On the systematic reuse of legacy data in distributed object-based enterprise resource planning software

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

Additional Information:


Metadata Record: https://dspace.lboro.ac.uk/2134/34563

Publisher: © Esther Prats-Abadia

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the published version.
Pilkington Library

Author/Filing Title: PLANTS - ACADIA

Vol. No. ........... Class Mark .........................

Please note that fines are charged on ALL overdue items.

REFERENCE ONLY
On the Systematic Reuse of Legacy Data in Distributed Object-Based Enterprise Resource Planning Software

by

Esther Prats-Abadia

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

June 2000

© by Esther Prats-Abadia 2000
ACKNOWLEDGEMENT

First, I wish to express my sincere thanks to my supervisor Professor Richard H. Weston for his invaluable supervision, encouragement and help throughout this research study.

I am also grateful to the staff of the MSI Research Institute for their generous assistance. Especially I would like to recognise the contribution of Mr Wouter Derks for his helpful discussions and ideas. My thanks are also to Mrs. M. E. Carden for her generous help and her kindness. Also my gratitude to J.I. Llorente and Arantxa Burgos at ETSII of Basque Country University for their help especially during the period of the preparation of this thesis.

My gratitude to personnel at Swan Ltd, especially to Mr. Richard Burke for providing useful information for the case study of this research. I am also grateful for the assistance provided by Mr Paul Brazier and Ms Anne Holmes.

I would like to take this opportunity to express my gratitude to the government of Spain for providing financial support for my study.

I wish to thank my mother, my father, my sisters and my brother for their unreserved support and encouragement throughout all this time. Many thanks to Tuan for his continual belief in me, encouragement and companionship. Thanks are also to my best friends in Spain: Carolina, Blanca, Isabel, Anabel, Susana and Sonia for their constant friendship and support.

Finally, I would like to thank all my friends at Loughborough that has shared with me great moments, especially to Anna, Keiko, Fernando and Carlos.
SYNOPSIS

The study concerns the development and testing of a systematic approach to reuse of legacy data. A key aspect of the approach is that it is designed to transform relational data, as might typically be stored in a source database, into object-oriented data. By building upon and extending the use of a set of general-purpose software engineering concepts the systematic approach has many potential application areas. However, within this study focus has been on testing the utility and practicality of the approach in the domain of enterprise resource planning (ERP). The systematic approach is based on a process consisting of four main steps, two of which are centred on the use of an algorithm which was conceived and developed to automate the processing of relational schema and data input by expert users so that object schema can be drawn out, new object classes can be added and redundant object classes removed. Such a capability is particularly suited for use in manufacturing application domains where relational data systems have been developed over time in an ad hoc way.

The utility and practicality of the systematic approach has been tested whilst deploying it to reuse manufacturing data obtained from a number of manufacturing businesses. Here the approach has provided an organised way of customising ERP software so that specific end user information requirements and operating conditions can be satisfied.

As part of the research a proof-of-concept software toolset and experimental environment has been conceived and developed. This has facilitated testing of the systematic approach.

The study has generated new knowledge with respect to:

- creating equivalent object-like data structures from relational database entities;
- understanding characteristic properties of particular end-user data structures;
- understanding common characteristic properties of end-user data sets used in similar application domains;
- reusing equivalent object-like data to define the requirements of object data storage systems;
- reusing equivalent object-like data structures in a systematic way when specifying and designing configurable ERP systems.
CONTENTS

Acknowledgement........................................................................................................... i
Synopsis............................................................................................................................ ii
Contents......................................................................................................................... iii
List of Figures.................................................................................................................... vii
List of Tables.................................................................................................................... x
Chapter 1 Introduction......................................................................................... 1
  1.0 Introduction........................................................................................................... 1
  1.1 Problems associated with the “AS IS” situation.................................................. 2
  1.2 Benefits of adopting “TO BE” solutions............................................................ 3
  1.3 Constraints arising from the adoption of “TO BE” systems............................... 4
  1.4 The research context............................................................................................ 6
Chapter 2 Literature Survey.................................................................................... 8
  2.0 Introduction........................................................................................................... 8
  2.1 The development of data standards................................................................... 11
     2.1.1 STEP (Standard for the Exchange of Product Model Data)....................... 13
     2.1.2 MANDATE (MANufacturing DATa Exchange)........................................ 13
     2.1.3 METADATA COALITION....................................................................... 14
     2.1.4 Conclusions............................................................................................... 15
  2.2 Generic reference models and customisation.................................................... 15
  2.3 Reengineering of data systems......................................................................... 20
  2.4 Incorporation of new technologies..................................................................... 25
     2.4.1 Approach 1: Use of an intermediate layer................................................. 25
     2.4.2 Approach 2: Creating an equivalent OODB.............................................. 28
     2.4.3 Comparative appraisal of the reuse approaches......................................... 29
  2.5 Summary.............................................................................................................. 30
Chapter 3 Justification and scope of the research.............................................. 33
  3.0 Introduction........................................................................................................... 33
3.1 Research aims, from a manufacturing perspective................................. 36
3.2 Research focus....................................................................................... 36
3.3 Research objectives................................................................................ 37
3.4 Key features envisaged that would be integral to and/or provided by the data reuse approach................................................................. 40
3.5 Summary................................................................................................. 42

Chapter 4 Current data structures and transformation processes....................... 43
4.0 Introduction.............................................................................................. 43
4.1 Assumptions and constraints related to current software application systems and computer tools................................................................. 44
4.2 A literature survey on data transformation processes.................................. 46
4.3 Classification of data transformation processes and associated open research questions.............................................................. 51
  4.3.1 Schema translation............................................................................... 51
  4.3.2 Object data population....................................................................... 53
4.4 Summary.................................................................................................. 54

Chapter 5 An approach to transforming relational data for its reuse in object-based systems........................................................................................................ 55
5.0 Introduction.............................................................................................. 55
5.1 A general approach to the transformation of relational data structures into equivalent OO data structures................................................................. 58
  5.1.1 Schema translation............................................................................... 58
  5.1.2 Object data population....................................................................... 67
5.2 A methodology in support of customisable vendor solutions.................... 68
5.3 Summary.................................................................................................. 73

Chapter 6 A software toolset to support the transformation of relational data structures into object-oriented data structures.................................................. 74
6.0 Introduction.............................................................................................. 74
6.1 A toolset specification to support the schema translation process............... 76
  6.1.1 Activity 1: Extracting the structure of the original information stored in the RDB........................................................................................................ 76
  6.1.2 Activity 2: Generating an OO schema.................................................. 77
### Contents

6.1.3 Activity 3: Analysis of data schema ........................................... 80

6.2 A toolset specification to support object data translation and population processes ............................................................. 83

6.2.1 Activity 4: Creating an OO database ........................................ 83

6.2.2 Activity 5: Object instance population ...................................... 85

6.3 Summary .............................................................................. 89

**Chapter 7  A Case Study................................................................**

7.0 Introduction ........................................................................... 90

7.1 Objectives and scope of the case study .................................... 90

7.2 An example classification of end-user companies .................... 92

7.3 Demonstration system set-up .................................................. 95

7.4 Results from the data analysis ............................................... 100

7.4.1 Example make_to_stock company ....................................... 102

7.4.2 Example assemble_to_order company ................................. 104

7.4.3 Example make_to_order company ....................................... 106

**Chapter 8  Analysis of Case Study results........................................**

8.0 Introduction ........................................................................... 108

8.1 Applicability of the approach ............................................... 108

8.1.1 Schema creation .................................................................. 109

8.1.2 Definition and population of the OODB ............................... 109

8.2 Gaining an improved understanding about legacy data in different manufacturing scenarios .................................................. 115

8.3 Conclusions ........................................................................... 123

**Chapter 9  Comparison of the research study approach with other approaches reported in the literature ................................................................**

9.0 Introduction ........................................................................... 125

9.1 Comparison between data transformation and reuse approaches ........ 125

9.1.1 Comparison between approaches developed in this study and other approaches reported in the literature capable of creating object schemas .................................................. 125

9.1.2 User support and information provision capabilities of the new approach compared with other data reuse approaches ......................... 129

v
9.1.3 OODB definition and data population capabilities of the new approach compared with other data reuse approaches........................... 130
9.2 Comparison of methods when detecting subsets of data structures.................. 130
  9.2.1 Capability to compare different data structures..................................... 131
  9.2.2 IT product customisation..................................................................... 131
9.3 Conclusions......................................................................................... 135

Chapter 10  Conclusions........................................................................ 136
10.0 Contributions to knowledge.............................................................. 136
10.1 Novel features of the research.......................................................... 137
10.2 Summary of research achievements.................................................. 139
10.3 Future work...................................................................................... 141

References............................................................................................. 142

Appendix A  Relational Databases vs. Object-Oriented Databases............... 151
  A.0 Relational Databases................................................................. 151
  A.1 Object-Oriented Databases......................................................... 151
  A.2 Comparison between RDBs and OODBs........................................ 153

Appendix B  Research software toolset specification............................... 154
  B.0 Diagram classes ........................................................................... 154
  B.1 Program structure........................................................................... 157
  B.2 Example code................................................................................ 160

Appendix C  Information files results from the case study........................... 164
  C.0 Example make_to_stock company................................................ 164
  C.1 Example assemble_to_order company.......................................... 173
  C.2 Example make_to_order company............................................... 182
LIST OF FIGURES

Figure 1.1 Characteristics of ‘AS IS’ and ‘TO BE’ systems to fulfil manufacturing today’s requirements................................................................. 6
Figure 2.1 Main areas of research on improving performance on Manufacturing Databases.................................................................................. 10
Figure 2.2 Facilitating interchange of data between different applications by using a standard data model. .......................................................... 15
Figure 2.3. SAP customisation processes. ................................................ 19
Figure 2.4 Approach 1: Solution deploying an intermediate mapping layer. ..... 26
Figure 2.5 Approach 2: Creating an equivalent OODB................................ 29
Figure 3.1 Common context of the transformation scenarios studied................. 39
Figure 4.1 Common issues in data transformation processes.......................... 51
Figure 5.1 Requirements and development concerns envisaged as being of prime concern in the development of the project methods and toolset............ 57
Figure 5.2 Layout of resources involved in the schema translation process........ 60
Figure 5.3 Removing the occurrence of Vertical Splitting Tables.......................... 62
Figure 5.4 Identifying missing entities by candidate keys.................................. 63
Figure 5.5 Identifying missing entities by attributes with null values................ 64
Figure 5.6 Mapping a table to a class.............................................................. 65
Figure 5.7 Determining an inheritance relationship.......................................... 65
Figure 5.8 Determining an aggregation relationship........................................ 66
Figure 5.9 Determining an association relationship.......................................... 67
Figure 5.10 Object data population process................................................... 68
Figure 5.11 Extraction of a generic object-based structure from the data structure underpinning a single vendor’s solution........................................ 71
Figure 5.12 Extraction of partial level data structures for one class of end user.... 72
Figure 5.13 Transformation processes for a single customer............................. 72
Figure 6.1 Elements of the translation processes............................................. 75
Figure 6.2 Main activities of the schema translation process............................. 76
Figure 6.3 Screen layout generated by the toolset after data is extracted from the
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>Screen layout for defining a new entity from a candidate key.</td>
<td>77</td>
</tr>
<tr>
<td>6.5</td>
<td>Example of information file about the use of data in the relational schema.</td>
<td>79</td>
</tr>
<tr>
<td>6.6</td>
<td>Screen layout for defining a new entity from a set of attributes belonging to the same class.</td>
<td>81</td>
</tr>
<tr>
<td>6.7</td>
<td>Screen layout generated by the toolset after schema mapping process is completed.</td>
<td>82</td>
</tr>
<tr>
<td>6.8</td>
<td>Java definition of the class <em>Table1</em>.</td>
<td>83</td>
</tr>
<tr>
<td>6.9</td>
<td>Object data population process.</td>
<td>84</td>
</tr>
<tr>
<td>6.10</td>
<td>Example of SQL queries to obtain data values from the RDB.</td>
<td>86</td>
</tr>
<tr>
<td>7.1</td>
<td>Proof-of-concept processes for transforming relational data tables into object data stored in an OO database.</td>
<td>87</td>
</tr>
<tr>
<td>7.2</td>
<td>Experimental set-up used to test the relational transformation approach.</td>
<td>97</td>
</tr>
<tr>
<td>7.3</td>
<td>File information process description.</td>
<td>98</td>
</tr>
<tr>
<td>7.4</td>
<td>Generic reference model drawn from standard main files of the Swan Software database.</td>
<td>99</td>
</tr>
<tr>
<td>7.5</td>
<td>Relationships between Swan Main Files used by the example make_to_stock company.</td>
<td>101</td>
</tr>
<tr>
<td>7.6</td>
<td>Class schema generated by the proof-of-concept toolset when processing test data set 1 from the example make_to_stock company.</td>
<td>103</td>
</tr>
<tr>
<td>7.7</td>
<td>Relationships between Swan Main Files used by the example assemble_to_order company.</td>
<td>105</td>
</tr>
<tr>
<td>7.8</td>
<td>Class schema generated by the proof-of-concept toolset for the data set 2 obtained from the example assemble_to_order company.</td>
<td>105</td>
</tr>
<tr>
<td>7.9</td>
<td>Relationships between Swan Main Files used by example make_to_order company.</td>
<td>107</td>
</tr>
<tr>
<td>7.10</td>
<td>Class schema generated by the proof-of-concept toolset when analysing data set 3 from the example make_to_order company.</td>
<td>107</td>
</tr>
<tr>
<td>8.1</td>
<td>Example application of the mapping rules.</td>
<td>113</td>
</tr>
<tr>
<td>8.2</td>
<td>Examples of verifying consistency during data population processes.</td>
<td>114</td>
</tr>
<tr>
<td>8.3</td>
<td>Illustration of the information types and application toolset used during...</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.4</td>
<td>Illustration of differences in the use of different types of data item within the three types of end user company</td>
<td>117</td>
</tr>
<tr>
<td>8.5</td>
<td>Example of split tables following an analysis of the use of attributes</td>
<td>119</td>
</tr>
<tr>
<td>8.6</td>
<td>Class hierarchy for each data set of the case study</td>
<td>120</td>
</tr>
<tr>
<td>9.1</td>
<td>Comparison drawn between generalised object schema under different relational parameters</td>
<td>122</td>
</tr>
<tr>
<td>9.2</td>
<td>Illustrative use of the project methods and application toolset to generate end user systems based on the use of customisable data structures</td>
<td>128</td>
</tr>
<tr>
<td>10.1</td>
<td>Illustrative summary of achievements made during the research study</td>
<td>134</td>
</tr>
</tbody>
</table>

Figure 10.1

Illustrative summary of achievements made during the research study.

140
LIST OF TABLES

Table 2.1 SEBPC projects................................................................................................. 24
Table 4.1 Summary of data transformation approaches described in the Literature.................. 50
Table 6.1 Main activities involved in the data transformation process.................................. 75
Table 7.1 Example classification of end-user companies....................................................... 93
Table 8.1 Factors considered when analysing the consistency of schema mapping approach.......................... 109
Table 8.2 Factors considered when analysing the accuracy of the OODB definition processes.................................................. 110
Table 8.3 Rationale used to evaluate object schema population ......................................... 112
Table 9.1 Main factors considered when comparing alternative schema mapping approaches........................................................................ 126
Table 9.2 Comparison between ERP vendor's data structures customisation approaches.............................................. 133
Table 10.1 Summary of the data transformation approach developed during this study.................. 137
Table A.1 Comparison between RDBs and OODBs.......................................................... 153
CHAPTER 1

INTRODUCTION

1.0 INTRODUCTION

Various forms of contemporary manufacturing software are used to facilitate the planning, control and monitoring of manufacturing activities [Porter et al, 1999]. Most software systems in current industrial use utilise relational database technology to manage their structured use of data [Zhang and Alting, 1994]. Use of general purpose database technology can facilitate a degree of separation between data issues, concerned with how user and system data is organised and managed, and application issues concerned more directly with how user application processes are structured and facilitated [McIntosh, 1995]. Common examples of such systems include engineering data management, product data management, manufacturing data management, enterprise resource planning (ERP), and shopfloor control and data acquisition (SCADA) systems. Indeed relational databases provide a well established and widely adopted approach to data organisation and management, which offers a capability to support some types of change and thereby a degree of "future proofing" [Prins, 1996].

Nonetheless today’s manufacturing companies have to operate in dynamic environments in which the rate of change is expected to increase [Bastos and Sousa, 1998]. To remain competitive a company must be able to respond and adapt by making product, process, resource and organisation changes in a timely and cost effective way [Popplewell and Bell, 1995]. To achieve this goal appropriate technology should be used [Cook, 2000]. Manufacturing systems\(^1\) should be supported by mechanisms that help realise and manage the generation and reuse of data in order to reduce the development cycle time of new products [Bullinger et al,

\(^1\) Systems that regulate the flow of goods and resources through the production cycle from raw material to final products.
1998] and processes [Vernadat, 1996]. Moreover enterprise solutions suitable for use by a world-class manufacturing company have to offer means of (1) integrating existing and new technologies, (2) interoperating with multiple applications, (3) sharing data between applications and (4) customising software products to meet companies’ necessities [Plachy and Hausler, 1999]. Coupled to this “user requirements pull” for improved data management systems are significant push forces arising from the availability of new forms of IT [Orfali et al, 1996]. According to Du and Wolfe the emergence of new types of computer application has resulted in demands to manage complex data types that are beyond the capabilities of existing relational databases [Du and Wolfe, 1997].

1.1 PROBLEMS ASSOCIATED WITH THE “AS IS” SITUATION

The provision of relational data management capabilities alone is unlikely to prove sufficient when seeking to respond to present day requirements for wider scope, faster changing and greater utility software systems [Orfali et al, 1996]. Moreover current generation vendor products have known deficiencies in the following respects.

(1) They embed some generalised model of requirements that significantly constrains the product capabilities and flexibility from a customer viewpoint [Leishman, 1999].

(2) Wide-scope IT systems are very time consuming and costly to specify, commission and develop, and even more costly to change to meet requirements outside the original system scope [Barber and Weston, 1998].

(3) They do not interoperate effectively with other multi-vendor IT systems without further significant developmental effort [Graham, 1995].

(4) It is impractical to configure and reconfigure large-scale systems so that they continue to operate effectively in alignment with enterprise-wide requirements, therefore typically duplicative human resources are deployed to ensure that multiple systems work harmoniously [Weston, 1999].
1.2 BENEFITS OF ADOPTING “TO BE” SOLUTIONS

Distributed object-based software systems\(^2\) have the potential to alleviate many of the problems outlined in 1.1. This is because inherent properties of such systems make them particularly suitable for use in complex, changing and distributed environments that have become commonplace in a modern manufacturing or service enterprises [Bullinger et al., 1998]. Appropriate use of distributed object technology can enable the development of software that can be integrated via an infrastructure. Such an infrastructure can provide (1) services that support data exchange and information sharing between physically distributed applications and (2) system configuration and management services that facilitate and manage systems integration and various forms of change [Orfali et al, 1996]. Some important and general advantages of distributed object-based systems are as follows:

- An inherent ability to capture the semantic\(^3\) of complex and heterogeneous information [Olsen et al., 1997]

- Software modules (sometimes called objects or components\(^4\)) may in certain cases be readily reconfigured and reused in different ways and possibly in application areas for which they were not originally intended [Sims, 1994].

- Software elements and software systems can be prototyped and developed rapidly. This property is very important, particularly in manufacturing environments that operate under uncertain market and other environmental conditions [Larsen et al, 1997].

- The maintainability of software elements can be much improved with respect to conventional industrial software systems [Taylor, 1998].

\(^2\) A distributed object-based software system is a client/server system that facilitates object modules or components\(^4\) to be distributed across the network and to interoperate with others distributed objects regardless of location, programming language and operating system.

\(^3\) Properties and behaviour that characterise the system that is being analysed

\(^4\) A component is a self-contained piece of software that can be developed, assembled and maintained independently of any application. A component should be able to interoperate with objects components thought a well-specified interface in unpredictable situations.
Chapter I

Introduction

• Generally distributed object systems can readily be extended as they may be represented by a formally defined structure that facilitates inheritance [Orfali et al., 1996].

In respect of their general application in the field of manufacturing systems, theoretically a move from conventional software systems to distributed object systems should deliver a significant improvement in terms of the capability of installed systems to be reconfigured, developed and reused [Larsen et al., 1997]. Therefore it is generally accepted that businesses that deploy distributed object systems will interoperate more effectively and possess a capability to be more responsive than companies that do not. Included within a contemporary distributed object system can be an arsenal of available object oriented methods, services and utilities that deploy object oriented databases as a data repository. Arguably therefore object oriented databases provide the modern alternative to relational databases and a “stronger” basis for building inherently reusable software components that can be flexibly configured into systems [Du and Wolfe, 1997].

1.3 CONSTRAINTS ARISING FROM THE ADOPTION OF ‘TO BE’ SYSTEMS

Unfortunately potential benefit arising from deploying distributed object-based technology may be offset by cost penalties. From the viewpoint of a manufacturing end user of software systems the design and implementation of new systems is likely to incur significant capital, developmental and staff training costs. Furthermore the magnitude of these costs (and associated risks) may be difficult to accurately predict [Juric et al., 1997]. Moreover the cost of recapturing and revalidating information and of facilitating the management of that information by an object database will: (i) constitute a major proportion of the overall developmental cost; (ii) make systems replacement more complex, therefore increasing the level of risk and utilisation of skilled person power, (iii) make longer the timescale of projects concerned with system replacement or major development [Graham, 1995]. Furthermore object database standards and object query languages (OQL) have yet to mature (particularly in comparison to relational database standards and standard query languages (SQL)) [Cooper, 1997]. Therefore there will be a finite risk for manufacturers of needing in
Chapter I Introduction

the near future to recapture, revalidate and facilitate reuse of information in a new type of object database [Lloyd and Galambos, 1999].

There will also be constraints on the rate at which vendors of software systems can support their end user clients in changing from contemporary to future distributed object based software systems [Guttman and Matthews, 1995]. Existing large-scale software system builders and integrators may well prefer to adopt new technology on a phased basis, so that they can progressively develop and use well-proven procedures and standardised tools to support the life cycle of distributed object-based systems [Prins, 1996].

Consequently the adoption of new technology can only be achieved at a pace at which users, vendors and systems integrators will accept it and invest in it. In general end users will prefer the existence of many pilot sites that prove that the technological risks are low and for which there is proven availability of support tools. In the interim there is a need to develop ways of reusing existing information (which has a high value and is often stored in a relational technology) into a form that can readily be reused by distributed object-based software systems. Figure 1.1 illustrates the main benefits of deploying object technologies and lists major constraints on the reuse of aspects of current systems.
1.4 THE RESEARCH CONTEXT

From the foregoing it is evident that the benefits and constraints arising from the reuse of legacy software within object-based systems will be influenced by:

- the "ease" with which legacy software can be reused and integrated into new systems. From a manufacturing end user viewpoint the "ease" may be effectively measured in terms of the time and cost involved relative to the cost of starting from scratch;

- any constraints imposed on new system properties and performance as a direct consequence of the integration and reuse of legacy elements, or indeed as a consequence of the outcomes from methods used to facilitate their integration and reuse.
Therefore to improve the ease with which software systems are engineered, it follows that effective methods are required to facilitate the reuse and integration of legacy software within modern distributed object system environments, i.e. those in which object oriented techniques are deployed to facilitate system reconfiguration and extension on a system-wide basis. As they constitute a common class of legacy software system, it is logical to focus attention on improving the reuse and integration of legacy software systems built upon relational database technology within wider scope systems comprising object oriented software.

*It follows that industry requires systematic methods and supporting tools that enable it to reuse data stored in relational database tables of existing manufacturing software systems.* Indeed this statement provides the research context for this particular PhD study. It follows that the research itself focuses on conceiving, developing and prototyping the use of methods and tools that enable:

- reuse of relational information pre-existing within contemporary manufacturing systems,

- potential advantages of object-based technology to be realised within contemporary manufacturing systems, and

- the development of data transformation approaches that support vendor and end user manufacturer requirements.
CHAPTER 2

LITERATURE SURVEY

2.0 INTRODUCTION

To enable a manufacturing enterprise (ME) to meet the complex and uncertain requirements of customers, stakeholder and environmental factors its system capabilities must be integrated in an effective yet flexible manner [Wang et al, 1997]. Distributed object technology is reported to be a key enabler of effective and flexible systems integration [Bullinger et al, 1998]. However as explained in Chapter 1 in practice often the behaviour of MEs is constrained by characteristic properties of contemporary (so called legacy) systems and MEs face major difficulties when deploying new forms of technology.

This chapter reviews the state-of-the-art in terms of the ability of enterprise data systems to meet changing requirements. Recent progress is considered in respect to various research thrusts that seek to reduce (i) constraints arising from the use of contemporary legacy data systems and particularly in scenarios where multi-vendor, multi-application integration is necessary, (ii) constraints arising from a mismatch between vendor solutions and end-user requirements with respect to satisfying data integration needs, and (iii) difficulties and problems associated with replacing old systems with new ones that incorporate distributed object technologies.

The literature survey will consider the objectives and status of such developments in four main areas of concern illustrated by Figure 2.1.

Discussion is structured with reference to the development of

---

5 Enterprise: a group of organisations sharing a set of goals and objectives to offer products, services or both (ISO 14258).
• Standards, designed to enable interoperation between different application and data systems.

• Common data structures and mechanisms designed to facilitate customisation of vendor software systems, so that they adequately meet end-users requirements.

• Data reengineering processes associated with legacy system elements, following changes in system requirements.

• New forms of object technology that can be introduced into existing systems, originally built upon a base of relational technology.
Chapter 2

Goals

Data sharing
Interoperability
Integration

Defining Data standards

Object-based technology

Exploiting new technologies

Customisation

Exploiting new technologies

Update source system
Implement target requirements
Keep data information

Figure 2.1 Main areas of research on improving performance on Manufacturing Databases
2.1 THE DEVELOPMENT OF DATA STANDARDS

Enterprise systems\(^6\) are required to structure and support human-centred activity so that the business processes of an enterprise are realised in an effective and timely way [Vernadat, 1996][Weston, 1999a]. In few situations are humans wholly replaced by technological systems [Yien and Tseg, 1997]. It follows that contemporary software applications\(^7\) and software tools\(^8\) (that form key building blocks of enterprise systems) are uniquely deployed in a variety of application domains [Vernadat, 1996]. Indeed end user businesses typically require their enterprise systems to be configured as unique compositions of software applications, software tools, enterprise personnel and various other supporting technology [Weston, 1999]. Common application domains include manufacturing planning and control, product design, product engineering, sales order processing, human resource management and enterprise engineering [Bullinger et al, 1998]. In general the data generated and used to underpin the operation of software applications and software tools will be unique [Zhang and Alting, 1994]. When considering enterprise operation as a whole various enterprise systems and their software building blocks must share information in order to co-ordinate the behaviours of individual application systems and thereby target those behaviours towards meeting common goals and objectives [Wang et al, 1997]. Consequently the use of an efficient data management system\(^9\) (to underpin the interoperation of enterprise systems) can much improve the competitiveness of a company [Aiken, 1998]. Because of the diversity and uniqueness of specific manufacturing applications requirements generally it is impractical for a single IT

---

\(^{6}\) Enterprise systems coordinate, manage, control, perform and regulate technical and human resources and operations in a structured and concurrent way to supply products and/or services to its customers.

\(^{7}\) Complete, self-contained programs designed to fulfil specific enterprise functions as directed by users.

\(^{8}\) A computer program, routine or other piece of software used to structure and/or support activities carried out by users.

\(^{9}\) Software that controls the operations related to data handling, such as data acquisition, coding, storage, retrieval and distribution.
vendor to (1) offer a full range of software solutions capable of supporting all unique and changing applications requirements of any end user enterprise or (2) offer unique solutions to more specific application requirements for a broad base of end user customers. Therefore the end user norm corresponds to the use of multi-vendor solutions. In any given ME this much complicates issues related to the integration and exchange of information between different enterprise systems and their component software applications and tools. Further complication arises because the application requirements of end user enterprises can change frequently and in an uncertain way during the lifetime of systems. Therefore any "software glue" used to integrate enterprise systems and their component software applications and tools should not overly constrain subsequent changes to enterprise systems and their interoperation [Feng and Zhang, 1998]. However, current practice often leads to inflexible integrated solutions when large-scale enterprise systems are engineered [Wang et al, 1997].

A common need for multi-vendor, multi-purpose interoperation of software applications and tools (as part of enterprise systems) gives rise to a requirement for application and data standards [Ganti and Brayman, 1995]. Data standard definitions typically represent a common vendor-neutral model of the essential properties of data structures in a given application domain. Use of such a common model can facilitate integrated operation between many different software applications and tools as part of a wider-scope system [Zhang and Alting, 1994]. Therefore, an aim when developing data standards is to define a reference model capable of:

(1) representing information used by different software applications and tools, so as to structure and support the way in which these activities are organised and supported by different applications and tools,

(2) being adopted by different vendors, so that they can supply software solutions and interface elements that conform to common end user needs.

Potentially therefore the use of an appropriate common data structure model can improve the performance of a manufacturing enterprise as a whole.

Diverse organisations have been working over a number of decades to define data standards aimed at facilitating data sharing between software applications and tools.
used in different manufacturing domains. Some of the primary initiatives involved are reviewed in the following sub section.

2.1.1 STEP (Standard for the Exchange of Product Model Data)

STEP provides a standard, neutral format for product data that is created and shared by different software applications and tools through their lifetime. The STEP standard describes various aspects of a product such as geometric data, tolerances, materials and features. STEP-compatible data models include product definition, product structure, shape representation, engineering change, approval and product scheduling [STEP] [Owen, 1997].

The STEP standard is designed to deal mainly with technical challenges such as, (1) data must be exchange accurately and without any changes, (2) models should be extensible to facilitate description of new products, processes and technologies, (3) the scope and complexity should include a broad variety of attributes and parameters (such as geometric shape, materials...) required to describe product data in a wide range of industries [Zhang and Alting, 1994].

The STEP standard uses EXPRESS as a modelling language to explicitly represent data. EXPRESS is an object-oriented data descriptive language. This classifies and constructs integrated sets of resources by describing their data entities, attributes, rules, relationships, functions and constraints [Owen, 1997].

2.1.2 MANDATE (MANufacturing DATa Exchange)

The MANDATE standard is focused on the domain of manufacturing management in MEs. MANDATE has been designed to standardise the capture of computerised information in a neutral form. Three main categories of entity are modelled, namely: (i) a model of data exchanged between a manufacturing company and its environment, (ii) a data model related to the management of resources in the manufacturing company, and (iii) a data model describing the control and monitoring of flows of material within the company from a manufacturing management viewpoint [MANDATE] [Shing, 1994]. A prime objective of the MANDATE standard is to promote the use of a standardised data model to facilitate integration between numerous application systems used to manage operations in companies and between
companies by defining (i) standard representations of production facilities and resources and (ii) standards that facilitate the exchange and sharing of production information.

2.1.3 METADATA COALITION

Metadata Coalition is an open, non-profit organisation that brings industry vendors and end users together to address a variety of problems and issues regarding the exchange, sharing, and management of metadata [Metadata Coalition]. Metadata is the information and documentation that defines and describes the structure and meaning of data thereby making data sets understandable and sharable for users [Aiken et al, 1999]. The Coalition seeks to define a metadata model to enable enterprise data management. This typically requires access, update and sharing of metadata in a multivendor scenario. Vendors' tools that comply with the metadata interchange specification would be able to exchange metadata. Hence, tool vendors will benefit through standardisation and end-users will benefit through the resulting tool integration.

The main objective of the Metadata Interchange Specification is to define an extensible mechanism that will allow vendors to exchange common metadata as well as to deploy 'proprietary' metadata. The main goals of the coalition can be summarised as:

(1) Creating a vendor-independent, industry-defined and maintained standard access mechanism and standard application programming interface for metadata,

(2) Enabling users to control and manage the access and manipulation of metadata in their unique environments through the use of interchange specification-compliant tools.

Therefore the common metadata model must provide sufficient mechanisms to allow vendors to extend the metadata model so that proprietary information can be exchanged while retaining an interchange capacity in multiapplication-multivendor scenarios [Metadata Coalition].
2.1.4 Conclusions

The intention of the aforementioned group of standards is to create a model reference for vendors. This can allow them to develop and deploy their software solutions in such a way that manufacturer end users can more readily achieve interoperation between the various software applications and tools they deploy to structure and support business activities. The STEP and MANDATE standards define a reference model that can act as an intermediary by representing the extent of all the variations of data of possible concern in a target domain. However, the Metadata Coalition takes a different approach. The purpose here is to deal with metadata, rather than the data itself, to avoid any mismatch between data semantic definitions in a multivendor environment.

Figure 2.2 illustrates how a standard reference model can facilitate the interchange of information in multi-applications and multi-vendor environments.

![Figure 2.2 Facilitating interchange of data between different applications by using a standard data model](image)

Goh [Goh et al, 1997] argues that object technology is able to represent and support the use of emerging data modelling standards more effectively than could current technology, based for example on the use of relational models.

2.2 GENERIC REFERENCE MODELS AND CUSTOMISATION

Most contemporary application systems as supplied by vendors, can be classified as either a “custom-designed system” (specially designed to meet specific end user requirements) or as a “general-purpose system” that can be configured in some way to meet a range of similar end-user requirements [Weston, 1998]. General-purpose (or generic) products offer advantage from a cost viewpoint, particularly for software
vendors but also in many cases for end user companies. However end-users find constraints arise, so that seldom can software products be configured to closely match specialist end user needs [Popplewell and Bell, 1995]. The number and value of variables needed to describe various aspects of business processes and ways that these processes and resources are theoretically infinite. Consequently it is logical that each company wants to shape its processes to optimally suit their particular requirements [Keller and Detering, 1995]. Moreover from a user company perspective generic solutions can limit the ability of that company to differentiate itself from its competitors [Weston, 1998].

Software vendors have developed various solutions to the problem of customising general-purpose products. Often they have developed structured methods and tools to alter a particular software product to improve its fit to customer’s requirements, from both technical and business viewpoints. However the process of designing and building customisable solutions implies an understanding of commonality and variability across industries, geographies, specific companies within industries, companies operating in different parts of the world, companies deploying various other enterprise systems and so forth [Keller and Teufel, 1998]. Consequently it is a non-trivial problem to design and implement configurable, interoperable and extensible software applications that will be used in unique multivendor environments. Ideally a customisable solution should not only provide means of fitting a generic solution into specific end-user functionality but additionally it should fit the context and requirements of many different end users. This implies that a customisation process should identify and enable use of common parts of generic business and data processes to provide a core system that reflects features of a general reference model. Furthermore the design of such a reference model and customisation process is complicated by the need to provide sufficient configurability to meet change (both predictable and uncertain) in general industry and specific customer requirements.

Customisation could be supported at different levels [Leishman, 1999] [Keller and Teufel, 1998] as indicated below:
1. Developing a generic architecture specification that explicitly specifies rules, etc. that govern how well defined component applications can be integrated into systems of wider scope.

2. Developing semi-generic architectures (i.e. as a degree of specialisation of the generic architecture) to support variability found with respect to a group of customers, possibly in a given industry sector, or via some other classification.

3. Providing structure methods and tools that support specific solution configuration with reference to the semi-generic architecture and component descriptions.

4. Facilitating the adoption of new version products and integration with other vendors' products under customers' specifications.

A reference model (generic or semi-generic) can represent common business processes and associated data models from where the commonalities and variability of the domain can be defined [Leishman, 1999]. Hence vendors should support customisation mechanisms to configure the solution to meet unique customer requirements. Variability in a 'common' reference model can be implemented and configured by (1) predefining a set of variation points in the vendor's solution where users and developers can generate and insert their own functions or data structures and/or (2) by providing a predefined set of scenarios from which customers can select the more appropriate to fit their processes [Busk-Emden and Galimow, 1996]. The latter technique may limit the capacity of the solution to meet customers' requirements since the range of choices provided may not cover the full range of functionality required by a given customer and, moreover, manufacturers often have to restructure their own system to fit the vendor classes provided [Porter et al, 1999]. However, a selecting technique will facilitate the upgrade of the customised solution into a new version [Leishman, 1999]. Several customisation techniques have been developed to implement variability in the reference model, some of them are next described:

- Multiple version of a solution. This technique permits an ability to define different solutions for different domains (e.g. industry sector, countries). The definition of multiple versions facilitates the description of multiple semi-generic reference
models, however the cost of maintaining multiple versions should be considered [Busk-Emden and Galimow, 1996].

- Use of components, subsystems and packages provides modularity in a given solution space that permits an encapsulation and separation of common parts and variation points [Jacobson et al, 1997].

- Properties of Object Technology (such as subclassing and inheritance) allow developers to define concrete classes from which subclasses can be defined. Also, the use of abstract classes supports the definition of functionality that can be introduce at customisation time [Gilbert and McCarthy, 1998].

- Parameter tables can be used to select variances in functions, procedures, user-interface screens and documentation that adapt the application behaviour to the customer requirements [Leishman, 1999]

- Customer exit\(^\text{10}\) definitions can be provided to enable users to add their own functionality potentially without affecting the operation of application code comprising the original vendor solution [Leishman, 1999][Busk-Emden and Galimow, 1996].

Therefore, according to Leishman, customisation could be implemented by defining a common generic reference model for a domain where a range of variable points can be defined to implement customisation. These variable points can be introduced in the form of a predefined set of selectable scenarios (mainly by using extensions, subclassing, overridden methods and parameters techniques) or/and by modifying the reference model to suit a specific customer domain using customisation techniques such as abstract classes, subclassing and customer exit definitions [Leishman, 1999].

