identified users as it is not the prime focus of the study. Rather wherever a user interface is considered in the remainder of this thesis, a human user interface will be assumed.

Another issue is that the user interface, regardless of the kind of user it supports, can be designed either as a centralised interface which only accepts requests at a single computer site of a network environment, or as a distributed interface which allows multiple users to submit their requests concurrently from multiple computer sites. For the same reason mentioned in the last paragraph, this issue will not be pursued further in this study. A centralised scheme will be assumed in the rest of the thesis.
4.3.2.2 The Common Database Access Interface

This module is responsible for enabling a uniform access to heterogeneous databases as required by exchange operations. It was decided that this should be achieved by providing for the support of a common query language and a common data representation format based on a chosen common data model. With this support, the user of the interface module (which is the exchange service module) will be able to interact with heterogeneous local databases as if they are all based on the common data model. Meanwhile, the interface must be capable of communicating with individual databases so that queries expressed in the common language can be translated into languages used by local DBMSs, and data returned by the DBMSs can be converted from their local representations into the common representation. This understanding is illustrated by Figure 4.5.

![Figure 4.5 Functionality of the common database access interface](image)

It is apparent that the above functionality cannot be achieved without the support of computer interconnection and interoperation, given the fact that manufacturing databases are normally geographically distributed over heterogeneous computing platforms. How to realise computer interconnection and interoperation will not be pursued by this study. At this stage, we simply assume that the common database access interface is capable of transferring queries and resultant data between different
computer sites connected by a network.

4.3.2.3 Interrelationships between the Functional Components

According to the above understanding, interactions between the three software components can be summarised as follows (see Figure 4.6).

- The user interface accepts requests from the user and then passes the request information (the content is yet to be determined) to the exchange service module.

![Figure 4.6 The interrelationships between the three functional components](image)

- The exchange service module will carry out necessary operations and then acknowledge the user interface module of the success or failure of the exchange task demanded by the request. The acknowledgement will subsequently be passed...
Chapter 4  System Design

on to the user who initiated the request.

- The exchange service module also needs to communicate with the common database access interface module in order to perform data retrieval and data update to multiple heterogeneous databases. Such communications are achieved by the use of a common query language and a common data format.

- The common database access interface will receive queries expressed in the common query language from the exchange service module, convert them into ones expressed in local database languages, and send them to destination databases. The data returned from the databases will be translated into a common format and sent back to the exchange service module for further processing.

4.3.3 The Exchange Service Module and its Information Requirements

The exchange service module and its information requirements are the central focus of the system design. So far the exchange service module has been viewed as a black box. In order to understand the internal behaviour and the information requirements of the module, we first study the exchange requirements in a basic data sharing situation.

We assume a situation where only two databases DB1 and DB2 are involved (see Figure 4.7). A and B are data objects which contain the same real world information, yet reside separately in the two databases. We also assume that the exchange requirement in the situation is to use the data records in A to update the records in B. With the support of the common database access interface, we can ignore data model heterogeneity between the two databases. However, database schema heterogeneity will have to be taken into account in this discussion.

To fulfil the A→B exchange task, the following operations must be performed:

- Retrieving the source data A from DB1;
• Making necessary conversions if A and B have different schema definitions;
• Updating the target data B in DB2 using the retrieved source data if no conversion was made, or the converted source data.

![Figure 4.7 A basic data exchange situation](image)

The above understanding is illustrated by Figure 4.8. The author believes that the three operations identified (i.e. data retrieval, data conversion, and data update) are commonly required by all exchange tasks including those involving more than one database and/or more than one data object (such as the situations addressed in section 4.2.4.3). Therefore, it was decided that the exchange service module should be decomposed into three functional modules, namely, a data retrieval module, a data conversion module, and a data update module. These modules should perform data retrieval, data conversion, and data update required by a particular data exchange.
task according to appropriate meta data supplied to them at run-time. The following paragraphs discuss the meta data requirements related to each of the modules and the services the modules should provide. Note that details of the meta data and of the behaviour of the modules can not be defined at this stage since they are implementation dependent. Therefore, the discussions presented below will aim at providing a general understanding rather than producing precise definitions.

**The Data Retrieval Module**

To retrieve a piece of data from a database, a database query must be specified. Such a query describes what database operations should be performed on which data structures in order to get the required data. Except for the simplest cases, constructing database queries requires human intelligence. In other words, query construction can not be automated. Therefore, it was decided in this research that the basic information required by the data retrieval function would be manually defined database queries, each of which should correspond to a particular piece of source data which needed to be retrieved. Furthermore, because of the existence of the common database access interface, all the queries will be defined in a common query language against local database schemas represented in a common data model. As the queries will be generated manually, the only task that the data retrieval module needs to perform at run-time will be to search the meta data for the predefined queries for a particular exchange request, and pass them to the common database access interface module.

**The Data Update Module**

To update a target data object, appropriate operations need to be performed on the target database, which overwrite records in the target object with the source data retrieved from a source database. Since the author has no experience of the use of non-relational databases, this discussion has to be limited to relational database situations.

Updating a relational database requires the construction of SQL update commands
addressed to relational tables. It is known that updating a single relational table is relatively straightforward. The commands can be created with the use of the schema definition of the relation and of the data to be used to replace the old records in the relation. Updating a relation which is derived from multiple relations, however, is a much more complicated task. Most current relational database products do not support such updates because of integrity considerations [Date, 1995] [INGRES, 1989] [ORACLE, 1990]. This point will not be pursued here, but it will be briefly explained in section 5.5.4.

With this understanding, we define the tasks of the data update module as: 1) to generate appropriate update commands (which should be specified in the common query language), and 2) to pass the commands to the common database access interface module. The meta data needed by the module should be the schema definitions of the target objects in a concerned environment.

Apparently the above comprehension is incomplete as it is based on a special situation. Nonetheless, it did provide an approximate understanding to the function of the module and its meta data requirement.

The Data Conversion Module

The data conversion function is required when a pair of matching source and target data objects have different schema definitions. However, the generic means of supporting data conversion has not been fully explored by the author during this study. Ideally a data conversion function should automatically translate data as necessary during exchange operations according to some form of schema mapping specification. The author does not know if it is achievable. However, it is believed that the diversity of schema heterogeneity would make it difficult for such a function to cater for all possible schema differences. In addition, because of the high level of complexity involved in schema mappings, this would demand a powerful representation method.

As a matter of fact, constructing a source data query can be regarded as part of the
process of data conversion, because it is the query that derives the source data in such a way that the resultant data matches the structure of the target data. Nonetheless, this cannot solve all schema heterogeneity problems. For example, if an attribute in the derived source data has to be mapped to three attributes in the target structure, a special function will be required to carry out the conversion. Conversion requirements like this will need sophisticated support which will not be pursued in this study.

There are, however, some minor mapping issues which are commonly required by all data exchange operations. For example, it is important to make sure that the source data used for updating a target object matches the attribute orders and attribute data types of that target data structure. To do it, the function needs to know attribute definitions of both source and target data objects as well as one to one mappings between the attributes. A software function which carries out the task can also be viewed as a 'data conversion function'. It was therefore decided that the data conversion function considered in this study would be one which resolves attribute order and attribute data type differences between source and target data objects at runtime. Such a function needs to know not only attribute definitions of both source and target objects, but also the one to one correspondence between the attributes.

In summary, the exchange service module is viewed by this study as being composed of three functions which are essential to data exchange operations; they are a data retrieval function, a data conversion function, and a data update function. A further study on each of the functions revealed that generalising the three functions is by no means an easy task due to the complexity involved in source and target data mappings. Nonetheless, the above analysis does provide some understanding of the problems, and it shows that some simple generalised tasks can be achieved with the support of some predefined information. The information required to support the three functions will become the major part of the meta data. Figure 4.9 demonstrates the above understanding.
4.3.4 Requests for Exchange

The exchange service module can only carry out exchange operations when it is requested to do so. It receives requests from the user through the user interface module. To understand what information should be provided by a request, we recall the A→B exchange situation discussed in the beginning of section 4.3.3. The request for this particular case can be specified as 'to use A in DB1 to update B in DB2'. The information involved in such a request can be generalised as follows. The request includes:

- The names of the data objects which are involved in the exchange operation;
- The names of their corresponding databases;
- The source and target nature of the data objects.

![Diagram](image)

Figure 4.9 The exchange service module and its information support

Once the information is received, the exchange service module can start to search the data dictionary for the meta data which is relevant to the data objects, and then carry out exchange operations accordingly.
In fact, the information required from a request can be further reduced if the knowledge of source-target data relationships is predefined and stored as part of the meta data in the data dictionary. With such a scheme, a request can include only the name of a data object and the name of its database. It will be the job of the exchange service module to find out whether the object is source or target, as well as the matching data objects and their locations, and then carry out exchange operations as necessary.

Therefore, it was decided that the meta data used by the exchange service module should include information about source-target data relationships, and that a user request should specify only the name of a data object and the name of its database. An advantage of the scheme is that the knowledge required from the user of a GDE system is minimal, which is what was expected from the system design (refer to subsection 4.3.1).

4.3.5 Conceptual Representation of the Meta Data

Up to this point we have known that the meta data supporting the exchange service module should include source-target data relationships, source data queries specified using a common language, and schema definition of each data object. However, this understanding is very approximate. In fact, the exact content of the meta data will also depend on implementation decisions such as selection of the common data model and selection of the common query language. Therefore, a conceptual model of the meta data will be discussed in the next chapter when a prototype implementation of the GDE system is described.

4.3.6 The Software Architecture

Figure 4.10 shows a software architecture which combines the initial system decomposition with the further decomposition of the exchange service module. It also illustrates the information flows between each functional modules as it was understood after the system design.
Chapter 4 System Design

The User

A GDE System

The User Interface Module

request information

The Exchange Service Module

retrieval results
converted results

Data retrieval module
Data conversion module
Data update module

Data Dictionary

modules

data queries in a common language
retrieval results in a common format
update commands in a common language

The Common Database Access Interface Module

A Multidatabase Environment

Figure 4.10 The software architecture of a GDE system

4.4 Summary

The system design has substantiated the idea of the proposed global data exchange approach by making a series of design decisions which led to a definition of a software architecture. It should be emphasised that the resulting design is only one of the
many possible ways of realising the idea. The major characteristics of the design was the use of a common database access interface to resolve data model heterogeneity problem, and the use of a data dictionary to achieve portability and flexibility of the exchange services, as well as to support less knowledgeable users.

A main problem encountered in the design, was how to handle database schema heterogeneity. Unfortunately this issue has not been explored in depth due to a lack of understanding of the general nature of schema heterogeneity. The subject has to be left for future investigation.
5.1 Introduction

This chapter describes a prototype implementation of a GDE system based on the software architecture described in Chapter 4.

The prototype system developed in this research is the result of a particular implementation among many possible alternatives for software realisation of the system design. The process of prototype development has enabled a more detailed study to the problems involved in realising the proposed global data exchange approach.

All three major functional modules defined in the software architecture (i.e. the common database access module, the exchange service module, and the user interface module) were implemented. The implementation aimed to build a minimum working software system rather than trying to be complete and comprehensive in software terms.

The main features of the prototype system were largely determined by the initial selection of a common data model. The decision affects the implementation of two modules, namely, the common database access interface module and the exchange service module. In this particular implementation, the common database access interface module was realised by the CIM-BIOSYS—a software platform available to the author. This enabled the development work to be concentrated on the exchange
service module and the user interface module.

In this chapter, the selection of a common data model is first discussed. Then the decision to use the CIM-BIOSYS is explained, followed by a brief description of the origin of CIM-BIOSYS and its capabilities. An overview of the prototype structure is then presented. A substantial part of this chapter is devoted to a discussion about the content of the meta data. This leads to the establishment of a conceptual representation of the meta data, which is subsequently used to generate the actual schema of the data dictionary. Finally, the behaviour of the user interface module and the exchange service module is defined.

### 5.2 Selection of a Common Data Model

Choosing a common data model was the most important decision that had to be made at the beginning of the implementation. It will not only affect the way in which the common database access interface is implemented, but will directly influence the implementation of the exchange service module and the meta data.

Theoretically, any formal data model can be used as the common model. These include models which are currently implemented in database systems, such as relational model, hierarchical model, network model, and more recently, object oriented model. Other candidates are models which have not yet had database products based on them, but are usually used for data analysis and database design. This later category of data model is often referred to as 'semantic data models', e.g. entity-relationship model [Chen, 1976], functional data model [Shipman, 1981], SDM (which stands for Semantic Data Model) [Hammer and McLeod, 1981] etc.

For the prototype development, the relational model was chosen as the common data model. The reason for the decision is mainly associated with a preference to use the SQL language as the common query language for the prototype implementation. The advantages of using SQL is explained as follows.
Chapter 5  A Prototype GDE System

- SQL is the most widely used database language in industry owing to the popularity of relational databases, and it is currently an ANSI standard as well as an ISO standard.

- Its 'non-procedural' feature is specially preferred by a GDE system, for a query can be defined in a much simpler way with an SQL statement than it would be if a procedural database language is used.

- Since a query specified in a common language has to be converted at run-time into one specified in a local database language, it is important that convertibility exists from the chosen common language to other database languages. It is known from research in multidatabase management system areas that translation from a non-procedural language to a procedural language is possible, while translation in the reverse direction is an open research problem\(^1\) [Sheth and Larson, 1990]. This further justifies the selection of SQL as the common language.

The main problem resulting from using the relational model as the common model is the inability of the model to represent complex data structures. This implies that when a local schema in a non-relational model is translated into one represented in the relational model (which is required for building a virtual environment for the GDE system, see 5.5.1), some information may be lost.

Nevertheless, because of the advantages of the SQL language, the benefits of adopting the relational model outweighs those of other models. Therefore, the relational model was used as the common model for this GDE implementation.