A solution that makes use of several customisation techniques is deployed with respect to SAP-based products [Keller and Teufel, 1998]. SAP-based products provide solutions that can be customised by offering a selection between predefined

---

\(^{10}\) Customer exits are locations within a vendor solution that vendors have predefined to allow users to insert their own specific application functions.
business activities that are claimed to characterise common user business processes in different application domains. Thereby users can select and configure processes that best match their particular requirements. Consequently the SAP methodology is based on use of a process-oriented reference model that differentiates between diverse industry sectors such as manufacturing, trade and services. Each industry sector is characterised by a set of enterprise business areas, such as production, sales and cost control [Busk-Emden and Galimow, 1996]. Also defined is a set of possible scenarios for each business area. Differences between the R/3 Reference model and the end user business processes model are claimed to be accommodated by selecting appropriate scenarios that match end-user requirements. Hence a reference model, corresponding to formal descriptions of different scenarios, has been developed from which end users can choose a scenario that matches most closely their own requirements [Keller and Teufel, 1998]. If the number of choices is high, then the configuration processes can be more effective but will also be more complicated. A further stage of SAP software product customisation is achieved by defining and attributing data values that specify detailed end user requirements in relation to selected scenarios [Busk-Emden and Galimow, 1996]. Each SAP scenario is described by a set of processes. Processes are defined as a flow of events and functions, associated with which communication, data, information flow, and function and organisation modelling viewpoint can be defined. Business scenario customising processes are controlled by fine-tuning parameters that define function variables [Hiquet and Kelly, 1998].

The overall customisation process is illustrated by Figure 2.3.
Chapter 2

Conclusions
The customisation of generic vendor software products, to produce specific end user solutions, can be achieved in alternative ways and with different results and levels of success. For example, ERP vendors (such as SAP) offer customisation processes based around predefined business scenarios that can be configured to reflect specific business needs. Here the number and coverage of scenarios will impact on the extent to which any given solution will match an individual set of customer requirements. Small companies that do not have sufficient resources to deploy complex software applications and tools (such as a SAP ERP package) may be forced to select proprietary solutions which are not particularly customisable but where limited functionality and customisation options offered have to prove adequate [Porter et al., 1999].

On considering customisation techniques reported in the literature, it can be concluded that the adoption of object technology (in underpinning the development of vendor solutions and their customisation options) can facilitate the development and modification of solutions that can better fit end user needs and where variability can be introduced through customising some form of common reference model. We may conclude that ability to customise software products will be influenced greatly by the choice of technology used to facilitate the definition and extent of generic reference models. However in addition the software system itself should be designed so that it is extensible, configurable and open. Upgrading software so that it deploys an improved enabling base of distributed object technology is by no means a trivial problem. However, if this can be achieved adequately well it is probable that the extent to which ‘general’ solutions (supplied by vendors) can be customised to meet unique customer needs will be much improved.

2.3 REENGINEERING OF DATA SYSTEMS
Various reengineering methods are reported in the literature that have been designed to provide means of analysing and developing current generation data systems. This section reviews this literature. Typically such a reengineering process involves creating an abstract system description, analysing and evaluating the impact of changes at the higher abstraction level, and then designing and re-implementing a new
According to Brodie and Stonebraker [Brodie and Stonebraker, 1993] the best architecture for migration purposes is a decomposable structure in which the interface, application, and database services are dealt with as distinct system entities.

The focusing of reengineering processes on data aspects of an existing system has proven to be practical and successful in industry, leading to an enhancement of understanding about a specific database that has deteriorated, or its operations have become unclear and this can facilitate database service redesign and improved operation [Aiken, 1998]. Such a reengineering process starts with an analysis of all current information pertaining to a system. Typically for a database system this requires analysis of existing schema models, data values, database queries, reports and documentation [Blaha, 1997]. Often weaknesses of existing DBMS models will be detected, such as poor design and outdated, or lack of, documentation that can make it very difficult to capture and interpret adequately all the necessary information from the system [Aiken, 1998] [Hainaut et al, 1997].

According to Menhoudj and Ou-Halima [Menhoudj and Ou-Halima, 1997] the objective when reengineering a data system is to develop an application system that must be equivalent or better than the legacy one, for example, so that improved performance can be realised under certain operating conditions. Various researchers have investigated alternative reengineering methods aimed at improving and re-implementing legacy systems so that they can perform differently and/or operate under new conditions [SEBPC] [Aiken, 1998] [Warren, 1999]. However, marked differences between system types make the definition and application of a generic method almost impossible [Blaha, 1999]. Usually re-engineering approaches correspond to various patterns of activity that result in the reengineering process. The importance of the upgrade of legacy systems is reflected by the activity of a large number of researchers in this area. For example there are many ongoing SEBPC (System Engineering and Business Process Change) projects funded by the EPSRC and dedicated exclusively to investigating issues arising from the reuse of legacy systems, particularly when new technologies are incorporated to the system [SEBPC]. The focus and findings of some of these projects are described below and summarised in Table 2.1.
SABA: Software as a Business Asset
The main objective of the SABA project is to create a decision model. The purpose of this model is to determine the design of an appropriate integrated system (comprising legacy system elements) that matches the requirements of a newly defined business process. The project is investigating the use of different parameters that characterise a business system (i.e. attributes of the business processes, software applications and the system management) that can facilitate the reengineering of legacy software. Results from the analysis are being used to determine how legacy parts should change and to enable calculations to determine about the cost-benefit and risk of modified solutions [Brooke et al, 1998].

The objective of the SABA project is not to create a unique best solution. Rather the objective is to analyse specific legacy problems and thereby to assist in the definition of a set of diverse solution scenarios. Solution development involves an iterative process that can help a user company to understand the problem, develop scenarios for organisational changes and define technical assets to support changes made to the system.

Business Process Change & Systems Design Strategies to overcome Problems of Legacy Systems
Like SABA, this project is addressing aspects of and the scale of the legacy problem. It is developing a framework to assist companies to upgrade their software technology. In this context the project is developing theoretical models that characterise legacy problems in terms of technical developments, and the impact of complex environments and changing organisations, from both management and information systems perspectives. Thereby the aim is to enable organisations to better exploit modern IT systems.

RAMASES: Risk Assessment of Legacy Systems within Business Process Change
This project seeks to quantify the risks involved when changing legacy systems within SMEs. The RAMASES method is designed to quantify effects of dependencies between legacy systems and the business processes they support, with respect to the risk involved when (1) making business process change and (2) taking decisions about legacy system solutions.
Chapter 2

Precise Visual Patterns for the Evolutionary Migration of Legacy Systems to Reusable Components.

This project is characterising various patterns used in support of the evolutionary transformation of legacy systems to component-based technology. The aim is to develop reusable patterns that assist in the processes of: (1) capturing business-process logic embedded within legacy systems, (2) achieving effective migration of legacy systems, and (3) specifying a component-based architecture.

AMORE: Methodology using Object-Orientation in Reengineering Enterprises

This project is developing a method and framework to facilitate business process change that can benefit from the deployment of new object-oriented technology. The method will facilitate an analysis and evaluation of legacy systems to support reengineering processes, thereby helping organisations to rapidly adjust their business processes.

ENRICH: Exploiting Legacy Data in New Product Development Processes

This project is concerned with issues arising from managing legacy product data that may be required by future product development processes. The main objective is to gain an understanding about relationships between new product processes and legacy product/process data.

It can be concluded that the reengineering of legacy systems is of current concern to industry and that a methodology is required to assist manufacturers to upgrade their systems in a more scientific and effective way.
## Chapter 2 Literature Survey

### Legacy systems Transformation approach TO BE system

<table>
<thead>
<tr>
<th></th>
<th>Legacy systems</th>
<th>Transformation approach</th>
<th>TO BE system</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SABA</strong></td>
<td>Analysis of model and technology to define a set of parameters that characterise the system.</td>
<td>Iterative process to assess cost, benefits and risks involved in transforming legacy systems with respect to a set of possible change scenarios options.</td>
<td>A set of possible scenarios regarding the integration of legacy systems, as the best decision options.</td>
<td></td>
</tr>
<tr>
<td><strong>Business Process Change</strong></td>
<td>Analysis about how legacy systems interoperate with new technologies</td>
<td>Theoretical framework to evaluate transformation techniques.</td>
<td>Systems that integrate new technologies with legacy systems</td>
<td></td>
</tr>
<tr>
<td><strong>RAMASES</strong></td>
<td>Analysis of how the legacy system can be adapted to changes in the system</td>
<td>Qualitative and quantitative techniques to evaluate legacy systems risks and costs with respect to various businesses process changes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Precise Visual Patterns</strong></td>
<td>Analysis of legacy business process logic.</td>
<td>A patterns definition to support evolutionary migration processes.</td>
<td>Component-based architecture</td>
<td></td>
</tr>
<tr>
<td><strong>AMORE</strong></td>
<td>Analysis of how the legacy system can be adapted to changes in the system</td>
<td>Structured method and framework to support the reengineering of legacy systems</td>
<td>Legacy systems integrated in object technology-based systems to facilitate business process change</td>
<td></td>
</tr>
<tr>
<td><strong>ENRICH</strong></td>
<td>Analysis of legacy data.</td>
<td>Software solutions to facilitate the management of legacy data within new manufacturing processes.</td>
<td>Access of legacy data from new technology and processes.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 SEBPC projects
2.4 INCORPORATION OF NEW TECHNOLOGIES

To keep pace with changing business demands and to take advantage of information technology developments, organisations have to look for alternative ways of replacing or redeveloping their systems, thereby increasing their functionality and performance [Hayes et al, 1994]. The success of a manufacturing company has become more closely linked to the speed and efficiency with which it can incorporate new technologies [Garetti and Bartolotta, 1995][Bullinger et al, 1998]. Therefore there is increased demand in industry to integrate old and new software applications.

Old applications may be procedural-oriented and run on mainframes in a batch mode while increasingly often new applications are object-oriented, distributed client/server types of system [Sneed, 1997]. Relational Databases are often built around central servers. These centralised client/server architectures typically comprise monolithic mainframe applications where the server provides access to shared resources, such as a RDB (Relational Database) [Cooper, 1997]. However with the availability of global information highways new client/servers have been developed where each machine can be both a client and a server. Distributed objects can take advantage of many computing resources by dividing applications into smart components that can work together across the network [Sneed, 1997].

Hence any method aimed at the reuse of data should be developed in such a way that users can realise and benefit from advantages of deploying object-based technologies.

Essentially the existing literature describes two main approaches to the reuse of relational data within object-oriented systems. These approaches are reviewed below.

2.4.1 Approach 1: Use of an intermediate layer

One class of approach to-date described in the literature is based on creating an intermediate layer of processing between the existing RDB and target object-based systems. The purpose of the intermediate layer is to automatically and transparently map database records to data objects and data objects to database records. It follows that the intermediate layer should function to translate data formats between RDB and OO models and it can achieve this purpose with reference to an object-oriented schema view of the pre-existing relational schema. The basis of this approach is that
legacy data stored in a relational format is retained so that data population operations
do not need to be executed during the translation process. However, schema mappings
relating RDB and OO data formats do need to be executed at run time in order to
manipulate data in the RDB. Figure 2.4 illustrates the approach.

Figure 2.4 Approach 1: Solution deploying an intermediate mapping layer

The literature reports on alternative methods of achieving approach 1 and thereby of
enabling pre-existing relational databases to interact with object applications. Agarwal
[Agarwal et al., 1995] proposed the creation of an interface layer (such as that
illustrated by Figure 2.4) as a first step in a staged transition towards the introduction
of object technology. In Agarwal’s method source data is maintained in a relational
database but new applications are generated using an object programming language
and an object interface. A similar method is proposed by Ambler [Ambler, 1999].
Based on results obtained from investigations Ambler argues that potentially approach
1 can allow objects to access relational databases in an effective way even though the
data remains in the relational system. Use of an intermediate layer 'mediator', that can
present an object-like view of the data and operate by accessing the services of an
existing relational database, was also described independently by Graham [Graham,
1995] and by Bergamaschi [Bergamaschi et al., 1997]. Apparently therefore a
common factor in the use of intermediate layer methods is the introduction of object
features by some form of mediator. However the features provided by such a mediator
and approach must be limited by general capabilities of the relational model, such as
its ability to only partially encode the semantic of a system.
There are examples of commercially available tools that facilitate ‘interaction’ between relational databases and object-oriented (OO) applications. Java Blend can transparently and automatically map database records to Java objects and Java objects to database records. Java Blend’s mapping tool performs the mappings required by referencing schema information (such as foreign keys) to determine an initial mapping schema. The mapping layer can then be customised by the end-user [Java Blend]. Java Blend is based on JDBC functionality. JDBC provides Java classes to enquire on databases and facilitates execution of SQL queries [Reese, 1997]. These classes supply a set of functions that can encapsulate database constructs, such as tables, cursors, and operations to enable users to create and define their own mapping layer. Also the ONTOS [Ontos] Data Service Layer corresponds to a middle layer solution that permits mapping between a typical relational database and a so-called business object model.

To support the development and use of a mapping layer in an effective way it is important that suitable object to relational mappings are chosen. Obviously when mappings between relational and object schemas involve major semantic differences (in terms of different entities and attributes) the design of such mechanism becomes more complex. Also use of such a complex mapping mechanism can slow down the performance of the system. A prime benefit of using a mapping layer is that programmers can deal with objects without any need to understand their underlying relational database structures [Srinivasan and Chang, 1997]. On the other hand, developers building object-relational applications do face difficult problems when: (i) mapping objects in the application model to related relational schema in the database, (ii) managing data locking and transactions such that data integrity is maintained, and (iii) considering the optimisation of performance characteristics. The retention of RDB data minimises data integrity problems during the translation processes but the target systems will not be able to fully support object features. Therefore the mediator should explicitly map data objects to relational tuples and should also ensure that data integrity is maintained by developing and using appropriate locking and transaction management mechanisms [Agarwal et al, 1995] [Vermeulen, 1996].
2.4.2 Approach 2: Creating an equivalent OODB

The second class of reused approach described in the literature involves the creation of an object-oriented database (OODB) that is equivalent to the source RDB. Use of an OODB can help to realise potential advantages associated with object-oriented technology. Indeed the goal of an object-oriented database management system (OODBMS) is to provide effective, extendible and reusable means of managing access to and update of information. This information may need to be accessed by complex software applications such as computer aided design and computer aided manufacturing software packages, product data management tools, project management tools, software engineering tools and so forth. Typically the use of software applications and tools in different domains of manufacturing businesses necessitates the sharing and management of a large number of data objects which are related to other data objects by some complex structure. Use of an OODBMS allows complex data of these types to be stored and accessed, either as a whole entity or as customised data fragments. The structure and behaviour of information objects can be modelled as an object-oriented model, thereby capturing more of the semantic content (or meaning) of information entities than is possible when using a relational model. In Appendix A a more extensive discussion about the relative capabilities of RDBs and OODBs and their modelling approaches can be found.

Methods conforming to approach 2 and reported in the literature use an object-oriented schema view to define an object-oriented structure representing the OODB. Then the OODB is populated with data from the pre-existing RDB in conformance with the defined object-oriented structure. Thereby data objects are stored in the OODB. Once RDB to OODB mapping processes are finished, potentially the legacy relational database can be replaced by the equivalent OODB. However in a practical industrial application of approach 2 subsequent data integration problems may well arise because of a need to periodically transfer data between databases if RDB data is updated or modified subsequently and/or periodically. Figure 2.5 illustrates approach 2.
The literature describes methods of realising approach 2 and thereby defining an equivalent OODB to a pre-existing RDB. Jahnke [Jahnke et al, 1996] described a method that systematically organises data extraction and data understanding related to the semantic of a relational schema, as a first stage in a more complete data translation process. Alternative means of creating OODBs from a pre-existing RDBs have also been described by Monk [Monk et al, 1996]. Currently much of the work reported in the literature is limited to schema translation from object to relational models and does not address the problem of transferring the data itself. However a method that supports a more complete approach to the replacement of a RDB by an equivalent OODB is described by Behm [Behm et al. 1997]. Behm’s method is designed to develop enhanced data storage systems that incorporate OO features. The approach is part of the SYNDAMMA project which deploys object-oriented systems designed to facilitate the incorporation of multimedia environments [Ayre et al, 1995].

2.4.3 Comparative appraisal of the reuse approaches

Experimental system building reported in the literature show that Approaches 1 and 2 have relative benefits and disbenefits. Use of an intermediate layer or gateway (i.e. Approach 1) can offer a good compromise when seeking to reuse information stored in different types of database. This is true where there is a need to enable the distribution of heterogeneous language objects in order to facilitate information access. Approach 1 can also provide a suitable way of underpinning the reuse of information retained in non-object-based legacy systems without diminishing the performance of the original system. Such a capability would clearly be important in companies wishing to retain use of existing systems in parallel with the use of new
distributed object systems. On the other hand use of a mediator (i.e. the basis of Approach 1) is less well suited for applications where complex objects need to be stored and accessed by object applications. Experiments conducted by Objectivity [Wade, 1998] found two main problems when deploying solutions conforming to Approach 1, viz.: (1) the life cycle cost of a mediator layer (in terms of the cost of designing and building a mediator and the cost of maintaining and changing its data model) and (2) diminishing system performance, due to the use of an extra layer of processing and particularly when accessing and updating multiple relational tables. The creation of an equivalent OODB (i.e. Approach 2) should provide better results in application scenarios where the benefits of distributed object systems need to be fully exploited. This is the case simply because the use of a mediator must place constraints on solutions [Srinivasan and Chang, 1997]. Moreover the functionality in OODBs is divided between client and server to improve the management of resources and to improve the scalability of solutions, thereby making them more suitable for use in a distributed environment [Orfali et al, 1996]. Complex objects, composite relationships and inheritance modelling properties can all be supported by a suitable schema mapping approach. A more detailed literature review about relational to object-oriented transformation algorithms is presented in sections of Chapters 4 and 5 of this thesis. However, if the RDB needs to remain live, code needs to be generated, maintained and upgraded to retain the integrity of the link between the object layer and the RDB and thereby to maintain consistency between data stored in tables and object-layer views. Unfortunately code generation will be complex in this case because it must implement sophisticated transaction and locking mechanisms.

2.5 SUMMARY

Present day business requirements dictate a need for faster, more comprehensive, accurate and efficient ways of accessing and managing enterprise data. Therefore this chapter has reviewed major research innovations, results from which can be deployed in a unified way to improve current practice in the management of manufacturing data systems.

Data standards have been developed centred on the definition of common data models designed to enable multivendor software applications and software tools to share data.
This can facilitate a degree of integrated operation between applications and tools and can facilitate some forms of large-scale system change. On the other hand software vendors often prefer to develop customised system solutions based on the use of their own reference model of domains in which their software applications and tools are to be used. However user manufacturers require solutions that closely match their specialist requirements thereby enabling them to assume a unique position in a product and/or service market.

A practical industrial application of the use of standard models and customisable solutions needs to be supported by appropriate technology. In this context, often the relational model lacks sufficient richness and flexibility and solutions based on use of an object-oriented model fare much better [Peng and Trappey, 1996]. Indeed the use of distributed object technology can lead to extensible, open and configurable systems.

However, major hurdles need to be overcome when upgrading systems to incorporate new distributed object technologies. Ongoing Research into software reengineering has been reported that, based on analysis of existing legacy systems, determines a suitable way of migrating to the use of new technologies. Certain approaches reported describe how AS IS database systems (such as relational systems) can be replaced by object-oriented databases. There are three primary reasons why existing systems deploying relational databases should be upgraded to allow them to benefit from use of object technology. Firstly the flexibility of the object-based model will improve the capacity of configured solutions to benefit from the use of standard models and thereby provide an improved capability to integrate and change systems. Secondly the ability to customise solutions to meet unique user requirements will be improved. And thirdly the performance of the storage systems underpinning software applications and tools can be enhanced.

Bearing these general conclusions in mind, the primary objective of this research study is to improve the process of transforming current systems based on the use of relational technology towards new object-based technology by achieving the following sub-objectives:
(1) Provide and support the reuse of object-like data structures (that are equivalent to relational data structures used to underpin the operation of legacy systems) and to develop concepts and supporting software tools that assist technology migration and thence the design and specification of upgraded systems based on the use of distributed object-based technology.

(2) Identify and classify common data structures and other selected properties of legacy systems, to gain a better understanding of common technology transformation requirements related to certain types of manufacturing software system.

(3) Gain an improved understanding of the reuse of common data structures, to facilitate their customised use.
CHAPTER 3

JUSTIFICATION AND SCOPE OF THE RESEARCH

3.0 INTRODUCTION

Data storage systems can be viewed as being important enabling components of most software systems. In business and manufacturing environments found in all industrial sectors they are used to support various types of decision and action making. Relational databases are the most common type of data storage system in practical use today [Bergamaschi et al., 1997]. On the other hand, as described in Chapter 2, operational performance, change and reuse capabilities of many types of manufacturing system would be advanced if their underlying relational databases were replaced in an effective way by a distributed object-based data storage system. This is the case because inherent properties of distributed object-based systems make them suitable for use in complex, changing and distributed environments of the type that has become commonplace within the contemporary manufacturing marketplace [Graham, 1995][Bullinger et al., 1998]. However, as discussed in Chapter 2, current and pending problems must be overcome before manufacturing companies can realise the full benefits of such a step change in enabling technology.

Common practice is for manufacturing end user companies to purchase semi-generic ‘application software’ from IT product and system supplier companies and to configure this software to meet their particular set of business and production requirements. Furthermore a typical manufacturing company (or end user) will purchase application software from many IT suppliers (or vendors). Commonly any new ‘product’ will be required to share common data and interoperate in other ways with pre-installed or other new application software products supplied by different vendors. Consequently both vendors and end users of application software have developed methods and software tools to support the configuration and integration of software applications into a new host end user environment. To-date the methods and tools developed by vendors have largely been centred on achieving an increased
number of specific product sales whereas in general the methods and tools required by end users should be vendor neutral and focused on achieving a close match between end user business and production requirements and the way that its various multi-vendor software applications interoperate to realise those requirements in a timely and cost effective way. It follows that a step change in enabling technology from the use of relational database technology to distributed-object database technology requires not only changes to the application products themselves, but also to the way the new products will be used (in conjunction with other application products to realise specific end-user business and production requirements) and in the way they are configured and integrated (via the use of methods and tools) into host end user environments.

Evidently it would be short-sighted to simply seek to design and develop new products and supporting methods and tools that replicate the current use of products built upon the use of centralised relational database technology. Otherwise it would not be possible to benefit fully from improved reuse, reconfigurability, extendibility and general change capability of new forms of distributed object-based manufacturing system so developed. Therefore there is a need for a radical new look at the design of next generation distributed-object based application products, the way that these multi-vendor products should be configured and integrated and how end users and/or vendors should be supported by methods and tools. Clearly each of these sets of issues is interlinked making this a very complex problem to tackle, as evidenced by the current EPSRC SEBPC special programme devoted to this area [SEBPC].

To illustrate this point further standards (such as OMG, MANDATE, STEP and the Meta Data Coalition Business Engineering and Information models) have sought with different scope and foci of concern and at different levels of granularity to develop conceptual models of possible distributed-object-based application components and their associated architectural frameworks [OMG][MANDATE][STEP][Metadata Coalition]. Collectively these public domain models cover business, manufacturing and technical aspects of next generation application systems but centred primarily on the latter. From such initiatives it is generally understood that in the future vendors will need to supply explicit models of the capabilities, interaction requirements and behaviours of their software application products so that they can be used in various
ways, as reusable and change-capable building blocks of next generation manufacturing enterprises [Barber and Weston, 1998]. Hence, as explained in Chapter 2, there remains much work to do to determine what the future big picture will become when nearly all application software is built using distributed-object technology and deploys a suitable underlying database.

It follows that over the next decade we can expect there to be significant constraints on the rate at which vendors can support their end user clients in changing from contemporary to future distributed-object based software systems [Juric et al, 1997]. The adoption of new technologies will only be achieved at a pace at which users and vendors will accept and invest in it. Nonetheless many distributed-object-based business and production application software products have already become available and are being used by end user manufacturers, and this trend has gathered pace through the period of this research project. In general, however, manufacturing companies have yet to benefit from any capability for these products to readily facilitate multi-vendor interoperability [Weston and Hodgson, 2000]. Also the configuration and integration methods and tools provided by vendors remain largely proprietary [Barber and Weston, 1998]. Therefore, for example, available methods and tools do not readily facilitate interoperation between new and distributed-object-based product and software applications previously installed into a host user environment. Consequently during the next decade, and possibly beyond that time, we can expect the industrial use of enabling distributed-object technology to be in a state of transition with industry requiring application software built on mixed enabling technologies to share common data and as appropriate facilitate the use of more sophisticated forms of interoperation so that specific commercial, logistical, technical and production activities can satisfactorily be achieved, on time and at the right cost.

As the use of distributed-object technology broadens and deepens, industry will have a growing need for effective and practical ways of sharing data contained in both relational and object-oriented data sources. For example, industry will require methods and tools to help it transform information stored in existing relational databases into a form that can be reused effectively by object-based applications. It follows that industry requires systematic methods and supporting tools to enable it to
reuse data stored in relational database tables which constitute part of existing manufacturing software systems.

3.1 RESEARCH AIMS, FROM A MANUFACTURING PERSPECTIVE

It is within the context outlined above that work reported in this thesis has researched, developed and tested ways of reusing, within distributed object environments, manufacturing, engineering and sales information contained in relational database and associated application software systems. The 'business' justification for doing this is that over the next decade and beyond industry will wish to:

(1) Recover some of the investment made (in terms of time and effort) in populating existing data storage systems with manufacturing, engineering and sales information.

(2) Maximise the magnitude of benefits it can gain (and the rate at which benefits can be gained) from using distributed-object-based application software.

(3) Improve an aspect of the means used by manufacturers and their IT suppliers to configure and integrate generic application software products so that they match more closely specific business and production requirements.

3.2 RESEARCH FOCUS

Clearly it is not practical within a single PhD study to conduct a thorough and general investigation of potential benefits and constraints on the reuse of equivalent object-like data structures within distributed object systems, nor of how such systems might interoperate in industrial situations whilst sharing data with multi-vendor object applications. Therefore it was necessary to focus this study. Some industrial advice was received from Swan Systems Ltd who supply ERP (Enterprise Resource Planning) software products to over 200 SMEs, mainly in the UK. Thereby the main focus of this research was defined as follows: to investigate how to reuse existing manufacturing, engineering and sales information contained in relational databases. Here the prime reuse focus was determined as being to support data reuse within new systems that only deploy object-based software applications. Therefore this research
has sought to develop and test novel means of (i) transforming manufacturing, engineering and sales data stored in the form of relational tables into an equivalent data-object form in such a way that data consistency is maintained and, (ii) analysing and systematising the reuse of common relational data structures to identify means of configuring these into new structures that match specific manufacturer requirements. It follows that a prime objective of this study has been to define, and support with computer tools, a structured and explicitly-defined methodology that can systemise the transformation of data used to underpin the operation of current vendor solutions into data that underpins the operation of new object-based customisable solutions. It also followed that there was a need to analyse stereotypical, current customer datasets and use these to test the practicality of the new transformation and configuration methods and tools and to illustrate how they can be beneficially applied in industrial scenarios. Therefore this study seeks to define and support a novel approach to analysing current data structures to underpin their transformation and reuse in new solutions requiring specific and customisable data structures.

The literature survey explained that previous research has focused on reengineering data structures with the prime purpose of gaining an in-depth understanding of database semantic structure, and/or providing mapping mechanisms that communicate between relational and object systems. Previous methods have also been reported that facilitate manufacturing domain analysis by defining and modelling different kinds of manufacturing scenario. However little work has been done to create, in a semi-automatic fashion, equivalent object-like data structures thereby facilitating data conversion and reuse as new structures by either IT vendors or manufacturing end users.

3.3 RESEARCH OBJECTIVES

Bearing the above factors in mind, this research has had the following objectives:

(1) Provide system developers with a method and computer tool that facilitates the creation of equivalent object schema by semi-automating data transformations on pre-existing relational database entities.
(2) Support the reuse of the equivalent object schema in application systems that fully exploit distributed object features.

(3) Identify and classify common data structures of legacy systems used in manufacturing, engineering and sales environments to gain a better understanding of common functional and interoperational requirements of these types of legacy system.

(4) Define and develop means of enhancing the performance of systems in which object applications must interoperate with legacy system elements and their embedded relational data.

(5) Provide data structure classifications that can assist in the future design and specification of new wholly distributed object-based systems.

It was decided that particular attention should be focussed on future scenarios in which existing monolithic software, built upon relational database technology, needs to be replaced by more advanced software built on object-oriented database technology. By considering requirements of such an implementation it was assumed that it would prove possible to specify methods and tools that support the configuration and integration of future vendors’ solutions so that they closely meet customers’ requirements. Figure 3.1 illustrates key aspects of these scenarios that have shaped the specification and development of the author’s approach.
Hence other related objectives of this study were determined as follows:

(6) To gain an improved understanding of common data structures (in use in the target domain) in the form of explicit data models, thereby providing knowledge that can be reapplied when customising data structures.

(7) To gain a general understanding of transformation requirements for legacy relational data structures, so they can conform to and fulfil object system requirements.

(8) To facilitate data capture and object view creation (and thence data reuse in distributed object systems) in a semi-automatic fashion.

(9) To identify and test a method that enables the commissioning of ERP (Enterprise Resource Planning) products based on use of data reuse approaches developed by the study.

(10) To investigate the capability and utility of the concepts and tools developed by the project.
3.4 KEY FEATURES ENVISAGED THAT WOULD BE INTEGRAL TO AND/OR PROVIDED BY THE DATA REUSE APPROACH

This project has sought to develop a structured methodology that can help support the development, commissioning and use of new manufacturing software applications that are built upon a base of distributed object technologies. To achieve its goals it was envisaged that the data reuse approach so developed should incorporate the following key features.

**Extracting information about data structures**

Before any data reuse approach can be deployed it was understood that certain details of existing data structures would be needed as input to the approach. Two types of information were expected to be essential, namely: 'semantic structure' of the database and 'data set values'. It was assumed that in general the semantic structure of the database would have previously been determined by the vendor of each software application and that this structure would be common to all customer data sets used to underpin that particular type of software product. Since one objective of the research is to implement the approach in a semi-automatic fashion, all sources of information would need to be accessible by the toolkit and/or the user(s) implementing the data reuse approach. Although all semantic information may not be accessible in an electronic form, it was assumed that mechanisms could be provided to introduce details not available electronically, thereby fulfilling preconditions of the research approach. It was assumed that the prime novelty of the approach would come from its capability to analyse and transform a broad range of information types, since diverse end-user data sets would need to be investigated and transformed using the approach and toolset.

**Creating equivalent object-like data structures**

It was decided that a mapping approach would be defined capable of creating object-like data structures from existing relational databases. General properties of relational databases and data values needed to be investigated to understand how to create an equivalent class hierarchy with appropriate object technology features. Therefore it would be necessary to build upon and extend the best mapping rules reported in the literature. Here it was understood from the literature review that previous mapping rules were not capable of detecting hidden entities and removing redundant tables to
facilitate the process of finding differences and commonalities between end-user data-
sets. It was therefore decided that a mapping algorithm was required with a capability to:

- Capture and define new entities.
- Detect redundant entities.
- Map relational database entities onto object entities and relationships.
- Provide decision support for users of the approach and toolset.

**Define an equivalent OODB**

Chapter 2 described how the main two approaches to connecting relational and object systems have been described in the literature, namely: (1) by constructing an intermediate layer of processing between the existing RDB and the object-based system and (2) creating an OODB which is equivalent to the pre-existing RDB. Solutions involving an intermediate layer between relational and object systems are more likely to be affected by difficulties associated with complex mappings between both data structures. Moreover any changes in the mapping mechanisms supported by the middle layer could have an adverse effect on the performance of the system as a whole. Hence it was considered to be beneficial to implement the object-like structures in an equivalent OODB since it was envisaged that the research approach would define complex mapping rules so that new entities can be detected.

**Transferring existing data records to populate the OODB**

When implementing object-based data structures it was therefore assumed that the new OODB would be populated with data values captured from the Relational Database. Relational records would be reformatted to match the object-like structures and to create data objects. It was decided that this process could and should be supported in a fully automatic manner.

**Abstracting and detecting specific structures of data**

A primary aim of this study was to gain an in-depth understanding of the use of data structures that are common to different manufacturer's requirements as this information could be valuable to the collaborating company, Swan Systems. Because of the need to analyse current vendor solutions that deploy different end users data-
sets, it was important that the data reuse approach should be assessed in the context of customising data structures. Therefore it was decided that different manufacturing scenarios would need to be studied to detect and abstract common properties of data structures and to classify them according to customers' usage. It was assumed that the detection of different structures of data, from predefined classifications, would naturally be facilitated by the way that schema mapping is carried out when detecting entities. It was understood that the data structures so generated might be used to facilitate the creation of new vendor solutions that support specific data structures in a way that closely matches manufacturers' requirements.

3.5 SUMMARY

It follows that the research reported in this thesis should aim to develop and test new ways of reusing data stored in relational data tables of a relational database management system, which itself forms some part of an existing manufacturing software system. The study has required an analysis of current manufacturer data to define and develop object-like data structures representing common customer requirements that can influence the design and provision of future application software products for manufacturing industry by IT vendors. The research seeks to offer benefits to manufacturing supply benefits businesses by:

- Transforming legacy data structures, so that their reuse conforms to object system requirements.
- Seeking to customise vendor data structures, so they match specific requirements of customers.
- Creating an equivalent OODB in a semi-automatic fashion that is supported by a toolkit.
- Detecting substructures of data according to current values and the use of data.
- Providing information that provides an enhanced understanding of systems used.
- Allowing ready access to existing relational data within object programming and operating environments.
CHAPTER 4
CURRENT DATA STRUCTURES AND TRANSFORMATION PROCESSES

4.0 INTRODUCTION

As described in previous chapters the structure of an underlying database system has an essential influence on the performance of software application systems and computer tools. In this research the decision was taken to investigate the reuse of relational database systems within software application systems and computer tools that deploy object technology. Benefits arising from the use of object-based technology within database systems is widely reported in the literature [Goh et al, 1997][Graham, 1995][Vermeulen, 1996][Behm et al, 1997]. This research will also investigate specific reuse requirements when defining and developing customisable software systems.

Some proprietary manufacturing software packages (such as ERP products from SAP and BAAN) take a distinctive approach to customising their software packages. A contemporary approach taken by such vendors is based on an analysis of business processes and subsequent configuration of an ERP model to reflect a class of business process architecture that provides a partial matching to customer needs [Keller and Teufel, 1998]. However the cost involved in configuring and installing these software packages is high and this restricts their wider industrial application. According to Porter this is a primary reason why small and medium sized organisations often deploy 'generic' software which has very limited configurability [Porter et al, 1999]. Bearing these problems in mind one can observe the latent potential of object technology in terms of its inherent capability to facilitate the development of reconfigurable, extendable and reusable solutions.

According to de Heij [de Heij and Caubo, 1996] in manufacturing environments data structures are considered to be more stable than corresponding application
Chapter 4  Current data structures and transformation processes

functionality. Wiggerts describes three scenarios in which objects can be detected as part of existing legacy systems, namely: function driven, data driven and object driven. He points out that the underlying data model often follows the domain more closely than does the functional structure and that the closeness of matching is often influenced by technical design decisions [Wiggerts et al, 1997]. Generally a data model can be described concisely and objectively whereas functionality cannot. Hence the aim of this research is focused on analysing current data structures with a view to enhancing the performance of manufacturing systems as a consequence of the way that they manage the storage of information.

This chapter will review general concepts about current data structures. This will specify a starting point for the approach developed during this PhD. Also key issues about data transformation processes involved in creating an equivalent OODB will be discussed to illustrate the main concepts on which the developed method will be based.

4.1 ASSUMPTIONS AND CONSTRAINTS RELATED TO CURRENT SOFTWARE APPLICATION SYSTEMS AND COMPUTER TOOLS

Any approach to the reuse and integration of legacy software within object-oriented environments will be conditioned by the degree of modularity (i.e. extent of decoupling) of elements which comprise the old system. Clearly it will be easier to migrate legacy software in which the main classes of component are well decoupled from each other than it will be for the case where software elements are not readily separable. Following Brodie’s reasoning [Brodie and Stonebraker, 1993] it is evident that the reuse and integration of legacy software can be attempted from three distinct and different starting points as follows.

Case I: in which none of the old system elements are separable. It is likely that such a system would need to be translated as a single reusable element via use of some kind of wrapper, which encapsulates the complete legacy system, i.e. as a single system building block or element. Inside the wrapper will be an interface that organises and realises communication between old and new system elements.
Case II: in which database and application elements are not separable but there may be loose coupling between certain elements that themselves comprise closely coupled data and application code. This starting point can be viewed as being similar to that of Case I, as design and implementation constraints are likely to arise because of the close coupling between data and application code that will limit the final OO solution and its performance.

Case III: in which databases, application and interface components are essentially self-standing. Under such conditions relational database elements of a legacy system can be independently translated into equivalent object oriented database elements of new and more readily configured and extended systems.

During recent decades advances in software and system engineering techniques have promoted a separation of application and user interface elements from information elements [Bamford and Curran, 1991]. Indeed the development and industrial acceptance of relational database technology, along with associated standards on database management and access have provided a platform for developing a common approach to promoting such a separation. Nonetheless according to Ganti [Ganti and Brayman, 1995] inappropriate structural organisation within a system has resulted in many contemporary software systems having data and application elements interwoven throughout the system, thereby raising significant barriers to the reuse and integration of existing systems. Thus, increasing the degree of decoupling between data and applications within system building blocks will result in performance enhancements in systems where they are used, namely by enabling: (1) improved degree of fit (and hence the effectiveness) of the business and production processes they support; (2) increased system responsiveness by reducing the time needed to realise system and process change, and; (3) improved access to information [Graham, 1995] [Bamford and Curran, 1991].

In contemporary industrial and commercial software applications, access and update of data stored in relational databases is typically achieved via queries expressed in a Standard Query Language (SQL). Essentially this type of Legacy Software can be considered to map onto a case which is at some intermediate position between the cases II and III as categorised above. In such cases potential benefits such as those listed in (1) through (3) above can be realised by moving from the use of a relational
database to an object oriented database, as the underlying means of organising data and separating it out from application code. For example this could lead to both “global” and “local” system benefits, since the modularity of the system allows “local” databases and applications to be separately defined and utilised. The translation process involved in moving from the use of relational to object oriented databases can thus be viewed as migrating further from case II towards case III.

For software systems that contain application modules with a degree of independence and maintain a separation between application code and a relational database, potentially additional benefits can accrue with respect to (1) through (3) through increased independence by deploying them as integral building blocks of object oriented systems. However, to achieve this and thereby improve the use of legacy systems it is necessary to devise suitable data translation processes. In this research study focus has been on data translation processes required to migrate elements of systems built on a RDBMS into elements of systems built on OODBMS.

4.2 A LITERATURE SURVEY ON DATA TRANSFORMATION PROCESSES

Chapter 2 reported on approaches designed to transform data stored in a RDB into a suitable form for use in an OODB. The main benefits of deploying an approach based on use of an intermediate processing layer located between relational data and object applications were discussed in Chapter 2. Russomanno [Russomanno, 1996] determined three evaluation parameters to decide between integrating a RDB in an object-based system or replacing the RDB. The parameters identified relate to: (1) the effectiveness of the relational application in terms of suitability of its technology and the capital invested, (2) the gap between new requirements and available functionality in the existing system, and (3) semantic differences between data models of the source and the target systems.

Inspection of the literature shows that the transformation processes involved in creating an OODB that is equivalent to an existing RDB can be divided into two

---

11 A modular system should enable both (a) the autonomy of a particular element of the system to be maintained, e.g. to improve its local effectiveness; and (b) the maintenance of interdependencies between system elements to effect global integration.
distinct stages, namely schema transformation and data population. Most approaches reported in the literature only support the schema transformation stage. For example Monk [Monk et al, 1996] tackles the problem of schema translation from relational to object structures. Classes are identified by investigating the nature of relational schema information and by detecting a hierarchy based on common attributes shared between classes. Monk developed a prototype tool to support schema transformation in a way that can be modified by a user, who therefore can modify or add attributes to the resultant schema. However Monk’s approach does not address the problem of transferring data to a target OODBMS.

Juric [Juric and Martin, 1998] also describes a mapping approach capable of transforming data structures of a relational database into a suitable form for use in an OO environment. Juric’s algorithm classifies relational tables, attributes and attribute relationships in order to draw an OO structure. To facilitate operation of this algorithm, assumptions have to be made about what information is available from the relational database dictionary. Mapping rules are not fully automated and expert user intervention is required. However, actual migration of data from an RDBMS to an OODBMS is postponed to a later stage.

A more complex approach to defining an object view, which is capable of achieving a complete database transformation, is described by Jahnke [Jahnke et al, 1996]. Jahnke’s approach emphasises semantic extraction so as to detect all the relational features (tables, attributes and keys) that will be used during the mapping process [Jahnke and Heitbreder, 1998]. Key attributes are deduced by analysing application code and SQL code. This information is later use to provide schema mapping rules that facilitate object view creation. Schema transformation, within the migration environment proposed by Jahnke, is based on the use of an adaptable set of schema mapping rules. Each mapping rule defines how a specific construct is mapped into an equivalent construct within the ODMG (Object Database Management Group) schema. An initial mapping is computed based on (user defined) priorities associated with a set of alternative mapping rules. In addition the database designer has an opportunity to edit any part of the ODMG schema. Users can add new attributes and/or classes and drop existing schema parts. A toolkit was developed to support
Jahnke’s approach which re-establishes dependencies between both schemas in a semi-automatic way [Jahnke et al, 1996].

Each of the approaches described above support some aspects of data transfer between relational and OO databases. However a transformation approach that supports both schema translation and data population stages is described by Behm [Behm et al, 1997]. Behm’s schema transformation is an iterative process designed to create an object schema by applying a predefined set of transformation rules that act upon the relational semantic information. Data migration is divided into two separate processes. During the first step (called an instance creation process) instances are created. While in the second step (called the attribute assignment process) values are assigned to attributes of instances.

Table 4.1 characterises the approaches reported in the literature with respect to how they support database transformation processes.

All approaches reported in the literature make use of relational features such as tables, attributes and keys relationship, by using these as the basic inputs for a mapping approach. Also schema mapping approaches are described that are used to define a set of classes. Relationships between classes are also described in most approaches in order to support OMG (Object Management Group) data model creation, namely through applying rules related to association, aggregation and hierarchy. Even Jahnke and Behm also provide a mechanism to split tables into two different classes by grouping a set of attributes to define a new entity. However, based on the literature review and the author’s experience of building systems it became evident that the generalised use in manufacturing domains of the approaches reported in the literature might be further improved by providing mechanisms that:

(1) Facilitate the use of actual data values, to help users to take decisions about the final object view.

(2) Enhance the user’s ability to detect candidate attributes when forming a new entity.

(3) Support the actual transfer of data values from the relational database to the equivalent OODB.
(4) Provide a suitable environment that semi-automatically realises all of the data transformation processes required.
## Current data structures and transformation processes

<table>
<thead>
<tr>
<th>Input information required</th>
<th>Mapping rules</th>
<th>Equivalent object-oriented schema</th>
<th>Implementation</th>
<th>Data population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monk</td>
<td>Iterative process to distribute entities in a hierarchy based on common attributes.</td>
<td>Initial class hierarchy based on common attributes. No associate and aggregation relationships. No new entities</td>
<td>Prototype tool that implements the mapping rules to create an initial class schema. This class schema can then be modified by adding or deleting attributes to the resultant schema.</td>
<td>Not supported</td>
</tr>
<tr>
<td>Juric</td>
<td>Algorithm based on attributes properties and relationships.</td>
<td>A set of classes related by inheritance, aggregation and association relationships. No new entities.</td>
<td>Software program developed to test the mapping algorithm.</td>
<td>Not supported</td>
</tr>
<tr>
<td>Jahnke</td>
<td>Adaptable set of mapping rules that are selected based on a priority classification.</td>
<td>Equivalent ODMG compliant schema. New entities can be introduced by grouping a set of attributes into a new class</td>
<td>Toolkit that implements the data transformation processes. Users can extend, edit and/or remove parts from the final class schema. An extraction tool supports an analysis of the relational database to infer relational properties</td>
<td>Not supported</td>
</tr>
<tr>
<td>Behm</td>
<td>Set of mapping rules based on relational semantic information.</td>
<td>A set of classes related by inheritance, aggregation and association relationships. New entities can be introduced by grouping a set of attributes into a new class</td>
<td>Toolkit to support the schema transformation processes.</td>
<td>Data transfer is created in two steps: first instances are created, then the OODB is populated. The toolkit supports the data transfer processes automatically.</td>
</tr>
</tbody>
</table>

Table 4.1 Summary of data transformation approaches described in the Literature.
4.3 CLASSIFICATION OF DATA TRANSFORMATION PROCESSES AND ASSOCIATED OPEN RESEARCH QUESTIONS

The literature review identified two types of data transformation process that are common to many data reuse scenarios, namely: schema translation and object data population. Figure 4.1 characterises essential features of these transformation processes, requirements of which are described in the following sub-sections.

![Figure 4.1 Common issues in data transformation processes.](image)

**4.3.1 Schema translation**

A systematic way of creating a class hierarchy is required that classifies relational tables into an equivalent object-like structural view. This will require a definition (in terms of the name) of objects in a set of classes, a definition of the attributes that characterise each class and a definition of relationships between classes. Collectively these definitions can be used to build a hierarchical data object structure. First it will be necessary to extract semantic information about the data stored in the relational database and to use this as input during schema translation. In this study it was assumed that (based on the use of current understandings and enabling technology) any fully automatic approach to object-oriented schema creation would most probably fail to detect all relevant information. This is because there are known general semantic differences between models of relational and object data in typical use during the runtime operation of conventional and distributed object software systems. Details about relation schema are generally made available within a runtime system. Indeed information about tables, attributes and keys is typically available via a relational database management system and therefore can readily be extracted.
automatically. However because of general semantic differences it was assumed that in general additional knowledge will be required which may have previously been defined as part of the original requirements specification or during the design of the original system and its RDBMS. It was also understood that in many practical situations this additional knowledge may not have been documented into a reusable form. Therefore it was assumed that in the general case it will be essential for a human expert to be involved in creating the equivalent object schema. On the other hand it was assumed that software tools should be capable of systemising and supporting activities carried out by expert humans, thereby improving the quality of the schema generated and removing the need for any mechanistic processing. However it became evident that to assess the utility of a computer-assisted approach to object schema creation answers were required to the following open research questions.

- What are the principal characteristics of the semantic information required to translate relational data schemata into equivalent object-oriented schemata for reuse in a distributed object system?

- What types of information are needed and what are the most suitable sources of this information, (i.e. what information fragments might best be extracted from the original runtime system or its documentation or input by a human expert) during object schema creation. Also might these information requirements be influenced by differences between application scenarios (e.g. different ways in which the distributed objects are required to interact with the relational data and/or its database management system)?

- Can legacy relational data be well documented so that it facilitates object schema creation? If so what form should this documentation take and can it be generated retrospectively?

- Can the nature of semantic differences between relational and object-oriented models be mapped onto decision types taken by a human expert during object schema creation? If so, to what extent might this knowledge help in predicting the benefits possible from use of a schema translation tool?
A further objective of the research was to investigate how manufacturing data systems can deploy customisable data structures. Hence choice of the schema mapping process will determine the ability of the approach to develop data structures that match closely customer requirements. To implement customisation capabilities it was assumed that the approach would not only need to support an investigation of schema structures but also to utilise understandings gained from analysing data values. Therefore the research would need to develop new understandings about the following issues.

- Since data structures that underpin a single vendor product are common to all customers using that product it was understood that a schema mapping approach based only on an analysis of relational semantic properties will not be able to generate class structures that closely meets specific customer requirements. Therefore other sources of information would need to be taken into account. Hence, the assumption that it would be necessary also to analyse the use of real data structures to determine how a given vendor solution and database can best fit specific manufacturer requirements?

- If it were proven to be the case that an analysis of data and schema properties can support the generation of equivalent classes of schema that closely match specific customer requirements, next the extent to which class schemas differ between different customers would need to be studied. Additionally it will be necessary to determine common elements shared by these schemas and how these common elements can be used to support the generation of customisable structures.

4.3.2 Object data population

The purpose here is to populate an equivalent object schema with corresponding fragments of original information stored in the legacy relational tables. An object can be defined by assigning values to its two tuples, namely: attributes and relationships. Obviously a set of object classes will need to have been created in advance so that properties of objects can be valued. Impedance mismatch between attribute types in the source (relational) and target (object) environments was expected to introduce added problems when designing and providing computer tools to support some data reuse scenarios. In the context of this study the main research issues involved in object population were:
- Understanding the nature of the impedance mismatch between fragments of RDB and OO data models, as this was expected to influence the way that a source data is interpreted and mapped into the target data model.

- Investigating ways of systemising the population of objects so that data integrity is maintained (e.g. mechanism to enable the correct introduction of objects to define a super/sub class relations without duplicate values in the hierarchy of objects).

4.4 SUMMARY

Two main processes have been defined with respect to the transformation of relational databases into OODB. A schema translation process is needed to create an equivalent OO schema. This process requires access to various types of information about the relational schema. Assumptions have been made about how much of this information can be made available. However no method previously reported in the literature utilises data values in support of its schema mapping approaches. Moreover automated support for data population is not widely supported by existing methods.
CHAPTER 5

AN APPROACH TO TRANSFORMING RELATIONAL DATA FOR ITS REUSE IN OBJECT-BASED SYSTEMS

5.0 INTRODUCTION

The primary aim of this study was to specify and develop an improved approach to the transformation of data contained within a legacy RDB into an equivalent OODB data structure and data values that can be reused in object-based systems. Hence appropriate algorithms needed to be devised and developed to analyse existing real data tables and to support the construction of equivalent object-oriented structures. A proof-of-concept-tool also needed to be designed, developed and tested to support the organisation and creation of equivalent class structures, to define and implement an equivalent OODB and to reuse existing relational data values when populating an OODB. A secondary aim of the study was to assess the practicality and benefits arising from applying the developed approach to customisation problems found in the domain of enterprise resource planning (ERP). Therefore project methods and tools so developed needed to support the reuse of existing data found in manufacturing systems. Here it was decided that new understandings would need to be developed concerning the reuse of knowledge about actual data values, in addition to the reuse of knowledge about data structures. It was assumed that the knowledge would be reused in support of configuring equivalent object-based structures in order to alleviate constraints arising from mismatches occurring between vendor solutions and end-user requirements. Here it was also assumed that use of a schema mapping algorithm would help to detect differences between data structures through making reference to predefined manufacturer classifications. Therefore new capacities needed to be specified, developed and included into the approach and software toolset. Figure 5.1 illustrates how it was envisaged that the research requirements would be met by developing a data reuse approach and toolset as part of this PhD study.
This chapter describes how the approach was conceived and developed. It also explains how it was envisaged that the approach would be applied in support of legacy data reuse and data structure and data customisation in manufacturing software systems.
An approach to transforming relational data for its reuse in object-based systems.

Figure 5.1 Requirements and development concerns envisaged as being of prime concern in the development of the project methods and toolset
5.1 A GENERAL APPROACH TO THE TRANSFORMATION OF RELATIONAL DATA STRUCTURES INTO EQUIVALENT OO DATA STRUCTURES

An algorithm for translating relational schemata is described in this section. This algorithm was conceived and developed within this study to offer an enhancement on earlier methodologies. The enhancements made concern the way in which information in existing relational databases and data structures is processed in order to draw out object schemata, add new object classes and remove redundant object classes.

As explained in chapter 4, in general the process of transforming data in a relational database into equivalent data in an OODB involves two main issues, namely, (1) schema translation, and (2) data transfer. The approach described addresses both of these issues by semi-automating the generation of OO schemata and, subsequent on the definition of a suitable OO schema, automatically transferring the data from a source RDB to a target OODB.

5.1.1 Schema translation

The literature describes various means by which a class hierarchy can be drawn from relational schema. For example Johannesson [Johannesson, 1994] and Chiang [Chiang et al., 1994] described the building of an extended ER model from a relational database. Castellanos [Castellanos, 1993] defined a canonical object model where different kinds of relationship exist between entities derived from a relational schema. All of these methods deploy information about inferences between keys and inclusion dependencies to determine an entity hierarchy. Moreover Fahrner [Fahrner and Vossen, 1995] describes a method for drawing out an object schema based on the identification of inclusion and exclusion relationships between attributes. This method provides a resultant object schema that consists of a hierarchy of classes joined by inheritance relationships. Here classes can have complex attributes that refer to other classes. Also Vermeer [Vermeer and Apers, 1995] described an approach to constructing an object view of a relational database. An improved algorithm was proposed by Ramanathan [Ramanathan and Hodges, 1997] who described the
development of an object schema by drawing out relationships between classes based on associations, aggregations and inheritance properties. The object schema drawn by Ramanathan leads to a richer resultant schema than that generated by the approach of Fahrner and Vermeer since due account is taken of a classification of the keys and inclusion dependencies when determining the structure of the class hierarchy. Moreover methods described by Vermeer and Johanesson and Castellanos identified means of introducing new entities from candidate keys.

Section 4.2 provided an overview of schema mapping approaches developed to support database transformation into equivalent OODBs. In these approaches, mapping rules were based on relational semantic properties as summarised by table 4.1. However, bearing in mind current and emerging requirements of manufacturing systems this thesis argues that previous schema mapping approaches fail in the way that they handle the inclusion of new entities and table splitting. Furthermore, as a consequence, it is argued that those approaches will not provide adequate structure and support for the reuse of relational data within next generation distributed object-based manufacturing systems.

Therefore a new schema translation algorithm was conceived and developed that builds on and enhances the capabilities of previous approaches earlier described.

In manufacturing environments, where relational data systems are typically developed over a period of time and in a relatively ad hoc manner, a common requirement will be to detect hidden entities. Otherwise a sub-optimal schema view will be generated. Indeed without taking due account of all entities, any schema generation method cannot (a) accurately generate mappings between classes nor (b) categorise relationships in an effective manner. Whereas by including a capability to detect vertical splitting tables, the removal of redundant tables can be facilitated and thereby the design of resultant object schemata can be much improved. By taking due account of these additional requirements the new algorithm was developed so that its application involves four steps. During the two first steps the inclusion of new entities and table splitting is handled, whilst during the last two steps the class hierarchy is generated.

Hence to enhance current data transformation approaches and meet the research
objectives, two types of information need to be made accessible:

- Database semantic. Access to information about ‘features’ of a relational database (such as tables, attributes, and keys) is a necessary prerequisite of the mapping approach when creating a hierarchy of object classes.

- Data entities. Information about ‘data values’ and properties of the distribution of these values within a data structure need to be made available to detect new entities and subsequently to define customisable data structures.

Figure 5.2 illustrates the enhanced schema mapping process conceived. Firstly semantic information is extracted from the relational database. Here it is necessary to reveal the ‘structure’ of the data tables in terms of table names, type and name of their attributes, primary keys, foreign keys and additional dependencies. Current generation relational database management systems support the extraction of semantic structure and typically store this type of information in a database dictionary. However the ‘richness’ of the information structures made available for subsequent data extraction purposes will depend on the capabilities made available by the database management system deployed. For example, the use of conceptual models, like EXPRESS or EER, can be used to facilitate a richer semantic extraction during software system design and implementation than is possible with some other conceptual models [Vernadat, 1996].

![Figure 5.2 Layout of resources involved in the schema translation process](image)

The information extracted about the ‘structure’ of an existing relational database can then be processed in order to construct a class hierarchy. The four-step algorithm was
conceived and developed to semi-automatically generate this class hierarchy. The first two steps of the algorithm support an identification of OO model entities in such a way that the removal of vertical splitting tables is organised and facilitated and thereby a consolidated set of new entities is generated in a methodological and semi-automated manner. It is mainly during the first two steps that information about data values is analysed to support the application of algorithmic rules. In general human user intuition and decision-making was found to be necessary during the first two steps of the mapping approach. However it was found to be practical to fully automate subsequent steps.

**Step 1. Removing the occurrence of vertical splitting tables**

During the first step of the schema mapping algorithm it is necessary therefore to detect the occurrence of vertical splitting tables. The occurrence of redundant tables in legacy relational data is often a result of a loss of semantic information (i.e. aspects of the meaning of the data are lost) following implementation and/or re-implementation of a legacy system design. Generally only a fraction of the original design semantic is retained and is accessible at system runtime. Subsequently during the operational lifetime of a manufacturing system data tables may be added to the legacy system in an ad hoc way, e.g. to facilitate system development including for example the development of new applications that access part of the original data but require additional table entries. However, because full details of the original design intent are not retained, often redundant data entities and data relationships are added [Johannesson, 1994]. The deletion of these tables is justified since a resultant enhancement to the relational model will normally result in a more compact data model that facilitates more efficient runtime operation.

The four-step algorithm was designed to detect the occurrence of a vertical splitting table if the following conditions occur.

All the attributes in a table match in both name and type with a subset of attributes of another table (or all the attributes in a table are foreign keys that refer to a subset of attributes of another table) and its primary key (PK) is also a foreign key (FK) that refers to this table. Both tables are then unified into a unique new table. Records in the new table will be defined as the set of records of the original two tables, thereby
removing any occurrence of duplicate records. The process is illustrated by Figure 5.3.

Table 1

<table>
<thead>
<tr>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Attr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>+++</td>
<td>+++</td>
<td>£££</td>
</tr>
<tr>
<td>VVV</td>
<td>+++</td>
<td>&amp; &amp; &amp;</td>
<td></td>
</tr>
<tr>
<td>»&gt; xxx</td>
<td>$$$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>+++</td>
<td>***</td>
</tr>
<tr>
<td>VVV</td>
<td>+++</td>
<td>&amp; &amp; &amp;</td>
</tr>
<tr>
<td>»&lt; $$$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input information: Name of tables, name and type of attributes. Data record values. User decision

Precondition:
- all the attributes in Table 2 match in both name and type with a subset of attributes of Table 1.
- the PK in Table 2 is also a FK that refers to Table 1.

Action: User decision may be required to remove the detected redundant table. If a decision is taken to remove a redundant table, then Tables 1 and 2 are unified to define a unique New Table. Record values are added to the New Table in a way that avoids duplication and the occurrence of inconsistent data.

Output: A modified set of relational tables.

Figure 5.3 Removing the occurrence of vertical splitting tables

Step 2. Identifying missing entities

During step 2 of object schema generation new entities can be introduced by splitting a table of the original relational schema. Based on schema translation approaches reported in the literature and on an analysis of common schema structures used in manufacturing systems the new approach was designed to support the introduction of such entities under two different scenarios, namely:

- (1) A new entity can be introduced if a candidate key is detected in a table. A candidate key is considered for any foreign key unless it is also a one-attribute PK or part of a PK that has all FK attributes that reference another table. In cases where a new entity is required the table must be split to create two different tables. Attributes should be distributed between the two new entities adding also the foreign key properties that identify a relationship between the entities. Hence,
attributes that define the new entity will be the candidate key (that will be defined as PK) and all other attributes that have a functional relation with the candidate key. Decisions as to whether more attributes need to be transferred from the original table to the new entity are left to a human developer. All these attributes will be deleted from the original table and one attribute will be added to define a relation between the original and the new entity. Also since the PK of the new entity was a FK in a third table (thereby meeting a pre-condition needed to be a candidate key), then there will be an inheritance relationship between the new table and the third table. Figure 5.4 illustrates this process by way of an example.

Figure 5.4 Identifying missing entities by candidate keys

- (2) The second scenario concerns the introduction of new entities based on an analysis of data values. Here reference is made to the way in which attributes have been populated with stored values or have been left null (or empty). The algorithm is designed to detect tables that have null values for two or more attributes (or for
two or set of attributes) and the total number of records found in a table can be divided into two or more distinct sets of records such that the null values concerned were introduced only for one of the attributes (or set of attributes). It follows that this type of table can be mapped onto a hierarchy of classes, for which subclasses are defined for each attribute (or set of attributes) where null values have been detected that meet the requirement explained above. The original table is mapped as a superclass within the hierarchy. Record values can then be distributed between the two subclasses. Figure 5.5 illustrates the processes involved.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attr1</td>
<td>Attr2</td>
</tr>
<tr>
<td>111</td>
<td>aaa</td>
</tr>
<tr>
<td>222</td>
<td>bb</td>
</tr>
<tr>
<td>333</td>
<td>aa</td>
</tr>
<tr>
<td>444</td>
<td>bbb</td>
</tr>
</tbody>
</table>

Input information: Name of tables, attributes. Value data.

Precondition: Table with attributes from which null values are detected for two or more attributes (or a set of attributes) and the total number of records found in a table can be divided into two or more distinct sets of records such that the null values concerned were introduced only for one of the attributes (or set of attributes).

Action: The table is split in the following way, viz.:
- Two or more new entities are created from each set of records that meet the precondition. Each new entity is defined by one set of null value attributes.
- The original table will be mapped as a superclass of the new entities. This class will have all attributes of the original table less those defined by the set of null value attributes.

Output: A modified set of relational tables.

Figure 5.5 Identifying missing entities by attributes with null values

Step 3. Mapping to a class
During step 3 all tables (original and new tables defined as a consequence of step 2) should be mapped onto independent classes. One exception was found in the case of tables that have a primary key made up of more than one attribute and where all of those keys are foreign keys referring to different classes. In such a case, classes are mapped as association classes. An exception to this latter case can be found when a table has attributes that are not a PK. Here a table is mapped onto a class. Classes will
be defined with the same name and attributes to those found in the equivalent table. Attribute domains will be mapped onto equivalent types in the object environment [Cooper, 1997]. The process is illustrated by Figure 5.6.

![Table](Image)

**Table I**

| Text Attr1 | Integer Attr2 |

**Input information:** Name of tables, name and type of attributes.
**Precondition:** Relational tables that do not conform to all of the following conditions, viz.:
- The primary key is made up of more than one attribute where all of them are foreign keys referring to different classes.
- All attributes are part of the primary key.
**Action:** Define a class from each table where the name and attributes of each class are the same as those in the original table. Attribute types are mapped onto equivalent types in the object environment.
**Output:** Set of classes.

**Figure 5.6 Mapping a table to a class**

**Step 4. Determining relationships**

Following step 3 the object classes of the equivalent object schema will have been determined. Hence, in step 4 relationships between these classes can be established. In general three types of relationship exist between OO classes, namely, inheritance, aggregation and association as defined in the OMG standard [Cooper, 1997].

- **Inheritance**

When two classes share a common primary key an inheritance relationship is drawn between them (as illustrated by Figure 5.7).

![Figure 5.7 Determining an inheritance relationship](Image)

**Input information:** PK and FK attributes of tables.
**Precondition:** Tables that have the same primary key
**Action:** Define an inheritance relationship between the equivalent classes of those tables.
**Output:** Inheritance relationship between classes.
Chapter 5  An approach to transforming relational data for its reuse in object-based systems.

- **Aggregation**
  For classes that have more than one attribute in the primary key and where at least one of them is not a foreign key an aggregation relationship will exist with the class for which the primary key corresponds to the largest subset of foreign keys. The former class will be determined as a contained class and the latter as a container class. Cardinalities are established as 1:1 when both classes contain a reference to each other. This design decision can be justified since the relationship is determined by PK attributes that have unique values. Otherwise a 1:m cardinality is drawn. The relationship will be determined by adding an attribute in the container class to refer to objects in the contained class. The type of this attribute will depend on the cardinality drawn. The processes involved are illustrated by Figure 5.8

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Attr11</th>
<th>Attr12</th>
<th>Attr13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2</td>
<td>Attr2</td>
<td>Attr22</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: Table1</th>
<th>Attr11; Attr12; Attr13; (Contained class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class: Table2</td>
<td>Attr2; Attr22; AttReferenceCh; (Container class)</td>
</tr>
</tbody>
</table>

**Input information:** PK and FK attributes of tables.

**Precondition:** Tables that have more than one attribute in the primary key and where at least one of them (but not all of them) is a foreign key.

**Action:**
- Define an aggregation relationship between these classes.
- Define cardinalities of the relationship as 1:1 when both classes involved in the relationship contain a reference to each other. Otherwise a 1:m cardinality should be drawn.

**Output:** Aggregation relationship between classes.

Figure 5.8 Determining an aggregation relationship

- **Association**
  An association relationship between two classes is determined for each foreign key unless it is possible to draw out either aggregation or inheritance relationships between these two classes. 1:1 cardinality is established when both classes reference each other. Otherwise the cardinality should be 1:m. It should be pointed out that cardinalities of m:n are represented by association classes. The relationship will be determined by adding an attribute in the class with multiple cardinality that refers to
the class that originally contained the FK and by changing the original FK into a reference to the associated class. Figure 5.9 illustrates the processes involved.

Input information: PK and FK attributes of tables.
Precondition: Tables with FK attributes that do not determine another kind of relationship.
Action:
- Define an association relationship.
  - Define cardinalities of the relationship as 1:1 when both classes involved in the relationship contain a reference to each other. Otherwise a 1:m cardinality should be drawn. M:n cardinality are represented by association classes.
Output: Association relationship between classes.

Figure 5.9 Determining an association relationship

5.1.2 Object Data population
Having semi-automated the establishment of object classes and relationships the resultant object-like schema can be used to define an Object-Oriented Database. Classes in the OODB can then be populated with data records originally stored in the source RDB. These persistent classes will have two types of attribute, those with simple primitive types and those that represent a relationship. Generally the latter will store objects of some other class in the database. When transferring data the difference between these attribute types must be noted as each type will have to be processed differently. Simple primitive attributes will store primitive types with the same value as found in the original table. However relationship attributes will store objects of some class of the class structure. Hence, data population should be processed by first populating and defining each class with objects where only simple attributes have been populated. Afterwards relationship attributes can be valued by referring to the objects that match the relation.

New entities introduced during the step 2 of the algorithm require special attention. Records from the original table should be distributed between the correspondent classes as illustrated earlier in Figures 5.4 and 5.5. Also classes with inheritance relationships should be examined to avoid the introduction of duplicate values between superclass and subclass objects. Therefore object values should be checked...
and if an object with the same values is found in the superclass and subclass, the object representing the superclass should be removed. Figure 5.10 illustrates the data population process where inheritance and association relationships have been found.

Figure 5.10 Object data population process

5.2 A METHODOLOGY IN SUPPORT OF CUSTOMISABLE VENDOR SOLUTIONS

When specifying and developing a methodology capable of supporting the customisation of vendor software, it was considered to be necessary to build on an understanding of how data structures that are common to different manufacturers are used in the domain of ERP. Based on such an understanding the customisation processes can be facilitated by defining and classifying two types of model of data structure requirements. One model type represents data structure requirements that are common to all customers and the other model type includes reference data structures that represent requirements of various groups of customers. Hence by deploying solutions with underlying data structures that match known companies preferences this project has sought to specify and generate reference data structures that have a dual capability to support vendor and manufacturer end user requirements when implementing and deploying object-based systems.

Prior to methodology design, therefore, ideally a wide ranging investigation and
analysis of existing ERP data structures, as currently supplied by vendors and deployed by various end users, would have been undertaken. However, in the context of a single PhD study, it was only practical to access and analyse the use of data by a limited number of end user companies. Furthermore it was only practical to focus project investigation on data structures used by end users of a single ERP product, namely an ERP product supplied by the collaborating vendor company, Swan Software.

A natural consequence of this decision was a problem simplification because the semantic of the data structures deployed by all the end user manufacturers investigated was essentially similar; being the data structure being defined and provided by a common product vendor. Hence residual differences between the end user data sets\(^{12}\) studied arose out of the way in which users deployed the underlying data structure. To help analyse such differences it was decided that a nominal classification of manufacturer users should be made, which reflected typical types of end user business. However it was assumed that the precision with which this classification would be developed was not a major concern in the context of this particular study. Rather it was considered to be important to the extent only that it demonstrated that it was practical and useful to refer to such classifications when interpreting alternative ways of deploying a common data structure. Indeed it was understood that various classifications of end user businesses could have been made, such as by grouping them in terms of typical processes they enact, common resources they require, the type of organisational structures they deploy, and so forth. Hence to focus this study, data sets selected for analysis meet two main requirements, namely that they were:

(1) derived from a common data structure, which was embedded by the collaborating vendor into a single ERP product. Essentially this product was considered to be a general-purpose solution to end user problems in the ERP domain and in this

\(^{12}\) A data set is one that is populated by an end-user in conformance with the data structure provided by the vendor’s applications.
study differences in data structure and data sets were considered to result from customising a ‘generic’ data structure.

(2) representative of a semi-generic class of data structure that corresponds to some specialisation of the ‘generic’ data structure to meet semi-generic requirements of the class (or type) of end user company concerned.

With this focus of investigation in mind it was envisaged that an appropriate number of end user data sets would be analysed in order to identify key properties of ERP data structures at three different levels, namely: ‘generic level’, ‘group level’, and ‘particular level’ as defined in the following.

Generic level: At this level the research sought to identify and develop the use of a generic data reference model of a single vendor solution. The aim here was to develop a reference model based on an abstract representation of the AS-IS structure of the Swan ERP product. Through applying such an abstraction process it was also assumed that, with reference to certain pre-conditions, it should prove practical to abstract and represent many vendor data structures into the form of a ‘generic’ reference model. Further it was assumed that other forms of reference model could be generated by applying schema transformation methods and processes developed by this research and by using as input the legacy relational data structure provided by ERP vendors (as illustrated conceptually in Figure 5.11). However it was envisaged that data values would not be required when carrying out the abstraction processes involved. It was also assumed that the generic reference model so generated could provide an object-like view of any current data structure underpinning the product of a single vendor and could also be used to enable comparisons to be drawn with other data structures, such as those used by groups of end user companies at the group level or those used by other vendors to underpin different ERP products.

13 The term ‘generic’ is used here to emphasise a general-purpose solution that can be configured to meet a range of similar end-user requirements.

14 The term ‘semi-generic’ class is used here to indicate a degree of specialisation of the generic data structure to represent the data structure needs of some grouping of customers.

15 The term ‘generic reference model’ was used to refer to a common data model that characterises commonalities and variability found within ‘semi-generic’ or ‘group’ data models.
An approach to transforming relational data for its reuse in object-based systems.

Transformation Process

Relational "AS IS" vendor data structure

Object oriented generic reference model

Figure 5.11 Extraction of a generic object-based structure from the data structure underpinning a single vendor's solution

Group level: At this level the research sought to identify and classify data structures representing and characterising common data structures and data sets found within example groupings of end user companies. It was assumed that such a data structure definition can be derived from a knowledge of similarities between data structures and data sets used by similar types of manufacturing company that deploy a common vendor product. Here it was envisaged that the data structures used by individual companies could be transformed into an object-like structure (using the transformation processes and methods developed by this research study) to facilitate the identification of common properties and thus a classification of end user data structures and data sets into group level data structures. Here it was assumed that both data structures and data values would need to be studied to extract common properties of data structures used by a group of companies. Hence it was assumed that results from different manufacturing data sets could be compared to extract a data structure that characterises and categorises how a group of companies apply an ERP product in a similar way. Transformation processes involved at the group level are illustrated conceptually by Figure 5.12. Reference back to the Generic Reference Model should enable comparisons to be drawn between data structures used by different groups of companies.

\[\text{The use of a similar ERP software by a group of companies was considered to correspond to a common application scenario as defined by Porter [Porter et al., 1999] and de Heij [de Heij and Caubo, 1996].}\]
Chapter 5  An approach to transforming relational data for its reuse in object-based systems.

Transformation Process

Comparison

Assessment

ManufacturerA data structure

ManufacturerB data structure

ManufacturerC data structure

Outputs for single data sets.

Industry categorisation data structure

Figure 5.12 Extraction of partial level data structures for one class of end user

Particular level: This level of investigation relates to understanding properties of specific data structures used by individual end user companies. The transformation methods and processes developed in this study can be used by or for individual companies with two objectives in mind, namely: (1) to provide manufacturers with new understandings about data structures suitable for use in object-based systems and (2) to migrate actual data stored in old (legacy) systems into new object-based systems. The transformation processes involved here are illustrated conceptually in Figure 5.13.

Transformation Process

Implementation

ManufacturerA data structure

Class hierarchy for a single customer

Equivalent OODB for a single customer

Figure 5.13 Transformation processes for a single customer

On the assumption that a suitable data transformation approach and toolset can be developed it was envisaged that benefits would accrue for both vendors and manufacturer users as a consequence of providing:

- Group level data structure models that correspond to different industry scenarios.
- Multiple level data structure models that assist in the definition of new systems that deploy object-based technology.
- Means of defining multi-level data structures that take as input AS-IS data
structure information.

- Mechanisms that assist individual manufacturers and their system builders in migrating and implementing new data structures.

- Mechanisms to transfer actual data instances between old and new systems

Hence a data transformation approach and toolset were developed to semi-automatically transform existing relational data structures into three levels of object-base view as defined above.

5.3 SUMMARY

A data transformation method was conceived and associated processes defined. The resultant transformation approach can systemise and support the translation of relational data into equivalent OO data structures that can be deployed within systems built using object-base technologies. The approach focuses on scenarios in which legacy data, stored in a target RDB, is to be reused. The approach can also be used to help interpret properties of different data structures and data sets and thereby gaining new understandings about these structures and sets that allow them to be classified and developed in useful ways. Here, the method provides output data structures in a form that enables human users to gain in-depth understanding of the potential uses of the data structures and thereby leads to the definition of customisable data structures.

Moreover, a translation algorithm that underpins the method and translation of legacy data has novel features in respect to the way in which it:

1. determines relationships between objects,

2. supports the splitting of tables,

3. transfers relational data from relational to OO database systems.
CHAPTER 6

A SOFTWARE TOOLSET TO SUPPORT THE TRANSFORMATION OF RELATIONAL DATA STRUCTURES INTO OBJECT-ORIENTED DATA STRUCTURES

6.0 INTRODUCTION

A new approach to transforming relational data into object-like data structures was described in chapter 5. This chapter describes the design and development of a software toolset, conceived to support the translation of data in accordance with the four-step approach. Thereby the resultant software toolset semi-automates the creation of object classes. Furthermore it takes as input actual relational data (stored as rows in the original relational database) and constructs object-oriented data that can be stored in an OODB in conformance with a defined object schema. The toolset and exemplar demonstration OO systems were developed as part of a Java environment, since characteristics of this language makes it potentially suitable for its use in the majority of situations in an enterprise where a software program is required [Cook, 2000]. This language readily supports links via JDBC drivers to various proprietary relational databases [Reese, 1997]. JDBC drivers are designed to achieve access to a relational database via SQL queries. The proof-of-concept toolset was also designed to store data in a Microsoft Access database, with a JDBC:ODBC driver being used to connect the database to the toolset. Also within the demonstration environment, OO schemas are described in the Java language and stored in a POET OO Database.