\(^1\) This is also one of the reasons why most current multidatabase systems use the relational model as the common model and the SQL language as the global query language.
5.3 Software Support for a Common Database Access Interface

The common database access interface is a crucial component of a GDE system as it enables uniform access to heterogeneous distributed databases. Its software development requires some form of software support for computer interconnection and interoperation. Following major advances in network technology over the last decade, commercial software platforms which support computer interoperation have started to emerge. However, although such platforms can serve as useful tools for building the common database access interface, the complexity involved in this kind of software development is immense. Since the relevant technologies are still maturing, there have been very few off-the-shelf software products available in the market during the period of this research.

Therefore, the author has chosen to use the CIM-BIOSYS (CIM-Building Integrated Open SYStem) to fill the gap. CIM-BIOSYS is a software prototype of an integrating infrastructure designed to support manufacturing application environments [Gascoigne, 1992] [Coutts et al, 1992]. CIM-BIOSYS was developed by members of the System Integration group at Loughborough University, now part of the Manufacturing Systems Integration (MSI) Research Institute.

The original ideas behind CIM-BIOSYS stemmed in the mid 1980's from the perception that future manufacturing enterprises would require applications to interoperate and share information with each other [Gascoigne, 1994] [Gascoigne, 1993]. Due to the lack of general support, companies usually chose to use ‘hard integration’ methods to achieve a certain level of interoperation and information sharing. The SI group recognised that a better approach to the problem was to use an integrating infrastructure which offers services commonly required by applications with integra-

---

1 'Hard integration' refers to approaches to systems integration in which interoperation and information sharing functions are hard-coded within the application systems [Weston et al, 1991]. Although such an approach can realise required interaction and information sharing, it usually leads to high software maintenance cost when manufacturing systems are required to change.
tion needs. The integration infrastructure should provide higher level interfaces so that applications can transparently access system resources without concern about their location and heterogeneity. In addition, the integrating infrastructure should provide management facilities to make the integrated environment easier to maintain when changes occur due to technological progress or organisational transformation. Such an approach is regarded as 'soft integration' or an 'open systems approach' [Weston, 1993] [Weston et al, 1991].

The services provided by CIM-BIOSYS can be classified into three categories, namely: communication services, information services, and function services (Figure 5.1) [Weston and Edwards, 1992]. The services are invoked by applications via the service interface of CIM-BIOSYS. The configuration and run-time manager of CIM-BIOSYS is responsible for coordinating communications between CIM-BIOSYS application processes and drivers which interface with system resources (such as databases, computer terminals supporting human operators, or robots) which themselves are distributed and are supported by heterogeneous computer systems. The configuration data describes the physical location of the applications and resources as well as mappings between logical names of system resources and names and access paths of their drivers. With this support, CIM-BIOSYS applications can communicate with each other or with system resources without having to cope with distribution and heterogeneity problems.

Since the prototype development of the GDE system only requires the use of CIM-BIOSYS's information services, the following paragraphs explains in some detail the capabilities that CIM-BIOSYS information services provide.

The CIM-BIOSYS information interface accepts database queries in the SQL language, and returns query results in a common format. Each database is interfaced via a software module called a 'database driver' which is specially designed for that particular type of database. Each information service request should contain a single SQL statement which comprises a character string, and the name of the database to which the statement should be sent. When such a request is submitted to CIM-
BIOSYS, its run-time manager will automatically invoke a database driver which links to that database and passes the SQL statement to it. The query result will be translated by the driver into the common format and sent back to the original sender of the query by the CIM-BIOSYS run-time manager.

![Functional View of CIM-BIOSYS](image)

**Figure 5.1** A functional view of CIM-BIOSYS (from [Weston and Edwards, 1992])

In principle, the information services of CIM-BIOSYS support heterogeneous database access via the common interface, although in practice only three types of relational databases (Ingres, Oracle, and Progress) are currently supported. To link a non-relational database, a driver must have built-in capabilities to translate SQL queries into local queries at run-time. Linking a relational database is more straightforward, as an SQL statement can be directly sent to the DBMS for execution.

### 5.4 An Overview of the Prototype GDE System

The two decisions (i.e. to use the relational model as the common model and CIM-BIOSYS as software support for the common database access interface) provide the base for the prototype implementation. The remaining tasks involved in the prototype system development are to establish what meta information will be required as a
result of these decisions and how the exchange service module will work in such a software environment.

Before we proceed, we take a look at what the two decisions mean to the prototype implementation in terms of software configuration. Figure 5.2 is an overview of the prototype GDE system supported by CIM-BIOSYS. It shows that the exchange service module of the GDE system interfaces with CIM-BIOSYS via its information service interface, and that CIM-BIOSYS fulfils the task of the common database access interface by interfacing with heterogeneous DBMSs and providing a uniform database access interface to the GDE system.

![Diagram of the prototype GDE system]

Figure 5.2 An overview of the prototype GDE system
A relatively minor decision made for the prototype development regards the implementation of the data dictionary of the GDE system. Although the meta data can be stored either in a data file or in a database, it was decided that a relational database should be used for this particular prototype implementation. It was also decided that at run-time the data dictionary should be accessed via CIM-BIOSYS's information interface as is the case for other databases. The benefit that can be expected from using a relational database is that off-line maintenance of the meta data would be easier with the support of the interactive SQL interface of the relational database system. By having the data dictionary accessible via CIM-BIOSYS, we are at liberty to place the data dictionary anywhere on the network. Figure 5.2 illustrates this effect of this decision.

5.5 The Meta Data

Although the basic elements of the meta data required by a GDE system were specified in Chapter 4, the use of different common data models and different common query languages will affect the content of the meta data, and consequently, the design of the data dictionary.

Therefore, in this section we will first define a virtual database environment viewed by the GDE system. Then the concept of shared data and their relationships will be redefined in the context of such a virtual environment. The definitions will be used in the subsequent discussion about the meta data required by retrieval and update operations. The understanding obtained will be used to establish a conceptual model of the meta data. Finally, a design of the data dictionary based on the model will be presented.

5.5.1 A Virtual Database Environment

The common database access interface is an enabling mechanism which allows the exchange service module to access multiple databases regardless of data model heterogeneity. In other words, the exchange service module will see a virtual data-
**base environment** in which every database supports the same data model. Since we have chosen the relational model as the common data model, such an environment is therefore relational as far as this prototype system is concerned.

![Diagram of A Virtual Relational Multidatabase Environment](image.png)

**Figure 5.3** A virtual database environment

This also implies that data in any of the databases should be specified in relational schemas. It is not a problem if a local database is a relational database. However, for a database which is non-relational, its original schema has to be translated into a relational equivalence. The translation process is known as 'schema conversion' or 'schema translation'. This study is not concerned about how an equivalent relational schema is generated out of a non-relational schema. It is known from the literature that schema conversion between different data models is achievable [Hsiao and Kamel, 1989] [Sheth and Larson, 1990].

We simply assume that for each local database, there exists a relational schema which represents only shared data in that database. Such a schema will be referred to as a
"component schema" (which is a subset of the conceptual schema of a local database), and such a component schema is a description of a "component database" (which is a virtual database). Consequently, we say that the prototype GDE system works in a virtual environment which is composed of a collection of component databases, where each of the databases is relational. Such a concept is illustrated in Figure 5.3. The successive discussions will be based on such a concept.

5.5.2 Shared Relations

Shared data in a virtual database environment are represented in relations. The relations which are originally defined in a component schema will be referred to as base relations\(^1\). In contrast to base relations there are derived relations\(^2\) which are relations that can be obtained by carrying out a series of relational operations on a number of base relations.

In the virtual environment, a pair of source and target data objects should be regarded as ‘equivalent relations’. By ‘equivalent relations’ we mean that the relations will contain the same real world information yet their schema specifications may be different. We have explained that although there exist many different types of schema heterogeneity (see section 4.2.3.2), this study will only consider the most common types of schema heterogeneity, i.e. those related to attribute definition differences (see section 4.3.3). In a relational environment, two relations are regarded as equivalent if they have the same set of attributes. However, the attribute definitions of the two relations can be different in two ways. One is that the order of

\(^1\) In relational terms, the relations which are originally defined in the design of a relational database are called ‘base relations’ [Date, 1995].

\(^2\) In relational database study, a derived relation is a relation which can be defined by means of some relational expression in terms of base relations. A relational view created during a database design is a derived relation with a name, whilst a query result is a derived relation without a name. [Date, 1995].
the attributes can be different. The other is that the names and data types of the matching attributes can be different. These are explained in the following example.

Example:

The two relations shown below are from two different component databases (DB1 and DB2), yet they are regarded as equivalent relations.

DB1: order (order_no, cust_name, priority, delivery_date);
DB2: cust_order (order_id, cust_name, ship_date, priority);

We can see that the two sets of attributes have a one to one correspondence by their semantics. However, the two relations have different names, and the names and the orders of the attributes are also different. Moreover, the two relations may also originate from different data structures in their component schemas. For example, the relation ‘order’ in DB1 may be a base relation, while the relation ‘cust_order’ in DB2 may be a derived relation which is obtained by the following SQL SELECT statement operating on two base relations ‘order’ and ‘customer’ in DB2.

```
SELECT o.order_id, c.cust_name, o.ship_date, o.priority
FROM order o, customer c
WHERE o.cust_id = c.cust_id;
```

In which, the schemas of the base relations ‘order’ and ‘customer’ are as follows:

```
order (order_id, prod_id, cust_id, ship_date, priority);
customer (cust_id, cust_name, cust_address);
```

End of Example.

It is necessary to point out that attribute definition heterogeneity explained in the example has been largely simplified to avoid unnecessary involvement in details.

---

1 According to relational theory, the order of attributes does not affect the semantics of a relation [Codd, 1990].
Identifying shared data and source-target relationships in local databases will require human intelligence. Once these are identified, a component schema for each local database should be established, in which shared data should be represented in relations. Subsequently, the shared relations and their interrelationships must be specified formally. Such a specification will become part of the meta data.

In order to formally represent source-target relationships, each source or target relation must be given a name. For a shared relation which is a base relation in its component schema, the name of the relation defined in the component schema should be used as its reference. For a shared relation which is a relational view in a component schema, the name of the view defined in the component schema should be used as its reference. A shared relation could be neither a base relation nor a relational view. It could be a relation which is derived from a single base relation or from multiple base relations. Such a relation will not have a name defined in its component schema. Relations in this category are known as 'unnamed relations' [Date, 1995]. To represent such relations in the meta data, a name must be assigned to it. In the above example, the name 'cust_order' is an assigned name to the derived relation specified by the SQL SELECT statement.

A shared relation may not be uniquely identified by its name in an environment since the same relation name may exist in other component databases. Consequently, a name of a shared relation must be associated with its database name whenever the relation is to be identified.

5.5.3 Meta Data for Source Data Retrieval

To retrieve data from a source relation, an SQL SELECT statement must be generated for the relation. If a source relation is a base relation or a relational view, the data retrieval will require a statement as follows:
SELECT * FROM relation_name;

The statement retrieves all the records in the relation. It can be used to retrieve data from any base relations or relational views as long as the name of the relation (or the view) is known.

For an unnamed derived relation, however, it is impossible to retrieve data without knowing how the relation is derived from other base relations. Therefore, the essential knowledge required for data retrieval of such a relation is an SQL SELECT statement, as shown in the example in section 5.5.2.

To unify the two situations, it was decided that for each source relation (base, view or unnamed), an SQL SELECT statement should be specified, and such a statement should be part of the meta data.

5.5.4 Meta Data for Target Data Update

SQL statements for update operations are DELETE, INSERT and UPDATE. At run-time the GDE system will have to generate these statements so that appropriate update operations can be performed on target relations.

For update operations, the fact that a target relation is a base relation or a derived relation (named or unnamed) will make a great difference. In theory, if a target relation is derived from more than one base relation, update operations addressed to it need to be converted (using some kind of algorithm) into equivalent operations on the base relations from which the target relation is derived. However, such a conversion is not always possible (see discussions on this issue in [Date, 1995]). As a consequence, most relational database products do not support update to relations which are derived from multiple base relations.

Since the problem has not been fully resolved in relational database theory, we assume in this research that all the target relations are either base relations or are derived relations which are based on single base relations.
In the virtual environment, the data used for updating a target relation is a set of records retrieved from a source relation. Different semantics of source-target mappings will determine how the data will be used in update operations. In this study, four kinds of update requirements are identified, namely, appendage update, table replacement update, row replacement update, and row distinction update. The following discussion explains the four types of update requirements and the SQL commands required to support them, which will lead to an identification of the meta data which is essential for automatic generation of update commands.

1. **Appendage Update (AP)**\(^1\). This kind of update is required in cases where the target relation holds history records while the source relation hold current records. The semantics dictates that retrieved source data should be appended to the current records of the target relation, leaving the existing records intact.

This kind of update can be supported by generating a set of SQL INSERT statements, each appending a row of data into the target relation. To insert a single row to a target relation which has \(n\) attributes and has a name 'example', the statement will take the following form:

\[
\text{INSERT INTO example VALUES (a_1, a_2, \ldots, a_n);}
\]

Where \(a_1, a_2, \ldots, a_n\) are the values corresponding to each attribute of the target relation 'example'. Such a statement can be generated automatically at run-time as long as the name of the relation and the set of attribute values are available. However, there are two more points which need to be noted when generating such a statement, as specified below:

---

\(^1\) 'AP' appeared here and 'TR', 'RR', 'RD which will appear in the later paragraphs are abbreviations for 'appendage update', 'table replacement update', 'row replacement update' and 'row distinction update' respectively. These abbreviations will be used in Figure 5.4 and Figure 5.5 as well as in Chapter 6.
• According to the SQL syntax, attribute names do not need to be listed in an INSERT statement if an update is addressed to the complete attribute set of a base relation. However, the attribute values in the INSERT statement must be in the same order as the attribute order defined in the base relation.