In this chapter the use of the toolset will be illustrated with respect to (1) how schema translation is achieved and (2) how an OODB structure is defined and populated with actual data derived from the original (legacy) RDB. In Appendix B more extensive information about the software toolset design and specification can be found. Figure 6.1 illustrates in conceptual form the elements of the data translation process enabled by the toolset. Table 6.1 summarises the activities involved in the translation
processes. The set of classes that comprise the new OO schema is generated in a semi-automatic fashion as the toolset enforces and supports the application of the schema translation algorithm. Here user input is required in respect to decision-making issues and in supplying certain types of information not accessible from the original relational database. Mapping information is stored during data transformation processes in a mapping information repository. This repository maintains information required during succeeding steps of the algorithm. Following OO schema generation, the original relational data can be transformed and transferred to the OODB in an automatic way.

Figure 6.1 Elements of the translation processes

<table>
<thead>
<tr>
<th>Activity</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Extracting the structure of the original information stored in the RDB.</td>
<td>Identification of documentation available and its state/version about the legacy data model. Identification of relational schema variables: tables, attributes, primary keys, and foreign keys.</td>
</tr>
<tr>
<td>2. Generating an OO schema.</td>
<td>Convert a relational schema into an equivalent object schema.</td>
</tr>
<tr>
<td>3. Analysis of data schema</td>
<td>Analyse initial data schema to determine the final object-based schema. Model data relevant to each functional area is represented by the data model components. Substructures can be detected according to predefined industrial classification</td>
</tr>
<tr>
<td>4. Creating an OO database</td>
<td>Define a valid OODB with the object-data model generated.</td>
</tr>
<tr>
<td>5. Object instance population</td>
<td>Generate object instances from relational records to populate the OODB conforming to the hierarchy of classes previously generated.</td>
</tr>
</tbody>
</table>

Table 6.1 Main activities involved in the data transformation process
6.1 A TOOLSET SPECIFICATION TO SUPPORT THE SCHEMA TRANSLATION PROCESS

As explained above, a schema translation process is required to create an equivalent OO schema that represents entities and entity relationships in legacy data when it is to be reused within OO environments. This new OO schema should function to restructure and maintain the properties of original information relationships contained within the tables of the relational system. The first part of the data transformation process is to create an OO schema which restructures but maintains the properties of information relationships contained within the tables of the original relational system. Figure 6.2 describes the main activities identified as being required, when generating an equivalent OO schema.

![Diagram of schema translation process]

Figure 6.2 Main activities of the schema translation process.

6.1.1 Activity 1: Extracting the structure of the original information stored in the RDB

First it is necessary to extract semantic information from the relational database. Here it is necessary to reveal the ‘structure’ of the data tables in terms of table names, type and name of their attributes, primary keys and foreign keys. Bearing these factors in mind, the proof-of-concept toolset was designed and developed to support schema translation processes in such a way that if the original RDB does not provide suitable means of revealing the ‘structure’ (as defined above) the toolset facilitates the use of mechanisms that enable a human user to achieve schema definition and creation. This capability is supported in terms of selecting, checking and detecting primary keys, and
defining foreign keys. Therefore the toolset was created in such a way that schema information can be automatically extracted, such as tables and attribute names (made available via JDBC functions) or data values (via SQL queries). However key properties can be identified by a human expert in cases where the relational database system does not provide sufficient information and/or mechanisms to extract them automatically. The toolset is also able to detect possible candidate attributes for primary keys and to check options introduced by users. This function was developed based on an understanding of uniqueness characteristics of primary keys. On completion of this step, the structure of the relational database is (1) displayed for the purpose of human users via a window-based representation and (2) stored in a repository in a format that facilitates subsequent mapping processes. Figure 6.3 shows an example screen layout generated by the toolkit immediately following the extraction of data from the source relational database. The left panel shows a tree representation used to display the relational schema to the expert human user of the toolset.

![Screen layout generated by the toolset after data is extracted from the source relational database](image)

**Figure 6.3** Screen layout generated by the toolset after data is extracted from the source relational database

### 6.1.2 Activity 2: Generating an OO schema

The four-step algorithm developed and described in Chapter 5 generates an OO schema characterising information extracted about the ‘structure’ of the relational database. In the example screen layout illustrated by Figure 6.3 the ‘Run Mapping’ button is used to initiate object schema generation. The checkboxes on the right of this prompt indicate the status of the mapping process. Here the algorithm can be used on
an iterative basis leading to the development of effective and efficient OO schema. In general human user intuition and decision-making was found to be necessary during the first two steps of this mapping procedure, namely: removal of vertical splitting tables and generating a set of new entities. However subsequent steps could be fully automated. Hence the proof-of-concept toolset was designed and implemented so that it generates a set of computer-executable object classes and establishes relationships between them, namely, inheritance, aggregation and association relationships.

**Step 1. Removing the occurrence of Vertical Splitting Tables**

The toolset was designed to display original attributes of a relational table and to compare these attributes with attributes of all other tables in the source RDB whilst systematically applying the algorithm. Vertical splitting tables are detected if the following conditions occurs: all the attributes in a table match in both name and type with a subset of attributes of another table and its primary key (PK) is also a foreign key (FK) that refers to this other table.

Therefore the toolset was designed to analyse all tables in a source RDB by applying the condition described above to detect instances of vertical splitting tables in a methodological way. When a table is defined by the toolset as being redundant, a human expert user is given the option to decide whether to remove that table or not. If the ‘removal’ option is chosen the toolset automatically removes the table (and its records) from the database. Since vertical splitting tables are detected because they have common attribute properties the occurrence of such a situation does not fully guarantee that a duplicate record exists. Therefore during stage 1 of the OO schema generation procedure, the toolset matches records stored in the redundant table to those in the splitting table. Where records are defined as new records\(^7\) the toolset stores them in the new table (as illustrated by Figure 5.5). Essentially the toolset facilitates a unification of two tables into a new table entity. It must be noted that an outcome of these procedural steps is that a table may be deleted from the database thereby modifying the original relational database system. The decision to allow this to happen was taken to facilitate the implementation of succeeding object schema generation activities.
Step 2. Identifying Missing Entities.

During step 2 of object schema generation new entities can be introduced. Such entities can be introduced under two different scenarios, namely: (1) when a candidate key is detected and (2) when attributes with alternative null values are found.

(1) The toolset was designed to detect candidate keys between each foreign key of the relational schema. When a candidate key is found a user decision is required to define the new entity. The toolset was designed to create new entities with at least two attributes, namely: the candidate key (as a PK) and an attribute as FK referring to the original table. Expert human user intuition and decision making is required to name the new entity. Human users can also transfer more attributes from the original table to the new entity than those defined automatically by the toolset. Figure 6.4 displays an example screen that allows the expert user to introduce the name and attributes of the new entity. In the example screen layout two attributes have already been defined for the new entity, i.e. \textit{attri13} and \textit{Table1_ass}. The expert human user is allowed to transfer more attributes from the table \textit{Table1} to the new entity. However the user can not add new attributes that have been defined as a PK (in this case \textit{attri1} in \textit{Table1} table).

![Figure 6.4 Screen layout for defining a new entity from a candidate key.](image)

\textsuperscript{17} New records are considered to be records that have different PK attributed values. Records with the same PK attributes but with different record attributes are removed from the relational database.
(2) The toolset analyses all the tables in the relational schema to detect cases where two or more sets of attributes were populated with null values and meet the conditions explained in the subsection 5.1.1. When a table meeting this condition is found a screen displays the proposed new entities. This allows users to decide whether to accept the mapping proposed. If the mapping is accepted the toolset is designed to automatically define the new entities as illustrated by Figure 5.5.

**Step 3. Mapping to a Class**

During step 3 of object schema generation sufficient information will have been made available for the prototype toolset to automatically create the set of classes required. In this step the toolset automatically defines classes with the same name and attributes to those that were specified in the original relational schema.

**Step 4. Determining relationships**

The prototype toolset was designed to automatically form object relationships in conformance with the inheritance, aggregation and association rules described in Chapter 5. On completion of this final stage of object schema generation, the resultant OO hierarchy is graphically displayed by the toolset. Here the final hierarchy is presented in the form of a tree.

**6.1.3 Activity 3: Analysis of data schema**

Thus the software toolset can provide an object-like view of relational data. Here schema mapping was based on an understanding of the nature of relational data structures to define class entities and relationships. By such means, data values can be analysed to detect redundant tables and candidate new entities. However further analysis of the data structure by an expert user will provide a better characterisation of specific application requirements, such as to facilitate the definition of common data structures for groups of manufacturers. Therefore the toolset was conceived to generate an improved understanding about data structures and their possible reuse, which includes a capability to facilitate the creation of new entities.

The toolset generates a file to display ‘use of data’ information for each entity defined during mapping processes. As illustrated by Figure 6.5 the files contain the following information:
• total number of records in each entity (i.e. Class table2 num. records=4)

• number of records that have a value for each field in each entity (i.e. int attri21 [In use=4])

• number of different record values for each field in each entity (i.e. int attri21 [diff 4]).

Figure 6.5 Example of information file about the use of data in the relational schema

Therefore for example an evaluation of the information displayed in these files can enable the user to gain a better understanding of customer requirements in terms of which type of table should be in use and how table attributes should be used. This information will help expert users to specify structures that characterise groups of customers.

Moreover it can be found that not all the attributes defined in the table are of value for a single manufacturer (or for a manufacturer class) since the original relational schema was defined to provide suitable functionality for a generic group of customers. This generic-purpose design and the fact that relational database management has inferior performance when dealing with queries involving joint tables could result in a requirement for very large tables.

Therefore the toolset was provided with mechanisms to split class entities and thereby improve the design of the initial class data schema. An expert user can select any set of attributes to define a separate entity. New entities will be created to:
- remove groups of attributes in class schema that are not in use or are of no value to a specific user, and

- compile a set of attributes for different classes of user and thereby to enable grouping of these attributes as a new entity where this is useful.

The toolset was developed to display classes of entity defined by the mapping algorithm, as illustrated in Figure 6.6. Users can choose any set of attributes to create a new entity and name it. An association relationship is automatically defined between both tables (the original class and the new entity). In the example screen of Figure 6.6 two attributes attri24 and attri25 were selected to create a new entity named New_class_fromTb2. The Extend panel in the upper part can be used to show the superclass for the Table2 class while the Relation panel in the lower part can show any class with an association or aggregation relationship with the Table2 class. In this example case the toolset has automatically defined the attribute New_class_fromTb2_ass in Table2 class and the attribute Table2_ass in the new entity to define an association relationship between both tables.

Figure 6.6 Screen layout for defining a new entity from a set of attributes belonging to the same class.

Figure 6.7 illustrates a screen window displayed by the toolkit when the schema translation process has been completed. The OO Schema is shown in the right half panel of this screen.
6.2 A TOOLSET SPECIFICATION TO SUPPORT OBJECT DATA TRANSLATION AND POPULATION PROCESSES

The foregoing explains how the prototype toolset can be used to semi-automate the creation of an equivalent schema by defining a hierarchy of classes. This set of classes will define the equivalent OODB. Records in the relational table can then be processed to conform within the class structure and therefore be used to populate a target OODB.

6.2.1 Activity 4: Creating an OO database

The toolset was designed so that it proceeds automatically to create files (written in the Java language) that represent an associated hierarchy of classes. Each class is characterised formally by the following constructs.

1. Name of class plus inheritance references, as applicable.

2. Simple attributes that do not define a relationship and that refer to columns in tables. Here the impedance mismatch between data types in SQL and Java was overcome by mapping data types in the manner specified by Hamilton [Hamilton et al, 1997]
3. Attributes referring to relationships between classes. Aggregation and association relationships are specified by an attribute in classes involved in the relationship, where the type of the attribute depends on the cardinality found. Multiple references will be defined with types named *SetOfObject* while single references will be determined with attributes referring to the class that defines the relationship.

4. Constructor functions that generate instances of each class. In the case of inheritance relationships this overrides the constructor of the superclass, thereby ensuring that inherited attributes have the same value in both super and sub classes.

5. Functions that can access values assigned to each attribute.

6. Functions that set the values of each attribute.

The classes and their constructs formally represent the object model that will be used to create an OODB. Figure 6.8 illustrates a class defined by the toolset. However, before populating a database, certain changes may need to be made to enable class files to be understood by the OODB. The actual changes required will depend upon properties of the OODB tool. Class files also need to be compiled. This can be achieved using a compiler tool provided by the OODB.

```java
import com.poet.odmg.collection.*;

public class Table1 {
    private int attri11;
    private String attri12;
    private New_candidate_entity attri13;

    public Table1 (int attri11_val, String attri12_val, New_candidate_entity attri13_val) {
        this.attri11 = attri11_val;
        this.attri12 = attri12_val;
        this.attri13 = attri13_val;
    }
    public Table1 (){
    }
    public int getAttri11 () { return (this.attri11); }
    public String getAttri12 () { return (this.attri12); }
    public New_candidate_entity getAttri13 () { return (this.attri13); }
    public void setAttri11(int stattri11) { this.attri11 = stattri11; }
    public void setAttri12(String stattri12) { this.attri12 = stattri12; }
    public void setAttri13(New_candidate_entity stattri13) { this.attri13 = stattri13; }
}
```

Figure 6.8 Java definition of the class *Table1*
6.2.2. Activity 5: Object instance population

The foregoing explains how the toolset creates an object-oriented database by defining the required hierarchy of classes. After the OODB has been created the toolkit populates instances of each class with values of data stored in the RDB. Record values in the relational database will be extracted to populate objects organised according the class structure generated earlier. The extraction of records can be readily achieved by making appropriate SQL queries on the relational database. Hence simple primitive attributes are populated first. The value of a simple primitive attribute type can be easily accessed by using a SQL Select query on the corresponding table, whereas relationship attributes require a SQL Select query on all tables involved in the relationship. When populating objects in the OODB, simple type attributes can be valued directly by the class constructor by using the result of the query, whereas relationship attributes should be valued after the query result has been matched to values of existing objects. Consequently the prototype toolkit was designed so that during data transfer (from a source RDB to a target OODB) firstly objects are stored with only simple type attribute values. After all objects have been stored those attributes that represent a relationships are valued, having first determined which objects match a relationship. Section B.0.5 of Appendix B illustrates how the toolset was designed to deal with object data population processes. Figure 6.9 illustrates the data transfer processes involved.
6.2.2.1 Obtaining row-value tables

Firstly the data stored in the relational database must be extracted and stored in a format that facilitates subsequent processing. To achieve this the toolset was designed to execute SQL conformant queries on the RDB, thereby obtaining row values for each table that is to be mapped onto a class. These values are stored in the Mapping Information Repository using the structure defined by the final class hierarchy. This approach facilitates the successive incorporation of relationship values and their storage in the OODB.

6.2.2.2 Obtaining row-value tables for each class relationship

When an OO schema is created, the toolset was designed to support the specification of three kinds of relationship. The toolkit treats relationship attributes in different ways dependent upon the kind of relationship they represent. Here the toolkit was designed to facilitate the definition of values for association and aggregation relationship attributes. The object value of the relationships is determined by making reference to attributes of both classes involved. Since all such relationships will be determined by foreign keys the toolset was designed to execute a SQL query to the relational database for each row value of the tables involved. The result of the query is...
stored in the mapping information repository together with results from step 6.2.2.1. Figure 6.10 illustrates the latter two steps.

6.2.2.3 Create store-data files

Before proceeding with storage processes it is necessary for the toolkit to create different files that need to be used to structure and support subsequent data translation and migration processes. This is necessary so that data stored in a relational format can be reformatted using rules imposed by the OODB. Files are created to assist in the storage of data and in its restructuring, based on properties of relationships between classes.

Therefore the toolset was designed to generate `Dtb_classname` files for each class mapped onto the OODB. These files can be used to support operations concerned with the storage of objects in the OODB. Each file needs to access data records of one class of object stored in the mapping information repository and to input them into objects used by the OODB. `Inher_classname` files are created for each subclass. The
prototype toolset was designed to generate files that can be used to support the
definition of attribute values that are shared between superclass and subclass objects.
*Aggr_classname* files are also generated by the toolset. Here objects that establish a
match between aggregation relationship attributes are stored as a reference in the
corresponding attribute. A file is created for each container class. The toolset was also
designed to support the generation of *refer_classname* files. Use of these files is
required during 1) the selection of objects that establish a match with the association
relationships attributes and 2) when storing the objects as a reference in the
corresponding attribute. A file is created for each class that has an association
relationship.

6.2.2.4 Storage of simple values

Once the source relational data has been analysed, object classes have been extracted
and the necessary files have been created, the prototype toolset was designed so that it
proceeds with automatically storing data into a target OODB. During the first step of
storage the toolset only stores simple attributes. Here one object is stored for each
row-value extracted in respect of each mapped class where reference is made to
*Dtb_classname* classes. Relationship attributes are declared as null values.

6.2.2.5 Storage of object relationship values

Following the storage of simple values (as explained in 6.2.2.4), all data previously
stored in the relational database will also be stored in the OODB. However as yet no
references will have been established between objects. Hence the toolset was designed
to store reference relationship attributes. To avoid problems of data duplication
between subclasses and superclasses it was decided that storage should be in the
following order.

1) *Inher_classname* classes are used to deal with inheritance relationships. An
*Inher_classname* class is created for each subclass that is populated by inheritance
relationships. Subsequent use of the *Inher_classname* class will enable the toolset
to restructure the data in an inheritance hierarchy. Superclass and subclass objects
are linked together by assigning values to inheritance attributes in each subclass.

2) The toolset uses *Aggr_classname* classes to handle aggregation relationships.
These classes are created for each container class involved in an aggregation
Chapter 6  A software toolset to support the transformation of relational data structures

relationship to store all objects of the aggregated classes that determine this relationship.

(3) refer_classname classes are used to store association relationship attributes. Each element of an association relationship stores a reference to other elements involved in the association.

6.3 SUMMARY

A proof-of-concept toolset has been designed and developed to facilitate the application of the transformation method described in Chapter 5. The toolset organises and supports the various translation processes involved in mapping a relational source of data onto a target object-oriented database. Expert human user input and decision making are assisted via dialog screens to facilitate schema translation processes. Also extra information is provided in the form of text files that describe semantic information about how data has been populated in the source database. Here schema translation processes are executed in a semi-automatic fashion. Thereby equivalent object-oriented schema are defined and compiled to create the data structure of an OODB. Actual data can be then be transferred automatically from the relational source to the object-oriented target. Subsequent use of the toolset was found to meet its design objectives of

- Supporting data transformation processes in a more complete way than reported elsewhere in the literature.
- Organising and supporting data transformation processes in a semi-automatic fashion
- Providing expert users with a data input and decision-support capability to determine a set of class schemata.
- Fully automate and support the transfer of real data between relational and object systems in accordance with structures defined by the class schema.
CHAPTER 7

A CASE STUDY

7.0 INTRODUCTION

This chapter describes a case study conducted to test the applicability of the data reuse approach and software toolset developed during this research study. Proof-of-concept experiments were carried out on three data sets provided by Swan Software Ltd (a collaborating company). Swan supplies ERP software, primarily to SMEs. The resultant case study work tests the:

(1) effectiveness of the data reuse approach and its associated steps and data transformation processes,

(2) applicability of the approach under different reuse scenarios,

(3) effectiveness and utility of the toolset which was developed to underpin the project methods and transformation processes.

7.1 OBJECTIVES AND SCOPE OF THE CASE STUDY

Many different types of manufacturing system that are in operation today were originally designed and built to utilise data stored in the form of relational tables. It is probable that the capability of a significant proportion of these systems could be enhanced and that reconfigurability and operational benefits could be gained by implementing some form of object orientation. However, alternative means of achieving such an enhancement can be deployed. Hence it was impractical (as part of a single PhD study) to seek to design a set of experiments that fully evaluate the benefits and disbenefits of the data transformation approach proposed. Therefore it was decided that a fairly focused set of experiments would be designed and carried out that reflect current interests of the collaborating company, Swan. It was also proposed that this simplified experimental study should be but one example of an
experimental approach that might be referenced during the design of future experiments aimed at achieving a more complete evaluation of the approach.

More specifically the main objectives of the proof-of-concept testing described in this chapter were to:

1. capture and develop an object-like data view of legacy data structures that are representative of 'as is' manufacturing data in the ERP domain. Also to classify prerequisite information that must be made available to allow the transformation approach to work successfully. Thereby to show that data capture and object view creation (and thence data reuse in object systems) can be achieved in a semi-automated fashion, such that manufacturing industry can benefit in several ways.

2. demonstrate application-oriented benefits and constraints associated with using the project methods. Here it was assumed that various application benefits would arise from an ability to gain an improved understanding of data structures in the form of explicit data models, thereby providing knowledge that can be reapplied when:
   - customising data structures,
   - enhancing the performance of systems in which object applications are required to interoperate with applications that embed legacy elements.

3. quantify and/or qualify technical benefits associated with applying project methods and in particular when:
   - mapping between relational and object entities and relationships;
   - capturing and defining new entities;
   - providing user decision-support;
   - achieving object data population;
   - abstracting and detecting substructures of data;
   - defining and implementing an OODB.
7.2 AN EXAMPLE CLASSIFICATION OF END-USER COMPANIES

Data sets analysed during case study experiments were selected according to two requirements. Firstly the selected data sets were representative of different classes of end-user company. Secondly they were each generated from a common ERP product data structure.

Consequently an example classification of manufacturing end-users had to be determined to illustrate a viable approach to experimentation. Different classifications of manufacturing systems reported in the literature were considered for this purpose. Porter [Porter et al, 1999] identified a number of ways of classifying manufacturing companies as follows:

1. Job-to-continuous. Here classification is made by referencing the nature of the production processes used by a company or enterprise, which can range from continuous process-based manufacture to discrete parts manufacture.

2. Make-to-stock vs. make and/or assemble to order. This classification differentiates between companies or enterprises in terms of their predominant use of stock-driven or order-driven manufacturing processes.

3. Complexity and uncertainty. Here organisations are differentiated by the extent to which their products, processes, relationships and environmental conditions are complex (e.g. in terms of volume and variety of products and product market relationships) and uncertain (e.g. in terms of the stability of product demands and competition).

4. Product sector. Here classification is based upon the type of product made.

After discussion with various relevant personnel at the collaborating vendor company (Swan) the classification described under 2 was chosen to focus and test the experimental methods developed and used in this study. Three different data sets were obtained from Swan Software, where these data sets correspond to manufacturing data used by three different types of end-user company. The end user companies from which the data sets were obtained had previously been considered by Swan personnel to belong to categories listed in the table below.

92
Table 7.1 Example classification of end-user companies

<table>
<thead>
<tr>
<th>Make-to-stock</th>
<th>Assemble-to-order</th>
<th>Company 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>Engineer-to-order</td>
<td>Company 2</td>
</tr>
<tr>
<td></td>
<td>Design-to-order</td>
<td>Company 2</td>
</tr>
<tr>
<td>Make-to-order</td>
<td></td>
<td>Company 3</td>
</tr>
</tbody>
</table>

It was assumed that choice of the data sets from the three companies would (1) allow testing of the data transformation approach developed in this study with respect to a broad range of data with distinctive differences between data sets, (2) illustrate its use in different but relevant manufacturing scenarios and (3) enable general conclusions to be drawn about the applicability of the approach with respect to manufacturing data.

The following paragraphs give an outline description of the companies from which sample data sets were obtained.

Company 1. MAKE_TO_STOCK (MTS)
This company produces fertilisers mainly to meet high volume orders. However the volume and variety of fertiliser production depends upon a seasonal forecast demand. The primary policy concern in the company regarding the instantiation of production processes is to keep a suitable balance between low inventory levels and customer demand delivery time. General concepts underpinning manufacturing operations in this company are as follows.

- Standard product, low level of variety.
- High unit volumes (near to continuous production).
- Short customer delivery lead times are critical.
- Production under forecast demands.
- Sales orders are filled directly from stock to meet short delivery lead times.
- Stock of finished goods.
- Low levels of WIP.
- Low levels of raw materials.
- JIT management to reduce process lead-times.
Chapter 7

- High Capital equipment costs.
- Relatively low labour costs.

*Company 2. ASSEMBLE_TO_ORDER (ATO)*

This company produces small-scale aircraft for the purpose of military exercises. The company offers a high variety of finished goods, producing most of these from standard basic components and subassemblies. Components and subassemblies are produced and stored in quantities that maintain a minimum safety stock level but final goods are assembled in response to sales orders. Primary policy issues related to production processes are concerned with reducing WIP, maintaining minimum component stock levels and minimising lead times. General concepts underpinning the company’s manufacturing operations are as follows:

- Production based on the assembly of standard product elements but with high variety of finished goods.
- Finished goods produced in medium to low unit volumes.
- Sub-assemblies manufactured to stock to reduce manufacturing lead times and thereby meet agreed customer delivery lead times.
- End product assemblies are scheduled to meet delivery dates for individual customer orders.
- Key task is to meet viable customer deadlines.
- Master production scheduling is used to enable the production and scheduling of component items under forecast demand.
- Basic components and subassemblies are in stock but final products are assembled to satisfy sales orders.
- High levels of sub-assembly and component WIP are commonplace.
- Low levels of WIP for finished goods.
- A large number of components are required but a flat bill of materials is used by the company.
Company 3. MAKE TO ORDER (MTO)
The third case study company produces Industrial Machine Tools capable of machining large components. The characteristics of finished goods are defined by sales orders. The main policy issues during the execution of production processes concern the need to keep budgets under control, a need to control WIP and to minimise the effect of redesigns on production quality, various lead times and costs. General concepts underpinning manufacturing operations of company 3 are as follows.

- Wide variety of finished goods.
- Low unit volumes.
- Overlapping schedules for design and manufacture.
- Delivery leadtime can be uncertain.
- Customer order information is used to control key aspects of production phases.
- High labour costs, hence budget controls are very important.
- Need to trace labour histories to keep contracts within budget.
- Sales orders generated by MRP processes.
- High levels of WIP.
- Large number of levels in a typical bill of materials.

7.3 DEMONSTRATION SYSTEM SET-UP

A proof-of-concept experimental arrangement was built to demonstrate the applicability of the project approach. This facility enables pre-existing information stored in relational databases to be transformed so that its reuse within object-oriented systems can be facilitated. It was also necessary to define and develop a stepwise procedure for deploying the proof-of-concept system. The procedure developed has the following steps, as illustrated by Figure 7.1.

- Step 1: Reorganise data files contained in the source relational database.
- Step 2: Extract relational database information and analyse the data tables concerned.
Step 3: Implement an object schema and populate a target OODB.

Figure 7.2 illustrates the experimental set-up used to test the software toolset, which (as explained in Chapter 6) supports schema creation and data object population. The following subsystems are utilised:

- SAGE dB, a hierarchical database, is used as a pre-source data repository that holds case study data sets provided by Swan. Here a specialised mechanism had to be provided to convert data from the hierarchical source into a suitable form for storage in the Access RDB, in order to facilitate use of the test data within the toolkit. A simple conversion from Sage dB files to equivalent relational tables was realised by developing a software application process, which functions in the manner shown in Figure 7.2. This conversion was achieved in an automated fashion by using ‘import table’ functions provided by the Access RDB.

- An Access relational database is used to store data to be analysed by the toolset.

- The software toolset developed during this research study is used to analyse relational data, implement the object data schema and populate an OODB.

  - The objective when analysing tables of the relational database is to determine an equivalent object schema that can be implemented in an OODB. This analysis is carried out by implementing the four-step algorithm developed during the research study. This algorithm deploys sets of classes. Also additional detailed information about files is provided to help users to determine a final choice of class schema. The additional information used includes: total number of records in each file; the number of records that have a value in each field; and the number of different record values in each field of each file. This stage of data analysis is illustrated by Figure 7.3.

  - The object schema is used to create an OODB and to define a format by which relational data is transformed so that it can be reused when populating a OODB.

- A Poet OODB is used as an example object-oriented data repository to store the final object schema and data records that are equivalents to original manufacturing data sets supplied by Swan.
Chapter 7

Figure 7.1. Proof-of-concept processes for transforming relational data tables into object data stored in an OO database

Tools used:
- Software tool to implement the object schema (see section B.0.4 of Appendix B).
- Software tool to transfer data and populate the OODB (see section B.0.5 of Appendix B).
Figure 7.2. Experimental set-up used to test the relational transformation approach
### File Information Process Description

**Figure 7.3.** File information process description

#### TABLE: WOP_WORKS_ORDERS

<table>
<thead>
<tr>
<th>Order_number</th>
<th>Order_status</th>
<th>Item_number</th>
<th>Routing_Ind</th>
<th>Document_number</th>
<th>Labor_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WO/0001</td>
<td>2</td>
<td>T/BA1</td>
<td>0003</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>WO/0002</td>
<td>6</td>
<td>T/TA1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>WO/0003</td>
<td>6</td>
<td>T/BA2</td>
<td>0005</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>WO/0004</td>
<td>14</td>
<td>T/CP1</td>
<td>0007</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>WO/0005</td>
<td>2</td>
<td>T/TM1</td>
<td>0009</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>WO/0006</td>
<td>8</td>
<td>T/TAI</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>MPS0001</td>
<td>5</td>
<td>T/MB1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>MPS0002</td>
<td>8</td>
<td>T/MB2</td>
<td>0001</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>MPS0003</td>
<td>5</td>
<td>T/MB1</td>
<td>0010</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>MPS0004</td>
<td>5</td>
<td>T/MCI</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**CLASS: WOP_WORKS_ORDERS**

- `num_records` = 10
- String `Order_number` (In use=10) [diff 10]
- int `Order_status` (In use=10) [diff 5]
- String `Item_number` (In use=10) [diff 8]
- String `Routing_Ind` (In use=0) [diff 0]
- String `Document_number` (In use=6) [diff 6]
- int `Labor_value` (In use=10) [diff 1 value 0.0]
7.4 RESULTS FROM THE DATA ANALYSIS

The three data sets were received in the database format deployed by Swan Software. Because of project resource and time constraints the analysis had to be limited to Swan ‘standard’ main files of this database. The files analysed were as follows:

- **STOCK_CONTROL_FILE**: This file provides the ability to maintain and report information relating to inventory activities, status and cost.

- **BOM_STRUCTURE_FILE**: This file provides information regarding the structure of manufactured items such as links between individual parts and their place in an assembly.

- **BOM_ALLOCS_ORDERS**: This file provides information about allocations of ‘on order’ transactions.

- **BOM_LEVELS_FILE**: This file is a batch file that holds all stock levels by price and depot.

- **BOM_ISSUED_SERIALS**: This file is a batch file that holds serial numbers than have been issued to Sales or Works Orders.

- **BOM_SERIALS_NUMBER**: This file is a batch file that holds serial numbers in stock or inspection.

- **MPS_SCHEDULE_FILE**: Master Production Schedule Master File.

- **POR_PURCHASE_ORDERS**: Purchase Orders Master File.

- **WOP_WORKS_ORDERS**: Works Orders Master File.

- **CAP_ROUTINGS_FILE**: Capacities Master File.

- **SOR SALES ORDERS**: Sales Orders Master File.

Here it was assumed that a generic reference model (as defined in Chapter 5) could be drawn out of the case study data by developing an abstract representation of the common and primary schema found in Swan ‘standard main files’. Figure 7.4 illustrates such a class hierarchy drawn out by applying the project methods and toolset to an ‘empty’ version of the Swan software tool. It means that no data values were analysed, only the data structure of the Swan software tool.
The following subsections illustrate results obtained from the proof-of-concept experiments carried out when analysing each of the three example data sets. The findings are presented in a form that facilitates further analysis which can be linked to stated objectives of the research as follows:

- General information about type of stock items and bill of material structures was extracted manually using Swan Software toolkit. This was achieved to exemplify characteristics of data sets within the context of the classification group to which the set belongs.

- Relationships amongst tables and size of tables are displayed to illustrate how generated versions of each data set are deployed. This information is created by using the toolkit while following guidelines provided by Swan Software manuals and information files. These information files are illustrated further in Appendix C.

- A class schema is also made available as a consequence of deploying the software toolset. This class schema represents the final output generated by the proof-of-concept toolset corresponding to each scenario tested.
7.4.1 Example make_to_stock company

General information about example data set 1

BOM Sequence: 10

Number of Items: 735:
- Raw Material: 158
- Bought Out Parts: 371
- Manufacturing Parts: 1

Subassembly: 74
Finished goods: 124

Table Sizes

Empty Tables:
- CAP_ROUTINGS_FILE
- BOM_ISSUED_SERIALS
- BOM_SERIAL_NUMBERS

Tables in use:
- STOCK_CONTROL_FILE: 735 records
- BOM_STRUCTURE_FILE: 1023 records
- BOM_ALLOCS_ORDERS: 127 records
- BOM_LEVELS_FILE: 9491 records
- MPS_SCHEDULE_FILE: 124 records
- POR_PURCHASE_ORDERS: 23 records
- WOP_WORKS_ORDERS: 431 records
- SOR_SALES_ORDERS: 3 records

Figure 7.5 illustrates relationships that exist between Swan Main Files, for example data set 1. This information was also provided by the company. Figure 7.6 illustrates the class schema deployed by the software toolset.
Relationships between Tables

Chapter 7

Figure 7.5 Relationships between Swan Main Files used by the example make_to_stock company

Class Schema

Figure 7.6 Class schema generated by the proof-of-concept toolset when processing test data set 1 from the example make_to_stock company

103
7.4.2 Example assemble_to_order company

General information about example data set 2

BOM Sequence: 285

Number of Items: 11452
- Raw Material: 279
- Bought Out Parts: 6976
- Manufacturing Parts: 1295
- Subassembly: 2517
- Finished goods:
- Phantom: 141

Table sizes

Tables in use:
- STOCK_CONTROL_FILE: 11452 records
- BOM_STRUCTURE_FILE: 25631 records
- BOM_ALLOCS_ORDERS: 127 records
- BOM_LEVELS_FILE: 20076 records
- BOM_ISSUED_SERIALS: 390 records
- BOM_SERIAL_NUMBERS: 322 records
- MPS_SCHEDULE_FILE: 38 records
- POR_PURCHASE_ORDERS: 3010 records
- WOP_WORKS_ORDERS: 5199 records
- CAP_ROUTINGS_FILE: 2326 records
- SOR_SALES_ORDERS: 177 records

Figure 7.8 illustrates the class schema generated by the toolset from example data set 2. Certain limitations of the current implementation of the data transformation approach were found when analysing this particular data set. Tables BOM_STRUCTURE_FILE and BOM_LEVELS_FILE could not be explored in a similar way to that of the other tables found in the example database. However it was confirmed that the constraint was related to memory space limitations imposed when implementing the toolset. As a consequence data values were not analysed for these tables and therefore the detection of new entities as in Step 2 of the schema mapping approach was cancelled. However it was found to be practical to map each of these tables as an equivalent class (as drawn out for the maketostock example company) so the remainder of the analysis could be carried out.
Chapter 7

A Case Study

Relationships between tables:

STOCK_CONTROL_FILE (11452)  BOM_STRUCTURE_FILE (25631)

Change status

SOR_SALES_ORDERS (177)

Minimum stock

MPS_SCHEDULE_FILE (38)

WOP_WORKS_ORDERS (5199)  CAP_ROUTINGS_FILE (2326)  BOM_ALLOCS_ORDERS (127)  BOM_LEVELS_FILE (20076)

Orders for components

receipts

On order

receipts

Allos convert.
to issues

BOM_SERIAL_NUMBERS (322)  BOM_ISSUED_SERIALS (390)

Figure 7.7 Relationships between Swan Main Files used by the example assemble_to_order company

Class schema:

Class BOM_ALLOCS_ORDERS {
  String Order_number
  Stock_Control_File Item_number
}

Class BOM_STRUCTURE_FILE {
  Stock_Control_File Item_number
  Stock_Control_File Parent_part
}

Class MPS_SCHEDULE_FILE {
  Stock_Control_File Item_number
}

Class STOCK_CONTROL_FILE {
  String Item_number
}

Class CAP_ROUTINGS_FILE {
  Stock_Control_File Item_number
  Wop_Works_Order Order_number
}

Class WOP_WORKS_ORDERS {
  String Order_number
  Cap_Routings_File Item_number
}

Class BOM_LEVELS_FILE {
  Stock_Control_File Item_number
  String Order_number
  String Batch_number
}

Class BOM_SERIAL_NUMBERS {
  Stock_Control_File Item_number
}

Class SOR_SALES_ORDERS {
  String Order_number
}

Class POR_PURCHASE_ORDERS {
  String Order_number
}

Figure 7.8 Class schema generated by the proof-of-concept toolset for the data set 2 obtained from the example assemble_to_order company.