To satisfy this requirement, the simplest thing to do is to retrieve source data in such a way that the order of the attributes matches that in the target relation. This will only require that a proper SQL SELECT statement is generated for each source relation (see section 5.5.3). Once this is done, the software mechanism responsible for update command generation can be free of concern of attribute order differences.

• An INSERT statement recognises two data types: numerical data type and character string data type. If an attribute is defined as a character string in the target relation, the value corresponding to that attribute must be quoted in the INSERT statement.

This requires that data type information for each attribute of a target relation be made available for automatic command generation.

We may conclude from the above two points that, in order to generate proper SQL statements to carry out an appendage update, the following knowledge must be available to the GDE system (this assumes that attribute order difference is resolved by source data retrieval):

a) The name of the target relation.

b) The data type of each attribute.

2. Table Replacement Update (TR). This kind of update is required when the nature of the exchange dictates that the records in the target relation be completely replaced by retrieved source data. For example, when a production planning application generates a new production plan, relevant data may need to be downloaded to
a shopfloor database to completely replace existing planning records.

In this situation, update operations should: 1) delete all existing records in the target relation, and 2) insert source relation records.

The correspondent SQL statements for 'table replacement update' on a target relation 'example' are as follows:

```
DELETE FROM example;
```

Execution of this statement should delete all the records in the relation 'example'. And then, for each row of source data:

```
INSERT INTO example VALUES (a1, a2, ..., an);
```

The command generation requirements discussed in 'appendage update' also apply to this situation.

3. **Row Replacement Update (RR).** In some situations a table replacement update may not be appropriate. For example, an MRP database may hold job status records of all the jobs which have been issued to the shopfloor. The records will be updated periodically using the records from different shopfloor databases which serve different production lines. Since the job status records in each of the shopfloor databases is a subset of the records in the MRP database, we have to make sure that each update operation using data from one of the shopfloor databases should not affect the MRP records regarding other production lines.

This can be done by comparing the source records and the target records by the values of their primary key attributes\(^1\), since each value in the key attribute is unique.

\(^1\) It is possible that more than one attribute in a single relation is defined as the 'primary key' of the relation [Date, 1990].
in a relation and is used to identify a particular record. When a key value in a source record is found in the target records, the following SQL statements should be generated:

DELETE FROM example WHERE key_attr = a;

INSERT INTO example VALUES (a1, a2, ..., an);

The first statement deletes the record whose key attribute value is \( a \), and the second statement inserts a row of source record which has the same key value.

In order to support row replacement update, the key values of the target relation should be retrieved. This may be done by either predefining key retrieval operation with an SQL SELECT statement and store it as part of the meta data, or automatically generating the retrieval command at run-time. In the latter case, the primary key definition (i.e. whether an attribute is a key attribute) of the target relation must be available to the exchange service module. It is also necessary for the GDE system to know which attribute(s) in the source relation is the primary key attribute(s), so that the right set of values from the retrieved data is used in the comparison operation. For the prototype implementation, it was decided to include primary key definitions in the meta data so that a key retrieval query can be generated automatically at run-time.

4. **Row Distinction Update (RD).** For some source-target relationships, neither appendage update nor table or row replacement update is appropriate. For example, when a new product is issued for production, its BOM must be downloaded from a design database to a production planning database. It is apparent that the BOM data of this particular product cannot be used to replace current BOM records in the production database, because production BOM records also contain BOMs for other products. Nor should the records be simply appended to the existing ones in the production BOMs, because it may result in record duplication due to possible common use of some subassemblies or components among different products. In such a situation, use of the row distinction update is required.
Like the row replacement update situation, row distinction update will compare records in source and target relations by their primary key attributes. If a key value in a source record is not found in the target records, an INSERT statement should be generated to insert this row of source record into the target relation. On the other hand, if a source key value exists in the target relation, this row of record in the target relation should be preserved and no update command needs to be generated.

In summary, the meta data that is required to support automatic update statement generation must provide the name of the target relation and the data type information for each attribute of the relation. To support the row replacement update and the row distinction update, key attribute definitions for both source and target relations have to be included in the meta data.

Due to the complexity of data semantics, more complicated update requirements may exist. Nonetheless, this prototype development will only consider the four situations discussed above.

5.5.5 A Conceptual Model of the Meta Data

The understanding provided by subsections 5.5.1 to 5.5.4 shows that the meta data of the prototype GDE system should include the following information:

- Information which can identify shared relations unambiguously within a virtual multidatabase environment.
- Information which specifies interrelationships between shared relations.
- Information which supports source data retrieval operations.
- Information which supports target data update operations.

Based on this understanding, a conceptual data model was established using EXPRESS a data modelling language with an object oriented flavour\(^1\).

\(^1\) EXPRESS was initially developed in the early eighties within the US Air Force for the writing of
The reason for building a conceptual model for the meta data is that it allows us to disregard implementation issues and concentrate on the semantic representation of the data. The model will be used to guide the design of the data dictionary of the GDE system.

The basic constructs used by EXPRESS are the ‘entity’ and the ‘attribute’. Entities are used to represent facts, physical objects, ideas, relationships etc. The attributes of an entity are characteristics which distinguish one entity from another.

To build a model using EXPRESS, the first task is to determine what entities are of interest. Once the entities are chosen, the next step is to specify the major attributes of the entities. Some attributes may be represented by simple data types such as strings or numbers, some will need to be represented by other entities (which shows relationships between entities). It is also necessary to find out if entities can be generalised or specialised (from which supertype/subtype relationships can be established).

Figure 5.4 is a model of the meta data represented in EXPRESS-G\(^1\), whereas Figure 5.5 is the same model represented using the EXPRESS language. From both figures we can see that the meta data model is composed of five entities, namely, a ‘shared_relation’ entity, a ‘source_relation’ entity, a ‘target_relation’ entity, a ‘database’ entity, and an ‘attribute’ entity.

---

\(^1\) The tool used to draw the EXPRESS-G diagram and to automatically generate the textual form of EXPRESS model from the drawing was developed by the researchers in the MSI Research Institute, Loughborough University as part of the Model-Driven CIM project [Weston et al, 1995].
The 'shared_relation' is modelled as a supertype entity with two subtype entities, i.e. 'source_relation' entity and 'target_relation' entity. The model shows that a shared relation has a name, a list of attributes, and belongs to a database. A source relation as a subtype of a shared relation, has all the properties that a shared relation does; in addition, it has a predefined retrieval command associated with it. The extra piece of information that a target relation has, is the type of update operation which should be performed on it (i.e. appendage, table replacement, row replacement, or row distinction update).

For the database entity, only the name of the database is required. The attribute entity has included three properties, i.e., the name of an attribute, the data type of an attribute, and whether or not the attribute is a key attribute.

Furthermore, the model also shows that each source relation may be associated with one or more target relations, while each target relation is only associated with one source relation. This semantics is represented by a 'one-to-many' link between the source relation entity and the target relation entity (see Figure 5.4).

5.5.6 Schema of the Data Dictionary

Guided by the conceptual model of the meta data, we can design a relational database in which the meta data will be stored and maintained.

To build such a database we need to convert the entities and the relationships between entities in the EXPRESS model into equivalent relational tables. The resultant database schema should preserve the semantics specified by the EXPRESS model. In fact there exist different ways to interpret the EXPRESS conceptual model into relational implementations. The one which will be explained below represents only one of these options.
Figure 5.4 The meta data model represented using EXPRESS-G
SCHEMA Meta_Data;
  TYPE name = STRING;
  END_TYPE;

  TYPE data_type = ENUMERATION OF (string, integer, float);
  END_TYPE;

  TYPE upd_req = ENUMERATION OF (AP, TR, RR, RD);
  END_TYPE;

  TYPE key_definition = ENUMERATION OF (y, n);
  END_TYPE;

  TYPE retrieval_cmd = LIST[1:?] OF STRING;
  END_TYPE;

ENTITY Shared_Relation
  SUPERTYPE OF (ONEOF(Source_Relation,TargeCRelation));
  called : name;
  has : LIST[1:?] OF Attribute;
  belongs_to : Database;
END_ENTITY;

ENTITY Attribute;
  called : name;
  is_of : data_type;
  with: key_definition;
END_ENTITY;

ENTITY Database;
  called : name;
END_ENTITY;

ENTITY Source_Relation SUBTYPE OF (Shared_Relation);
  is_retrieved_by : retrieval_cmd;
  match : SET[1:?] OF Target_Relation;
END_ENTITY;

ENTITY Target_Relation SUBTYPE OF (Shared_Relation);
  has : upd_req;
END_ENTITY;

END_SCHEMA;

Figure 5.5 The meta data model represented in the EXPRESS Language
Figure 5.6 shows the relational tables generated for the data dictionary. The primary key field of a table is marked by a ‘#’ sign. The main principles adopted in the design of the schema were that:

1. Corresponding to each entity in the EXPRESS model, a relation was created. The relations which fit in this category are ‘shared_rel’, ‘src_rel’, ‘tgt_rel’, ‘attr’, and ‘db’.

2. Corresponding to each relationship between entities, a relation was created. The relations ‘attr_of_rel’, ‘rel_of_db’, and ‘s_t_mapping’ are of this category.

Each relation which represents an entity in the EXPRESS model was designed to have a single primary key attribute. For example, ‘rel_id’ is the primary key attribute of relation ‘shared_relation’, and ‘src_id’ is the primary key attribute of relation ‘src_relation’, etc. The supertype-subtype semantics represented in the model (i.e. the entities ‘source_relation’ and ‘target_relation’ are subtypes of the entity ‘shared_relation’) is handled by including the primary key attribute of the supertype relation ‘shared_relation’ (i.e. ‘rel_id’) in the attribute sets of the subtype relations ‘src_relation’ and ‘tgt_relation’.

By contrast, a relation which represents a relationship between entities in the model will not have a single primary key attribute. Instead, the primary key of such a relation is composed of the primary key attributes of the other relations related by the relationship relation. For example, the relation ‘s_t_mapping’ represents the relationship between ‘source_relation’ entity and ‘target_relation’ entity in the EXPRESS model. Its primary key is a composite key which consists of two attributes: ‘src_id’ (which is the primary key of ‘src_relation’), and ‘tgt_id’ (which is the primary key of ‘tgt_relation’).

It is important that the data dictionary of a particular application environment is created and maintained by authorised users so that the correctness and consistency of meta data can be guaranteed. The meta data will not only be used by the exchange service module, but also be accessed by an authorised human system administrator to
obtain necessary knowledge about data sharing relationships within a particular multi-database environment.

shared_relation (#rel_id, rel_name);
src_relation (#src_id, rel_id, retrieval_cmd);
tgt_relation (#tgt_id, rel_id, upd_req);
s_t_mapping (#(src_id, tgt_id));
attribute (#attr_id, attr_name, attr_type, key_def);
db (#db_id, db_name);
attr_of_rel (#(#attr_id, rel_id));
rel_of_db (#(#rel_id, db_id));

Figure 5.6 Relational tables in the data dictionary

5.6 The User Interface

In section 4.3.4 it was decided that the only information that a user should specify when submitting an exchange request should be the name of a data object and the name of its database. In the context of the prototype implementation, the information required should be the name of a shared relation and its database name.

It is also possible to support requests which involve multiple relation names and their database names. Such situations could arise in some situations such as those mentioned in 4.2.4.3. The user interface of the prototype GDE system has been designed to support this requirement. In cases where multiple relation names are involved in a user request, the user interface will acknowledge the success or failure of exchange operations corresponding to each relation, so that the user could single out the relations whose exchange operations had failed, and re-initiate their exchange as necessary.
5.7 The Exchange Service Module

The exchange service module is the central software module of a GDE system. It reacts to user requests, it functions according to the knowledge it obtains from the meta data, and it interacts with the common database access interface to send queries and receive resultant data. Since we have defined the meta data, the user interface and the common database interface of the prototype GDE system, the behaviour of the exchange service module can now be determined.

Figure 5.7 is an abstract representation of the essential functions carried out by the exchange service module upon receipt of an exchange request. A complete exchange process can be broadly seen as comprising three operational steps, namely, meta data acquisition, source database query, and target database update. Each of these steps is explained in the following paragraphs. Note that the discussion is based on a request which only specifies a single relation name and its database name. For requests which involve multiple relations, the exchange service module will handle each relation individually.

1. Meta Data Acquisition. When the exchange service module receives a request, it downloads relevant meta data from the data dictionary to the memory. The first step of such a downloading is to send an SQL query to the data dictionary to find out whether the relation is a source relation or a target relation. This is necessary because it will affect subsequent query generation for downloading the rest of the meta data for this particular exchange task. The logical steps involved in the query generation process are explained in Figure 5.8.

The meta data retrieved will be accessed by different functional modules within the exchange service module throughout the process of execution, as shown in Figure 5.7.
The name of a relation and the name of its database

The Exchange Service Module

The Common Database Access Interface

Chapter 5  A Prototype GDE System

Figure 5.7 A conceptual representation to the internal functions of the exchange service module
2. **Source Data Retrieval.** Any single exchange task will involve only one source relation owing to the nature of data exchange. To retrieve the source data, the exchange service module needs to find the source data query as well as the name of the source database from the downloaded meta data. The query and the database name will then be submitted to CIM-BIOSYS via its information service interface. CIM-BIOSYS will subsequently invoke a particular database driver which interfaces with the physical database, and pass the query to the DBMS for execution. The query result returned by CIM-BIOSYS will be processed and readily prepared for update command generation.

![Table: Request information and logical steps](image)

<table>
<thead>
<tr>
<th>Request information:</th>
<th>Logical steps involved in querying the data dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>name of the relation (rel_name in the DD schema)</td>
<td>1. Generate queries to retrieve all the relevant information about the relation, which includes its attribute information and its update requirement.</td>
</tr>
<tr>
<td>name of the database (db_name in the DD schema)</td>
<td>2. Generate a query to determine the matching source relation.</td>
</tr>
<tr>
<td></td>
<td>3. Generate queries to retrieve all relevant information about the source relation, which includes its attribute information, its database name, and its retrieval command.</td>
</tr>
<tr>
<td>Start: Generate a query to find out if the relation is a source relation or a target relation.</td>
<td>If it is a target relation:</td>
</tr>
<tr>
<td></td>
<td>1. Generate queries to retrieve all the relevant information about the relation, which includes its attribute information and its update requirement.</td>
</tr>
<tr>
<td></td>
<td>2. Generate a query to determine the matching source relation.</td>
</tr>
<tr>
<td></td>
<td>3. Generate queries to retrieve all relevant information about the source relation, which includes its attribute information, its database name, and its retrieval command.</td>
</tr>
<tr>
<td></td>
<td>If it is a source relation:</td>
</tr>
<tr>
<td></td>
<td>1. Generate queries to retrieve all relevant information about the relation, which includes its attribute information and its retrieval query.</td>
</tr>
<tr>
<td></td>
<td>2. Generate a query to find out all matching target relations (as we know that one source relation may have a number of matching target relations).</td>
</tr>
<tr>
<td></td>
<td>3. Generate queries to retrieve all relevant information about each target relation, which includes its attribute information, its database name, and its update requirement.</td>
</tr>
<tr>
<td>End.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.8 Logical steps involved in querying the data dictionary

3. **Target Data Update.** A single exchange task may involve one or more target relations. For each target relation, the exchange service module must first find out its update requirement from the downloaded meta data. According to the information,
the exchange service module will choose to execute a set of procedures. Procedures for row distinction update and row replacement update are more complicated than those for appendage update and table replacement update. They will include the generation of an SQL query for retrieving the keys of the target relation and a key comparison operation. Since update command generation for different update requirement situations has been covered in section 5.5.4, it will not be repeated here.

During an exchange process, various problems may arise. For example, database retrieval or update may fail, or the network may break down. When failure occurs, the exchange service module should be able to notify the user so that the user can re-initiate the failed exchange operation later. Meanwhile, it should also acknowledge the user when an exchange operation is successful. This capability is conceptually represented in Figure 5.7 by two functional modules in the lower right corner.

5.8 Summary

The prototype development has produced a working software system which involves all the functional components defined in the software architecture specified in Chapter 4. The central part of the prototype development was the implementation of the exchange service module and its data dictionary. A detailed investigation on the essential functions of the exchange service module (i.e. data retrieval, data conversion, and data update), was conducted in the context of a virtual relational environment. This has led to the definition of a conceptual model of the meta data as well as a detailed functional decomposition of the exchange service module.

It is apparent that the implementation of the exchange service module is dependent on the selection of a common data model. The common data model determines how data should be presented to the exchange service module by the common database interface module, and how data should be queried and manipulated from the exchange service point of view. This particular prototype development has chosen to use the relational model as the common data model and SQL as the common data access language. If a different data model were chosen, the exchange service module
would have to be implemented differently.

In its current form, the prototype GDE system has the following limitations: it does not support data exchange between non-relational databases; only limited types of schema heterogeneity can be handled by this prototype system; and only limited types of update requirement are supported. These limitations have to be taken into account when conducting the case study.
Chapter 6

A CASE STUDY

6.1 Introduction

The case study presented in this chapter will illustrate how the prototype GDE system can support the exchange of data in a multidatabase environment.

The study involves four activities:

- Investigating the data sharing situation in a real industrial environment;
- Building an experimental multidatabase environment which models the industrial environment with certain simplifications;
- Setting up the prototype GDE system to support the exchange requirements in the experimental environment;
- Investigating a change in the industrial environment and the way in which an established GDE system should adapt itself to support the change.

6.2 An Industrial Multidatabase Environment

The company which provided the data for the investigation is D2D (Design to Distribution), a UK manufacturing company located at Kidsgrove, UK. D2D, once a manufacturing plant of ICL (International Computers Ltd.) enterprise, is now an autonomous manufacturing company. It now accepts a major proportion of its
customer orders from external computer companies as well as from companies within the ICL Enterprise.

Computer application systems at D2D have undergone a continuous process of incremental growth over the last decade. Today such systems support most of the product realisation activities (e.g. design engineering, process planning, manufacturing engineering, and production management and control) in the company.

6.2.1 Computer Application Systems at D2D

The computer application systems used in D2D and their main features are illustrated in Figure 6.1. The data flow between them will be discussed in detail in the next subsection.

The oldest systems at D2D are ENDB, OMAC, and QUART. They were developed in the mid 80s by 'in-house' IT system developers at ICL. QUART (QUality And Reliability Tracking) is a file based quality control system. It holds information collected from terminals in the ATE (Automatic Testing Equipment) area of the shopfloor. ENDB is an ICL proprietary system which holds ICL product structure data (i.e. bill of materials) and related information generated by design engineers at the ICL design centre located at West Gordon, UK. ENDB is built on a network database known as a ‘CODASYL system’ (which has been mentioned in Chapter 2, section 2.3.2). OMAC is basically the same kind of software system as ENDB except that it was designed to support MRP II and inventory control at the Kidsgrove plant. The system itself is located at Belfast, UK. Apart from product structure data, OMAC also contains customer orders, inventory information, job status, and so on.

SPEAR (Strategic Planning Environment for Assembly Routes) is an ICL project

---

1 The material described in this section was obtained during a number of visits to D2D Kidsgrove plant and from discussions with Mr. Chris Sumpter, a consultant engineer who currently works for ICL, and Mr. Rick Standish, an ICL PDM (Product Data Management) consultant.
started four years ago which aimed to support automatic process planning and manufacturing information generation (e.g. NC data generation, operations card generation, etc.). Unlike older generation systems (such as ENDB and OMAC), SPEAR is developed on top of an open database product---Ingres. Its database contains design definitions of printed circuit assemblies (PCAs).

![Diagram of computer application systems at D2D](image)

Figure 6.1 Computer application systems at D2D

SMS, FAST and FAMS are newer systems introduced during the last two years. The software development for these systems was 'outsourced', i.e. they were designed and implemented by software houses. SMS is a stock management system supported by an Informix database (which is a relational database) running on a SUN server. Both FAST and FAMS are cell control systems supported by separate Ingres databases. FAST is used in the surface mount assembly area, while FAMS is used in the
bareboard manufacturing area.

Recently, D2D has introduced a new system EM3 to replace the old ENDB. EM3 runs on an ICL super server and is based on an Ingres database. The role of EM3 is the same as that of ENDB. The difference is that EM3 stores not only the design data of ICL's own products, but also the design data of D2D's customer companies.

The D2D environment is a typical manufacturing information environment in which multi-level heterogeneity (computer hardware, computer software, database models and products, and database schemas) exists as a result of incremental system growth and constant technological innovation.

6.2.2 Data Sharing Relationships Between the D2D Databases

The focus of this subsection is on the data flow between the databases. Because the database schemas and data sharing relationships in the D2D environment are very complex, the author has decided not to involve the details of the D2D schema definitions in this discussion. However, Appendix A has been produced in order to provide some understanding of how complex the situation is. In Appendix A, four different schema definitions of BOMs in four D2D databases (i.e. ENDB, OMAC, SPEAR, and EM3) are presented. It shows that mappings between heterogeneous schema definitions are not always straightforward. At this point, however, schema mappings are not the concern. What is interested here, is how data sharing is currently handled in the environment.

In the D2D environment, shared data is duplicated in heterogeneous and autonomous databases. The data sharing is supported by point-to-point interface programs. Figure 6.2 shows major data duplications in the D2D databases. Data flow between these databases is illustrated in Figure 6.3. Here we divide the data sharing activities into two groups. One is product data related data sharing activities. The activities include the exchange of part lists, BOMs and supplier information between ENDB, OMAC, and SPEAR. The other is production data related data sharing activities.
They include the exchange of job order information, job status information, and inventory information between OMAC, FAST, FAMS, and SMS\textsuperscript{1}.

\textit{Product Data Exchange}

Although part lists, BOMs and supplier information are all stored in three database systems (i.e. ENDB, OMAC and SPEAR), each system has a different focus of concern on them. ENDB includes BOMs of all products which are either currently in production, or have been taken out of production, or are still at the design stage. By contrast, OMAC only contains BOMs of those products which are in production, while SPEAR is only interested in those BOMs which are related to printed circuit assemblies (PCAs) of the products that are currently being manufactured.

Data downloading needs to be performed when a new product is released from design to production. In D2D, the downloading is initiated by the users of OMAC and SPEAR via two separate direct interfaces to ENDB. An intermediate file using an ICL proprietary neutral format is generated during each exchange process to facilitate data translation between different database schema definitions.

\textit{Production Data Exchange}

The OMAC MRP system runs every weekend to calculate overall material requirements for the D2D manufacturing plant. The OMAC system also monitors inventory levels and work-in-progress (WIP) on the shopfloor. It receives updates of job status information from FAST and FAMS every ten minutes, whilst the inventory information in OMAC is updated every night from SMS. OMAC also downloads job schedules to FAST and FAMS over night. The periodic uploading and downloading operations are executed automatically by direct interface programs linking the FAST, FAMS and SMS systems to OMAC.

\textsuperscript{1} QUART is not included here because it will not be modelled in the experimental environment as it is not a database system. Furthermore, its data sharing relationships with OMAC is the same as that of FAST and FAMS.
### Shared Data vs In Databases

<table>
<thead>
<tr>
<th>Shared Data</th>
<th>In Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part lists</td>
<td>ENDB, OMAC, SPEAR</td>
</tr>
<tr>
<td>BOMs</td>
<td>ENDB, OMAC, SPEAR</td>
</tr>
<tr>
<td>Supplier information</td>
<td>ENDB, OMAC, SPEAR</td>
</tr>
<tr>
<td>Job orders</td>
<td>OMAC, FAST, FAMS</td>
</tr>
<tr>
<td>Job status</td>
<td>OMAC, FAST, FAMS</td>
</tr>
<tr>
<td>Inventory</td>
<td>OMAC, SMS</td>
</tr>
</tbody>
</table>

**Figure 6.2 Data duplications in the D2D databases**

**Figure 6.3 Data exchange relationships between the databases**

### 6.3 Building an Experimental Environment

The purpose of building an experimental environment is to allow a demonstration of the use of the prototype GDE system. In the experimental environment, six databases are established, each corresponding to a D2D database except for QUART. Each database is populated with data that can be used to demonstrate various data
exchange situations.

Despite its correspondence to the D2D environment, the experimental environment is much simpler than its counterpart due to current limitations of the prototype system. The features of the D2D environment which are incorporated into the experimental environment, are the database names, the data objects that are shared between the databases, and the source-target data relationships. What has been excluded from the experimental environment, is: the heterogeneity of the computing environment, heterogeneity of database models, complicated database schemas and consequently complex data retrieval, conversion and update problems.

Nonetheless, the experimental environment does include a limited degree of database schema heterogeneity. Furthermore, the database schemas are so designed that various types of update requirements discussed in section 5.5.4 are covered.

6.3.1 The Database Systems

Figure 6.4 lists the database products used when building each experimental database as well as the data dictionary (DD), and the data that is stored in the databases.

As both Ingres and Oracle databases are relational databases, data model heterogeneity does not exist. All the data will be represented in relational tables. As a result, schema conversion can be avoided as the data model supported by the local databases is the same as the common data model.

The reason for using only Ingres and Oracle in the experiment is that they were the only database products available to the author at the time of the study, and that both are supported by the CIM-BIOSYS integrating infrastructure.

6.3.2 The Database Schemas

Figure 6.5 is a list of database schemas defined for each of the experimental databases. The schemas were created to model data sharing relationships in the D2D
environment. However, certain simplifications are made so that the exchange requirements can be handled by the prototype GDE system. Moreover, not all the shared data listed in Figure 6.2 are included in the schemas. Part lists and supplier information are not modelled because the patterns of their exchange can be covered by the exchange of the other data objects.

<table>
<thead>
<tr>
<th>Databases</th>
<th>Products Used</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENDB</td>
<td>Oracle 6.0</td>
<td>Product BOMs</td>
</tr>
<tr>
<td>OMAC</td>
<td></td>
<td>Product BOMs, Job Information, Inventory information</td>
</tr>
<tr>
<td>SPEAR</td>
<td>Ingres 6.4</td>
<td>PCA level BOMs</td>
</tr>
<tr>
<td>FAST</td>
<td></td>
<td>Job Information</td>
</tr>
<tr>
<td>FAMS</td>
<td></td>
<td>Job Information</td>
</tr>
<tr>
<td>SMS</td>
<td>Oracle 6.0</td>
<td>Inventory information</td>
</tr>
<tr>
<td>DD</td>
<td>Ingres 6.4</td>
<td>Meta data</td>
</tr>
</tbody>
</table>

Figure 6.4 Database products and information stored in the databases

One major simplification is that each relation includes only the most basic attributes which define the semantics of the table. As can be seen from Appendix A, BOMs in a real industrial database contain many attributes which are necessary for industrial applications. Yet the BOMs defined for the experimental databases ENDB, OMAC, and SPEAR only include three essential attributes: 'parent part number', 'child part number', and 'quantity' (in the case of ENDB the fourth attribute 'type' is included, which will be used for qualifying different BOM retrieval demands).

In the schema definition, some forms of heterogeneity have been deliberately included. For example, the three BOM relations in three different databases are
defined using different attribute names. The same inventory information is defined by two relations in OMAC and SMS respectively using different relation names. Furthermore, data structures of some corresponding relations are not completely identical. For example, the ‘stock’ relation in SMS has an attribute ‘last_updated’ which does not appear in its target relation ‘inventory’ in OMAC. It shows that some information is only of local concern. The attribute ‘type’ in ENDB’s BOM relation and the ‘area_code’ in OMAC’s ‘job_orders’ relation are included for a different reason: they are required to qualify certain data retrievals when updating different target relations.