105
7.4.3 Example make_to_order company

General information about example data set 3

BOM Sequence: 315
Number of Items: 87727
- Raw Material: 6866
- Bought Out Parts: 49046
- Manufacturing Parts: 15171

Subassembly: 15012
- Finished goods: 136
- Phantom: 143

Table sizes

Empty tables:
- BOM_ISSUED_SERIALS
- BOM_SERIAL_NUMBERS

Tables in use:
- STOCK_CONTROL_FILE: 87727 records
- BOM_STRUCTURE_FILE: 202095 records
- BOM_ALLOCS_ORDERS: 26027 records
- BOM_LEVELS_FILE: 120558 records
- SOR_SALES_ORDERS: 138 records
- POR_PURCHASE_ORDERS: 1945 records
- WOP_WORKS_ORDERS: 12598 records
- CAP_ROUTINGS_FILE: 10172 records

Memory space implementation constraints also placed restrictions on the ability of the user to analyse the complete example data set 3. As in example data set 2 it were not possible to analyse data values to detect new entities in some large tables of the data set. However, the results obtained showed that it was practical to map these tables as if no new entities were found, in the same way that was carried out for data set 2. The results are as illustrated by Figure 7.10.
Chapter 7

Chapter 7

Relationships between tables:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOCK_CONTROL_FILE (87727)</td>
<td></td>
<td>Change status</td>
</tr>
<tr>
<td>BOM_STRUCTURE_FILE (202095)</td>
<td></td>
<td>Orders for components</td>
</tr>
<tr>
<td>SOR_SALES_ORDERS (138)</td>
<td></td>
<td>Alloca orders</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WOP_WORKS_ORDERS (12598)</td>
<td>On order</td>
<td></td>
</tr>
<tr>
<td>CAP_ROUTINGS_FILE (10172)</td>
<td>On order</td>
<td></td>
</tr>
<tr>
<td>POR_PURCHASE_ORDERS (1945)</td>
<td>On order</td>
<td></td>
</tr>
<tr>
<td>BOM_ALLOCS_ORDERS (26027)</td>
<td>On order</td>
<td></td>
</tr>
<tr>
<td>BOM_LEVELS_FILE (120558)</td>
<td>Alloca, convert to issues</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.9 Relationships between Swan Main Files used by the example make_to_order company

Class schema

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Attributes/Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOCK_CONTROL_FILE</td>
<td></td>
<td>Stock_Control_File Component_part Stock_Control_File Parent_part:</td>
</tr>
<tr>
<td>CAP_ROUTINGS_FILE</td>
<td></td>
<td>String Item_number Wop_Works_Order Order_number:</td>
</tr>
<tr>
<td>WOP_WORKS_ORDERS</td>
<td></td>
<td>String Order_number Cap_Routings_File Item_number:</td>
</tr>
<tr>
<td>BOM_ALLOCS_ORDERS</td>
<td></td>
<td>String Order_number Stock_Control_File Item_number:</td>
</tr>
<tr>
<td>SOR_SALES_ORDERS</td>
<td></td>
<td>String Order_number:</td>
</tr>
<tr>
<td>BOM_LEVELS_FILE</td>
<td></td>
<td>Stock_Control_File Item_number:</td>
</tr>
<tr>
<td>POR_PURCHASE_ORDERS</td>
<td></td>
<td>String Order_number:</td>
</tr>
</tbody>
</table>

Figure 7.10 Class schema generated by the proof-of-concept toolset when analysing data set 3 from the example make_to_order company.
CHAPTER 8

ANALYSIS OF CASE STUDY RESULTS

8.0 INTRODUCTION

In chapter 7 a proof-of-concept experiment was reported that illustrates the applicability of the approach to data reuse developed in this research study. The results show potential benefits arising from using the approach in different manufacturing scenarios. By testing the approach on three data-sets (that correspond to different types of manufacturer end-user, namely MTS, MTO and ATO) the aim has been to establish: (i) the effectiveness of the approach to data reuse, and (ii) the potential of the approach to detect particular properties of data-sets used in different, yet commonly occurring, manufacturing scenarios. This chapter analyses the results obtained from the case study with the purpose of appraising the capabilities of the approach from two perspectives. Firstly the capability of the approach to create and capture object-like data views is assessed, as is its ability to create and populate an OODB from legacy relational data. Secondly, potential industrial benefits are explored that arise from the object-like data structures generated, in terms of their inherent capability to provide an enhanced understanding of data sets used by manufacturer end-users.

8.1 APPLICABILITY OF THE APPROACH

Primary building blocks of the data reuse approach are its schema mapping and data transformation processes. In the case study experiment the application toolset was successfully deployed to inspect tables, attributes and data values and thereby to extract information that can be reused when building a class hierarchy and defining an OODB data structure which is semantically equivalent to the original relational data structure used in the source data repository. In this section the results are analysed with respect to the capability and general utility of the schema mapping and data transformation approach.
8.1.1 Schema creation
The schema outputs generated during the case study were explored to determine the consistency of results by generating semantically equivalent schema (e.g. which have consistent relationships, missing entities, etc) and the relevance of the mapping rules included in the transformation approach. The consistency of the schema mapping processes was successfully studied by analysing resultant outputs and comparing them with the input (or source) relational schemas. The factors considered during this section of study are reported in the Table below.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Parameters</th>
<th>Evaluation Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness of the class schema</td>
<td>Table attributes.</td>
<td>Have all the attributes in the relational tables been converted into an equivalent attribute in the class hierarchy?</td>
</tr>
<tr>
<td>Equivalency of relational parameters</td>
<td>Primary and foreign keys</td>
<td>Are all foreign key relations represented in the class hierarchy?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Are primary keys in relational tables considered when creating object-like entities?</td>
</tr>
</tbody>
</table>

Table 8.1 Factors considered when analysing the consistency of the schema mapping approach.

An analysis of results from the case study experiments confirmed the relevance and suitability of the mapping rules included into the data transformation processes. Here, use of the software toolset (which implements the transformation approach) on the case study data showed an ability to (1) successfully detect the main semantic features in an automatic fashion and (2) to support the user in making decisions about how an equivalent OO data structure should be implemented. Figure 8.1 shows some examples of how the completeness and consistency of mapping rules was investigated during the experimental work.

8.1.2 Definition and population of the OODB
Output data from the schema mapping process provides an input class hierarchy that can be used to define an equivalent OODB data structure. This new structure can then be used when transferring (and hence reusing) relational data records into a new OODB (i.e. when populating an object database). The issues considered when analysing the accuracy of the OODB definition processes are listed in the following table.
Chapter 8 Analysis of Case Study Results

<table>
<thead>
<tr>
<th>Factors</th>
<th>Parameters</th>
<th>Evaluation Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data consistency</td>
<td>Data records value</td>
<td>Are all data values in the legacy database transferred to the OODB?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Are there any duplicate record values following transfer to the OODB?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do foreign key attribute values keep their relation values following transfer to the OODB?</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>Is consistency of data types maintained between legacy and object data?</td>
</tr>
<tr>
<td>Scalability</td>
<td>Number of records per table</td>
<td>Does the volume of records stored in a table affect the process and the results?</td>
</tr>
<tr>
<td></td>
<td>Number of relation_attributes per class</td>
<td>Does the number of attributes defining a relationship affect the process and the results?</td>
</tr>
</tbody>
</table>

Table 8.2 Factors considered when analysing the accuracy of the OODB definition processes.

Unfortunately, practical limitations were placed on the experimental study because of implementation constraints associated with the development of the toolset. This meant that it was not possible to fully analyse all data sets made available by the collaborating company. This was due to memory space limitations related to use of the toolset, which meant that the application toolset could not be used to explore tables containing in excess of 30,000 records in a single file. Had sufficient time been available the problem could have been overcome either by optimising the design of the data extraction code, by improving the way that memory management was realised or by developing a new approach that enhances the scalability of the toolset. Therefore the experimental work showed that the way in which the project concepts were implemented by the toolset needed significant improvement in terms of scalability. Unfortunately, however, insufficient project resource was available to action such an enhancement. Nonetheless the approach and toolset were successfully tested on real, representative and large manufacturing, engineering and sales data sets, as originally proposed.

Because of the practical limitation on size imposed by the toolset, only the data set corresponding to the MTS example company was fully tested. The testing here led to the reuse of actual legacy data from a MTS company and resulted in the creation and population of an equivalent OODB. Subsequently object-like data values for this data
set were manually examined to validate data consistency, mainly in instances where consistency was likely to have been an issue such as where relationship references and new entities are introduced. Table 8.3 illustrates the rationale applied to evaluate the results when populating the class schema. This rationale was based on the use of defined mapping rules which structure and inform the creation of the class schema. Here each example mapping rule was analysed by checking the data consistency related to exemplar records of the entities related to each rule. In all instances of testing the consistency of data was maintained in terms of attribute type mappings, relationship references and the completeness of the process (i.e. the same number of records in a source table as in an equivalent class). Figure 8.2 illustrates examples of data consistency examined as part of the case study work.

It was concluded that equivalent object data population of the real data set obtained from the make_to stock company can be considered to be consistent and complete. Moreover, since the data evaluation was based on mapping rules defined by the research method (and having evaluated examples for all the mapping rules) these results were assumed to be valid for any set of relational data input to the project methods and toolset. Consequently, provided that implementation constraints related to the use of the software toolset can be overcome, data population tests on large data sets (e.g. ATO and MTO example companies) would lead to similarly satisfactory results.

Because of the large number of records in real case data sets (such as those provided by the collaborator) it was considered to be impracticable to test all data values in a manual fashion, whilst developing a semi-automatic tool to carry out a consistency test was considered to be outside the scope of this study. Therefore only some of the records in each class were examined manually. However consistency has been verified successfully in all experimental examples tested manually and compared with tool outputs. Moreover it is believed that test results carried out on a few records can be extrapolated to the whole set of records from the same table.
# Chapter 8 Analysis of Case Study Results

## Mapping rules

<table>
<thead>
<tr>
<th>Mapping rule</th>
<th>Input (relational schema)</th>
<th>Output (object schema)</th>
<th>Evaluation checks</th>
</tr>
</thead>
</table>
| Class definition          | A relational table        | One class for each relational table | - Each primary key value in the relational table defines a unique object in the output class.  
- Correct values and attribute types for each object.  
- Number of data objects per class and records per table. |
| Association & aggregation relationships | Foreign key | Bi-directional reference between two classes. | - Classes keep correct number and value of objects for the relation. |
| Inheritance relationships | Foreign key between PKs.  | Super/sub class relationship between two classes. | - Classes keep correct object values for the relation.  
- No object values are repeated in the hierarchy. |
| New entity definitions    | A relational table        | Two or more class as a result of split the relational table.  
A relationship between these new classes | - Relation with origin source is correctly maintained.  
- No repeated object values are introduced into the object set. |

Table 8.3 Rationale used to evaluate object schema population
Chapter 8

Mapping rule: Association relationship

### Class STOCK_CONTROL_FILE
- **String** Item_number
- **int** Part_type
- SetOfObjects Bom_allocs_orders

### Class WORKS_ORDER_ITEM
extends STOCK_CONTROL_FILE
- **String** Item_number
- **Wop_Works_Order** Order_number

### Class WOP_WORKS_ORDER
- **String** Order_number
- **Works_order_Item** Item_number

---

Figure 8.1 Example application of the mapping rules
Chapter 8 Analysis of Case Study Results

Figure 8.2. Examples of verifying consistency during data population processes
8.2 GAINING AN IMPROVED UNDERSTANDING ABOUT LEGACY DATA IN DIFFERENT MANUFACTURING SCENARIOS

Many software product vendors like Swan seek to sell and install their manufacturing software into various manufacturing end-user companies, each with specific data and application requirements. Therefore the vendor's product must provide some means of satisfying disparate requirements of their customers; hence from a vendor perspective their products must satisfy a broader range of requirements than those of any single customer. However, the vendors' domain of concern (which will determine the type of functionality included within their product provision) will normally be but a subset of the full domains of concern to an individual customer. Hence, from a customer perspective they will in general need to deploy more than one application software product to realise IT support for the full range of functions and data usage they require. Also in any given end-user vendor scenario, the mapping between product and end-user application and data requirements will be unique, as invariably each customer's set of requirements will be unique. Hence, vendors must find a suitable way of supporting data and application structures within its products that (a) is sufficiently rich to cover disparate customer requirements as completely as possible or at least as well as their main vendor competitors, and (b) can function as a suitable and differing subset of a wider data and application requirements of any individual customer. Therefore, by gaining an improved understanding of differences and similarities between end user data structures, it was assumed that it should prove possible to assist software product vendors with respect to improving their product customisation and integration processes. For example it was assumed that it might prove possible to help vendors to develop a capability to specify and develop data structures that can readily be modified to meet specific end user requirements rather than expecting each customer to adjust their requirements to suit a generic structure provision. Following this line of reasoning further, it was assumed that it should prove beneficial to define classes of companies, where classification is made with respect to similarities and differences between data and application requirements. Furthermore by developing an improved understanding of similarities between application requirements and data structures it was assumed that it would prove possible to define and develop models of data structures that are common either (1) to all manufacturer
end users or (2) groupings of companies by type. After developing preliminary
classification proposals and holding discussions with technical and sales personnel at
Swan Systems the three company classes, Make To Stock (MTS), Assemble To Order
(ATO) and Make To Order (MTO), were selected and used to help structure
subsequent thinking about possible customisation approaches. Here it was envisaged
that a data structure classification might be reused as a part of an improved
customisation approach. To begin to test this line of reasoning (as explained in
chapter 7) case study investigation was focused on gaining improved understandings
about three large sets of legacy data obtained from representative MTS, ATO and
MTO customers of Swan. Discussions held with Swan personnel confirmed that data
and application sets selected for testing were typical of their type amongst Swan’s
customer base of over 200 end-user businesses.

In addition to case study evaluation activities reported in chapter 7, the case study data
sets were analysed with respect to (a) the total number of files and types of file used
by each company, and (b) the way in which record values and attributes in those files
were deployed. Figure 8.3 illustrates the experimental method developed to analyse
the data sets and generate various classes of schema. Essentially two distinct forms of
information were generated by the software toolset developed to support the
experimental method, namely: object-like schemas and data set information files. For
each data set, values of data records were examined by accessing data contained in the
information files. This information allowed the user/system developer to gain an
improved understanding about the legacy data in such a way that the initial class
schema toolset could be enhanced by grouping attributes that were found to be useful
when generating new classes or when removing redundant classes from the final class
schema. Generalised assumptions about representative class schemas for each
grouping of companies could then be drawn.
Figure 8.3 Illustration of the information types and application toolset used during the process of analysing the test data sets
Distinct differences between end-user databases were observed when applying the experimental method depicted by Figure 8.3. For example, when comparing the number of data items in each data set it was observed that both the number of records stored and the number of table attributes which have a value, are significantly lower in the example MTS company than for the ATO company example, which in turn was significantly lower than that for the MTO example. Figure 8.4 illustrates graphical comparisons drawn to illustrate some of the experimental observations made. This led to the conclusion that important distinctive characteristics of each class type company can be drawn by gaining understandings about the frequency with which data objects are used and how data schemata are populated in data repositories. Furthermore that the distinctions and similarities so drawn about the ‘dynamic use of data’ may be of equal significance to system developers as are classifications made based on improved understandings about ‘more static properties of data structures’. As they were drawn from application systems used in the ERP domain it is probable that the dynamic differences between the three data sets analysed were directly related to how stock and production orders are generated and treated in the various host companies.

The differences observed were reported to the collaborator. Subsequent discussions were held with Swan personnel to consider if the differences and similarities found were likely to be representative of a wider base of end user within each company class and thereby to received feedback from experienced ERP system developers on the probable utility of the observations and data structure classifications. These discussions have given weight to the view that the three legacy data sets used for project experimentation are each representative of a distinctive company group, i.e. which distinguish dissimilarities between data sets from companies in each group. Hence the results presented in chapter 7 could prove beneficial both to specific application system developers and new ERP product developers (at companies like Swan). Although the study results are very preliminary and only include test results from one example of each company group they do provide an illustrative starting point from which an analysis of a greater number of example companies in each group would prove or disprove the observations as being typical trends. Results could then be generalised to refine these or other company classifications in such a way that application developers have an improved understanding of both common and distinctive needs of end user manufacturers.
Further the results described in this chapter indicate that a more detailed and broader study of the profile of use of files in different company classes could well determine further sub-classes (or entity groupings) thereby explicitly encoding into a model additional understandings about similarities and differences between smaller groupings of companies. From information drawn out about how attributes are used in each data set, decisions can be taken with respect to how classes can be split further. This possibility was considered earlier in subsection 6.1.3. For example, from an analysis of the profile of use of the class POR_PURCHASE_ORDER, with respect to mapping results and information files listed in Appendix C, it can be deduced that attributes describing free_format parameters have different uses in each data set. Similar deductions can be made with respect to attributes describing last orders. Following this line of reasoning the table POR_PURCHASE_ORDERS can be mapped onto the classes shown in Figure 8.5. Moreover by determining the relative frequency of use of records within each file, other information can be drawn out to
gain an improved understanding and possibly more definitive classification of data sets. For example, information that attributes are seldom used (e.g. \texttt{VAT\_INCLUSIVE\_PRICES} in \texttt{POR\_PURCHASE\_ORDERS} File) might be identified and used to reduce the diversity in data transactions during the operation of a given application system that in turn may avoid the provision of unnecessary functionality and overhead in managing the database. Therefore by using the proposed methods and experimental toolset to analyse both initial class schemas and the information files, an application system developer can conceive and develop an enhanced set of classes; such as where groups of attributes have been separated out to define a new entity, or where a schema has been simplified following the removal of redundant entities from the final class schema.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure8.5.png}
\caption{Example of split tables following an analysis of the use of attributes.}
\end{figure}
An important general observation illustrated by the findings of this study is that the project methods and toolset can be used to draw out import distinctions between subclasses, contained within a given class of data structure, based on the type and number of files in use or planned to be used in a given end user company.

A second important general observation is that despite differences between classes and subclasses of data structure a common hierarchy of classes can be identified which is shared by company groupings and indeed by all the case study data structures analysed. Figure 8.6 shows class hierarchies drawn out for each company type by using the project methods and toolset. These hierarchies can be viewed as being semi-generic reference models based on an identification of common data models shared by companies within a type. These hierarchies can be referenced by system developers in such a way that they improve the usability of the research approach when seeking to define and use customisable data structures. A generic reference model can also be identified (refer again to Figure 7.4) which corresponds to an abstraction of common entities found in the class hierarchies of Figure 8.6.
Therefore a common reference model has been generated that represents a common structure for use in the ERP domain. Also as more specific models can be developed to fit group or individual customer requirements we may conclude that the representation of commonalities and differences in data structure can be facilitated by the use of object technology and more specifically by the use of the methods and toolset developed during this study.
The case study findings provide a basis for developing and extending the research approach so that ERP applications developed by a single vendor can be deployed more widely and effectively. Potentially it might induce the development of customisable application systems that better match current and future manufacturer requirements. As discussed earlier, manufacturers usually need to deploy a number of application systems, supplied by multiple vendors, in order to structure and support their various business processes. Potentially an enhanced understanding of data user requirements can facilitate data sharing between application systems and thereby their interoperation in specific multivendor scenarios. Additionally means of implementing specified improvements to data sharing arrangements could in part be based on the use of project methods and toolset to customise and unify use of data structures that underpin the operation of the constituent application systems.

Moreover potentially the research methods and toolset can assist in the generation of data standards as they can support the generation of generic, semi-generic and custom-designed object schemata in a manner that can improve the understandings about and the presentation of common manufacturing data scenarios.

8.3 CONCLUSIONS

This chapter reports on testing of the data reuse approach and its supporting toolset. All legacy data used as input to the project methods and toolset has been transformed exactly into an equivalent object form, without information loss and without introducing inconsistency. Some practical difficulties associated with the size (and hence complexity) of real data structures did limit the extent of the testing. The software toolset was not able to facilitate a complete analysis of data sets which contained large data tables. However even where the test data sets were very large, use of the application toolset did provide sufficient information to draw conclusions about how equivalent data structures can be generated and reused, thereby helping to develop new understandings about manufacturing data scenarios. Hence it is believed that the developed approach, its embedded algorithms and supporting toolset have been adequately tested to verify that they operate as required and that consequently they have the capability to support the identified need in a unique and effective way.
It can also be concluded that the data reuse approach and its supporting toolset have been used successfully to analyse real data. Here they have demonstrated a new capability to:

- detect object-like data in a semi-automatic fashion.
- create equivalent OODBs.
- support OODB population.

A second set of tests on real data proved to be less conclusive. Here the aim was to seek to advise IT vendors (like Swan) about (a) the nature of the data and application structures they should provide for their customers and (b) about ways in which they might customise and integrate their products more effectively. Nonetheless initial concepts and results were generated that are of interest from both academic and industrial perspectives as they indicate how researchers, standards makers and IT vendors might redesign their data and application structures with reference to company types. It is argued that the results indicate that product provision related to company types would allow a closer matching to specific user needs than is typical with current best industrial practice and hence could ease product customisation difficulties and lead to efficient multivendor interoperation of application systems. The data reuse approach and experimental toolset were used to facilitate analysis of what were assumed to be stereotypical types of data and application structure. Although these stereotypes may eventually not prove to be typical or particularly useful and although both broader and deeper testing is required before derived classifications and findings can really be of benefit to vendors, standards makers and researchers, it is believed that a useful and general application of the data reuse approach and its manifestation as an application toolset has been achieved.
CHAPTER 9

COMPARISON OF THE RESEARCH STUDY APPROACH WITH OTHER APPROACHES REPORTED IN THE LITERATURE

9.0 INTRODUCTION

This chapter reviews the advantages and disadvantages of the data reuse approach developed in this research study against other approaches reported in the literature and used by the collaborating company. Comparison is drawn based on an analysis of results presented in Chapters 7 and 8.

More specifically the data reuse approach is compared with other approaches that exploit existing relational semantic information to define object-oriented schemas. Contrasts are drawn in terms of conceptual differences and potential benefits when transforming typical data used in manufacturing scenarios. Key comparisons are made by referencing the outputs of each approach. This highlights important advantages of the new approach when seeking to develop an improved understanding of data structures. Further comparison is drawn between current customisation methods used by the case study ERP vendor and new customisation methods enabled by this research which points towards potential industrial benefits.

9.1 COMPARISON BETWEEN DATA TRANSFORMATION AND REUSE APPROACHES

9.1.1 Comparison between approaches developed in this study and other approaches reported in the literature capable of creating object schemas

Different ways of creating object schemas from a relational semantic have been reviewed in earlier chapters of this thesis. Previously available methods and techniques were assigned to one of two classes, namely (1) those in which an object schema is created so that it can function as an intermediary between relation and
Chapter 9  Comparison of the research study approach with other approaches reported in the literature

object-oriented systems and (2) those in which schema mapping is focussed on creating and implementing an OODB. Because these two classes of approach are different in concept, resultant object schemas have necessary differences. The prime concern when developing an intermediary middle layer solution is to simplify (as far as possible) the mapping between relational objects and relational schema. This is necessary because if a complex mapping results its runtime reuse can adversely effect the performance of resultant object systems. On the other hand in general those approaches aimed at defining an OODB seek accurate and complete mappings between relational and object schemas, irrespective of their resultant complexity, so that the features of object-oriented technology can be fully and effectively exploited.

The main factors considered in this study when comparing alternative approaches are listed in Table 9.1.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Parameters</th>
<th>Evaluation Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>Input information:</td>
<td>The type and quantity of information required for each approach?</td>
</tr>
<tr>
<td>available</td>
<td>Quantity and Format</td>
<td>What effect does the documentation format have on the performance of the approach?</td>
</tr>
<tr>
<td>Scalability</td>
<td>Number of tables</td>
<td>Will the number of tables lead to increased complexity and does this affect performance?</td>
</tr>
<tr>
<td></td>
<td>Size of tables</td>
<td>Will the size of tables lead rapidly to increased complexity and does this affect performance?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will the size of tables affect comparisons drawn between approaches?</td>
</tr>
</tbody>
</table>

Table 9.1 Main factors considered when comparing alternative schema mapping approaches.

Information input to class (1) approaches (i.e. those requiring a ‘middle layer’) comprises only the semantic structure of relational schema such as primary key and foreign key. As simple mapping rules need to be deployed, generally there will not be a requirement to input information about data values, use of data or candidate keys because subsequent analysis would lead to the definition of new entities and consequently increase the complexity of resultant mappings. It follows also that generally class (1) approaches will not investigate certain features of schemata, such as candidate keys. It also follows that normally no new entities are introduced when building a middle layer. However a common objective of class (2) approaches (aimed at defining an equivalent OODB) is to exploit all available input information so as to optimally define object entities. Consequently class (2) approaches have been
developed which use input information about candidate keys to detect and define new entities by observing functional dependencies between attributes found in relational tables. However, previous class (2) approaches reported in the literature only make use of the semantic of relational schema, candidate keys and attribute domains as the basis of their transformation approach. This means that in general means of analysing other sources of information, such as data values profiles of data use have not been included into previous transformation approaches in order to improve the ‘quality’ of outputs generated.

Therefore on comparing the new data transformation methods and techniques developed in this study with previous approaches (as characterised by class (1) and class (2) approaches) it can be claimed that the new approach analyses information on data structures (e.g. semantic schema and data values) in greater depth than previous approaches. As a consequence, it can be further claimed that the new approach can provide users with improved decision-support capabilities about introducing new entities into object schema. The circumstances under which this claim is likely to be true are as follows:

(a) When a candidate key is found.

(b) When attributes in a table have null values.

(c) When a set of attributes can be grouped and insulated from the original table.

In general, previous approaches reported in the literature do not analyse key aspects of how the database has been populated, such as in terms of attribute values and data values. Therefore previous approaches cannot support the detection of new entities under circumstances described under (b). System developers can also be provided with improved decision support capabilities when defining new entities by grouping attributes (such as in circumstances described under (c)) if relevant aspects of data values have been analysed. Because previous approaches only provide mechanisms to analyse mainly schema information, attributes domain and candidate keys they can only support the introduction of new entities when a candidate key has been found by some other means.
Chapter 9  Comparison of the research study approach with other approaches reported in the literature

In Figure 9.1 a conceptual graphic is drawn to distinguish between previous class (1) and class (2) approaches and the new approach when the input information comprises different numbers of tables and tables of different size. The graphic shows that the size of input tables will have little effect on class (1) and class (2) approaches because no transformation rules are used as part of those approaches in order to split tables. However class (2) approaches are likely to detect more new entities than class (1) approaches. Whereas generally the new approach developed in this research study will generate object schemas that contain a greater number of entities but with a smaller number of attributes associated within each class.

<table>
<thead>
<tr>
<th>Case</th>
<th>Num. tables</th>
<th>Size of tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
<td>Small</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>Large</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
<td>Small</td>
</tr>
<tr>
<td>4</td>
<td>High</td>
<td>Large</td>
</tr>
</tbody>
</table>

Initial parameters in the relational schema

- Middle layer approaches
- Approaches that convert to an OODB
- Approach proposed in this study.

Note: The size of icons in the graphic represents the expected outputs covered by the results of the approach.

Figure 9.1 Comparison drawn between generalised object schema under different relational parameters

Consider now the context in which there is a need to transform and analyse real industry data. Generally we may expect this to comprise large tables that make partial use of schema. Apparently therefore in such a context the new approach developed in the research study will provide benefits over and above previous approaches since it (1) allows large tables to be split to create new entities and (2) it can detect attributes and tables not in use. It also follows that the output schema generated by the new approach should provide system developers with an improved understanding of input data structures, including legacy data structures generated and poorly documented by previous system developers. Therefore it is argued here that the new approach is
better suited to manufacturing scenarios (at least by those represented by case study experiments) than other approaches described in the literature, since it provides improved mechanisms for introducing new entities and can capture a richer set of information about pre-existing data structures. Consequently it is claimed that the new approach has promising potential to underpin human-centred decision making involved in transformation and reuse of legacy relational data.

9.1.2 User support and information provision capabilities of the new approach compared with other data reuse approaches

The literature and findings of this study show that thus far it has not proven possible to develop a fully automatic approach to mapping between relational and OO schemata. Consequently it is necessary to adopt a policy of supporting the activities of human users by providing schema mapping tools that allow them to make good quality and timely decisions leading to the generation of effective OO schemata. In general therefore the decision-support capability provided should enable human users to interpret key characteristic properties of input data structures. A primary use of this kind of improved understanding about data structures will be to decide when to introduce new entities. Previous approaches reported in the literature have already considered this requirement. Indeed it has been suggested that a user should be offered a set of transformation rules (e.g. [Jahnke et al., 1996]). However it is argued here that the application tool of Jahnke does not provide sufficient information to assist all human users when making all types of mapping decision governing the transformation of relational to OO data structures. Whereas the new approach developed in this research study can improve the decision-making processes involved by providing:

- Information on use of attributes that assists a human user to group attributes and split them from the original table, thereby creating a new entity.

- Graphical displays of models of new class definitions so that users have an improved visualisation of these classes and can consider the new entities in relation to other entities and their interrelationships.

- Information about foreign key and primary key attributes that can be used to investigate the consistency of resultant OO data structures; such as by automatically
indicating when mapping relationships should be tested or by flagging cases where sufficient primary key information has not been provided.

**9.1.3 OODB definition and data population capabilities of the new approach compared with other data reuse approaches**

OODB definition and population capabilities of the new approach cannot directly be compared to those of class (1) approaches (since these approaches seek to maintain the original source relational database). However, on the assumption that consistent and automated data population mechanisms can be developed and used without loss of data integrity, it was originally thought that a comparison could be drawn between class (2) approaches and the new approach with respect to the time required to define and populate an OODB. Necessarily, however, this time will be dependent upon parameters of the input data structure, particularly in terms of the size of its legacy tables and object classes. It is also understood that in general these parameters will be application dependent. Indeed in the context of this thesis it should be remembered that manufacturing, engineering and sales data structures can vary widely in size and complexity. Having considered these issues in some detail it was concluded that a meaningful comparison between new and pre-existing data reuse approaches could not be drawn with respect to their capability to support OODB database population. Moreover only one pre-existing approach [Behm et al., 1997] was found to support the data population process and in this particular case sufficient information could not be obtained from the literature to draw meaningful conclusions about whether this pre-existing approach was scalable in this respect. Therefore regretfully it did not prove to be practical or meaningful to compare approaches with respect to their data population performance.

**9.2 COMPARISON OF METHODS WHEN DETECTING SUBSETS OF DATA STRUCTURES**

In the previous chapter it was explained that the data reuse approach developed in this study can be used to gain an improved understanding of existing manufacturing data structures and that this capability might be utilised by IT vendors when customising their products. In this context the advanced capability of the new approach to detect and differentiate between classes in the output schema is considered to be beneficial when manufacturing data structures are compared.
9.2.1 Capability to compare different data structures

In comparison to previous approaches the new data reuse approach provides advanced capabilities to detect substructures within large, existing data structures that are commonly found in existing manufacturing, engineering and sales relational databases. This is the case because additional detail can be provided so that it is easier for human users to detect commonalities and differences between data sets in the manner illustrated in chapter 8. Other approaches reported in the literature do not utilise information about data values when creating object schemata. Hence they do not have a capability to compare and thereby differentiate between data sets that utilise the same data structure.

9.2.2 IT product customisation

An issue of major concern to end-user customers of vendor companies (like Swan) is that the software applications and tools they use are normally built upon generalised data structures that may have been pre-defined by a third party software vendor. This is the case for Swan whose application software runs over a database product developed by Sage\(^\text{19}\). Furthermore in the case of Swan products (and as can be the case for many other IT vendors including many of Swan’s competitors) the syntax of the data schema was originally defined for use in financial application domains, and therefore is unlikely to match particularly well requirements found in manufacturing domains. Hence there is broad agreement that generic functionality provided by application software products should be matched more closely to specific requirements of customers. The methods that Swan deploys (or provides to its users) to customise its software systems are based on modifying screen layouts that depict data tables so that they reflect end user specific company labels. In reality Swan underpins its software application products with a complete and fixed data structure even if customers use only a small fraction of this fixed structure. This situation is illustrated by case study results that are collated in Appendix C and represented conceptually by Figure 7.3.

\(^\text{19}\) Sage, like Swan, is a vendor of IT products. Sage specialises in database products and their applications in finance domains. Current version ERP products from Swan are built up Sage database products.
In Chapter 2 a reference to customisation methods developed by SAP was outlined. SAP ERP product customisation is centred on business process modelling to define a set of ERP product parameters and variables that can be initialised to meet customer preferences. However the database structure itself comprises a large set of tables that are predefined, independently of the customisation process, hence in this respect they are of a similar nature to data structures supplied by Swan. The SAP customisation approach does facilitate the introduction of user-defined structures by offering a so-called customer exit mechanism to modify the application model within a fairly strict set of constraints. Hence it can be expected that if the new approach developed in this study were used to analyse a relational database populated with a SAP tool, the system developer would have found similar classes of entity to that in the database of a Swan ERP system and where only a proportion of it is in use in respect to specific customer data sets.

Theoretically, the aim of a customisation process should be to provide individual customers with software packages that closely match their specific data-processing needs, and yet can readily be changed to meet future needs that have yet to be determined. The customisation approach proposed in this study is centred on defining and classifying common data structure requirements in groups of cognate companies, in the manner described in chapter 7. The analysis in this thesis of the stereotypical use of real data structures lends weight to the assumption that it is practical to develop and use a relatively well-defined classification of (1) information objects, that are common to all customers (2) and other object classes that are common only to some types of customer. By starting with a data structure which matches known preferences of a company type, manufacturing end users should be provided with data structures that more closely match their needs than could a more general and lowest common denominator data structure which can only be modified in vendor-defined ways, typically by modifying the labelling of data representations on a screen. A possible customisation process proposed on the basis of findings from this study is illustrated conceptually in Figure 9.2.

Logically, the use of such classifications should provide a better starting point for creating solutions than that offered by vendors of current generation ERP products which is based on the provision of a common data structure for all types of
companies. Hence this study suggests an improvement to current customisation practice typified by Swan and many other IT vendors. Table 9.2 illustrates the main differences found when comparing the use of data structures and customisation techniques by a small selection of contemporary IT product vendors with corresponding approaches enabled by this research.

<table>
<thead>
<tr>
<th>Nature of the vendor’s data structure.</th>
<th>Swan</th>
<th>SAP</th>
<th>Research approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic data structure of the product offered to all clients.</td>
<td>Generic data structure of the product offered to all clients.</td>
<td>Semi-generic data structure.</td>
<td></td>
</tr>
<tr>
<td>The complete generic data structure with modification based only on changing data values.</td>
<td>The complete data structure with limited modification capability.</td>
<td>A modifiable, semi-generic data structure that matches known preferences of a group of similar companies.</td>
<td></td>
</tr>
<tr>
<td>Modify screen layouts to reflect end user specific company labels.</td>
<td>Limited introduction of user-defined data structures plus screen layout modification capability.</td>
<td>Data structure that can be modified and extended by using methods and tools to add or remove objects, so as to match end user requirements within IT product and application system constraints.</td>
<td></td>
</tr>
</tbody>
</table>

Table 9.2 Comparison between the use of data structures and customisation approaches in the ERP domain.
Chapter 9

Comparison of the research study approach with other approaches reported in the literature

Stage 1: Analysis of current end-user data structures to develop data structures that match generic and group levels.

Tool used: Research tool to analyse data sets.

Stage 2: Definition of new vendor’s application based in the data structures generated by the application tool.

Figure 9.2 Illustrative use of the project methods and application toolset to generate end user systems based on the use of customisable data structures.

Stage 3: Implementation of new data structures for individual end-users.

Tool used: Research tool to implement and populate equivalent OO schemas.

Data Set End User 1

Data Set End User 2

Data Set End User 3

Data Set End User 4

Data Set End User 5

Data Set End User 6

Data Set End User 7

Data Set End User 8

Data Set End User 9

Data Set End User 10

Data Set End User 11

Data Set End User 12

Data Set End User 13

Data Set End User 14

Data Set End User 15

Data Set End User 16

Data Set End User 17

Data Set End User 18

Data Set End User 19

Data Set End User 20

Data Set End User 21

Data Set End User 22

Figure 9.2 Illustrative use of the project methods and application toolset to generate end user systems based on the use of customisable data structures.
9.3 CONCLUSIONS

It can be concluded that the data transformation and reuse approaches developed in this research study have the potential to provide advanced capabilities when compared with other approaches reported in the literature and reported as best industry practice. Essentially this is achieved because the data transformation methods and toolset developed take as input data values as well as data structures and apply state-of-the-art mapping and decision-support mechanisms. The methods and software toolset enable system developers to understand and reuse data contained in existing relational data tables. Outputs are generated semi-automatically in the form of data schemas. Such schemas can be used to represent different classes of end user data structure. Thereby data schemas generated can be used to improve customisation processes and more generally support the development of OO application systems.

Chapters 7, 8 and 9 report on a proof-of-concept demonstration of improved means of:

- Deploying object schemas, where the introduction of new entities results in an improvement in the quality of the final object schema.

- Implementing data transformation processes that are appropriate when seeking to reuse large manufacturing, engineering and sales data sets, also as required, automatically populate a target OODB.

- Providing enriched understandings for system developers to enable them to make better decisions about the definition of data structures and associated data transformation processes.

- Gaining an improved understanding of the viability of reusing legacy data structures.

- Understanding, comparing and customising data sets that share a similar data structure.

- Deploying existing and new IT products by improving the process of customising their underlying data structures.
10.0 CONTRIBUTIONS TO KNOWLEDGE

This research study has developed novel ways of reusing manufacturing, engineering and sales data and has shown how its methods and tools can be applied when developing new forms of application system built on a base of distributed object technology. In this context the thesis reports on the design, development and testing of a practical set of methods and toolset that structure and support the reuse of legacy data found in ERP databases. When compared with existing best practice and other research approaches reported in the literature, the project methods and toolset offer improved means of: (i) supporting human-centred reasoning about the reuse of relational data structures, data values and information profiles about data use, and (ii) transforming manufacturing data stored as relational tables into an equivalent data-object form.