<table>
<thead>
<tr>
<th>Databases</th>
<th>Schema Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENDB</td>
<td>BOM (#(part_no[char 4], comp_part_no[char 4]), qty[int], type[char 4]);</td>
</tr>
</tbody>
</table>
| OMAC       | BOM (\#(partnumber[char 4], comp_partno[char 4]), qty_per_assy [int]);  
             | Job_orders (\#job_id[char 4], due_date[char 10], area_code[char 2]);  
             | Job_status (\#job_id[char 4], status[char 5]);  
             | Inventory (\#part_no [char 4], qty[int]); |
| SPEAR      | BOM (\#board_no[char 4], comp_part_no[char 4], qty[int]); |
| FAST       | Job_orders (\#job_id[char 4], due_date[char 10]);  
             | Job_status (\#job_id[char4], status[char 5]); |
| FAMS       | Job_orders (\#job_id[char 4], due_date[char 10]);  
             | Job_status (\#job_id[char4], status[char 5]); |
| SMS        | Stock (\#part_no[char 4], qty[int], last_updated[char 10]); |

Figure 6.5 Database schema definitions

The relational tables were created using the SQL data definition language. Here is an example of the creation of the BOM relation for OMAC.
CREATE TABLE BOM (partnumber char(4) not null,
comp_partno char(4) not null,
qty_per_assy integer);

The tables were populated with example data. In fact, the data has no significance in the case study. What will be of concern at the stage of the GDE system installation is the schemas of the databases.

6.4 The GDE System Set Up

In order to support data exchange in the experimental environment, the databases in the environment must be properly linked, and the data dictionary (DD) must be populated with appropriate meta data. The following subsections will describe how this was done in the case study.

6.4.1 System Configuration

The whole experiment involves three SUN workstations connected via Ethenet. One workstation runs the CIM-BIOSYS run-time manager, the prototype GDE system, and the database drivers. Ingres and Oracle database products are installed on the other two workstations (see Figure 6.6).

The Ingres drivers and Oracle drivers were developed as part of CIM-BIOSYS's information services by the researchers in the MSI Research Institute. As one driver is required to link to each database, in total seven drivers are used in this case study. Among these three are Oracle drivers (drivers 1 to 3) and four are Ingres drivers (drivers 4 to 7).

At run-time, the prototype system generates SQL commands and sends the commands together with the destination database names to the CIM-BIOSYS run-time manager. Upon receipt of the information, the CIM-BIOSYS run-time manager searches its configuration files for the mappings between databases and their drivers, and then passes the commands to the appropriate drivers. The drivers will then
invoke correspondent DBMSs and pass on the SQL commands for execution. The resultant data returned from a DBMS will first be converted by the database driver into a CIM-BIOSYS defined format in which every data item is represented as a character string. Then the data is transmitted in the reverse direction, i.e. from the driver to the CIM-BIOSYS run-time manager, and finally to the prototype GDE system.

Figure 6.6 Physical setup of the experimental environment

In order for the CIM-BIOSYS run-time manager to correctly manage the multidatabase access required by the prototype system, the CIM-BIOSYS's configuration files must be populated with necessary information. The exact content of the configuration information will not be discussed here, as it will involve too much detail about current CIM-BIOSYS implementation.
6.4.2 Data Analysis

The purpose of data analysis is to understand data sharing relationships in an environment so as to generate the meta data required by the prototype GDE system.

From previous discussions in this chapter, we already have the knowledge of source-target object relationships, object-database relationships, and database schemas in the experimental environment. However, we still need to know what kind of update should be performed on each target relation, what SQL command should be generated to retrieve each source relation, and what attribute information should be included in the meta data and what should not. This subsection presents an analysis for each pair of source-target relations and explains the decisions made about their meta data generation.

**Exchange of BOMs**

This case study assumes that the BOM relation in ENDB contains only the Bill of Materials of a new product which is to be released from design to production. This assumption allows us to avoid unnecessary discussions on how to separate the BOMs of a new product from those of other products (i.e. the products which are either in production, out of production, or at the design stage).

The ENDB BOM has two target objects, i.e. the OMAC BOM and the SPEAR BOM. While OMAC requires a complete downloading of the BOMs of a new product, SPEAR only needs the BOMs of PCAs. Therefore, two different SQL retrieval commands must be generated against the ENDB BOM schema.

---

1 As explained in Appendix A, the D2D SPEAR database does not have a BOM relation. The BOM data is downloaded from ENDB by using a direct interface program, and processed and populated into a number of relations. Unfortunately, the prototype GDE system currently does not support this kind of exchange requirement. Hence, the situation has been simplified and a single BOM relation has been created in the experiment database.
For updating the BOM relation in OMAC, the following SQL statement is generated to retrieve all the BOM records in the ENDB BOM relation.

```
SELECT part_no, comp_part_no, qty
FROM BOM;
```

For updating the BOM relation in SPEAR, a retrieval condition must be specified. The attribute ‘type’ in the ENDB BOM relation is designed for this purpose. The value of the attribute ‘type’ will be set to ‘PCA’ if the record is a PCA BOM, otherwise it will be set to ‘NULL’. Therefore, we can define the following SQL statement for the data retrieval:

```
SELECT part_no, comp_part_no, qty
FROM BOM
WHERE type = 'PCA';
```

Update to both BOM relations in OMAC and SPEAR are typical ‘row distinction update’. It implies that the update operations will include the retrieval of the key values from the target relations so that they can be compared with the key values of the source relation. Only those source records which are not found in the target relations need to be inserted into the target relations.

To perform such operations, the prototype system needs to know not only the attribute definitions (i.e. attribute names, data types, and key definitions) of the target BOM relations, but also those of the source BOM relation. However, we need to decide which of the attributes in these relations are relevant and should be included in the meta data.

The author has made the decision that for this case study, the attribute definitions of

---

1 In a BOM relation, the primary key is composed of two attributes, i.e. the parent part number attribute, and the child part number attribute (see Figure 6.5).
all the matching attributes in source and target relations should be included in the meta data, even though not all the definitions will be used in exchange operations. In the case of BOM data exchange, the above decision means that the definition of the attribute ‘type’ in the ENDB BOM will not be part of the meta data, because it does not match any attribute in the target BOM relations.

The attribute information is required for data type conversion and update command generation. In fact, the attribute information of source relations is not used by the prototype system, because data type conversion is not performed due to the fact that the current CIM-BIOSYS implementation does not distinguish different data types returned by DBMSs.

**Exchange of Job Orders**

The downloading of job orders from OMAC to FAST and FAMS is similar to the exchange of BOM data in that data flow is from a single source relation to multiple target relations. However, the update requirements in the two cases are different. Although this may not be the case in the D2D environment\(^1\), we assume that each time when job order data is downloaded from OMAC, the current job order records in FAST and FAMS are completely replaced. Therefore, the update required by the exchange of job orders is a ‘table replacement update’.

The source data retrieval for the two exchange situations is fairly straightforward. The attribute ‘area_code’ in the OMAC Job_orders relation can be used to separate orders for different manufacturing areas. Its value will be assigned to ‘SM’ if an order is for the surface mount assembly area, or to ‘BB’ if it is for the bareboard manufacturing area. The two SQL data retrieval statements for the two exchange situations are listed below.

---

\(^1\) Information regarding production data exchange at D2D was not fully available to the author, because the software company which designed FAST, FAMS and SMS systems did not release the database schemas.
For updating `job_orders` in FAST:

```sql
SELECT job_id, due_date
FROM job_orders
WHERE area_code = 'SM';
```

For updating `job_orders` in FAMS:

```sql
SELECT job_id, due_date
FROM job_orders
WHERE area_code = 'BB';
```

Like the BOM case, information about the attribute ‘area_code’ in the OMAC `job_orders` relation will not be included in the meta data.

**Exchange of Job Status**

The exchange of job status information demonstrates a different situation where the source data object is distributed in two different databases (job_status relation in FAST and FAMS). The main point to be made about the exchange is that the update to the OMAC `job_status` relation has to be a ‘row replacement update’. This is because each of the `job_status` relations in the two source databases contains only status records of jobs which are currently processed in each of the production areas, while the target relation in OMAC holds the status records of all the jobs which has been issued to the shopfloor.

The SQL statement for retrieving source data from both FAST and FAMS is as follows:

```sql
SELECT *
FROM job_status;
```

All the attribute definitions of `job_status` relations should be included in the meta data.
Exchange of Inventory Data

Inventory data exchange requires a 'table replacement update' to the OMAC inventory relation. The correspondent source data retrieval required from the SMS stock relation can be defined by the following SQL statement:

\[
\text{SELECT part_no, qty} \\
\text{FROM stock;}
\]

The attribute 'last_updated' in the SMS stock relation plays no role in the exchange, and therefore its definition will not be included in the meta data.

Figure 6.7 illustrates the update requirements demanded by each exchange situations described above.

<table>
<thead>
<tr>
<th>Source Relations</th>
<th>Update Requirements</th>
<th>Target Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM in ENDB</td>
<td>Row Distinction(RD) update</td>
<td>BOM in OMAC</td>
</tr>
<tr>
<td>BOM in ENDB</td>
<td>Row Distinction(RD) update</td>
<td>BOM in SPEAR</td>
</tr>
<tr>
<td>Job_orders in OMAC</td>
<td>Table Replacement(TR) update</td>
<td>Job_orders in FAST</td>
</tr>
<tr>
<td>Job_orders in OMAC</td>
<td>Table Replacement(TR) update</td>
<td>Job_orders in FAMS</td>
</tr>
<tr>
<td>Job_status in FAST</td>
<td>Row Replacement(RR) update</td>
<td>Job_status in OMAC</td>
</tr>
<tr>
<td>Job_status in FAMS</td>
<td>Row Replacement(RR) update</td>
<td>Job_status in OMAC</td>
</tr>
<tr>
<td>Stock in SMS</td>
<td>Table Replacement(TR) update</td>
<td>Inventory in OMAC</td>
</tr>
</tbody>
</table>

Figure 6.7 Update requirements

6.4.3 Populating the Data Dictionary

The meta data obtained from the above data analysis can now be populated into the data dictionary. Appendix B presents the SQL statements which have been used to
create the relational tables in the data dictionary, as well as the meta data which has
been populated to the tables. The population has been done manually by the author.
It is obvious that in an environment where large quantities of shared data objects are
involved, manually populating the data dictionary could be cumbersome. In fact, data
population could be assisted by a special user interface built on top of DBMS of the
data dictionary. However, such an interface was not developed in this study due to
time constraints.

6.4.4 User Support

As explained in section 5.6, the prototype GDE system only needs two pieces of
information from the user to start an exchange operation. One is the name of a
shared relation. The other is the name of the database in which the relation is stored.

If the relation is a target relation, there will be only one matching source relation be-
cause of the master copy assumption made in section 4.2.4.2. In such a situation, the
prototype system will automatically update the target relation using the data retrieved
from the source relation. If the relation is a source relation, the prototype system will
find out the matching target relations and automatically update them all. The system
also allows the user to specify multiple relation names and their corresponding data-
base names, in which case each relation will be handled individually. There is no re-
striction on whether an exchange should be initiated from a source site or a target
site.

Figure 6.8 shows some possible user inputs for the exchange of different data in the
experimental environment and the operations that the GDE system performs in re-
sponse to each input. Each of these operations was tested on the established GDE
system and it proved to work satisfactorily.

Situations 1 and 2 listed in Figure 6.8 are about the exchange of BOM data. The
BOM sharing relationship is a typical ‘one source to many targets’ relationship.
Therefore, when the source relation name and its database name is given as the input,
all the matching target relations will be updated automatically by the GDE system (situation 1). The user may also choose to update each of the target relations separately (which is the practice adopted in the D2D environment). This can be done by specifying one of the target relation names (in this case, both target relation names are identical) and the correspondent database name (situation 2).

Situations 3 and 4 are the same as 1 and 2 except that the data to be exchanged are job_orders between OMAC as the source, and FAST and FAMS as the targets.

Situations 5 and 6 are about the exchange of job_status data. Unlike the exchange relationships of the BOM and job_orders, the source data of job_status is distributed in two source databases (FAST and FAMS), while the target data is in a single database (OMAC). Situation 5 shows that if the GDE system is given the target relation name and its database name, it will automatically update the target relation using the data retrieved from both source databases. However, it is also possible to update part of the target relation with data from one of the source databases. To do this, the user needs to specify one of the source relation names and its database name, as shown in situation 6.

Situation 7 illustrates an input which involves multiple relation names and database names. In this case all the relations are source relations, and their matching target relations are in a single database (OMAC). The same exchange activities can be achieved if the user specifies the target relation names and their database name (situation 8).

Of course, the eight situations have not covered all exchange possibilities. Nevertheless, they demonstrated that the GDE system can effectively insulate users from details about application systems and exchange operations. In addition, the input scheme is sufficiently flexible to satisfy different user requirements, such as to initiate a request from a source site or a target site, to specify a single relation name or multiple relation names, etc.
Chapter 6  A Case Study

User Input | Exchange Operations
--- | ---
1 | BOM, ENDB
Update BOMs in both OMAC and SPEAR with data from ENDB.
2 | BOM, OMAC (or SPEAR)
Update BOM in OMAC (or SPEAR) with data from ENDB.
3 | Job_orders, OMAC
Update Job_orders in both FAST and FAMS with data from OMAC.
4 | Job_orders, FAST (or FAMS)
Update Job_orders in FAST (or FAMS) with data from OMAC.
5 | Job_status, OMAC
Update Job_status in OMAC with data from both FAST and FAMS.
6 | Job_status, FAST (or FAMS)
Update Job_status in OMAC with data from FAST (or FAMS).
7 | Stock, SMS; Job_status, FAST; Job_status, FAMS
Update Inventory and Job_status in OMAC with data from SMS, FAST, and FAMS.
8 | Inventory, OMAC; Job_status, OMAC
Update Inventory and Job_status in OMAC with data from SMS, FAST, and FAMS.

Figure 6.8 User inputs and correspondent exchange operations

6.5  Supporting Systems Evolution

As mentioned in section 6.2, the D2D multidatabase environment has been through a continuous process of transformation over the past ten years. A very recent change has been the use of a new system EM3 to replace the old ENDB system. The EM3 system is based on an Ingres database. It contains not only product data from the
ICL design centre, but also that from D2D’s customer companies\(^1\).

The introduction of EM3 is an important step towards maintaining alignment between D2D’s product data management function and its organisational change. This particular change had resulted in redevelopment of two direct interface programs that had previously been used in D2D: one for downloading product data from ENDB to OMAC, the other from ENDB to SPEAR. As a consequence, D2D staff had to be retrained to use the new interfaces.

In this section we discuss how an established GDE system can be adapted to support such a change. The discussion will be based on the assumption that a prototype GDE system has been installed in the real D2D environment, and it will be focused on conceptual issues rather than detailed implementation issues.

Figure 6.9 is a partial reproduction of Figure 6.6, containing only elements relevant to this discussion. Note that the original picture shows a physical system setup for the experimental environment, while in Figure 6.9, the databases involved are regarded as real D2D databases, and the drivers are regarded as ones that match the types of the real databases. To illustrate the database replacement concerned by this discussion, ENDB and its driver have been faded to indicate that they are to be replaced, while EM3 and its database driver are added.

As understood from previous discussions in this thesis, system components which can be affected by changes in local database systems are the database drivers, the data dictionary, and the system configuration files. Here we examine the implications of the database replacement in regard to each of these system elements.

1. **The database drivers.** Each database driver is specially developed for a particular database type. Therefore, when there is a change to the type of a database in an

\(^1\) Before EM3 was introduced, the product data from customer companies was fed directly into OMAC and SPEAR through specially designed local interfaces.
environment, the database driver associated to that database will be affected. Here the situation is that a CODASYL database (used by ENDB) was replaced by an Ingres database (used by EM3). Consequently, the CODASYL driver should be replaced by an Ingres driver. In fact, in this particular case there is no need to create an Ingres driver from scratch, as several other databases (SPEAR, FAST, and FAMS) in D2D are Ingres databases. What is needed is to simply deploy a copy of the existing Ingres drivers for the new EM3. If, however, EM3 was based on a new database type, a new driver must be developed for it.

2. The data dictionary. The new EM3 contains the same product information (such as part lists, BOMs, and supplier information) as does the old ENDB except that the range of the products that EM3 covers is wider. Therefore, OMAC and SPEAR will have the same data sharing relationships with EM3 as they did with ENDB. However, the component schema (see section 5.5.1 for the definition of a ‘component schema’) of EM3 will be different from that of ENDB, since the original schema definitions of the two databases are different (see Appendix A). This
will also affect the data retrieval commands. Hence, the relevant meta data will have to be modified as necessary.

3. The system configuration files. The system configuration files concern the physical location of the databases and mappings between the databases and their drivers. Therefore, replacing ENDB with EM3 will inevitably result in modifications to these files.

Once the above modifications are made, the GDE system is ready to support the new environment. It should be noted that database replacement is only one of the many ways in which software systems could evolve. Other forms of evolution may also be possible, such as a change to the schema of an existing database, a change of data distribution between several databases, or adding a new database to the environment to support a new application, etc.

Not all the changes that could happen to a multidatabase environment will affect shared data and data sharing relationships. In cases where these are affected, modifications required to the three sets of system components could differ from one situation to another. It is not practical to discuss all possible situations in this study. However, the case that has been discussed above can be regarded as a typical example which involves modifications to all three types of components. Based on the example, the following features of the GDE system in support of systems evolution can be observed:

- **Software reusability.** The example shows that most of the software code of the GDE system and the supporting infrastructure does not need to be modified when a change occurs. Even the redevelopment of a database driver can be avoided if a newly added database is of the same type as an existing one. Compared with point-to-point exchange approaches, the GDE approach demonstrates improved software reusability.

- **User support.** The example shows that users of the GDE system are almost un-
affected by the database replacement. Referring back to Figure 6.8 in section 6.4.4, the input for situation 1 only needs a change to the database name (from 'ENDB' to 'EM3'), while the input for situation 2 needs no change at all. Of course, run-time operations related to the two inputs will be affected, but the users will not be aware of it. In this respect, the contrast between the GDE approach and the point-to-point exchange support in the D2D environment is obvious, for the same change has resulted in user retraining at D2D.

- **Support of Autonomy.** The example shows that the changes to the GDE system setup for supporting the replacement of ENDB did not affect other database systems nor their applications. This may not have been the case had the environment been supported by a multidatabase management system. This point will be discussed further in section 7.2.1.

### 6.6 Summary

The case study has demonstrated the use of a GDE system to support data sharing requirements in a multidatabase environment. The data sharing situations that have been handled in the case study are reproductions of those found in an industrial environment. Since the prototype GDE system built in this research is limited in scope, simplifications have been made in respect of database types, database schemas, schema heterogeneity patterns, and sharing relationships involved in the experimental environment. Nevertheless, the case study shows that the prototype system is capable of:

- Accessing remote heterogeneous databases;
- Carrying out data retrieval, conversion, and update according to predefined metadata;
- handling certain types of data exchange situations;
- supporting certain types of update requirements;
- Shielding users from details of the underlying systems.
In addition, the study discussed a case of change in the industrial environment and the way in which an established GDE system could have adapted to it. This provided a level of understanding of the capability of the GDE system to support systems evolution.
Chapter 7

DISCUSSION

7.1 Introduction

This chapter reviews the capability of a global data exchange system from two perspectives. Firstly, the GDE system is compared with multidatabase management systems and other data exchange methods. The comparison highlights the advantages and disadvantages of using a GDE system in a multidatabase environment against other methods. Secondly, three major software attributes of the system corresponding to the three initial design decisions (i.e. the data driven approach, the common database interface approach, and the transparent user support) are discussed in separate sections, in which their major merits, limitations and possible future improvements are outlined.

7.2 The Global Data Exchange Approach versus Other Approaches

7.2.1 Comparison with Multidatabase Management Systems

Support for Autonomy

Fundamentally, a GDE system provides a looser integration than does a multidatabase management system because of its adoption of an off-line data exchange
scheme. As can be seen from the case study described in Chapter 6, the main tasks required when installing a GDE system in a particular environment are: 1) to establish links to individual databases, and 2) to populate the data dictionary with appropriate data. There is no need to modify database schema definitions, nor is it necessary to migrate existing applications. With the support of a GDE system, pre-existing autonomous application systems will continue to function as independent systems. The data exchange requirements of these systems will be looked after by the GDE system. Therefore, we may conclude that a GDE system provides a better support for autonomy than will a multidatabase management system. It can be argued that the approach is more suited to manufacturing environments where application systems are autonomous and are likely to remain autonomous due to broad differences in their software requirements.

A problem associated with the support for autonomy is that the GDE system may have to cope with more complicated schema heterogeneity than will a multidatabase system. Since a multidatabase management solution will gradually migrate all the local applications to the global level, the original schema heterogeneity will be resolved during the process. For a GDE system, however, the differences between schema definitions of different database systems will be preserved because of the nature of the solution. Consequently, the data-driven exchange service module of the GDE system must be sophisticated enough to cope with complex situations which result from schema heterogeneity. This implies that the meta data must be capable of describing complex schema mappings, and that the exchange service functions must be capable of performing complicated data conversion operations. The capability of the exchange service module will be discussed later in section 7.3. At this point, we can see that handling schema heterogeneity is an issue which is more important to a GDE system than it is to a multidatabase system.

Support for Systems Evolution

The basic idea of multidatabase management methods is to encourage future applications to be built as global level applications which access data in distributed databases from a global interface. This will minimise the modification to an established multida-
tabase management system when a new application is added. However, it will be much more difficult for an installed multidatabase system to accommodate an autonomous application system. One of the reasons is that this would require a reconstruction of the global schema to incorporate the new local schema. The schema modifications which result from this could require modifications in the existing global application software, since global applications are dependent on the global schema.

By contrast, in an environment where a GDE system is installed, none of the applications will be affected by changes made in other systems, as each application is independent. However, as shown in the case study, the changes will affect the meta data definition (i.e. shared data definitions and sharing relationships) of the environment, the relevant database interface modules, and the system configuration. The modifications resulted do not affect any of the applications in the environment, but will affect the run-time behaviour of the GDE system.

In this respect, the GDE system could be a favourable solution for those environments which are more likely to evolve as a collection of autonomous systems rather than towards a uniform system. Arguably, manufacturing environments fall into the former category.

7.2.2 Comparison with Other Data Exchange Methods

A GDE system differs from point-to-point data exchange methods in that it aims to provide a generic and flexible approach to data exchange between any types of database, and it has shown the capability to reduce the maintenance effort required as a multidatabase environment evolves over time (as discussed in section 6.5). By contrast, point-to-point exchange methods are closed solutions. They are designed to integrate two particular systems, and therefore have poor software reusability. In cases where old systems are replaced with new systems, relevant interface programs may have to be discarded and new interface programs may have to be re-written from scratch.
Compared with point-to-point exchange methods, neutral format methods are relatively ‘open’. However, neutral format methods are most commonly used in file data exchange. As a consequence, neutral formats are usually designed to support efficient file data processing. By contrast, a GDE system is a database biased solution which does not address file data exchange problems. It identifies data by its database schema definition, not by format. Furthermore, a GDE system is also a network based solution, while neutral format methods are not concerned about how data is transferred from one computer to another.

One disadvantageous aspect of a GDE system compared with other data exchange methods, is its capability to handle complexity. As can be seen from the industrial scenario presented in the case study, a multidatabase environment may involve different types of heterogeneity, such as computer operating system heterogeneity, database management system heterogeneity, database model heterogeneity, and database schema heterogeneity. These will cause complications in respect of building system interfaces, data communications, and in schema mappings and data conversions. Point-to-point exchange approaches can cope with the complexity because each interface program is customised to support data exchange between two particular application systems. Neutral format approaches can cope with the complexity because each data format translator is specially developed for a single application system. By contrast, a GDE system is supposed to be a generic solution. Consequently it will be a great challenge for such a solution to be capable of coping with all the complexities.

The case study shows that the prototype system built in this research has certain limitations in this respect. Further improvement could be possible by refining the implementation of the system, as will be discussed in section 8.3. On the other hand, the author believes that it is important to find the right compromise between supporting local system autonomy and supporting a common approach to systems integration. The issues as to where the balance of the GDE system should be, and how such a balance can be achieved, have to be left for future research.
7.3 The Data-Driven Exchange Services

The exchange service module is the most important part of a GDE system. It is this module which accomplishes major exchange functions. In this research this module was designed to be 'data-driven'. A relational-based implementation of the module has been carried out, and a meta data model and exchange service functions have been defined for this particular implementation.

The data-driven approach was adopted for incorporating generality and flexibility into the GDE system. As shown in the case study, such a system does not 'hard-link' particular application systems. Instead, its behaviour is determined by the meta data previously defined and stored in the data dictionary. Therefore, the system can be used in any environment provided that proper links to individual database systems are established and the data dictionary is populated with appropriate meta data specifying exchange requirements in that environment. It is also easier to modify the meta data than to reconstruct hard-coded interface programs when exchange requirements change, which has been demonstrated in the case study.

However, the potential of the data-driven approach to cope with complex schema mappings and data conversion is uncertain, as the nature of schema heterogeneity was not fully studied in this research. The prototype implementation of the exchange service module has only been able to support certain types of schema heterogeneity and update requirements which were identified in this research. To devise a more sophisticated solution supporting more complicated exchange situations, a more advanced level of understanding of schema heterogeneity issues must be acquired.

7.4 The Common Database Access Interface

Use of a uniform database access interface was proposed at the beginning of the research for handling data directly from databases rather than using an intermediate file. Such an interface also provides a homogeneous view of a multidatabase environment to the exchange service module so that the latter can concentrate on dealing with
Chapter 7 Discussion

only schema heterogeneity.

Such an interface was not explored in depth in this research except that its main functionality and its interface to the exchange service module were principally defined. The research has shown that the design of the meta data model and of the internal functions of the exchange services is dependent on the selection of the common data model. At the time of the research, the relational model was the most mature data model and SQL language was the most widely accepted data query language. If a different data model had been chosen, the implementation would have been different.

A well-known problem with the relational model is its lack of representational power with respect to complex data structures. Other more sophisticated models such as the entity-relationship model or an object oriented model might be better candidates in the future when they are more mature.

The implementation of the common database interface requires the support of network computing technology to provide necessary computer interconnection and interoperation within a heterogeneous computing environment. Although substantial progress has been made in this area and several commercial platforms (see the survey in section 2.4.2) which support distributed computing are now available in the market, building such an interface still requires sophisticated computer programming. On the other hand, the use of network technology in manufacturing companies is still at a very basic level. Computers may be interconnected, but only simple network services such as email and file transfer are used. Despite all these deficiencies, there is no doubt that the situation will improve in the future as network computing technology is currently one of the fastest developing technologies in the information technology domain.

Standardisation could alleviate the difficulties involved in building a common interface which supports remote database access. The only relevant standard available at present is the ISO's RDA (Remote Database Access) standard (see the survey in section 2.4.3). However, a problem with standards in the information technology area is that they become obsolete quickly due to rapid technological innovation. It is
not known at this stage whether the RDA standard will be widely applied.

7.5 The User Support

In this research, the GDE system is designed to shield users from the details of underlying application systems and the details of data sharing relationships in an environment. It has been shown in the case study that exchange operations are executed without user’s intervention apart from initiating a request containing the minimum information required by the system to uniquely identify the exchange requirements from the predefined meta data.