The novelty of the project methods and toolset stems from the way in which they deploy new algorithms and techniques in order to provide improved understandings about legacy data, to transform legacy inputs in object-oriented data structures, (and thereby facilitate the generation and reuse of customisable data structures) and more generally facilitate the reuse of legacy data in new forms of application systems.

Proof-of-concept experiments were designed and developed to test and demonstrate the applicability and practicability of the methods and toolset. A case study was conducted centred on the analysis and reuse of real and typical customer sets of ERP data. This demonstrated how the project methods and toolset can be beneficially applied in example manufacturing scenarios.

Evaluation of the case study findings has shown that for the example scenarios the research methods and toolset demonstrated improved capabilities over previous
approaches primarily because they help generate rich and explicit understandings about legacy data structures and data sets, thereby enabling system developers to:

(1) create and reuse equivalent object-like data structures, in support of the design and specification of new wholly distributed object-based systems,

(2) identify and classify similarities and differences between industrial data sets, in order to gain a detailed and explicit understanding of common data requirements of manufacturing system and,

(3) deploy customisable data structures in the ERP domain.

Table 10.1 summarises characteristic capabilities of the data transformation approach developed. The reader can compare the capabilities with those of other approaches reported in the literature and summarised by Table 4.1.

<table>
<thead>
<tr>
<th>Schema transformation</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input information required</strong></td>
<td><strong>Mapping rules</strong></td>
</tr>
<tr>
<td>Relational schema: table, attributes and key relationships</td>
<td>A 4-step approach based in the relational schema and data values information.</td>
</tr>
<tr>
<td>Data values, information about the use of the data structure.</td>
<td>Toolkit that implements the data transformation approach by semi-automatically extracting the input information and generating an equivalent object schema. The toolkit provides information about data values to guide developers when taking decisions to introduce new entities or remove existing entities.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data population</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transfer rules</strong></td>
<td><strong>Implementation</strong></td>
</tr>
<tr>
<td>Data is extracted from the relational source and is restructured to conform the equivalent object-based schema. Data population is achieved in two steps, first values of simple attributes are transferred and then relationship attribute values are stored.</td>
<td>The toolkit automatically creates an OODB which conforms to the equivalent object data structure and populates it with values found in the source relational database.</td>
</tr>
</tbody>
</table>

Table 10.1 Summary of the data transformation approach developed during this study.

10.1 NOVEL FEATURES OF THE RESEARCH

Novel features of the research approach are considered below in relation to the research objectives stated in Chapter 3.
Chapter 10  Conclusions

Extracting information about data structures

- The research has analysed two types of information, namely: (1) semantic structure of the database and (2) data values contained in the database. Generally it was assumed that the semantic structure will have been determined by an IT vendor (such as a vendor of an application software product) and that this structure would be common to all customer data sets whereas differences in the actual data values will account for variation in the data populated and used by each manufacturer business.

- In this research methods and tools have been developed and tested such that they can access both main sources of information in a semi-automatic fashion and thereby can structure and support users (e.g. system developers) in their decision-making and where they are required to introduce information which cannot be made available in an automated way.

Creating equivalent object-like data structures.

- The novel data transformation approach has been developed that is capable of creating object like data structures from existing relational databases. In so doing it improves upon previous data transformation approaches by supporting the removal of redundant entities and the introduction of new entities.

- The object-like data structure defined by the approach has improved the understanding of users about input data structures and particularly with respect to informing decision-making about when to split large tables and create new entities.

- The proof-of-concept toolset has shown to generates explicit computer processable models that encode understandings about data structures and data sets in the form of database information and screen displays so as to enable users to apply well-defined transformation rules.

Define an equivalent OODB

- For legacy data which is representative of various manufacturing domains, the new approach and toolkit was found to have a capability to define and create
equivalent object-based data structures and to use these structures when automatically populating an OODB.

**Transferring and reusing existing data records when populating OODBs**

- The toolset has been proven to successfully translate data derived from a source relational database to create data entries in an OODB that conforms to an equivalent object data structure. A data transfer approach was specified, developed and automated that extracts data values, reformats data to match defined object-like structures and populates OODB entities with suitable data values.

- For legacy data, representative of many manufacturing domains, this data transfer process has proven to populate target OODBs that are free of information loss or inconsistency.

**Abstracting and detecting specific structures of data**

- By testing the approach on representative types of manufacturer end-user data sets the potential of the approach to detect and differentiate a rich set of object properties of legacy manufacturing data has been demonstrated.

- The research has also developed new understandings about the reuse of data sets in different application scenarios. Focus here was on reusing general-purpose software applications (and particularly ERP products and their underlying legacy data structure) in different end user application scenarios. The observations made offer a starting point for further investigation and solution development, where custom designed data structures can readily be derived from a common data structure to closely match end-user requirements.

**10.2 SUMMARY OF RESEARCH ACHIEVEMENTS**

Figure 10.1 illustrates and summarises the main achievements of this study and how the research methods and approach were developed, implemented and tested.
Figure 10.1 Illustrative summary of achievements made during the research study.

Conclusions
10.3 FUTURE WORK

This research has developed and tested a methodology and toolset that have the potential to facilitate an improved reuse of legacy data in common manufacturing scenarios. However further investigation and development is required before the method and toolset can be used effectively as part of a broader approach to accessing and managing enterprise data. Therefore additional research and development is recommended in the following key areas:

Facilitating data sharing: Although the approach has been developed and tested primarily for use in the ERP domain, evidently its use can be extended to other manufacturing domains. This would facilitate the definition of common data structures for typical manufacturing applications and thereby data sharing.

Customisation: The research experiments have derived and reused an initial set of semi-generic data structures that characterise common classes of manufacturing scenarios. Hence further research is required to investigate how these data structures might be related and used to support the development of ‘open’ and readily customisable vendor solutions.

Defining data standards: By enabling access and analysis of new understandings about common types of manufacturing data the research findings would be reapplied with a view to developing data standards. However further research is required to consider how common data structure definitions developed using the method and toolset might align with ongoing standards initiatives, such as MANDATE or STEP. Clearly many forms and types of common data structure could be derived and developed related to the integration and interoperation of multi-vendor products in multi-application, multipurpose scenarios.

Exploiting new technologies: A primary objective of the research was to reuse legacy data in such a way that resultant target systems can fully exploit distributed object features. Here the research approach and toolset apparently provide generally applicable means of creating and populating equivalent OODBs. However to meet this objective more completely, further analysis should be conducted with respect to how equivalent object data entities might interoperate effectively within distributed object systems.
REFERENCES


142


[Java Blend] www.sun.com/software/javaBlend


References


[Metadata Coalition] [www.mdcinfo.com/index.htm](http://www.mdcinfo.com/index.htm)


[OMG] [www.omg.org](http://www.omg.org)
[Ontos] www.ontos.com


References

[SEBPC] www.staff.ecs.soton.ac.uk/~ph/sebpc


APPENDIX A

RELATIONAL DATABASES VS. OBJECT-ORIENTED DATABASES

A.0 RELATIONAL DATABASES

The relational model was originally proposed by Codd in 1970 and is the most widely purchased database technology today. In the relational model, data is represented in two-dimensional tables called relations. In a table, the columns are the attributes of entities and all values of an entity are stated in a row, also called a tuple [Du and Wolfe, 1997]. Relationships are determined by defining shared attributes between different tables.

Relational technology also provides a mathematical basis for manipulating records and maintaining consistency. An important feature of relational databases is data independence. This means that applications developed to use the database are not dependent on physical data structure [Ryan and Smith, 1995].

A.1 OBJECT-ORIENTED DATABASES

Research prototypes of object oriented databases began to emerge in the late 1980s. Essentially an object-oriented database approach combines concepts from two areas, namely object-oriented programming and semantic modelling [Du and Wolfe, 1997]. The objects in an OODB encapsulate attributes that represent structure and operations. Collectively these attributes describe the behaviour of software applications. Objects are represented by classes that are grouped in a hierarchy. OODBs deploy unique and permanent identifiers for each object. This identifier is implemented and known by the system but is independent of the value of the object. Since the identification of an object does not depend on the value of attributes the identification of information
objects is improved when compared with the use of keys of tuples in the case of a RDB [Cooper, 1997].

A further major advantage of OODBs over relational databases lies in their rich modelling power. This facilitates the representation of complex data, an integration of data operations with capabilities of conventional programming languages, and features like encapsulation and inheritance.

There remain concerns with respect to the maturity (i.e. lack of technological stability) of OODBs. The ODMG (Object Database Management Group) has sought to address this problem. The ODMG seeks to bring ODBMS vendors, toolmakers and end users together to put together with the aim of providing a standard industry-wide architecture for OODBs to which all object-oriented systems can conform. The ODMG-standard synthesises existing SQL, OMG object and object programming language standards into a common standard for building ODBMS applications. The standard ensures that applications will be portable across compliant ODBMSs. The ODMG object model is intended to be address more specifically database technology, and to include the OMG model as a subset. The standard reduces user dependence on a particular ODBMS vendor and preserves software development investment as users needs grow and change [Cattell et al, 1997].

Therefore the ODMG seeks to define:

- An ODBMS architecture;
- A common data model for ODBMSs to enable a similar level of inter-operability to be achieved to that in relational systems;
- A data definition language, ODL, that forms a concrete specification of the operations permitted with respect to schemata defined in the data model;
- A query language, OQL, which provides an interface for posing ad hoc queries, and
- A number of bindings to existing object-oriented programming languages, such as C++, Smalltalk and Java [Cooper, 1997].
# A.2 COMPARISON BETWEEN RDBs AND OODBs

<table>
<thead>
<tr>
<th></th>
<th>RDBMS (Query performance)</th>
<th>OODB (Design performance)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data structure</strong></td>
<td>Flat structure represented by relational tables.</td>
<td>Hierarchical structure of classes.</td>
</tr>
<tr>
<td></td>
<td>Pre-defined types for field records.</td>
<td>Any user-defined structure and relationship is support to define types for fields' records.</td>
</tr>
<tr>
<td></td>
<td>Manages simple data structures well.</td>
<td>Manages complex data and complex operations</td>
</tr>
<tr>
<td><strong>Semantic schema</strong></td>
<td>Information focuses on number and value of data.</td>
<td>Information focuses on structure and relationships between tables</td>
</tr>
<tr>
<td></td>
<td>Semantic mismatch with OO languages</td>
<td>Support OO modelling.</td>
</tr>
<tr>
<td><strong>Operation support</strong></td>
<td>SQL queries. Use of joins limit performance of search queries.</td>
<td>Any user-definition operation. Data navigation increases the performance of applications accessing to high-related data.</td>
</tr>
<tr>
<td></td>
<td>Data duplication and null common entries increase storage requirements and limit application of object-oriented methods</td>
<td>No data duplication or null entries.</td>
</tr>
<tr>
<td><strong>Technology records</strong></td>
<td>Proven technology (10+ years) with many applications</td>
<td>Still experimental with few substantive applications.</td>
</tr>
<tr>
<td><strong>Schema evolution</strong></td>
<td>Potentially complex normalisation of entity-relation diagram required</td>
<td>Complex initial design but easy to maintain and previously-defined objects may be reused</td>
</tr>
<tr>
<td><strong>Architecture issues</strong></td>
<td>Centralised systems that support the entire requests for applications deferring scalability</td>
<td>Distributed architecture that permits scalability in clients and servers and extensibility in terms of moving physically objects dynamically and transparently.</td>
</tr>
</tbody>
</table>

Table A.1 Comparison between RDBs and OODBs. This table was constructed by unifying information obtained primarily from the following sources: [Layden, 1993] [Srinivasan and Chang, 1997] [Barghouti et al, 1995]
This section describes the code structure of the software toolset developed during this research. The software toolset were generated in Java language. Java classes were divided within five Packages according the functionality that they provide. Firstly class diagrams that illustrate the main classes involved in each activity of the research toolset is described. Secondly the overall structure of the research toolset is outlined in terms of the function classes contained in each package. Finally some of the Java code is reproduced.

**B.0 DIAGRAM CLASSES**

**B.0.1 Activity 1: Extracting the structure of the original information stored in the RDB**  
Objective: Extract semantic information from the relational database to reveal the structure of the data tables in terms of table names, type and name of their attributes, primary keys and foreign keys.

**B.0.2 Activity 2: Generating an equivalent OO schema**  
Objective: Analyse the structure of the RDB according to the fourth-step algorithm to define and generate an equivalent class hierarchy.
B.0.3 Activity 3: Analysis of data schema

Objective: Provide information about the potential uses of data in the relational database and facilitate the reuse of this information by enabling modification of the class schema generated in Activity 2.

B.0.4 Activity 4: Defining an OO database

Objective: Create the class files (in the Java language) that represent the hierarchy of classes and description files required by the OODB compiler.
**B.0.5 Activity 5: Object instance population**

**Objective:** Populate the OODB with data records from the RDB.

Data values $\rightarrow$ transfdata $\rightarrow$ storedata $\rightarrow$ OODB

---

**JAVA Compiler**

GetRdb.connecting $\rightarrow$ GetRdb.query $\rightarrow$ TransData.transfData $\rightarrow$ MngMapping.mainManager

StoreDb.classmapin $\rightarrow$ StoreDb.classchange $\rightarrow$ StoreDb.databaseIn

---

**OODB Compiler**

TransData.Createinherit $\rightarrow$ TransData.Createreferclass $\rightarrow$ TransData.creatswitch

Dbt_classes.java $\rightarrow$ Inher_classes.java $\rightarrow$ Aggr_lasses.java

---

MngMapping.mainManager $\rightarrow$ TransData.runProcess $\rightarrow$ Compile files using the POET compiler

Dbt_classes.java $\rightarrow$ Classes.java $\rightarrow$ Database_opt.

Inher_classes.java $\rightarrow$ Aggr_lasses.java $\rightarrow$ refer_lasses.java $\rightarrow$ Switch_relation.java

156
B.1 PROGRAM STRUCTURE

**GETRDB Package:** classes that provide functionality to connect and access the RDB.

*class connecting* It provides connection to a RDB through a JdbcOdbc driver.

*class keylnfo* It gets information about PK and FK of the RDB. If it is not available from the RDB dictionary a set of dialogs are display for the user to introduce this information. *It uses storeDB classes to store the information results.*

*class queries* It supports functionality to query the RDB to get information, introduce and remove values using JDBC methods. *It uses the class connecting to connect with the RDB.*

*class table info* It supports functionality to get information from the RDB. It uses JDBC methods to get tables and attributes information. *It uses storeDB classes to store the information results.*

**STOREDB Package:** classes to implement the Mapping Information Repository.

*class databaseinfo* class to store information about the RDB structure such as tables, attributes, PK and FK. It contains classes column and foreignKey.

*class column* Class to store relational attributes.

*class foreignKey* Class defined to store foreign keys relationship.

*class classmapinfo* Class to store the structure of the equivalent OO structure. It contains classes column and relation.

*class relation* Class to store association and aggregation relationships.

*class classchanges* It contains information that relates new entities with the original tables from where they were split.

**MAPOO Package:** classes that implement the research transformation algorithm.

*class algorithm* It implements the 4 steps of the algorithm to define an equivalent OO schema.

*class algor5* It implements algor5 step of the algorithm that permits user to modify further the equivalent class structure. It uses storeDB to store the results for the mapping process.

*class Createclasses* It creates java files that describes the classes generated during the mapping algorithm. It gets the information for the storeDB files.

*class Createreports* It creates a report about data values and use of the relational structure.

**TRANSDATA Package:** classes to support the data transfer process.

*class transfdata* It provides functionality needed to create and define the OODB and to transfer the data from the RDB to the OODB. It uses the information from the storeDb classes.
class Creatrclass This class provides functionality needed to create temporal java files that will be used to store simple attribute values in the OODB. It creates a file for each class in the OODB with the attribute name and the functionality to create persistent objects.

class Createinherfile This class provides functionality needed to create temporal java files that will be used to store inheritance values in the OODB. It creates a file for each subclass and provides the functionality required to modify objects in the OODB to keep super/sub integrity.

class createreferclass This class provides functionality needed to create temporal java files that will be used to give values to relationship attributes. It creates a file for each association and aggregation relationships and the functionality required to add the reference objects in each persistent object of the OODB.

class creatswitch It creates a temporal java file that allows the program to run the java files created for the last three classes.

class runprocess It defines and creates an OODB by:

Creating an file.opt that is required for the Poet OODB to compile the persistent classes.
Run a process that compiles the classes that will be define the OODB and all the files that will be used to populate the OODB using an special compiler for POET OODB.

GUI Package: User Interface classes.

class Algodialog extends Dialog It shows a dialog with a question and two buttons option.

class chanfra extends Dialog It shows a dialog to display a source table from where attributes can be transferred to define a new table. It Contains:

class labtext extends Panel

class attripanel extends Panel It contains:
class tabpanel extends Panel

class pantable extends Panel

class choosendial extends Dialog Displays attributes of a table where some can be selected. It contains: Panel tabpanel.

class fltable extends choosendial It displays a set of attributes to select an attribute as foreign key,
class refrays extends choosendial It displays a set of attributes to select an attribute referred by a foreign key.

class showPk extends choosendial It displays a set of attributes to select attributes as PKs.

class chosessubcl extends Dialog It displays a set of alternative choices. It contains

class chgroupscroll extends ScrollPane

class dgtabsplinull extends Dialog It displays the outputs of user decision to confirm changes results.
It contains:
Appendix B

Research Software Toolset Specification

class showtablab extends Panel
It displays each mapped class to allow user to make further changes.

It contains:

class scrollpanelhori extends ScrollPane It displays tables and attributes

class scrollpanelvert extends ScrollPane Display changes introduced by the user

class panelcentral extends Panel It displays the mapped class that users can make further changes
to. It contains:

class groupscroll extends ScrollPane

class mulop extends Dialog It contains a set of choices

class namedbdil extends Dialog Dialog to request a user input. It contains labtext Panel.

class notedial extends Dialog information dialog with an OK button.

class process extends Dialog Information dialog.

class viewFrame extends Frame Main screen of the application. It displays parameters of the process
and information about the relational and the OODB database. It contains:

class panelalg extends Panel Information about algorithm processing state. It contains:

class groupbuttonpanel extends Panel Check box with algorithm steps.

class panconect extends Panel It displays information about RDB name and driver use to
access the RDB.

class paneldbs extends Panel It contains two treetab panels to display the structure of the
relational database and the equivalent object-oriented schema.

class treetab extends Panel It contains a tree-view of table or class with the
attributes.

MNGMAPPING Package:

class mainmanager Main class. It displays the main Frame viewFrame that includes the interface to
start the process.

Other classes

class storedata This class is outside the Java compiler environment. It is compiled together with the
persistent classes and the temporal java files using the OODB compiler. This class links the application
class and the OODB classes. An object of this class is created by transfdata class. This object will start
the transfer process by calling the temporal classes created for this purpose.

159
B.2 EXAMPLE CODE

Class querys

/* class querys manages querys invocation to the Relational database. */
package getRdb;

import java.sql.*;
import java.util.Vector;
import java.util.Hashtable;
import storeDb.column;

public class querys{
    private connecting conq; //attribute to provide database connection properties.
    private Statement stmt; //attribute to query the RDB
    private Hashtable table; // attribute to collect results of the query
    private int numberOfColumns; //attribute to collect number of columns in query outputs.
    private Vector namecol=new Vector();

    public querys OO //empty constructor
    public querys(connecting con){ //constructor that open a link with the RDB for queries.
        try{
            this.conqu=con;
            this.stmt=conqu.con.createStatement();
        }catch(SQLException ex) {
            System.err.print("SQLException: ");
            System.err.println(ex.getMessage());
        }
    this.table=new Hashtable();
}

    /* method to execute queries that modify the relational tables (e.g. DROP or INSERT)*/
    public void upquery(String que){
        try{
            int rs=this.stmt.executeUpdate(que);
        }catch(SQLException ex) {
            System.err.print("SQLException: ");
            System.err.println(ex.getMessage());
    }

    /*method to execute queries that returns a set of relational records.
The results of the query are stored in the Hashtable table*/
    public void exequery(String que){
        try{
            ResultSet rs= this.stmt.executeQuery(que);
            ResultSetMetaData rsmd = rs.getMetaData();
            numberOfColumns = rsmd.getColumnCount();
            int numrows=0;
            while (rs.next()) {
                Vector row=new Vector();
            }
}
for (int i = 1; i <= numberOfColumns; i++) {
    Object data = rs.getObject(i);
    if (rs.wasNull()) {
        String type = rsmd.getColumnTypeName(i);
        if (type.equals("TEXT")) data = "";
        else if (type.equals("LONG")) data = new Integer(0);
        else if (type.equals("CURRENCY")) data = new 
                java.math.BigDecimal("0");
        else if (type.equals("DATETIME")) data = new 
                java.util.Date(19,0,1);
    } catch(SQLException ex) {
        System.err.println("SQLException: ");
        System.err.println(que);
        System.err.println(ex.getMessage());
    }
}

for(int nr=1;nr<=numberOfColumns;nr++){
    namecol.addElement(rsmd.getColumnName(nr));
    row.addElement(data);
    Integer numRow= new Integer(numrows);
    this.table.put(numRow,row);
    numrows++;
}

public void exequery(String que, Vector classattri) {
    exequery(que);
    int max = 0;
    if (classattri.size() > namecol.size()) max = namecol.size();
    else max = classattri.size();
    for (int co = 0; co < max; co++) {
        String shortclass = ((co < namecol.size()) ? (String) namecol.elementAt(co) :"");
        String shortresult = (String) namecol.elementAt(co);
        if (!shortclass.equals(shortresult)) {
            boolean isInDb = false;
            for (int it = 0; it < namecol.size(); it++) {
                if (shortclass.equals(((String) namecol.elementAt(it))){
                    isInDb = true;
                } if (isInDb == true) {
                    this.short_columns(co, shortclass, shortresult);
                }
            }
        }
    }
}

/* method to sort columns of a results of a query according the order given by a vector of attribute names*/
void sort_columns(int pos, String nameattri, String nameintab) {
    int tabpos = 0;
    for (int nc = 0; nc < namecol.size(); nc++) {
        if (((String) namecol.elementAt(nc)).equals(nameattri)) tabpos = nc;
    } for (int cc = 0; cc < table.size(); cc++) {
        Integer num = new Integer(cc);
    }
Vector row=(Vector)table.get(numr);
Object posobj=row.elementAt(pos);
Object tabposobj=row.elementAt(tabpos);
row.setElementAt(tabposobj,pos);
row.setElementAt(posobj,tabpos);
}

namecol.setElementAt(nameattri,pos);
namecol.setElementAt(nameintab,tabpos);
}

public Hashtable getTable(){
    return (this.table);}

public int getnumberofColumns(){
    return (this.numberofColumns);}
}

Class runprocess

/* class transdata compile java files that will be used in the OODB environment
   using the POET compiler*/

package transdata;
import java.util.Vector;
import java.io.*;
import GUI.processing;
import GUI.noteDialog;
import GUI.viewFrame;

public class runprocess{
    private Runtime r;
p

    /*constructor of the class.
    Input: name of the java class that will be declared persistent*/
    public runprocess(Vector Nameclasses){
        this.createfileopt(Nameclasses); }

    /* method to start a new process to compile java files with the Poet compiler
    the compiler required
    dir: directory where the java classes are stored
    compclass: name of the new OODB database
    dir: directory where ptjavac.opt is located */
    public void runcomp (String oodbname, viewFrame parent){
        this.r=Runtime.getRuntime();
        this.p=null;
        String dir="C:\access_db\mapping\classes\poetSO\";
        String comppoeet[]={"ptjavac","-d "+dir+"oodbname","-conf "+dir+"ptjavac.opt"];
        processing poetcomp=new processing(parent, "Ptjavac compiling "+oodbname+"... ");
        //information dialog
        try{llcode
            to capture messages from the Poet compiler process to be display in the
            Ilcurrent Java environment.
            poetcomp.show();
            this.p=r.exec(comppoeet);
            InputStream ins=this.p.getInputStream();
            DataInputStream insr=new DataInputStream(ins);
            String input="";

            //code to capture messages from the Poet compiler process to be display in the
            //current Java environment.
            poetcomp.show();
            this.p=r.exec(comppoeet);
            InputStream ins=this.p.getInputStream();
            DataInputStream insr=new DataInputStream(ins);
            String input="";
```java
while((input=insr.readLine())!=null){
    System.out.println(input);
    insr.close();
    ins.close();
    this.p.waitFor();
    poetcomp.hide();
}
catch(Exception e){
    notedial ncomp=new notedial(parent,"Exception in compiling ptjavac");
    poetcomp.hide();
    ncomp.show();}
}
/*method to create the file ptjavac.opt that name all java classes that
will be persistens as required by the Poet OODB compiler.*/
void createfileopt(Vector Nameclasses){
    try{
        File ffile=new File("/access_db/mapping/classes/poet50","ptjavac.opt");
        FileWriter ffilewr=new FileWriter(ffile);
        BufferedWriter buffile=new BufferedWriter(ffilewr);
        String oodb="[schemata]\OOdicty]\n"
            +"onefile=true\n"
            +"[databases]\ OOpro]\n"
            +" oneFile = true\n";
        buffile.newLine();
        buffile.write(oodb,0,oodb.length());
        buffile.newLine();
        for (int ac=0;ac<Nameclasses.size();ac++){
            String storeclass="[c1asses]\"+(String)Nameclasses.elementAt(ac)\"]\n"
                +" persistent=true\n"
                +"hasextent=true"
            ;
            buffile.newLine();
            buffile.write(storeclass,0,storeclass.length());
            buffile.newLine();
        }
        buffile.close();
        ffilewr.close();
    }catch(java.io.IOException e){
        System.out.println("error writen file: "+e.getMessage());
    }
}
```
APPENDIX C

INFORMATION FILES RESULTS FROM THE CASE STUDY

C.0. EXAMPLE MAKE-TO-STOCK COMPANY

Empty tables:
- CAP_ROUTINGS_FILE
- BOM_ISSUED_SERIALS
- BOM_SERIALS_NUMBER

Tables in Use:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS_STOCK_CONTROL_FILE</td>
<td>num. records=735</td>
</tr>
<tr>
<td>double ABNORMAL_DEMAND_FAC</td>
<td>[ in use=735] [ diff 1 value 0.0 ]</td>
</tr>
<tr>
<td>String ALT_SUPPLIER</td>
<td>[ in use=11 ] [ diff 7 ]</td>
</tr>
<tr>
<td>double ASSEMBLY_LEAD_TIME</td>
<td>[ in use=735] [ diff 1 value 0.0 ]</td>
</tr>
<tr>
<td>double AVERAGE_PRICE</td>
<td>[ in use=735 ] [ diff 34 ]</td>
</tr>
<tr>
<td>double AVERAGE_WEIGHT</td>
<td>[ in use=735] [ diff 1 value 0.0 ]</td>
</tr>
<tr>
<td>String BAR_CODE</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>short BAS_QTY_PER_PACK_CODE</td>
<td>[ in use=735 ] [ diff 3 ]</td>
</tr>
<tr>
<td>String BATCH_TRACEABILITY</td>
<td>[ in use=735 ] [ diff 2 values= [N],[Y]]</td>
</tr>
<tr>
<td>String BESPOKE_IND</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>int BESPOKE_NUM</td>
<td>[ in use=735 ] [ diff 1 value 0 ]</td>
</tr>
<tr>
<td>String BIN_NUMBER_1</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String BIN_NUMBER_2</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String BOM_EXPLOSION</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String BOM_IGNORES_IND</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String BULK_ISSUE_IND</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String BUYER_CODE</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CATEGORY</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CE_COMMODITY_CODE</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CHANGE_NOTE_NUMBER</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CHANGE_PENDING</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CLASSIFICATION_1</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CLASSIFICATION_2</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String COMMODITY_CODE</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String CONFIG_IND</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>double CONVERSTION_1</td>
<td>[ in use=735 ] [ diff 1 value 1.0 ]</td>
</tr>
<tr>
<td>double CONVERSTION_2</td>
<td>[ in use=735 ] [ diff 1 value 0.0 ]</td>
</tr>
<tr>
<td>String COS_NOMINAL_CODE</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>String COS_VAR_NOMINAL_CODE</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>double COST_VALUE_SOLD_MTH</td>
<td>[ in use=735 ] [ diff 1 value 0.0 ]</td>
</tr>
<tr>
<td>double COST_VALUE_SOLD_YTD</td>
<td>[ in use=735 ] [ diff 1 value 0.0 ]</td>
</tr>
<tr>
<td>String COSTING_ONLY_IND</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
<tr>
<td>java.util.Date DATE_LAST_ISSUE</td>
<td>[ in use=735 ] [ diff 173 ]</td>
</tr>
<tr>
<td>java.util.Date DATE_LAST_ORDER</td>
<td>[ in use=735 ] [ diff 45 ]</td>
</tr>
<tr>
<td>java.util.Date DATE_LAST_STOCK_TAKE</td>
<td>[ in use=735 ] [ diff 1 value 1999-11-30 00:00:00.00 ]</td>
</tr>
<tr>
<td>java.util.Date DATE_NEXT_ORDER_DUE</td>
<td>[ in use=735 ] [ diff 1 value 1999-11-30 00:00:00.00 ]</td>
</tr>
<tr>
<td>java.util.Date DATE_OF_LAST_SALE</td>
<td>[ in use=735 ] [ diff 78 ]</td>
</tr>
<tr>
<td>String DRAWING_NUMBER</td>
<td>[ in use=0 ] [ diff 0 ]</td>
</tr>
</tbody>
</table>
Appendix C

Information Files results from the Case Study

String EBQ_IND [ In use=9] [ diff 1 value 0]
java.util.Date END_DATE [ In use=735] [ diff 1 value 1999-11-30 00:00:00.0]
String ENG_CHANGE_REF [ In use=0] [ diff 0]
String ENG_DEPT_USE_1 [ In use=0] [ diff 0]
String ENG_DEPT_USE_2 [ In use=101] [ diff 1 value Y]
double ENG_PRICE [ In use=735] [ diff 1 value 0.0]
java.util.Date EXPIRY_DATE [ In use=735] [ diff 1 value 1999-11-30 00:00:00.0]
double EXPECTED_BUYING_PRICE [ In use=735] [ diff 1 value 0.0]
String FACTOR_PRICES_IND [ In use=0] [ diff 0]
String FAMILY_BILL_PART_NO [ In use=0] [ diff 0]
double FORWARD_ORDER_QTY [ In use=735] [ diff 1 value 0.0]
String FREE_ISSUE_IND [ In use=0] [ diff 0]
double HEIGHT [ In use=735] [ diff 1 value 0.0]
double INSP_QTY_ALLOCATED [ In use=735] [ diff 1 value 0.0]
double INSPECTION_QUANTITY [ In use=735] [ diff 9]
java.util.Date ISSUE_DATE [ In use=735] [ diff 1 value 1999-11-30 00:00:00.0]
String ISSUE_NUMBER [ In use=723] [ diff 6]
String ITEM_DESCRIPTION_1 [ In use=734] [ diff 719]
String ITEM_DESCRIPTION_2 [ In use=249] [ diff 247]
String ITEM_NUMBER [ In use=735] [ diff 735]
String KIT_STOP_IND [ In use=0] [ diff 0]
String LAST_GEN_SERIAL_NO_1 [ In use=0] [ diff 0]
String LAST_GEN_SERIAL_N_2 [ In use=0] [ diff 0]
double LAST_MONTHS_STOCK [ In use=735] [ diff 246]
double LAST_MTHS_AVER_STOCK [ In use=735] [ diff 1 value 0.0]
double LAST_SERIAL_NO_1 [ In use=735] [ diff 1 value 0.0]
double LAST_SERIAL_NO_2 [ In use=735] [ diff 1 value 0.0]
double LATEST_BUYING_PRICE [ In use=735] [ diff 32]
double LEAD_TIME [ In use=735] [ diff 12]
double LENGTH [ In use=735] [ diff 1 value 0.0]
String MAIN_SUPPLIER [ In use=632] [ diff 42]
double MAXIMUM_STOCK_LEVEL [ In use=735] [ diff 1 value 0.0]
double MIN_ORDER_QTY [ In use=735] [ diff 1 value 0.0]
short MIN_STOCK_MONTHS [ In use=735] [ diff 1 value 0]
double MINIMUM_STOCK_LEVEL [ In use=735] [ diff 1 value 0.0]
String MPS_IND [ In use=356] [ diff 2 values= [N], [Y]]
String MRP IGNORES_IND [ In use=0] [ diff 0]
String MRP_SORT_KEY [ In use=0] [ diff 0]
String MULTIPRICE_IND [ In use=0] [ diff 0]
String MULTI_PUR_IND [ In use=0] [ diff 0]
String MULTI_SALE_IND [ In use=735] [ diff 1 value N]
String MULTI_SPRICE_IND [ In use=735] [ diff 1 value N]
String NOMINAL_COD_1 [ In use=0] [ diff 0]
String NOMINAL_COD_2 [ In use=0] [ diff 0]
String NOMINAL_COD_3 [ In use=0] [ diff 0]
double NORMAL_ORDER_QTY [ In use=735] [ diff 1 value 0.0]
double OLD_BUYING_PRICE_1 [ In use=735] [ diff 1 value 0.0]
double OLD_BUYING_PRICE_2 [ In use=735] [ diff 1 value 0.0]
double OLD_BUYING_PRICE_3 [ In use=735] [ diff 1 value 0.0]
double OLD_BUYING_PRICE_4 [ In use=735] [ diff 1 value 0.0]
double OLD_BUYING_PRICE_5 [ In use=735] [ diff 1 value 0.0]
double ORDER_UP_TO [ In use=735] [ diff 1 value 0.0]
double OVERALL_STOCK_LEVEL [ In use=735] [ diff 249]
String OWNER [ In use=0] [ diff 0]
String PARTCATEGORY [ In use=0] [ diff 0]
short PARTTYPE [ In use=735] [ diff 6]
short PART_TYPE_BOM_FLAG [ In use=735] [ diff 1 value 0]
String PERCENT_BILL_IND [ In use=166] [ diff 1 value N]
short PI_REPORTING_YEAR_IND [ In use=735] [ diff 1 value 0]
short PI_REPORTING_IND [ In use=735] [ diff 1 value 0]
Appendix C

Information Files results from the Case Study

String PLANING_ONLY_IND [ In use=0 ] [ diff 0 ]
double PREV_MONTHS_STOCK [ In use=735 ] [ diff 256 ]
String PRIMARY_ITEM_IND [ In use=735 ] [ diff 1 value N ]
String PRODN_DEPT_U_1 [ In use=0 ] [ diff 0 ]
String PRODN_DEPT_U_2 [ In use=0 ] [ diff 0 ]
String PRODUCT_GROUP [ In use=735 ] [ diff 9 ]
short PURCHASE_CURRENCY [ In use=735 ] [ diff 1 value 0 ]
String QC_IND [ In use=0 ] [ diff 0 ]
double QTY_ALLOCATED [ In use=735 ] [ diff 33 ]
double QTY_ON_ORDER [ In use=735 ] [ diff 23 ]
double QTY_ORD_THIS_MONTH [ In use=735 ] [ diff 1 value 0.0 ]
double QTY_ORD_YTD [ In use=735 ] [ diff 1 value 0.0 ]
short QTY_PER_PACK_CODE_1 [ In use=735 ] [ diff 4 ]
short QTY_PER_PACK_CODE_2 [ In use=735 ] [ diff 3 ]
short QTY_PER_PACK_CODE_3 [ In use=735 ] [ diff 1 value 0 ]
double QTY_SOLD_MTH [ In use=735 ] [ diff 14 ]
double QTY_SOLD_YTD [ In use=735 ] [ diff 86 ]
String RELIFE_CODE [ In use=0 ] [ diff 0 ]
double RELIFE_COST [ In use=735 ] [ diff 1 value 0.0 ]
short REORDER_LEVEL [ In use=735 ] [ diff 1 value 0.0 ]
short REORDER_POLICY [ In use=735 ] [ diff 2 values= [ 0 ], [ 1 ] ]
double REORDER_QTY [ In use=735 ] [ diff 2 values= [ 0.0 ], [ 114.0 ] ]
String REVISION_NUMBER [ In use=0 ] [ diff 0 ]
double SAFETY_STOCK [ In use=735 ] [ diff 1 value 0.0 ]
short SALES_CURRENCY [ In use=735 ] [ diff 1 value 0 ]
String SCRAP_IND [ In use=735 ] [ diff 4 ]
String SCRAP NOMINAL_CODE [ In use=0 ] [ diff 0 ]
double SCRAP_PERCENT [ In use=735 ] [ diff 2 values= [ 0.0 ], [ 1.0 ] ]
double SELLING_PRICE_1 [ In use=735 ] [ diff 1 value 0.0 ]
double SELLING_PRICE_2 [ In use=735 ] [ diff 1 value 0.0 ]
double SELLING_PRICE_3 [ In use=735 ] [ diff 1 value 0.0 ]
double SELLING_PRICE_4 [ In use=735 ] [ diff 1 value 0.0 ]
double SELLING_PRICE_5 [ In use=735 ] [ diff 1 value 0.0 ]
String SERIAL_NO_IND [ In use=0 ] [ diff 0 ]
String SERIAL_NO_GEN [ In use=0 ] [ diff 0 ]
int SHELF_LIFE [ In use=735 ] [ diff 1 value 0 ]
double STANDARD_COST [ In use=735 ] [ diff 46 ]
java.util.Date START_DATE [ In use=735 ] [ diff 1 value 1999-11-30 00:00:00.0 ]
double STK_NO_OF_BASE_UNITS [ In use=735 ] [ diff 1 value 1.0 ]
short STK_QTY_PER_PACK_CODE [ In use=735 ] [ diff 3 ]
String STOCK_MAY_GO_NEGATIVE [ In use=0 ] [ diff 0 ]
String SUB_CON_NOMINAL_CODE [ In use=0 ] [ diff 0 ]
String SUBCON_BULK_ISSUE [ In use=0 ] [ diff 0 ]
String SUMM_0SSUE_IND [ In use=0 ] [ diff 0 ]
String SUPERSEDES_ITEM [ In use=0 ] [ diff 0 ]
String SUPERSEDED_BY_ITEM [ In use=0 ] [ diff 0 ]
double SYS_QTY_ALLOCATED [ In use=735 ] [ diff 450 ]
double SYSTEM_LEAD_TIME [ In use=735 ] [ diff 1 value 0.0 ]
double TBA_QUANTITY [ In use=735 ] [ diff 1 value 0.0 ]
double THIS_MONTHS_STOCK [ In use=735 ] [ diff 1 value 0.0 ]
double VAL_ORD_THIS_MONTH [ In use=735 ] [ diff 1 value 0.0 ]
double VAL_ORD_YTD [ In use=735 ] [ diff 1 value 0.0 ]
double VALUE_SOLD_MTH [ In use=735 ] [ diff 1 value 0.0 ]
double VALUE_SOLD_YTD [ In use=735 ] [ diff 1 value 0.0 ]
short VAT_CODE [ In use=735 ] [ diff 1 value 1 ]
double WEIGHT [ In use=735 ] [ diff 8 ]
double WIDTH [ In use=735 ] [ diff 1 value 0.0 ]
String WIP_NOMINAL_CODE [ In use=0 ] [ diff 0 ]