Such an approach is different from those of multidatabase management methods and other data exchange methods where the users must be sufficiently knowledgeable to operate the systems. In addition, the approach can also protect the user (a user could also be an application package, a software timer, as well as a human being, as defined in section 4.3.2.1) from being affected by changes in the environment, as explained in section 6.5.
Chapter 8

CONCLUSIONS

8.1 Summary of the Research

The literature study conducted as part of this research has led to the recognition of a demand in manufacturing companies for preserving autonomy of manufacturing information systems while pursuing systems integration. Having studied current information systems integration methods, the author proposed a global data exchange (GDE) approach to the integration of manufacturing databases, which provides off-line data exchange services based on the support of a uniform multi-database access interface.

The GDE approach has been investigated from the perspective of its software system development. A particular system design was proposed, which is characterised by: 1) a 'data-driven' approach to the realisation of data exchange functionality; 2) the use of a common database access interface to facilitate communications between the GDE system and heterogeneous database systems; and 3) a user interface designed to support less sophisticated users. The design led to a specification of a software architecture which defines functional components of a GDE system and the inter-relationships between the components.

Based on the design, a prototype GDE system was developed which partially realised the functions envisaged. The prototype implementation was based on the use of the relational model as the common data model and the CIM-BIOSYS inte-
grating infrastructure as the platform to support a uniform access to multiple autonomous database systems. Within such a context, the structure of the meta data and the detailed behaviour of the exchange service module were specifically defined.

To demonstrate how a GDE system can be applied, a case study was conducted. In the case study, an experimental environment was established based on an investigation of an industrial multidatabase environment. The prototype GDE system was then set up to support data exchange requirements in the experimental environment. In addition, a case of change which happened recently in the industrial environment was examined, and the capability of the GDE system to support such a change was discussed.

8.2 Contributions to Knowledge

The contributions of the research can be summarised as follows:

• The research has proposed a global data exchange (GDE) approach which addresses the requirements of data sharing between autonomous manufacturing databases.

• The research has suggested a particular functional decomposition of the proposed GDE system based on an understanding of the essential functionality required and a preference for incorporating openness and flexibility. Such a decomposition is formally represented by the software architecture presented in Chapter 4.

• The research has tested the proposed decomposition by developing a software prototype of the GDE system and by deploying the prototype system in an experimental multidatabase environment.

• The research has shown that it is possible to build a working prototype of a GDE system based on the use of the relational model as the common data model and on the support of the CIM-BIOSYS integrating infrastructure to provide a uniform
access to heterogeneous databases.

- The case study result shows that the prototype GDE system was capable of: 1) enabling data sharing between autonomous application systems without requiring modifications to the underlying databases and the applications; 2) handling certain types of data exchange requirement; 3) supporting the evolution of a manufacturing application environment with reduced maintenance effort; 4) shielding users from details of the underlying systems.

- The research shows that the support of network computing technology is a necessity for the realisation of the GDE approach. Although at present such support may not be widely available in most manufacturing companies, this situation is expected to improve in the near future, as network computing technology is one of the fastest developing technologies in the information technology domain.

- The research shows that the data-driven approach adopted at the system design stage has been able to provide generic and flexible exchange services. However, the potential of the approach to handle complex types of schema heterogeneity is not clear due to the lack of sufficient understanding of schema heterogeneity per se. This issue has to be left for future research.

- The research shows that the GDE system's commitment to preserving autonomy has a price to pay. A GDE system will have to cope with high levels of complexity which result from various types of heterogeneity. Ironically, its other commitment to providing generic and flexible services inevitably constrains its capability to handle such complexity. The author believes that the problem is one that is commonly found in developing systems integration solutions. The key issue here is to find the right balance between supporting local system autonomy and supporting a common approach to systems integration with respect to a particular problem domain. This issue has to be left for future investigation.
8.3 Future Work

A major issue which needs further investigation (as previously mentioned) concerns the way in which schema heterogeneity is handled by a GDE system. The author believes that it is possible to devise a more sophisticated algorithm for the data-driven exchange service module so that it can cope with more complex data conversion. Such an algorithm will have to be based on a more advanced research into the nature of schema heterogeneity.

The meta data model may also be enhanced to include more information about shared data and data sharing relationships in an environment, so that more complex exchange situations can be supported. In fact, the types of data exchange situation discussed in this research (see sections 4.2.4.3 and 5.5.4) are only a subset of what might exist in a multidatabase environment. Further research should identify other possible types of data exchange situation and provide a necessary understanding of the meta information required to support such situations.

User interface capabilities such as supporting 'application-triggered' requests, 'time-triggered' requests, and of supporting a distributed user interface were not explored in this research. Yet it would be very important for a GDE system to have these capabilities so that it can satisfy different request patterns of data exchange required by manufacturing companies. Therefore, further investigation should be carried out into the design and implementation issues involved in realising these capabilities in a GDE system.

This research has concentrated on the integration of those information systems which support engineering activities and production planning and control activities. The main features of data sharing between such systems are that: 1) the sharing relationships are usually stable and well-understood; and 2) the data structures are relatively simple. This may not be true for other information systems deployed in manufacturing environments, such as those supporting product design activities.
In recent years, the demand for integrating different design systems is growing especially in companies (such as those in the car industry or aeroplane industry which have their products manufactured in different countries over the world) where several design systems are geographically distributed. In general, design data structures in these industries are more complex, and data sharing required by product design activities is relatively more dynamic. In addition, design systems are even more heterogeneous and autonomous in comparison to the systems mentioned in this thesis. Consequently, enabling data sharing between design systems will present a bigger challenge to information systems integration. Further work could investigate whether the work described in this thesis could be extended to support information sharing in this area.

**8.4 Conclusion**

The global data exchange (GDE) approach proposed in this research offers an alternative way of tackling information sharing problems in manufacturing companies which use multiple heterogeneous and autonomous database systems to support manufacturing activities. It is an approach which has combined features of two classes of IT methods supporting information sharing, namely, multidatabase management methods and neutral data format methods. This research has investigated major issues involved in software realisation of the approach, and has built a prototype GDE system which has the capabilities and benefits envisaged. The research also reveals that there are outstanding issues which require further investigation, such as the nature of schema heterogeneity and the way in which the GDE system should handle it. The author believes that the understanding provided by this research has added to the knowledge about information systems integration in manufacturing environments, which will help to bring about more advanced solutions.
REFERENCES


Vol. 25, No. 3 (March), pp. 50-60.


References


123.


References


Appendix A

The BOM Schemas in D2D Databases

A.1 Introduction

In this appendix the original BOM schemas of four D2D databases, namely, ENDB, OMAC, SPEAR, and the newly installed EM3\(^1\), are included. The purpose of the appendix is to show data duplication and data heterogeneity in a real manufacturing environment.

As can be seen from this appendix, the information associated with the BOMs is different from one database to another even though all the schemas represent the Bill of Materials in the same manufacturing environment. It illustrates that different applications have different concerns. For example, ENDB has attributes such as 'design_auth' and 'cad-file' in its 'part-rec' record definition. These are clearly the concern of design engineers. In OMAC, attributes such as 'opno' (operation number) and 'componentscrap' in the 'structure' record definition are of concern in production areas. The information included in SPEAR's schema is closely related to the PCA designs, hence the attributes associated with its BOM schema reflect this nature (for example, 'position_x', 'orientation' in 'component_instance' table definition).

The information which is of common interest between the databases is the part

\(^{1}\) The schemas were supplied by Mr. Rick Standish who works at D2D as a PDM (Product Data Management) consultant.
assembly structure and the part-supplier relationships (which can also be found in the schemas). The following paragraphs describe the heterogeneous nature of part assembly schemas in different databases.

Both ENDB and OMAC are CODASYL network databases. Section A.2 shows a typical CODASYL schema definition of the BOM in ENDB. It represents BOM with two record types 'part-rec' and 'structure', together with two sets 'used-on' and 'made-from' (which specify relationships between the two record types). OMAC should have a similar schema definition represented by means of record types and sets. However, the schema supplied to the author is in a different presentation format of the network schema definition, which shows two record types 'part' and 'structure' (see section A.3). Although based on the same data model, we can see that there are differences in record name definitions ('part-rec' for ENDB and 'part' for OMAC) and attribute name definitions (e.g. 'part_no' in ENDB and 'partnumber' in OMAC).

SPEAR is based on a relational database, therefore its schema definition is relational. However, SPEAR does not have an explicit BOM data structure. After BOM data is downloaded from ENDB to SPEAR via a direct interface program, the data has to be disseminated into three tables (namely, component_instance, component_type, and design_definition) shown in section A.4. From these tables a BOM can be deduced. This situation is rather unconventional, and therefore is not supported by the GDE system designed in this research. However, a possible solution to this particular problem is to create a BOM table in SPEAR for BOM data downloading, and then to run a special purpose local program to read the data from the BOM table and to update the other three tables as required.

EM3 is also based on a relational database. However, the database design of EM3 has adopted an object oriented approach. As a consequence, it is not straightforward to find out which attribute represents 'part_no' from the tables listed in section A.5. In fact, 'part' is modelled as an object known as a 'process'. Each process has a unique 'process_id'. From the table 'obj_iss_link' we can find the product hierarchy
Appendix A  The BOM Schemas in D2D Databases

represented by ‘process_id’, ‘child_process_id’, and ‘child_qty’. Information related to each ‘process_id’, such as ‘part_no’, has to be obtained from the table ‘process’. Again, this case shows how the same information can be represented in different logical schemas, even though databases support the same data model.

A.2  The BOM Schema in ENDB

ENDB record definitions:

Record name: part-rec

Key endb-calc-0101 is r101-part-no duplicates not allowed.

<table>
<thead>
<tr>
<th>03</th>
<th>r101-part-no</th>
<th>pic x(14).</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>r101-design-auth</td>
<td>pic x(10).</td>
</tr>
<tr>
<td>03</td>
<td>r101-tot-cost</td>
<td>pic 9(5)v9(4) comp.</td>
</tr>
<tr>
<td>03</td>
<td>r101-auth-code</td>
<td>pic x(10).</td>
</tr>
<tr>
<td>03</td>
<td>r101-chng-date</td>
<td>pic 9(6) comp-3.</td>
</tr>
<tr>
<td>03</td>
<td>r101-category</td>
<td>pic x.</td>
</tr>
<tr>
<td>03</td>
<td>r101-class-no</td>
<td>pic x(10).</td>
</tr>
<tr>
<td>03</td>
<td>r101-class-seq</td>
<td>pic x(4).</td>
</tr>
<tr>
<td>03</td>
<td>r101-supd-by</td>
<td>pic x(14).</td>
</tr>
<tr>
<td>03</td>
<td>r101-sup-status</td>
<td>pic x.</td>
</tr>
<tr>
<td>03</td>
<td>r101-file-issue</td>
<td>pic 9(3).</td>
</tr>
<tr>
<td>03</td>
<td>r101-cad-file</td>
<td>pic x(14).</td>
</tr>
<tr>
<td>03</td>
<td>r101-desc-leng</td>
<td>pic 999 comp.</td>
</tr>
<tr>
<td>03</td>
<td>r101-full-desc.</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>r101-desc-char occurs 0 to 150 depending r101-desc-leng</td>
<td>pic x.</td>
</tr>
</tbody>
</table>
Appendix A  The BOM Schemas in D2D Databases

Record name: structure

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>r103-im</td>
</tr>
<tr>
<td>03</td>
<td>r103-spare-1</td>
</tr>
<tr>
<td>03</td>
<td>r103-alpha-qty</td>
</tr>
<tr>
<td>03</td>
<td>r103-assm-lev</td>
</tr>
<tr>
<td>03</td>
<td>r103-spare-2</td>
</tr>
<tr>
<td>03</td>
<td>r103-qty-used</td>
</tr>
</tbody>
</table>

ENDB set definitions:

used-on set

set name used-on.
  order next.
  owner part-rec.
  member structure.
  insertion is manual retention is mandatory.

made-from set

set name made-from.
  order sorted.
  owner part-rec.
  member structure.
  key is ascending r103-im duplicateds last.
  insertion is automatic retention is mandatory.

A.3 The BOM Schema in OMAC

OMAC record definitions:

Record-type: part

<table>
<thead>
<tr>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>part</td>
<td>permanent</td>
</tr>
</tbody>
</table>
### Items: part

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Unique</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>partnumber</td>
<td>characters</td>
<td>1-32</td>
<td>no</td>
<td>fast</td>
</tr>
<tr>
<td>partdesc</td>
<td>characters</td>
<td>1-32</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>change-note-id</td>
<td>characters</td>
<td>1-12</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>doa-flag</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>change-status</td>
<td>characters</td>
<td>1-1</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>struc-date</td>
<td>date</td>
<td>8</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>low-lev-code</td>
<td>integer</td>
<td>1-8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>altvepartnumber</td>
<td>characters</td>
<td>1-32</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>lastupdatedate</td>
<td>date</td>
<td>8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>stockunit</td>
<td>characters</td>
<td>1-4</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>altveunit</td>
<td>characters</td>
<td>1-4</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>conversionfactor</td>
<td>decimal</td>
<td>11.6</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>vatcode</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>delete-flag</td>
<td>characters</td>
<td>1-1</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>part-type</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>drawingsize</td>
<td>characters</td>
<td>1-2</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>roundreqts</td>
<td>integer</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>doa-change</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>forecastpart</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>mms-control-part</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>currstrucmodnum</td>
<td>integer</td>
<td>1-8</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>nextstrucmodnum</td>
<td>integer</td>
<td>1-8</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>nextstrucmoddate</td>
<td>date</td>
<td>8</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>commoditycode</td>
<td>characters</td>
<td>1-4</td>
<td>no</td>
<td>slow</td>
</tr>
</tbody>
</table>
### Record-type: `structure`

<table>
<thead>
<tr>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>permanent</td>
</tr>
</tbody>
</table>

### Items: `structure`

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Unique</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp-partno</td>
<td>characters</td>
<td>1-32</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>change-note-id</td>
<td>characters</td>
<td>1-12</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>change-status</td>
<td>characters</td>
<td>1-1</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>delete-flag</td>
<td>characters</td>
<td>1-1</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>unitflag</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>by-product</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>lastupdatedate</td>
<td>date</td>
<td>8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>on-date</td>
<td>date</td>
<td>8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>off-date</td>
<td>date</td>
<td>8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>component-seq-no</td>
<td>integer</td>
<td>1-8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>qty-per-assy</td>
<td>decimal</td>
<td>11.6</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>assembly-doa</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>component-doa</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>loop-flag</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>dup-comp-flag</td>
<td>characters</td>
<td>1-1</td>
<td>yes</td>
<td>slow</td>
</tr>
<tr>
<td>feed-time</td>
<td>integer</td>
<td>1-8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>opno</td>
<td>characters</td>
<td>1-8</td>
<td>no</td>
<td>slow</td>
</tr>
<tr>
<td>componentscrap</td>
<td>decimal</td>
<td>2.6</td>
<td>no</td>
<td>slow</td>
</tr>
</tbody>
</table>
A.4 The BOM Schema in SPEAR

SPEAR table definitions:

Table name: component_instance

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nulls</th>
<th>Defaults</th>
<th>Key Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd_id</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>component_instance</td>
<td>char</td>
<td>20</td>
<td>no</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>type</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>part_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>attach_side_code</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>position_x</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>position_y</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>orientation</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>pitch</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>endb_ref</td>
<td>char</td>
<td>10</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>associated_with</td>
<td>char</td>
<td>20</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>pkge_id</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Table name: component_type

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nulls</th>
<th>Defaults</th>
<th>Key Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>part_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>part_number</td>
<td>char</td>
<td>20</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>base_part_number</td>
<td>char</td>
<td>20</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>text_description</td>
<td>char</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>component_name</td>
<td>char</td>
<td>15</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nulls</td>
<td>Defaults</td>
<td>Key Seq</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>--------</td>
<td>-------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>rejection_code</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>supply_rotation</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>supply_format_id</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>high_tolerance</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>low_tolerance</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>polarised</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>axial_depth_stop</td>
<td>char</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>refire_required</td>
<td>integer</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>polarity_test</td>
<td>char</td>
<td>1</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Table name: design_definition

Column Information:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nulls</th>
<th>Defaults</th>
<th>Key Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd_id</td>
<td>integer</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>board_name</td>
<td>char</td>
<td>30</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>mod_level</td>
<td>char</td>
<td>10</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>pbn</td>
<td>char</td>
<td>20</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>date_of_modification</td>
<td>date</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>engineers_name</td>
<td>char</td>
<td>20</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>associated_ecps</td>
<td>char</td>
<td>20</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>modification_details</td>
<td>char</td>
<td>20</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>assembly_number</td>
<td>char</td>
<td>20</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>product_group</td>
<td>char</td>
<td>12</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
A.5 The BOM Schema in EM3

EM3 table definitions:

Table name: objiss_link

Column information

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nulls</th>
<th>Defaults</th>
<th>Key Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>objiss_link_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>otype_link_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>process_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>proc_on_rev_no</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>proc_off_rev_no</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>objiss_link_seq</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>child_process_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>child_rev_no</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>child_qty</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>objiss_link_name</td>
<td>varchar</td>
<td>48</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

Table name: process

Column information

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nulls</th>
<th>Defaults</th>
<th>Key Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>process_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>proc_rev_no</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>otype_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>process_name</td>
<td>varchar</td>
<td>16</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>process_issue</td>
<td>varchar</td>
<td>6</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>process_desc</td>
<td>varchar</td>
<td>36</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>route_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>proc_priority_id</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>---</td>
<td>----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>process_state</td>
<td>integer</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>issue_state</td>
<td>char</td>
<td>1</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

The Data Dictionary: the Schema Definitions
And the Meta Data

B.1 The DD Schema Definitions

The data dictionary used by the prototype GDE system is a relational database. The conceptual schema of this database was created in accordance with the conceptual model established in section 5.5.5. In section B.1.1 below, a list of the SQL data definition statements used for creating the relational tables is presented. Since the tables were designed to be normalised, directly querying them will be a tedious task. Usually long and multi-layered SQL SELECT statements will have to be used to retrieve a piece of information from such tables. To alleviate this problem, the author created a set of relational views. The SQL statements used for the view creation are included in section B.1.2.

B.1.1 SQL Statements for DD Relational Table Creation

create table shared_relation (rel_id integer1 not null,
rel_name char(10) not null);

create table src_relation (src_id integer1 not null,
rel_id integer1 not null,
retrieval_cmd varchar(70));

create table tgt_relation (tgt_id integer1 not null,
rel_id integer1 not null,
upd_req char(7));
create table s_t_mapping (src_id integer1 not null,
                      tgt_id integer1 not null);

create table attribute (attr_id integer1 not null,
                        attr_name char(15) not null,
                        attr_type char(9),
                        key_def char(7));

create table db (db_id integer1 not null,
                 db_name char(7) not null);

create table attr_of_rel (attr_id integer1 not null,
                          rel_id integer1 not null);

create table rel_of_db (rel_id integer1 not null,
                        db_id integer1 not null);

B.1.2 SQL Statements for DD View Creation

create view rel_db_view as
    select a.rel_id, a.rel_name, b.db_id, b.db_name
    from shared_relation a, db b, rel_of_db c
    where a.rel_id = c.rel_id
         and b.db_id = c.db_id;

create view src_view as
    select b.rel_id, a.src_id, b.rel_name, a.retrieval_cmd, b.db_name
    from src_relation a, rel_db_view b
    where a.rel_id = b.rel_id;

create view tgt_view as
    select b.rel_id, a.tgt_id, b.rel_name, a.upd_req, b.db_name
    from tgt_relation a, rel_db_view b
    where a.rel_id = b.rel_id;

create view st_view (s_rel_id, s_rel_name, s_db_name,
                     t_rel_id, t_rel_name, t_db_name)
Appendix B  The Data Dictionary: the Schema Definitions and the Meta Data

as select a.rel_id, a.rel_name, a.db_name,
        b.rel_id, b.rel_name, b.db_name
from src_view a, tgt_view b, s_t_mapping c
where a.src_id = c.src_id
    and b.tgt_id = c.tgt_id;

create view attr_rel_db_view as
    select a.attr_id, a.attr_name, a.key_def, c.rel_name, c.db_name
from attribute a, attr_of_rel b, rel_db_view c
where a.attr_id = b.attr_id
    and b.rel_id = c.rel_id;

B.2 A Populated Data Dictionary for the Experimental Environment

The data dictionary was populated with the meta data generated during the data analysis process described in section 6.4.2. In the following subsections, the meta data is presented in a number of views which are obtained by performing data retrieval over the relational views defined in B.1.2.

B.2.1 A View of the Relation-Database Relationships

Retrieval command:  select rel_name, db_name from rel_db_view.

<table>
<thead>
<tr>
<th>rel_name</th>
<th>db_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM</td>
<td>endb</td>
</tr>
<tr>
<td>BOM</td>
<td>omac</td>
</tr>
<tr>
<td>job_orders</td>
<td>omac</td>
</tr>
<tr>
<td>job_status</td>
<td>omac</td>
</tr>
<tr>
<td>inventory</td>
<td>omac</td>
</tr>
<tr>
<td>BOM</td>
<td>spear</td>
</tr>
<tr>
<td>job_orders</td>
<td>fast</td>
</tr>
<tr>
<td>job_status</td>
<td>fast</td>
</tr>
</tbody>
</table>
## B.2.2 A View of the Source Relations

Retrieval command:  
\[
\text{select rel\_name, db\_name, retrieval\_cmd from src\_view.}
\]

<table>
<thead>
<tr>
<th>rel_name</th>
<th>db_name</th>
<th>retrieval_cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM</td>
<td>endb</td>
<td>select part_no, comp_part_no, qty from BOM</td>
</tr>
<tr>
<td>BOM</td>
<td>endb</td>
<td>select part_no, comp_part_no, qty from BOM where type = 'PCA'</td>
</tr>
<tr>
<td>job_orders</td>
<td>omac</td>
<td>select job_id, due_date from job_orders where area_code = 'BB'</td>
</tr>
<tr>
<td>job_orders</td>
<td>omac</td>
<td>select job_id, due_date from job_orders where area_code = 'SM'</td>
</tr>
<tr>
<td>job_status</td>
<td>fast</td>
<td>select * from job_status</td>
</tr>
<tr>
<td>job_status</td>
<td>fams</td>
<td>select * from job_status</td>
</tr>
<tr>
<td>stock</td>
<td>sms</td>
<td>select part_no, qty from stock</td>
</tr>
</tbody>
</table>

## B.2.3 A View of the Target Relations

Retrieval command:  
\[
\text{select rel\_name, db\_name, upd\_req from tgt\_view.}
\]

<table>
<thead>
<tr>
<th>rel_name</th>
<th>db_name</th>
<th>upd_req</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM</td>
<td>omac</td>
<td>RD</td>
</tr>
<tr>
<td>job_status</td>
<td>omac</td>
<td>RR</td>
</tr>
<tr>
<td>inventory</td>
<td>omac</td>
<td>TR</td>
</tr>
<tr>
<td>BOM</td>
<td>spear</td>
<td>RD</td>
</tr>
<tr>
<td>job_orders</td>
<td>fast</td>
<td>TR</td>
</tr>
<tr>
<td>job_orders</td>
<td>fams</td>
<td>TR</td>
</tr>
</tbody>
</table>
B.2.4 A View of the Source-Target Relationships

Retrieval command: \[ \text{select } s\_rel\_name, s\_db\_name, t\_rel\_name, t\_db\_name \text{ from } st\_view; \]

<table>
<thead>
<tr>
<th>s_rel_name</th>
<th>s_db_name</th>
<th>t_rel_name</th>
<th>t_db_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM</td>
<td>endb</td>
<td>BOM</td>
<td>omac</td>
</tr>
<tr>
<td>job_status</td>
<td>fams</td>
<td>job_status</td>
<td>omac</td>
</tr>
<tr>
<td>job_status</td>
<td>fast</td>
<td>job_status</td>
<td>omac</td>
</tr>
<tr>
<td>stock</td>
<td>sms</td>
<td>inventory</td>
<td>omac</td>
</tr>
<tr>
<td>BOM</td>
<td>endb</td>
<td>BOM</td>
<td>spear</td>
</tr>
<tr>
<td>job_orders</td>
<td>omac</td>
<td>job_orders</td>
<td>fast</td>
</tr>
<tr>
<td>job_orders</td>
<td>omac</td>
<td>job_orders</td>
<td>fams</td>
</tr>
</tbody>
</table>

B.2.5 A View of the Database-Relation-Attribute Relationships

Retrieval command: \[ \text{select } db\_name, rel\_name, attr\_name, key\_def \text{ from } attr\_rel\_db\_view; \]

<table>
<thead>
<tr>
<th>db_name</th>
<th>rel_name</th>
<th>attr_name</th>
<th>key_def</th>
</tr>
</thead>
<tbody>
<tr>
<td>endb</td>
<td>BOM</td>
<td>comp_part_no</td>
<td>y</td>
</tr>
<tr>
<td>endb</td>
<td>BOM</td>
<td>part_no</td>
<td>y</td>
</tr>
<tr>
<td>endb</td>
<td>BOM</td>
<td>qty</td>
<td>n</td>
</tr>
<tr>
<td>omac</td>
<td>BOM</td>
<td>comp_partno</td>
<td>y</td>
</tr>
<tr>
<td>omac</td>
<td>BOM</td>
<td>partnumber</td>
<td>y</td>
</tr>
<tr>
<td>omac</td>
<td>BOM</td>
<td>qty_per_assy</td>
<td>n</td>
</tr>
<tr>
<td>omac</td>
<td>job_orders</td>
<td>due_date</td>
<td>n</td>
</tr>
<tr>
<td>omac</td>
<td>job_orders</td>
<td>job_id</td>
<td>y</td>
</tr>
<tr>
<td>omac</td>
<td>job_status</td>
<td>job_id</td>
<td>y</td>
</tr>
<tr>
<td>omac</td>
<td>job_status</td>
<td>status</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>omac</td>
<td>inventory</td>
<td>part_no</td>
<td>y</td>
</tr>
<tr>
<td>omac</td>
<td></td>
<td>qty</td>
<td>n</td>
</tr>
<tr>
<td>spear</td>
<td>BOM</td>
<td>board_no</td>
<td>y</td>
</tr>
<tr>
<td>spear</td>
<td>BOM</td>
<td>comp_part_no</td>
<td>y</td>
</tr>
<tr>
<td>spear</td>
<td>BOM</td>
<td>qty</td>
<td>n</td>
</tr>
<tr>
<td>fast</td>
<td>job_orders</td>
<td>due_date</td>
<td>n</td>
</tr>
<tr>
<td>fast</td>
<td>job_orders</td>
<td>job_id</td>
<td>y</td>
</tr>
<tr>
<td>fast</td>
<td>job_status</td>
<td>job_id</td>
<td>y</td>
</tr>
<tr>
<td>fast</td>
<td>job_status</td>
<td>status</td>
<td>n</td>
</tr>
<tr>
<td>fams</td>
<td>job_orders</td>
<td>due_date</td>
<td>n</td>
</tr>
<tr>
<td>fams</td>
<td>job_orders</td>
<td>job_id</td>
<td>y</td>
</tr>
<tr>
<td>fams</td>
<td>job_status</td>
<td>job_id</td>
<td>y</td>
</tr>
<tr>
<td>fams</td>
<td>job_status</td>
<td>status</td>
<td>n</td>
</tr>
<tr>
<td>sms</td>
<td>stock</td>
<td>part_no</td>
<td>y</td>
</tr>
<tr>
<td>sms</td>
<td>stock</td>
<td>qty</td>
<td>n</td>
</tr>
</tbody>
</table>