166
### Appendix C Information Files results from the Case Study

#### CLASS BOM_STRUCTURE_FILE  num. records=1023
- double ALT_NUMBER_OFF [In use=1023] [diff 1 value 0.0]
- String AMENDED_BOM [In use=0] [diff 0]
- short BOM_SEQ [In use=1023] [diff 11]
- String BY_PRODUCT [In use=0] [diff 0]
- String COMPONENT_PART [In use=1023] [diff 509]
- double CONVERSION [In use=1023] [diff 1 value 0.0]
- java.util.Date DATE_AMD [In use=1023] [diff 113]
- java.util.Date END_DATE [In use=1023] [diff 1 value 1999-11-30 00:00:00.0]
- String ENG_CHANGE_REF [In use=0] [diff 0]
- String FREE_ISSUE [In use=0] [diff 0]
- double NUMBER_OFF [In use=1023] [diff 310]
- short OPERATION [In use=1023] [diff 1 value 0]
- String PARENT_PART [In use=1023] [diff 152]
- short QTY_PER_PACK_CODE [In use=1023] [diff 1 value 0]
- String REFERENCE_TEXT [In use=21] [diff 18]
- double STAGE_LEAD_OFFSET [In use=1023] [diff 1 value 0.0]
- java.util.Date START_DATE [In use=1023] [diff 1 value 1999-11-30 00:00:00.0]

#### Stock_Control_File Component_part
#### Stock_Control_File Parent_part

#### CLASS BOM_LEVELS_FILE  num. records=9491
- String ACCOUNT_NUMBER [In use=8229] [diff 42]
- double ALT_CONVERSION_1 [In use=9491] [diff 1 value 0.0]
- double ALT_CONVERSION_2 [In use=9491] [diff 1 value 0.0]
- short ALT_QTY_PER_PACK_1 [In use=9491] [diff 1 value 0]
- short ALT_QTY_PER_PACK_2 [In use=9491] [diff 1 value 0]
- double ALT_QTY_RECEIVED_1 [In use=9491] [diff 1 value 0.0]
- double ALT_QTY_RECEIVED_2 [In use=9491] [diff 1 value 0.0]
- short AUXILIARY_STATUS [In use=9491] [diff 1 value 0]
- String BATCH_DESCRIPTION [In use=0] [diff 0]
- String BATCH_NUMBER [In use=9491] [diff 9491]
- String BATCH_SOURCE [In use=9491] [diff 3]
- short BATCH_STATUS [In use=9491] [diff 3]
- double BUYING_RATE [In use=9491] [diff 44]
- String CONTRACT_NUMBER [In use=0] [diff 0]
- String COSTED_IND [In use=0] [diff 0]
- double CST_LABOUR_VALUE [In use=9491] [diff 1 value 0.0]
- double CST_MATERIAL_VALUE [In use=9491] [diff 44]
- double CST_MATERIAL_OHEAD [In use=9491] [diff 1 value 0.0]
- double CST_OTHER_COSTS [In use=9491] [diff 1 value 0.0]
- double CST_OVERHEAD_VALUE [In use=9491] [diff 1 value 0.0]
- double CUST_QTY_RETURNED [In use=9491] [diff 2 values=[0.0],[1320.0]]
- short CUSTOMS_SEQ [In use=9491] [diff 1 value 0]
- java.util.Date DATE_EFFECTIVE [In use=9491] [diff 429]
- java.util.Date DATE_ENTERED [In use=9491] [diff 429]
- java.util.Date DATE_INVOICED [In use=9491] [diff 1 value 1999-11-30 00:00:00.0]
- String DEPOT_CODE [In use=9491] [diff 2 values=[1],[2]]
- short DROP_SEQUENCE_NO [In use=9491] [diff 1 value 0]
- double EXP_LABOUR_VALUE [In use=9491] [diff 1 value 0.0]
- double EXP_MATERIAL_VALUE [In use=9491] [diff 44]
- double EXP_MATERIAL_OHEAD [In use=9491] [diff 1 value 0.0]
- double EXP_OTHER_COSTS [In use=9491] [diff 1 value 0.0]
- double EXP_OVERHEAD_VALUE [In use=9491] [diff 1 value 0.0]
- double INSP_QTY_ALLOCATED [In use=9491] [diff 1 value 0.0]
- double INV_EXCH_RATE [In use=9491] [diff 1 value 0.0]
- String INVOICE_NUMBER [In use=0] [diff 0]
- String ISSUE_NUMBER [In use=0] [diff 0]
- String ITEM_NUMBER [In use=9491] [diff 665]
- String LOCATION [In use=9485] [diff 937]
<table>
<thead>
<tr>
<th>Field</th>
<th>In use</th>
<th>Diff</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_OF_CARTONS</td>
<td>9491</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ORDER_NUMBER</td>
<td>8962</td>
<td>1527</td>
<td></td>
</tr>
<tr>
<td>ORG_SUPP_BATCH</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>PHASE_NUMBER</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>PRICE_AGreed</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC_DOC_NUMBER</td>
<td>8342</td>
<td>3212</td>
<td></td>
</tr>
<tr>
<td>QC_IND</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC_STATUS</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QTY_ALLOCATED</td>
<td>9491</td>
<td>2</td>
<td>values=[-70.0], [0.0]</td>
</tr>
<tr>
<td>QTY_IN_STOCK</td>
<td>9491</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>QTY_INVOICED</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>QTY_ISS_AT_EXP_COST</td>
<td>9491</td>
<td>1698</td>
<td></td>
</tr>
<tr>
<td>QTY_ISSUED</td>
<td>9491</td>
<td>1449</td>
<td></td>
</tr>
<tr>
<td>QTY_RECEIVED</td>
<td>9491</td>
<td>1358</td>
<td></td>
</tr>
<tr>
<td>QTY_REWORKED</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>QTY_SCRAPPED</td>
<td>9491</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>QTY_TRANSFERRED</td>
<td>9491</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>QTY_ALLOCATED</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>RTG_ISSUE_NUMBER</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SUPP_DESP_NOTE</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SUPP_QTY_RETURNED</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>SUPPLIER_BATCH_NO</td>
<td>8553</td>
<td>3436</td>
<td></td>
</tr>
<tr>
<td>SUPPLIER_BATCH_NO</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>QC_IND</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC_STATUS</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ACCOUNT_NUMBER</td>
<td>36</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ANALYSIS</td>
<td>127</td>
<td>2</td>
<td>values=[PUR], [WIP]</td>
</tr>
<tr>
<td>EFFECTIVE_DATE</td>
<td>124</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EFFECTIVE_TIME</td>
<td>124</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>EXCLUDEFROM_OTL_STK</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FIRM_IND</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>INSPECTION_IND</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ISSUE_NUMBER</td>
<td>56</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ITEM_NUMBER</td>
<td>124</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MPS_TYPE</td>
<td>124</td>
<td>1</td>
<td>value M</td>
</tr>
<tr>
<td>MPS_VERSION</td>
<td>124</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>ROUTING_VERSION</td>
<td>124</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>START_DATE</td>
<td>124</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>SUPP_QTY_RETURNED</td>
<td>9491</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>DEPOT_CODE</td>
<td>102</td>
<td>1</td>
<td>value 1</td>
</tr>
<tr>
<td>DROP_SEQUENCE_NO</td>
<td>127</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>EFFECTIVE_DATE</td>
<td>127</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>EFFECTIVE_TIME</td>
<td>127</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>EXCLUDEFROM_OTL_STK</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FIRM_IND</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>INSPECTION_IND</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ISSUE_NUMBER</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ITEM_NUMBER</td>
<td>127</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>NUMBER_OFF</td>
<td>127</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>OPERATION</td>
<td>127</td>
<td>1</td>
<td>value 0</td>
</tr>
<tr>
<td>ORDER_FILL</td>
<td>21</td>
<td>2</td>
<td>values=[7], [8]</td>
</tr>
<tr>
<td>ORDER_NUMBER</td>
<td>126</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>ORDER_TYPE</td>
<td>127</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Information Files results from the Case Study

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>In Use</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>short PHANTOM_SEQ</td>
<td>[In use=127] [diff 1 value 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short PHASE_OR_SEQ</td>
<td>[In use=127] [diff 1 value 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String PROGRAM_NAME</td>
<td>[In use=0] [diff 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String PROJECT_NUMBER</td>
<td>[In use=0] [diff 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double QTY_ISS_OR_REC</td>
<td>[In use=127] [diff 31]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double QUANTITY</td>
<td>[In use=127] [diff 115]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String REFERENCE_NUMBER</td>
<td>[In use=127] [diff 30]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String RTG_ISSUE_NUMBER</td>
<td>[In use=0] [diff 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short SCON_SEQUENCE_NO</td>
<td>[In use=127] [diff 1 value 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short SEQUENCE_NO</td>
<td>[In use=127] [diff 19]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double SHORTAGE_QTY</td>
<td>[In use=127] [diff 29]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double STAGE_LEAD_OFFSET</td>
<td>[In use=127] [diff 1 value 0.0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double STAGE_QTY_ISSUED</td>
<td>[In use=127] [diff 1 value 0.0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short SUPERSEDED_ISSUED</td>
<td>[In use=127] [diff 1 value 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short TRANSACTION_TYPE</td>
<td>[In use=127] [diff 5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.util.Date TRANSACTION_DATE</td>
<td>[In use=127] [diff 11]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short TRANSACTION_TIME</td>
<td>[In use=127] [diff 27]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double UNIT_PRICE</td>
<td>[In use=127] [diff 5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short USER_NUMBER</td>
<td>[In use=127] [diff 4]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stock_Control_File item_number

CLASS SOR_SALES_ORDERS num. records=3

String ACCOUNT_NUMBER [In use=3] [diff 2 values=[11],[13]]
String ACCOUNT_NAME [In use=3] [diff 2 values=[##########],[&&&&&&&&&]]
String ANALYSIS_CODE_9 [In use=0] [diff 0]
String ANALYSIS_CODE_10 [In use=0] [diff 0]
String ANALYSIS_CODE_11 [In use=0] [diff 0]
String ANALYSIS_CODE_12 [In use=0] [diff 0]
String ANALYSIS_CODE_13 [In use=0] [diff 0]
String ANALYSIS_CODE_14 [In use=0] [diff 0]
String ANALYSIS_CODE_15 [In use=0] [diff 0]
double ANALYSIS_AMOUN_1 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_2 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_3 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_4 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_5 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_6 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_7 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_8 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_9 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_10 [In use=3] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_11 [In use=3] [diff 1 value 0.0]
String ANALYSIS_TYPE_1 [In use=3] [diff 1 value C]
short AUXILIARY_STATUS [In use=3] [diff 2 values=[0],[6]]
String CONTACT [In use=0] [diff 0]
String CREDIT_ORDER_NUMBER [In use=0] [diff 0]
double CURRENT_ORDER_VALUE [In use=3] [diff 1 value 0.0]
short DEL_ADDRESS_CODE [In use=3] [diff 1 value 0]
String DEL_ADDRESS_NAME [In use=3] [diff 2 values=[xxxxxxxx],[xxxxxxxxxxx]]
String DEL_ADDRESS_LINE_1 [In use=0] [diff 0]
String DEL_ADDRESS_LINE_2 [In use=0] [diff 0]
String DEL_ADDRESS_LIN_3 [In use=0] [diff 0]
String DEL_ADDRESS_LINE_4 [In use=0] [diff 0]
String DELIVERY_NOTE_NUMBER [In use=3] [diff 3]
double DISCT_SURCENT [In use=3] [diff 1 value 0.0]
double EXCHANGE_RATE [In use=3] [diff 1 value 0.0]
double FC_CURRENT_ORD_VALUE [In use=3] [diff 1 value 0.0]
String FREQUENCY_DESC_1 [In use=0] [diff 0]
String FREQUENCY_DESC_2 [In use=0] [diff 0]

short FREQUENCY_CD_1 [In use=3] [diff 1 value 0]
short FREQUENCY_CD_2 [In use=3] [diff 1 value 0]

double DISCOUNT_AMT [In use=3] [diff 1 value 0.0]
double EXCHANGE_RATE [In use=3] [diff 1 value 0.0]

double FULL_ORDER_VALUE [In use=3] [diff 1 value 0.0]

java.util.Date INVOICE_DATE [In use=3] [diff 2 values= [1997-01-21 00:00:00.0], [1999-11-30 00:00:00.0]]

String PROCESS_IND [In use=0] [diff 0]
String PROCESSED_BY_MRP [In use=394] [diff 2 values= [N], [Y]]
int SALES_HEADER_REC [In use=0] [diff 0]
double SETTLEMENT_DISC_PCT [In use=3] [diff 1 value 0.0]
short SETTLEMENT_DISC_DAYS [In use=3] [diff 1 value 0]

String ALLOW_MRP_RESCHED [In use=394] [diff 2 values= [N], [Y]]
APPENDIX C

Information Files results from the Case Study

String AUTO_RELEASE_IND [In use=0] [diff 0]
String BOM_IND [In use=23] [diff 1 value N]
java.util.Date CLOSE_DATE [In use=431] [diff 206]
String COMPONENT_DEPOT_CODE [In use=398] [diff 1 value 1]
int COMPONENT_COUNT [In use=431] [diff 11]
short CUSTOMS_SEQ [In use=431] [diff 1 value 0]
String DOCUMENT_NUMBER [In use=28] [diff 28]
java.util.Date DUE_DATE [In use=431] [diff 197]
String EXCLUDE_FROM_POTL_STK [In use=394] [diff 1 value N]
String EXCLUDE_FROM_MRP [In use=394] [diff 1 value N]
String ISSUE_NUMBER [In use=0] [diff 0]
String ITEM_NUMBER [In use=430] [diff 146]
short JOB_CARDS_PRINTED [In use=431] [diff 1 value 0]
short KIT_LIST_PRINTED [In use=431] [diff 2 values= [0],[1]]
double LABOUR_VALUE [In use=431] [diff 1 value 0.0]
java.util.Date LAST_ISSUE_DATE [In use=431] [diff 225]
short LAST_STAGE_NO [In use=431] [diff 1 value 0]
double MATERIAL_VALUE [In use=431] [diff 1 value 0.0]
double MATERIAL_OHEAD [In use=431] [diff 1 value 0.0]
String OPS_BOOKED_IND [In use=0] [diff 0]
String ORDER_DEPOT_CODE [In use=430] [diff 1 value 1]
String ORDER_DESCRIPTION [In use=86] [diff 44]
String ORDER_NUMBER [In use=431] [diff 431]
short ORDER_STATUS [In use=431] [diff 8]
int ORDER_SUFFIX [In use=431] [diff 13]
short ORDER_TYPE [In use=431] [diff 2 values= [2],[4]]
double OTHER_COSTS [In use=431] [diff 1 value 0.0]
double OVERHEAD_VALUE [In use=431] [diff 1 value 0.0]
String OWNER [In use=0] [diff 0]
String PARENT_ORDER_NO [In use=0] [diff 0]
short PARTIAL_COMPONENT_ISS [In use=431] [diff 1 value 0]
short PHASE_NUMBER [In use=431] [diff 1 value 0]
String QC_IND [In use=0] [diff 0]
double QTY_COMPLETE [In use=431] [diff 306]
double QTY_ISSUED [In use=431] [diff 295]
double QTY_ORDERED [In use=431] [diff 267]
double QTY_REWORKED [In use=431] [diff 1 value 0.0]
double QTY_SCRAPPED [In use=431] [diff 1 value 0.0]
double QTY_THIS_STAGE [In use=431] [diff 1 value 0.0]
java.util.Date RELEASE_DATE [In use=431] [diff 219]
String ROUTING_IND [In use=0] [diff 0]
short ROUTING_PRINTED [In use=431] [diff 1 value 0]
String RTG_ISSUE_NUMBER [In use=0] [diff 0]
double SCRAP_ALLOWANCE [In use=431] [diff 68]
short SEQUENCE_NO [In use=431] [diff 1 value 0]
String SHORTAGE_IND [In use=4] [diff 1 value Y]
short STAGE_ISSUED [In use=431] [diff 1 value 0]
java.util.Date START_DATE [In use=431] [diff 190]
String TEMP_COST_IND [In use=0] [diff 0]
String TSSI_HIST_SHEET_IND [In use=0] [diff 0]
short VERSION [In use=431] [diff 1 value 0]
String WALLCHART_IND [In use=0] [diff 0]
String WOP_BOM_AMENDED [In use=4] [diff 1 value Y]
String WOP_ROUT_AMENDED [In use=0] [diff 0]
Stock_Control_Item_number

CLASS PURCHASE_ORDERS num. records=23
String ACCOUNT_NUMBER [In use=23] [diff 7]
String ACCOUNT_NAME [In use=23] [diff 7]

171
Appendix C

Information Files results from the Case Study

short AUXILIARY_STATUS [In use=23] [diff 2 values= [0],[3]]
String BUYER_CODE [In use=0] [diff 0]
String CONTACT [In use=14] [diff 4]
double CURRENT_ORDER_VALUE [In use=23] [diff 1 value 0.0]
short DEL_ADDRESS_CODE [In use=23] [diff 2 value 0]
double EXCHANGE_RATE [In use=23] [diff 1 value 0.0]
double FC_ORDER_VALUE [In use=23] [diff 1 value 0.0]
String FREE_FORMAT_DESC_1 [In use=0] [diff 0]
String FREE_FORMAT_DESC_2 [In use=0] [diff 0]
short FREE_FORMAT_VAT_CD_1 [In use=23] [diff 1 value 0]
short FREE_FORMAT_VAT_CD_2 [In use=23] [diff 1 value 0]
double FREE_FORMAT_AMOUNT_1 [In use=23] [diff 1 value 0.0]
double FREE_FORMAT_AMOUNT_2 [In use=23] [diff 1 value 0.0]
String FREE_FORMAT_CC_1 [In use=0] [diff 0]
String FREE_FORMAT_CC_2 [In use=0] [diff 0]
String FREE_FORMAT_DEPT_1 [In use=0] [diff 0]
String FREE_FORMAT_DEPT_2 [In use=0] [diff 0]
String FREE_FORMAT_ACCOUNT_1 [In use=0] [diff 0]
String FREE_FORMAT_ACCOUNT_2 [In use=0] [diff 0]
String HISTORY_ANALYSIS [In use=0] [diff 0]
String LAST_DELIVERY_NOTE_NO [In use=0] [diff 0]
String LAST_INVOICE_NO [In use=0] [diff 0]
java.util.Date LAST_INVOICE_DATE [In use=23] [diff 1 value 1999-11-30 00:00:00.0]
short LAST_SEQUENCE_NO [In use=23] [diff 11]
short NO_OF_SCHED_ITEMS [In use=23] [diff 1 value 0]
double NOM_EXCH_RATE [In use=23] [diff 1 value 0.0]
java.util.Date ORDER_DATE [In use=23] [diff 9]
java.util.Date ORDER_DUE_DATE [In use=23] [diff 9]
String ORDER_NUMBER [In use=23] [diff 23]
String ORDER_PRIORITY_LEVEL [In use=0] [diff 0]
short ORDER_STATUS [In use=23] [diff 2 values= [3],[10]]
short ORDER_TYPE [In use=23] [diff 1 value 1]
double ORDER_VALUE [In use=23] [diff 1 value 0.0]
double OVERALL_DISCT_PCT [In use=23] [diff 1 value 0.0]
int PURCHASE_HEADER_REC [In use=23] [diff 7]
String QC_IND [In use=0] [diff 0]
String REPEAT_ORDER [In use=23] [diff 1 value M]
String RETURN_NOTE_EXPECTED [In use=0] [diff 0]
String SHORT_NAME [In use=23] [diff 7]
String SPECIAL_INS_1 [In use=0] [diff 0]
String SPECIAL_INS_2 [In use=0] [diff 0]
short SSD_ENTRY_REQUIRED [In use=23] [diff 1 value 0]
String SUPP_ADDRESS_NAME [In use=0] [diff 0]
String SUPP_ADDRESS_LINE_1 [In use=0] [diff 0]
String SUPP_ADDRESS_LINE_2 [In use=0] [diff 0]
String SUPP_ADDRESS_LINE_3 [In use=0] [diff 0]
String SUPP_ADDRESS_LINE_4 [In use=0] [diff 0]
String THEIR_REFERENCE [In use=0] [diff 0]
String VAT_INCLUSIVE_PRICES [In use=0] [diff 0]
int WAITING_DELIVERY [In use=23] [diff 1 value 0]
int WAITING_INVOICE [In use=23] [diff 1 value 0]
C.1. EXAMPLE ASSEMBLE-TO-ORDER COMPANY

Empty tables:

Tables in Use:
CLASS STOCK_CONTROL_FILE num. records=11452
  double ABNORMAL_DEMAND_FAC [ In use=11452 ] [ diff 1 value 0.0]
  String ALT_SUPPLIER   [ In use=516 ] [ diff 88]
  double ASSEMBLY_LEAD_TIME [ In use=11452 ] [ diff 23]
  double AVERAGE_PRICE   [ In use=11452 ] [ diff 4978]
  double AVERAGE_WEIGHT  [ In use=11452 ] [ diff 1 value 0.0]
  String BAR_CODE        [ In use=0 ] [ diff 0]
  short BAS_QTY_PER_PACK_CODE [ In use=11452 ] [ diff 13]
  String BATCH_TRACEABILITY  [ In use=11447 ] [ diff 1 value N]
  String BESPOKE_IND     [ In use=0 ] [ diff 0]
  int BESPOKE_NUM        [ In use=11452 ] [ diff 1 value 0]
  String BIN_NUMBER_1    [ In use=7424 ] [ diff 1171]
  String BIN_NUMBER_2    [ In use=305 ] [ diff 138]
  String BOM_EXPLOSION   [ In use=0 ] [ diff 0]
  String BOM_IGNORES_IND [ In use=0 ] [ diff 0]
  String BULK_ISSUE_IND  [ In use=2 ] [ diff 1 value N]
  String BUYER_CODE      [ In use=0 ] [ diff 0]
  String CATEGORY        [ In use=8722 ] [ diff 3]
  String CE_COMMODITY_CODE [ In use=31 ] [ diff 5]
  String CHANGE_NOTE_NUMBER  [ In use=0 ] [ diff 0]
  String CHANGE_PENDING  [ In use=0 ] [ diff 0]
  String CLASSIFICATION_1 [ In use=0 ] [ diff 0]
  String CLASSIFICATION_2 [ In use=0 ] [ diff 0]
  String COMMODITY_CODE  [ In use=0 ] [ diff 0]
  String CONFIG_IND      [ In use=0 ] [ diff 0]
  double CONVERSION_1    [ In use=11452 ] [ diff 45]
  double CONVERSION_2    [ In use=11452 ] [ diff 2 values= [0.0], [1.0]]
  String COS_NOMINAL_CODE [ In use=0 ] [ diff 0]
  String COS_VAR_NOMINAL_CODE [ In use=0 ] [ diff 0]
  double COST_VALUE_SOLD_MTH [ In use=11452 ] [ diff 10]
  double COST_VALUE_SOLD_YTD [ In use=11452 ] [ diff 10]
  String COSTING_ONLY_IND [ In use=0 ] [ diff 0]
  java.util.Date DATE_LAST_ISSUE [ In use=11452 ] [ diff 486]
  java.util.Date DATE_LAST_ORDER [ In use=11452 ] [ diff 988]
  java.util.Date DATE_LAST_STOCK_TAKE [ In use=11452 ] [ diff 1 value 1999-11-30 00:00:00.0]
  java.util.Date DATE_NEXT_ORDER_DUE [ In use=11452 ] [ diff 1 value 1999-11-30 00:00:00.0]
  java.util.Date DATE_OF_LAST_SALE [ In use=11452 ] [ diff 210]
  String DRAWING_NUMBER   [ In use=0 ] [ diff 0]
  String EBQ_IND          [ In use=5840 ] [ diff 4]
  java.util.Date END_DATE  [ In use=11452 ] [ diff 1 value 1999-11-30 00:00:00.0]
  String ENG_CHANGE_REF   [ In use=0 ] [ diff 0]
  String ENG_DEPT_USE_1   [ In use=0 ] [ diff 0]
  String ENG_DEPT_USE_2   [ In use=990 ] [ diff 1 value Y]
  double ENG_PRICE       [ In use=11452 ] [ diff 1 value 0.0]
  java.util.Date EXPIRY_DATE [ In use=11452 ] [ diff 1 value 1999-11-30 00:00:00.0]
  double EXPTED_BUYING_PRICE [ In use=11452 ] [ diff 2387]
  String FACTOR_PRICES_IND [ In use=11447 ] [ diff 2 values= [N], [Y]]
  String FAMILYBILL_PART_NO [ In use=0 ] [ diff 0]
  double FORWARD_ORDER_QTY [ In use=11452 ] [ diff 33]
  String FREE_ISSUE_IND   [ In use=0 ] [ diff 0]
  double HEIGHT          [ In use=11452 ] [ diff 1 value 0.0]
  double INSP_QTYALLOCATED [ In use=11452 ] [ diff 1 value 0.0]
  double INSPECTION_QUANTITY [ In use=11452 ] [ diff 16]
  java.util.Date ISSUE_DATE [ In use=11452 ] [ diff 1 value 1999-11-30 00:00:00.0]
### Appendix C Information Files results from the Case Study

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>In Use</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>double QTY_SOLD_YTD</td>
<td>[In use=11452]</td>
<td>diff 8</td>
<td></td>
</tr>
<tr>
<td>String RELIFE_CODE</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>double RELIFE_COST</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double REORDER_LEVEL</td>
<td>[In use=11452]</td>
<td>diff 91</td>
<td></td>
</tr>
<tr>
<td>short REORDER_POLICY</td>
<td>[In use=11452]</td>
<td>diff 4</td>
<td></td>
</tr>
<tr>
<td>double REORDER_QTY</td>
<td>[In use=11452]</td>
<td>diff 90</td>
<td></td>
</tr>
<tr>
<td>String REVISION_NUMBER</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>double SAFETY_STOCK</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>short SALES_CURRENCY</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0</td>
</tr>
<tr>
<td>String SCRAP_IND</td>
<td>[In use=11396]</td>
<td>diff 1</td>
<td>value 0</td>
</tr>
<tr>
<td>String SCRAP_NOMINAL_CODE</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>double SCRAP_PERCENT</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double SELLING_PRICE_1</td>
<td>[In use=11452]</td>
<td>diff 1301</td>
<td></td>
</tr>
<tr>
<td>double SELLING_PRICE_2</td>
<td>[In use=11452]</td>
<td>diff 4</td>
<td></td>
</tr>
<tr>
<td>double SELLING_PRICE_3</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double SELLING_PRICE_4</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double SELLING_PRICE_5</td>
<td>[In use=11452]</td>
<td>diff 182</td>
<td></td>
</tr>
<tr>
<td>String SERIAL_NO_IND</td>
<td>[In use=3528]</td>
<td>diff 1</td>
<td>value N</td>
</tr>
<tr>
<td>String SERIAL_NO_GEN</td>
<td>[In use=3528]</td>
<td>diff 1</td>
<td>value N</td>
</tr>
<tr>
<td>int SHELF_LIFE</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0</td>
</tr>
<tr>
<td>double STANDARD_COST</td>
<td>[In use=11452]</td>
<td>diff 3272</td>
<td></td>
</tr>
<tr>
<td>java.util.Date START_DATE</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 1999-11-30 00:00:00.0</td>
</tr>
<tr>
<td>double STK_NO_OF_BASE_UNITS</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 1.0</td>
</tr>
<tr>
<td>short STK_QTY_PER_PACK_CODE</td>
<td>[In use=11452]</td>
<td>diff 13</td>
<td></td>
</tr>
<tr>
<td>String STOCK_MAY_GO_NEGATIVE</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>String SUB_CON_NOMINAL_CODE</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>String SUBCON_BULK_ISSUE</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>String SUMM_ISSUE_IND</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>String SUPERSEDED_ITEM</td>
<td>[In use=105]</td>
<td>diff 97</td>
<td></td>
</tr>
<tr>
<td>String SUPERSEDED_BY_ITEM</td>
<td>[In use=191]</td>
<td>diff 174</td>
<td></td>
</tr>
<tr>
<td>double SYS_QTY_ALLOCATED</td>
<td>[In use=11452]</td>
<td>diff 312</td>
<td></td>
</tr>
<tr>
<td>double SYSTEM_LEAD_TIME</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double TBA_QUANTITY</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double THIS_MONTHS_STOCK</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double VAL_ORD_THIS_MONTH</td>
<td>[In use=11452]</td>
<td>diff 18</td>
<td></td>
</tr>
<tr>
<td>double VAL_ORD_YTD</td>
<td>[In use=11452]</td>
<td>diff 18</td>
<td></td>
</tr>
<tr>
<td>double VALUE_SOLD_MTH</td>
<td>[In use=11452]</td>
<td>diff 9</td>
<td></td>
</tr>
<tr>
<td>double VALUE_SOLD_YTD</td>
<td>[In use=11452]</td>
<td>diff 9</td>
<td></td>
</tr>
<tr>
<td>short VAT_CODE</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 1</td>
</tr>
<tr>
<td>double WEIGHT</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>double WIDTH</td>
<td>[In use=11452]</td>
<td>diff 1</td>
<td>value 0.0</td>
</tr>
<tr>
<td>String WIP_NOMINAL_CODE</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
</tbody>
</table>

#### CLASS BOM_STRUCTURE_FILE
num. records=25631

#### CLASS BOM_LEVELS_FILE
num. records=20076

#### CLASS BOM_ISSUED_SERIALS extends BOM_LEVELS_FILE
num. records=390

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>In Use</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>String ACCOUNT_NUMBER</td>
<td>[In use=316]</td>
<td>diff 8</td>
<td></td>
</tr>
<tr>
<td>String AUX_ORDER_NUMBER</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>short AUX_SEQ_NO</td>
<td>[In use=390]</td>
<td>diff 1</td>
<td>value 0</td>
</tr>
<tr>
<td>short AUXILIARY_STATUS</td>
<td>[In use=390]</td>
<td>diff 1</td>
<td>value 0</td>
</tr>
<tr>
<td>String BATCH_NUMBER</td>
<td>[In use=390]</td>
<td>diff 59</td>
<td></td>
</tr>
<tr>
<td>String CONTRACT_NUMBER</td>
<td>[In use=0]</td>
<td>diff 0</td>
<td></td>
</tr>
<tr>
<td>java.util.Date DATE_ISSUED</td>
<td>[In use=390]</td>
<td>diff 47</td>
<td></td>
</tr>
<tr>
<td>String DEPOT_CODE</td>
<td>[In use=390]</td>
<td>diff 3</td>
<td></td>
</tr>
<tr>
<td>String DESPATCH_NOTE_NUMBER</td>
<td>[In use=96]</td>
<td>diff 11</td>
<td></td>
</tr>
<tr>
<td>short DROP_SEQUENCE_NO</td>
<td>[In use=390]</td>
<td>diff 1</td>
<td>value 0</td>
</tr>
<tr>
<td>String INVOICE_NUMBER</td>
<td>[In use=316]</td>
<td>diff 36</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix C

Information Files results from the Case Study

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Use</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Date ISSUE_DATE</td>
<td>[In use=390] [diff 2 values= 1954-07-07 00:00:00.00,1999-11-30 00:00:00.0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String ISSUE_NUMBER</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>String ITEM_NUMBER</td>
<td></td>
<td>[In use=390] [diff 16]</td>
<td></td>
</tr>
<tr>
<td>String ORDER_NUMBER</td>
<td></td>
<td>[In use=390] [diff 25]</td>
<td></td>
</tr>
<tr>
<td>short PHASE_NUMBER</td>
<td></td>
<td>[In use=390] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>short SEQUENCE_NO</td>
<td></td>
<td>[In use=390] [diff 27]</td>
<td></td>
</tr>
<tr>
<td>String SERIAL_NUMBER_1</td>
<td></td>
<td>[In use=390] [diff 345]</td>
<td></td>
</tr>
<tr>
<td>String SERIAL_NUMBER_2</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>short STATUS</td>
<td></td>
<td>[In use=390] [diff 2 values= 10,99]</td>
<td></td>
</tr>
</tbody>
</table>

**Stock Control File Item_number**

**CLASS BOM_SERIAL_NUMBERS extends BOM_LEVELS_FILE**

num. records=322

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Use</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>String ACCOUNT_NUMBER</td>
<td></td>
<td>[In use=314] [diff 8]</td>
<td></td>
</tr>
<tr>
<td>String AUTOGEN_IND</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>short AUXILIARY_STATUS</td>
<td></td>
<td>[In use=322] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>String BATCH_NUMBER</td>
<td></td>
<td>[In use=322] [diff 40]</td>
<td></td>
</tr>
<tr>
<td>String CONTRACT_NUMBER</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>java.util.Date DATE_ISSUED</td>
<td></td>
<td>[In use=322] [diff 31]</td>
<td></td>
</tr>
<tr>
<td>java.util.Date DATE_RETURNED</td>
<td></td>
<td>[In use=322] [diff 1 value 1999-11-30 00:00:00.00]</td>
<td></td>
</tr>
<tr>
<td>String DEPOT_CODE</td>
<td></td>
<td>[In use=322] [diff 1 value M]</td>
<td></td>
</tr>
<tr>
<td>String DESPATCH_NOTE_NUMBER</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>short DROP SEQUENCE_NO</td>
<td></td>
<td>[In use=322] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>String INVOICE_NUMBER</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>java.util.Date ISSUE_DATE</td>
<td></td>
<td>[In use=322] [diff 2 values= 1954-07-07 00:00:00.00,1999-11-30 00:00:00.0]</td>
<td></td>
</tr>
<tr>
<td>String ISSUE_NUMBER</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>String ITEM_NUMBER</td>
<td></td>
<td>[In use=322] [diff 12]</td>
<td></td>
</tr>
<tr>
<td>String OLD_AUX_SAL_ORDER</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>short OLD_AUX_SEQ_NO</td>
<td></td>
<td>[In use=322] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>String OLD_AUX_SAL_DESP_NOTE</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>String OLD_AUX_SAL_INV</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>String OLD_AUX_SAL_DESC</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>String ORDER_NUMBER</td>
<td></td>
<td>[In use=322] [diff 15]</td>
<td></td>
</tr>
<tr>
<td>short PHASE_NUMBER</td>
<td></td>
<td>[In use=322] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>String PURCH_SALES_IND</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>short SEQUENCE_NO</td>
<td></td>
<td>[In use=322] [diff 26]</td>
<td></td>
</tr>
<tr>
<td>String SERIAL_NUMBER_1</td>
<td></td>
<td>[In use=322] [diff 314]</td>
<td></td>
</tr>
<tr>
<td>String SERIAL_NUMBER_2</td>
<td></td>
<td>[In use=0] [diff 0]</td>
<td></td>
</tr>
<tr>
<td>short STATUS</td>
<td></td>
<td>[In use=322] [diff 1 value 10]</td>
<td></td>
</tr>
<tr>
<td>short TEMP_USER_NO</td>
<td></td>
<td>[In use=322] [diff 1 value 0]</td>
<td></td>
</tr>
</tbody>
</table>

**Stock Control File Item_number**

**CLASS MPS_SCHEDULE_FILE**

num. records=38

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Use</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Date END_DATE</td>
<td>[In use=38] [diff 1 value 1999-11-30 00:00:00.00]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.util.Date ENTRY_DATE</td>
<td>[In use=38] [diff 14]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String ISSUE_NUMBER</td>
<td></td>
<td>[In use=33] [diff 2 values= 01,11]</td>
<td></td>
</tr>
<tr>
<td>String ITEM_NUMBER</td>
<td></td>
<td>[In use=38] [diff 38]</td>
<td></td>
</tr>
<tr>
<td>String MPS_TYPE</td>
<td></td>
<td>[In use=38] [diff 1 value M]</td>
<td></td>
</tr>
<tr>
<td>short MPS_VERSION</td>
<td></td>
<td>[In use=38] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>String OWNER</td>
<td></td>
<td>[In use=37] [diff 3]</td>
<td></td>
</tr>
<tr>
<td>short ROUTING_VERSION</td>
<td></td>
<td>[In use=38] [diff 1 value 0]</td>
<td></td>
</tr>
<tr>
<td>java.util.Date START_DATE</td>
<td></td>
<td>[In use=38] [diff 1 value 1999-11-30 00:00:00.00]</td>
<td></td>
</tr>
</tbody>
</table>

**Stock Control File Item_number**

**CLASS BOM_ALLOCS_ORDERS**

num. records=127

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Use</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>String ACCOUNT_NUMBER</td>
<td></td>
<td>[In use=36] [diff 10]</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS</td>
<td></td>
<td>[In use=127] [diff 2 values= 15]</td>
<td></td>
</tr>
</tbody>
</table>

176
<table>
<thead>
<tr>
<th>String</th>
<th>In use</th>
<th>diff</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPOT_CODE</td>
<td>102</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>short DROP_SEQUENCE_NO</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>java.util.Date EFFECTIVE_DATE</td>
<td>127</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>short EFFECTIVE_TIME</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>String EXCLUDE_FROM_POTL_STK</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String FIRM_IND</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String INSPECTION_IND</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String ISSUE_NUMBER</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String ITEM_NUMBER</td>
<td>127</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>double NUMBER_OFF</td>
<td>127</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>short OPERATION</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>String ORDER_FILL</td>
<td>21</td>
<td>2</td>
<td>[7],[8]</td>
</tr>
<tr>
<td>String ORDER_NUMBER</td>
<td>126</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>short ORDER_TYPE</td>
<td>127</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>short PHANTOM_SEQ</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>short PHASE_OR_SEQ</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>String PROGRAM_NAME</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double QTY_ISS_OR_REC</td>
<td>127</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>double QUANTITY</td>
<td>127</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>String REFERENCE_NUMBER</td>
<td>126</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>String RTG_ISSUE_NUMBER</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short SCON_SEQUENCE_NO</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>short SEQUENCE_NO</td>
<td>127</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>double SHORTAGE_QTY</td>
<td>127</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>double STAGE_LEAD_OFFSET</td>
<td>127</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>double STAGE_QTY_ISSUED</td>
<td>127</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>short SUPERSEDED_ISSUED</td>
<td>127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>short TRANSACTION_TYPE</td>
<td>127</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>java.util.Date TRANSACTION_DATE</td>
<td>127</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>short TRANSACTION_TIME</td>
<td>127</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>double UNIT_PRICE</td>
<td>127</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>short USER_NUMBER</td>
<td>127</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**CLASS SOR_SALES_ORDERS**

num. records=177

<table>
<thead>
<tr>
<th>String</th>
<th>In use</th>
<th>diff</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOCATE_ON_ORD_ENTRY</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_CC_1</td>
<td>176</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_CC_3</td>
<td>21</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_CC_14</td>
<td>48</td>
<td>2</td>
<td>[EXT],[ING]</td>
</tr>
<tr>
<td>String ANALYSIS_CC_15</td>
<td>48</td>
<td>2</td>
<td>[EXT],[ING]</td>
</tr>
<tr>
<td>String ANALYSIS_DEPT_1</td>
<td>176</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_DEPT_2</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_DEPT_3</td>
<td>21</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_CODE_1</td>
<td>176</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_CODE_3</td>
<td>21</td>
<td>2</td>
<td>[100],[702]</td>
</tr>
<tr>
<td>String ANALYSIS_CODE_6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>String ANALYSIS_CODE_14</td>
<td>49</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>String ANALYSIS_CODE_15</td>
<td>49</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_1</td>
<td>177</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_2</td>
<td>177</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_3</td>
<td>177</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_4</td>
<td>177</td>
<td>2</td>
<td>[0.0],[400.0]</td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_5</td>
<td>177</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_6</td>
<td>177</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_7</td>
<td>177</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_8</td>
<td>177</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>double ANALYSIS_AMOUNT_9</td>
<td>177</td>
<td>1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

177
double ANALYSIS_AMOUNT_10 [In use=177] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_11 [In use=177] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_12 [In use=177] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_13 [In use=177] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_14 [In use=177] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_15 [In use=177] [diff 1 value 0.0]
String ANALYSIS_TYPE_1 [In use=176] [diff 2 values= [C],[S]]
String ANALYSIS_TYPE_2 [In use=176] [diff 2 values= [C],[S]]
short AUXILIARY_STATUS [In use=177] [diff 1 value 0.0]
String CONTACT [In use=30] [diff 22]
String CREDIT_ORDER_NUMBER [In use=0] [diff 0]
double CURRENT_ORDER_VALUE [In use=177] [diff 19]
short DEL_ADDRESS_CODE [In use=177] [diff 1 value 0.0]
String DEL_ADDRESS_NAME [In use=164] [diff 66]
String DEL_ADDRESS_LINE_1 [In use=92] [diff 41]
String DEL_ADDRESS_LINE_2 [In use=79] [diff 31]
String DEL_ADDRESS_LINE_3 [In use=77] [diff 28]
String DEL_ADDRESS_LINE_4 [In use=63] [diff 17]
String DELIVERY_NOTE_NUMBER [In use=0] [diff 0]
double DISCT_SUR_PERCENT [In use=177] [diff 1 value 0.0]
double EXCHANGE_RATE [In use=177] [diff 0]
double FC_CURRENT_ORDER_VALUE [In use=177] [diff 19]
String FREE_FORMAT_DESC_1 [In use=0] [diff 0]
String FREE_FORMAT_DESC_2 [In use=0] [diff 0]
short FREE_FORMAT_VAT_CD_1 [In use=177] [diff 3]
short FREE_FORMAT_VAT_CD_2 [In use=177] [diff 3]
double FREE_FORMAT_AMOUNT_1 [In use=177] [diff 1 value 0.0]
double FREE_FORMAT_AMOUNT_2 [In use=177] [diff 1 value 0.0]
String FREE_FORMAT_NOM_CC_1 [In use=48] [diff 2 values= [EXT],[ING]]
String FREE_FORMAT_NOM_CC_2 [In use=48] [diff 2 values= [EXT],[ING]]
String FREE_FORMAT_NOM_CD_1 [In use=97] [diff 1 value 100]
String FREE_FORMAT_NOM_CD_2 [In use=97] [diff 1 value 100]
double FULL_ORDER VALUE [In use=177] [diff 77]
String HISTORY_ANALYSIS [In use=117] [diff 6]
java.util.Date INVOICE_DATE [In use=177] [diff 27]
String INVOICE_NUMBER [In use=66] [diff 66]
short INVOICED_VAT_CODE_1 [In use=177] [diff 4]
double INVOICED_VAT GOODS_1 [In use=177] [diff 54]
double INVOICED_VAT GOODS_4 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT GOODS_7 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT GOODS_8 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT GOODS_9 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT GOODS_10 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_1 [In use=177] [diff 15]
double INVOICED_VAT AMT_2 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_3 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_4 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_5 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_6 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_7 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_8 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT AMT_9 [In use=177] [diff 1 value 0.0]
double INVOICED_VAT_AMT_10 [In use=177] [diff 1 value 0.0]
double INVOICED_DISCOUNT_AMT [In use=177] [diff 1 value 0.0]
short LAST_SEQUENCE_NO [In use=177] [diff 28]
String MAINT_CONTRACT [In use=0] [diff 0]
short NO_OF_VAT_CODES_USED [In use=177] [diff 4]

178
Appendix C

Information Files results from the Case Study

```java
java.util.Date ORDER_DATE [In use=177] [diff 87]
java.util.Date ORDER_DUE_DATE [In use=177] [diff 102]
String ORDER_NUMBER [In use=177] [diff 177]
String ORDER_PRIORITY_LEVEL [In use=0] [diff 0]
short ORDER_STATUS [In use=177] [diff 5]
short ORDER_TYPE [In use=177] [diff 2 values=[-1],[1]]
double ORDER_VALUE [In use=177] [diff 19]
double ORDER_VALUE_INC_OUT [In use=177] [diff 38]
double ORDER_VALUE_DISC_PCNT [In use=177] [diff 1 value 0.0]
java.util.Date POSTED_DATE [In use=177] [diff 26]
String PROCESS_IND [In use=0] [diff 0]
String PROCESSED_BY_MRP [In use=0] [diff 0]
String QC_IND [In use=0] [diff 0]
int SALES_HEADER_REC [In use=177] [diff 36]
double SETTLEMENT_DISC_PCENT [In use=177] [diff 1 value 0.0]
short SETTLEMENT_DISC_DAYS [In use=177] [diff 1 value 0]
String SHORT_NAME [In use=177] [diff 37]
String SL_ANALYSIS_1 [In use=108] [diff 2 values=[EXT],[ING]]
short SSD_ENTRY_REQUIRED [In use=177] [diff 3]
String THEIR_REFERENCE [In use=109] [diff 76]
String USE_HEAD_OFFICE [In use=177] [diff 1 value N]
String VAT_INCLUSIVE_PRICES [In use=0] [diff 0]
String WORKS_INST_IND [In use=0] [diff 0]

CLASS WOP_WORKS_ORDERS num. records=5199
String ALLOW_MRP_RESCHED [In use=3914] [diff 1 value Y]
String AUTO_RELEASE_IND [In use=0] [diff 0]
String BOM_IND [In use=133] [diff 1 value N]
java.util.Date CLOSE_DATE [In use=5199] [diff 90]
String COMPONENT_DEPOT_CODE [In use=1275] [diff 1 value M]
int COMPONENT_COUNT [In use=5199] [diff 82]
short CUSTOMS_SEQ [In use=5199] [diff 1 value 0]
String DOCUMENT_NUMBER [In use=1280] [diff 125]
java.util.Date DUE_DATE [In use=5199] [diff 251]
String EXCLUDE_FROM_POTL_STK [In use=5194] [diff 2 values=[N],[Y]]
String EXCLUDE_FROM_MRP [In use=5194] [diff 2 values=[N],[Y]]
String ISSUE_NUMBER [In use=0] [diff 0]
String ITEM_NUMBER [In use=5195] [diff 930]
short JOB_CARDS_PRINTED [In use=5199] [diff 1 value 0]
short KIT_LIST_PRINTED [In use=5199] [diff 2 values=[0],[1]]
double LABOUR_VALUE [In use=5199] [diff 289]
java.util.Date LAST_ISSUE_DATE [In use=5199] [diff 129]
short LAST_STAGE_NO [In use=5199] [diff 1 value 0]
double MATERIAL_VALUE [In use=5199] [diff 331]
double MATERIAL_OHEAD [In use=5199] [diff 1 value 0.0]
String OPS_BOOKED_IND [In use=0] [diff 0]
String ORDER_DEPOT_CODE [In use=5194] [diff 1 value M]
String ORDER_DESCRIPTION [In use=5195] [diff 18]
String ORDER_NUMBER [In use=5199] [diff 5199]
short ORDER_STATUS [In use=5199] [diff 9]
int ORDER_SUFFIX [In use=5199] [diff 1 value 0]
short ORDER_TYPE [In use=5199] [diff 3]
double OTHER_COSTS [In use=5199] [diff 2 values=[0.0],[13.05]]
double OVERHEAD_VALUE [In use=5199] [diff 296]
String OWNER [In use=0] [diff 0]
String PARENT_ORDER_NO [In use=0] [diff 0]
short PARTIAL_COMPONENT_ISS [In use=5199] [diff 1 value 0]
short PHASE_NUMBER [In use=5199] [diff 1 value 0]
String QC_IND [In use=0] [diff 0]
```
Information Files results from the Case Study

Appendix C

```
double QTY_COMPLETE [ In use=5199] [ diff 50]
double QTY_ISSUED [ In use=5199] [ diff 51]
double QTY_ORDERED [ In use=5199] [ diff 66]
double QTY_REWORKED [ In use=5199] [ diff 1 value 0.0]
double QTY_SCRAPPED [ In use=5199] [ diff 1 value 0.0]
double QTY_THIS_STAGE [ In use=5199] [ diff 1 value 0.0]
java.util.Date RELEASE_DATE [ In use=5199] [ diff 118]
String ROUTING_IND [ In use=0] [ diff 0]
short ROUTING_PRINTED [ In use=5199] [ diff 2 values= [0],[1]]
String RTG_ISSUE_NUMBER [ In use=0] [ diff 0]
short ROUTING_RINTED [ In use=5199] [ diff 2 values= [0],[1]]
double SCRAP_ALLOWANCE [ In use=5199] [ diff 1 value 0.0]
short SEQUENCE_NO [ In use=5199] [ diff 2 values= [0],[1]]
short STAGE_ISSUED [ In use=5199] [ diff 1 value 0]
java.util.Date START_DATE [ In use=5199] [ diff 268]
String TEMP_COST _ IND [ In use=0] [ diff 0]
String TSSI_HIST_SHEET_IND [ In use=0] [ diff 0]
short VERSION [ In use=5199] [ diff 10]
String WALLCHART_IND [ In use=0] [ diff 0]
String WOP_BOM_AMENDED [ In use=107] [ diff 1 value Y]
String WOP_ROUT_AMENDED [ In use=0] [ diff 0]
Cap_Routings_File Item_number

CLASS CAP_ROUTINGS_FILE num. records=2326
String ADDL_DESCRIPTION [ In use=2266] [ diff 729]
String BRANCH_OPS [ In use=1185] [ diff 1 value N]
java.util.Date DATE_AMENDED [ In use=2326] [ diff 451]
java.util.Date DATE_CREATED [ In use=2326] [ diff 451]
java.util.Date END_DATE [ In use=2236] [ diff 1 value 1999-11-30 00:00:00.0]
String ENG_CHANGE_REF [ In use=0] [ diff 0]
short FIRST_PRINT_IND [ In use=2236] [ diff 2 values= [0],[1]]
String ITEM_NUMBER [ In use=2236] [ diff 2303]
double LAST_RELEASE_QTY [ In use=2236] [ diff 1 value 0.0]
java.util.Date LAST_RELEASE_DATE [ In use=2236] [ diff 1 value 1999-11-30 00:00:00.0]
short PRINT_CODE [ In use=2236] [ diff 2 values= [0],[1]]
short RELEASE_CODE [ In use=2236] [ diff 1 value 0]
double RELEASE_QTY [ In use=2236] [ diff 20]
java.util.Date START_DATE [ In use=2236] [ diff 1 value 1999-11-30 00:00:00.0]
short TIME_CODE [ In use=2236] [ diff 1 value 1]
short VERSION [ In use=2236] [ diff 10]
Stock_Control_File Item_number
Wop_Works_Order Order_number

CLASS POR_PURCHASE_ORDERS num. records=3010
String ACCOUNT_NUMBER [ In use=3010] [ diff 342]
String ACCOUNT_NAME [ In use=3010] [ diff 354]
short AUXILIARY_STATUS [ In use=3010] [ diff 2 values= [0],[3]]
String BUYER_CODE [ In use=0] [ diff 0]
String CONTACT [ In use=1441] [ diff 227]
double CURRENT_ORDER_VALUE [ In use=3010] [ diff 1 value 0.0]
short DEL_ADDRESS_CODE [ In use=3010] [ diff 8]
double EXCHANGE_RATE [ In use=3010] [ diff 23]
double FC_ORDER_VALUE [ In use=3010] [ diff 1695]
String FREE_FORMAT_DESC_1 [ In use=0] [ diff 0]
String FREE_FORMAT_DESC_2 [ In use=0] [ diff 0]
short FREE_FORMAT_VAT_CD_1 [ In use=3010] [ diff 1 value 0]
short FREE_FORMAT_VAT_CD_2 [ In use=3010] [ diff 1 value 0]
double FREE_FORMAT_AMOUNT_1 [ In use=3010] [ diff 1 value 0.0]
double FREE_FORMAT_AMOUNT_2 [ In use=3010] [ diff 1 value 0.0]
String FREE_FORMAT_CC_1 [ In use=0] [ diff 0]
```
Appendix C

Information Files results from the Case Study

String FREE_FORMAT_CC_2 [In use=0] [diff 0]
String FREE_FORMAT_DEPT_1 [In use=0] [diff 0]
String FREE_FORMAT_DEPT_2 [In use=0] [diff 0]
String FREE_FORMAT_ACCOUNT_1 [In use=0] [diff 0]
String FREE_FORMAT_ACCOUNT_2 [In use=0] [diff 0]
String HISTORY_ANALYSIS [In use=7] [diff 4]
String LAST_DELIVERY_NOTE_NO [In use=1537] [diff 1390]
String LAST_INVOICE_NO [In use=1454] [diff 1381]
java.util.Date LAST_INVOICE_DATE [In use=3010] [diff 246]
short LASTSEQUENCE_NO [In use=3010] [diff 67]
short NO_OF_SCHED_ITEMS [In use=3010] [diff 1 value 0]
double NOM_EXCH_RATE [In use=3010] [diff 22]
java.util.Date ORDER_DATE [In use=3010] [diff 385]
java.util.Date ORDER_DUE_DATE [In use=3010] [diff 400]
String ORDER_NUMBER [In use=3009] [diff 3009]
String ORDER_PRIORITY_LEVEL [In use=0] [diff 0]
short ORDER_STATUS [In use=3010] [diff 8]
short ORDER_TYPE [In use=3010] [diff 2 values= [-1], [1]]
double ORDER_VALUE [In use=3010] [diff 1989]
double OVERALL_DISC_PERCENT [In use=3010] [diff 1 value 0.0]
int PURCHASE_HEADER_REC [In use=3010] [diff 342]
String QC_IND [In use=0] [diff 0]
String REPEAT_ORDER [In use=1205] [diff 1 value M]
String RETURN_NOTE_EXPECTED [In use=0] [diff 0]
String SHORT_NAME [In use=3009] [diff 345]
String SPECIALINS_1 [In use=14] [diff 11]
String SPECIALINS_2 [In use=3] [diff 3]
short SSD_ENTRY_REQUIRED [In use=3010] [diff 3]
String SUPP_ADDRESS_NAME [In use=1815] [diff 355]
String SUPP_ADDRESS_LINE_1 [In use=1812] [diff 362]
String SUPP_ADDRESS_LINE_2 [In use=1811] [diff 333]
String SUPP_ADDRESS_LINE_3 [In use=1805] [diff 228]
String SUPP_ADDRESS_LINE_4 [In use=1664] [diff 325]
String THEIR_REFERENCE [In use=404] [diff 386]
String VATINCLUSIVE_PRICES [In use=0] [diff 0]
int WAITING_DELIVERY [In use=3010] [diff 12]
int WAITING_INVOICE [In use=3010] [diff 10]
C.2. EXAMPLE MAKE-TO-ORDER COMPANY

Empty tables:
MPS_SCHEDULE_FILE
BOM_ISSUED_SERIALS
BOM_SERIALS_NUMBER

Tables in Use:
CLASS STOCK_CONTROL_FILE  num. records=87727
CLASS BOM_STRUCTURE_FILE num. records=202095
CLASS BOM_LEVELS_FILE num. records=120558
CLASS BOM_ALLOCS_ORDERS num. records=26027

CLASS SOR_SALES_ORDERS num. records=138
String ACCOUNT_NUMBER [ In use=138 ] [ diff 90]
String ACCOUNT_NAME [ In use=138 ] [ diff 88]
String ALLOCATE_ON_ORD_ENTRY [ In use=0 ] [ diff 0]
String ANALYSIS_CC_1 [ In use=134 ] [ diff 1 value AB]
String ANALYSIS_CC_2 [ In use=118 ] [ diff 10]
String ANALYSIS_CC_3 [ In use=60 ] [ diff 7]
String ANALYSIS_CC_4 [ In use=19 ] [ diff 5]
String ANALYSIS_CC_5 [ In use=7 ] [ diff 3]
String ANALYSIS_CC_6 [ In use=1 ] [ diff 1 value G7]
String ANALYSIS_CC_7 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_8 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_9 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_10 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_11 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_12 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_13 [ In use=0 ] [ diff 0]
String ANALYSIS_CC_14 [ In use=65 ] [ diff 1 value AB]
String ANALYSIS_CC_15 [ In use=65 ] [ diff 1 value AB]
String ANALYSIS_DEPT_1 [ In use=134 ] [ diff 4]
String ANALYSIS_DEPT_2 [ In use=118 ] [ diff 2 values= [MAT],[MFG]]
String ANALYSIS_DEPT_3 [ In use=60 ] [ diff 2 values= [MAT],[MFG]]
String ANALYSIS_DEPT_4 [ In use=19 ] [ diff 2 values= [MAT],[MFG]]
String ANALYSIS_DEPT_5 [ In use=7 ] [ diff 2 values= [MAT],[MFG]]
String ANALYSIS_DEPT_6 [ In use=1 ] [ diff 1 value MAT]
String ANALYSIS_DEPT_7 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_8 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_9 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_10 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_11 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_12 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_13 [ In use=0 ] [ diff 0]
String ANALYSIS_DEPT_14 [ In use=65 ] [ diff 3]
String ANALYSIS_DEPT_15 [ In use=65 ] [ diff 2 values= [GEN],[MFG]]
String ANALYSIS_CODE_1 [ In use=134 ] [ diff 10]
String ANALYSIS_CODE_2 [ In use=118 ] [ diff 1 value 200010]
String ANALYSIS_CODE_3 [ In use=60 ] [ diff 4]
String ANALYSIS_CODE_4 [ In use=19 ] [ diff 2 values= [10002],[200010]]
String ANALYSIS_CODE_5 [ In use=7 ] [ diff 1 value 200010]
String ANALYSIS_CODE_6 [ In use=1 ] [ diff 1 value 200010]
String ANALYSIS_CODE_7 [ In use=0 ] [ diff 0]
String ANALYSIS_CODE_8 [ In use=0 ] [ diff 0]
Information Files results from the Case Study

String ANALYSIS_CODE_9 [In use=0] [diff 0]
String ANALYSIS_CODE_10 [In use=0] [diff 0]
String ANALYSIS_CODE_11 [In use=0] [diff 0]
String ANALYSIS_CODE_12 [In use=0] [diff 0]
String ANALYSIS_CODE_13 [In use=0] [diff 0]
String ANALYSIS_CODE_14 [In use=65] [diff 2 values= [10002], [60095]]
String ANALYSIS_CODE_15 [In use=65] [diff 1 value 10002]
double ANALYSIS_AMOUNT_1 [In use=138] [diff 75]
double ANALYSIS_AMOUNT_2 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_3 [In use=138] [diff 4]
double ANALYSIS_AMOUNT_4 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_5 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_6 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_7 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_8 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_9 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_10 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_11 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_12 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_13 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_14 [In use=138] [diff 1 value 0.0]
double ANALYSIS_AMOUNT_15 [In use=138] [diff 1 value 0.0]
String ANALYSIS_TYPE_1 [In use=134] [diff 1 value S]
String ANALYSIS_TYPE_2 [In use=134] [diff 1 value C]
String ANALYSIS_TYPE_3 [In use=60] [diff 2 values= [C],[S]]
String ANALYSIS_TYPE_4 [In use=19] [diff 2 values= [C],[S]]
String ANALYSIS_TYPE_5 [In use=7] [diff 1 value C]
String ANALYSIS_TYPE_6 [In use=1] [diff 1 value C]
String ANALYSIS_TYPE_7 [In use=0] [diff 0]
String ANALYSIS_TYPE_8 [In use=0] [diff 0]
String ANALYSIS_TYPE_9 [In use=0] [diff 0]
String ANALYSIS_TYPE_10 [In use=0] [diff 0]
String ANALYSIS_TYPE_11 [In use=0] [diff 0]
String ANALYSIS_TYPE_12 [In use=0] [diff 0]
String ANALYSIS_TYPE_13 [In use=0] [diff 0]
String ANALYSIS_TYPE_14 [In use=0] [diff 0]
String ANALYSIS_TYPE_15 [In use=0] [diff 0]
short AUXILIARY_STATUS [In use=138] [diff 5]
String CONTACT [In use=56] [diff 46]
String CREDIT_ORDER_NUMBER [In use=1] [diff 1 value BPC002-1]
double CURRENT_ORDER_VALUE [In use=138] [diff 77]
short DEL_ADDRESS_CODE [In use=138] [diff 1 value 0]
String DEL_ADDRESS_NAME [In use=91] [diff 80]
String DEL_ADDRESS_LINE_1 [In use=87] [diff 75]
String DEL_ADDRESS_LINE_2 [In use=87] [diff 76]
String DEL_ADDRESS_LINE_3 [In use=85] [diff 68]
String DEL_ADDRESS_LINE_4 [In use=68] [diff 52]
String DELIVERY_NOTE_NUMBER [In use=0] [diff 0]
double DISCt_SURCENT [In use=138] [diff 1 value 0.0]
double EXCHANGE_RATE [In use=138] [diff 6]
double FC_CURRENT_ORD_VALUE [In use=138] [diff 77]
String FREE_FORMAT_DESC_1 [In use=2] [diff 2 values= [CARRIAGE & PACKING],[TRANSPORT]]
String FREE_FORMAT_DESC_2 [In use=0] [diff 0]
short FREE_FORMAT_VAT_CD_1 [In use=138] [diff 3]
short FREE_FORMAT_VAT_CD_2 [In use=138] [diff 3]
double FREE_FORMAT_AMOUNT_1 [In use=138] [diff 3]
double FREE_FORMAT_AMOUNT_2 [In use=138] [diff 1 value 0.0]
String FREE_FORMAT_NOM_CC_1 [In use=72] [diff 1 value AB]
String FREE_FORMAT_NOM_CC_2 [In use=72] [diff 1 value AB]
Appendix C

Information Files results from the Case Study

String FREE_FORMAT_NOM_DP_1 [In use=72] [diff 1 value MFG]
String FREE_FORMAT_NOM_DP_2 [In use=72] [diff 1 value MFG]
String FREE_FORMAT_NOM_CD_1 [In use=72] [diff 2 values=[10002],[10005]]
String FREE_FORMAT_NOM_CD_2 [In use=72] [diff 1 value 10002]
double FULL_ORDER_VALUE [In use=138] [diff 120]
String HISTORY_ANALYSIS [In use=96] [diff 5]
java.util.Date INVOICE_DATE [In use=138] [diff 120]
String INVOICE_NUMBER [In use=88] [diff 88]
short INVOICED_VAT_CODE_1 [In use=138] [diff 4]
short INVOICED_VAT_CODE_2 [In use=138] [diff 4]
short INVOICED_VAT_CODE_3 [In use=138] [diff 2 values=[0],[2]]
short INVOICED_VAT_CODE_4 [In use=138] [diff 1 value 0]
short INVOICED_VAT_CODE_5 [In use=138] [diff 1 value 0]
short INVOICED_VAT_CODE_6 [In use=138] [diff 1 value 0]
short INVOICED_VAT_CODE_7 [In use=138] [diff 1 value 0]
short INVOICED_VAT_CODE_8 [In use=138] [diff 1 value 0]
short INVOICED_VAT_CODE_9 [In use=138] [diff 1 value 0]
short INVOICED_VAT_CODE_10 [In use=138] [diff 1 value 0]
double INVOICED_VAT_GOODS_1 [In use=138] [diff 75]
double INVOICED_VAT_GOODS_2 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_3 [In use=138] [diff 2 values=[0.0],[965.0]]
double INVOICED_VAT_GOODS_4 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_5 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_6 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_7 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_8 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_9 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_GOODS_10 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_1 [In use=138] [diff 15]
double INVOICED_VAT_AMT_2 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_3 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_4 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_5 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_6 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_7 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_8 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_9 [In use=138] [diff 1 value 0.0]
double INVOICED_VAT_AMT_10 [In use=138] [diff 1 value 0.0]
double INVOICED_DISCOUNT_AMT [In use=138] [diff 1 value 0.0]
short LAST_SEQUENCE_NO [In use=138] [diff 17]
String MAINT_CONTRACT [In use=0] [diff 0]
short NO_OF_VAT_CODES_USED [In use=138] [diff 4]
java.util.Date ORDER_DATE [In use=138] [diff 100]
java.util.Date ORDER_DUE_DATE [In use=138] [diff 94]
String ORDER_NUMBER [In use=138] [diff 138]
short ORDER_STATUS [In use=138] [diff 6]
short ORDER_TYPE [In use=138] [diff 2 values=[-1],[1]]
double ORDER_VALUE [In use=138] [diff 76]
double ORDER_VALUE_INC_OUT [In use=138] [diff 81]
double ORDER_VALUE_DISC_PCNT [In use=138] [diff 1 value 0.0]
java.util.Date POSTED_DATE [In use=138] [diff 45]
String PROCESS_IND [In use=0] [diff 0]
String PROCESSED_BY_MRP [In use=0] [diff 0]
String QC_IND [In use=0] [diff 0]
String RECORD_TYPE [In use=1] [diff 1 value V]
int SALES_HEADER_REC [In use=138] [diff 90]
double SETTLEMENT_DISC_PCNT [In use=138] [diff 1 value 0.0]
short SETTLEMENT_DISC_DAYS [In use=138] [diff 1 value 0]
String SHORT_NAME [In use=138] [diff 85]

184
Information Files results from the Case Study

```java
String SL_ANALYSIS_1 [ In use=0 ] [ diff0]
String SL_ANALYSIS_2 [ In use=0 ] [ diff0]
String SL_ANALYSIS_3 [ In use=0 ] [ diff0]
short SSD_ENTRY_REQUIRED [ In use=138 ] [ diff3]
String THEIR_REFERENCE [ In use=123 ] [ diff11]
String USE_HEAD_OFFICE [ In use=138 ] [ diff1 value N]
String VAT_INCLUSIVE_PRICES [ In use=0 ] [ diff0]
String WORKS_INST_IND [ In use=0 ] [ diff0]

CLASS WOP_WORKS_ORDERS num. records=12598
String ALLOW_MRP_RESCHED [ In use=9484 ] [ diff2 values= [N],[Y] ]
String AUTO_RELEASE_IND [ In use=0 ] [ diff0]
String BOM_IND [ In use=387 ] [ diff1 value N]
java.util.Date CLOSE_DATE [ In use=12598 ] [ diff220]
String COMPONENT_DEPOT_CODE [ In use=8242 ] [ diff1 value ]
int COMPONENT_COUNT [ In use=12598 ] [ diff182]
short CUSTOMS_SEQ [ In use=12598 ] [ diff1 value 0]
String DOCUMENT_NUMBER [ In use=10295 ] [ diff166]
java.util.Date DUE_DATE [ In use=12598 ] [ diff854]
String EXCLUDE_FROM_POTL_STK [ In use=11427 ] [ diff2 values= [N],[Y] ]
String EXCLUDE_FROM_MRP [ In use=11427 ] [ diff2 values= [N],[Y] ]
String ISSUE_NUMBER [ In use=0 ] [ diff0]
String JOB_CARDS_PRINTED [ In use=12598 ] [ diff2 values= [0],[1] ]
short KIT_LIST_PRINTED [ In use=12598 ] [ diff2 values= [0],[1] ]
double LABOUR_VALUE [ In use=12598 ] [ diff2459]
java.util.Date LAST_ISSUE_DATE [ In use=12598 ] [ diff854]
short LAST_STAGE_NO [ In use=12598 ] [ diff1 value 0]
double MATERIAL_VALUE [ In use=12598 ] [ diff3712]
double MATERIAL_OHEAD [ In use=12598 ] [ diff1 value 0.0]
String OP_TABLE_CODE [ In use=6 ] [ diff1 value Y]
String ORDER_DEPOT_CODE [ In use=10707 ] [ diff1 value ]
String ORDER_DESCRIPTION [ In use=5796 ] [ diff126]
String ORDER_NUMBER [ In use=12598 ] [ diff12598]
short ORDER_STATUS [ In use=12598 ] [ diff8]
int ORDER_SUFFIX [ In use=12598 ] [ diff5]
short ORDER_TYPE [ In use=12598 ] [ diff6]
double OTHER_COSTS [ In use=12598 ] [ diff9]
double OVERHEAD_VALUE [ In use=12598 ] [ diff2464]
String OWNER [ In use=0 ] [ diff0]
String PARENT_ORDER_NO [ In use=3531 ] [ diff282]
short PARTIAL_COMPONENT_ISS [ In use=12598 ] [ diff1 value 0]
short PHASE_NUMBER [ In use=12598 ] [ diff53]
String QC_IND [ In use=0 ] [ diff0]
double QTY_COMPLETE [ In use=12598 ] [ diff55]
double QTY_ISSUED [ In use=12598 ] [ diff60]
double QTY_ORDERED [ In use=12598 ] [ diff57]
double QTY_REWORKED [ In use=12598 ] [ diff1 value 0.0]
double QTY_SCRAPPED [ In use=12598 ] [ diff1 value 0.0]
double QTY_THIS_STAGE [ In use=12598 ] [ diff1 value 0.0]
java.util.Date RELEASE_DATE [ In use=12598 ] [ diff645]
String ROUTING_IND [ In use=0 ] [ diff0]
short ROUTING_PRINTED [ In use=12598 ] [ diff2 values= [0],[1] ]
String RTG_ISSUE_NUMBER [ In use=0 ] [ diff0]
double SCRAPP_ALLOWANCE [ In use=12598 ] [ diff1 value 0.0]
short SEQUENCE_NO [ In use=12598 ] [ diff6]
String SHORTAGE_IND [ In use=1525 ] [ diff2 values= [N],[Y] ]
short STAGE_NUMBER [ In use=12598 ] [ diff1 value 0]
java.util.Date START_DATE [ In use=12598 ] [ diff840]
String TEMP_COST_IND [ In use=0 ] [ diff0]
```
Information Files results from the Case Study

CLASS CAP_ROUTINGS_FILE  num. records=10172
String ADDL_DESCRIPTION  [ In use=4163] [ diff 169]
String BRANCH_OPS  [ In use=5643] [ diff 1 value N]
java.util.Date DATE_AMENDED  [ In use=10172] [ diff 684]
java.util.Date DATE_CREATED  [ In use=10172] [ diff 699]
java.util.Date END_DATE  [ In use=10172] [ diff 1 value 1999-11-30 00:00:00.0]
String ENG_CHANGE_REF  [ In use=0] [ diff 0]
short FIRST_PRINT_IND  [ In use=10172] [ diff 2 values= [0],[1]]
String ITEM_NUMBER  [ In use=10172] [ diff 7943]
double LAST_RELEASE_QTY  [ In use=10172] [ diff 1 value 0.0]
java.util.Date LAST_RELEASE_DATE  [ In use=10172] [ diff 1 value 1999-11-30 00:00:00.0]
short PRINT_CODE  [ In use=10172] [ diff 2 values= [0],[1]]
short RELEASE_CODE  [ In use=10172] [ diff 1 value 0]
double RELEASE_QTY  [ In use=10172] [ diff 13]
java.util.Date START_DATE  [ In use=10172] [ diff 1 value 1999-11-30 00:00:00.0]
short VERSION  [ In use=10172] [ diff 6]
Stock_Control_File item_number
Wop_Works_Order Order_number

CLASS POR_PURCHASE_ORDERS  num. records=1945
String ACCOUNT_NUMBER  [ In use=1945] [ diff 344]
String ACCOUNT_NAME  [ In use=1945] [ diff 357]
short AUXILIARY_STATUS  [ In use=1945] [ diff 1 value 0]
String BUYER_CODE  [ In use=0] [ diff 0]
String CONTACT  [ In use=420] [ diff 98]
double CURRENT_ORDER_VALUE  [ In use=1945] [ diff 1 value 0.0]
short DEL_ADDRESS_CODE  [ In use=1945] [ diff 8]
double EXCHANGE_RATE  [ In use=1945] [ diff 27]
double FC_ORDER_VALUE  [ In use=1945] [ diff 1503]
String FREE_FORMAT_DESC_1  [ In use=0] [ diff 0]
String FREE_FORMAT_DESC_2  [ In use=0] [ diff 0]
short FREE_FORMAT_VAT_CD_1  [ In use=1945] [ diff 1 value 0]
short FREE_FORMAT_VAT_CD_2  [ In use=1945] [ diff 1 value 0]
double FREE_FORMAT_AMOUNT_1  [ In use=1945] [ diff 1 value 0.0]
double FREE_FORMAT_AMOUNT_2  [ In use=1945] [ diff 1 value 0.0]
String FREE_FORMAT_CC_1  [ In use=0] [ diff 0]
String FREE_FORMAT_CC_2  [ In use=0] [ diff 0]
String FREE_FORMAT_DEPT_1  [ In use=0] [ diff 0]
String FREE_FORMAT_DEPT_2  [ In use=0] [ diff 0]
String FREE_FORMAT_ACCOUNT_1  [ In use=0] [ diff 0]
String FREE_FORMAT_ACCOUNT_2  [ In use=0] [ diff 0]
String HISTORY_ANALYSIS  [ In use=1162] [ diff 27]
String LAST_DELIVERY_NOTE_NO  [ In use=1167] [ diff 1097]
String LAST_INVOICE_NO  [ In use=879] [ diff 878]
java.util.Date LAST_INVOICE_DATE  [ In use=1945] [ diff 302]
short LAST_SEQUENCE_NO  [ In use=1945] [ diff 72]
short NO_OF_SCHED_ITEMS  [ In use=1945] [ diff 1 value 0]
double NOM_EXCH_RATE  [ In use=1945] [ diff 21]
java.util.Date ORDER_DATE  [ In use=1945] [ diff 558]
java.util.Date ORDER_DUE_DATE  [ In use=1945] [ diff 558]
String ORDER_NUMBER  [ In use=1944] [ diff 1944]
String ORDER_PRIORITY_LEVEL  [ In use=0] [ diff 0]
Appendix C

Information Files results from the Case Study

short ORDER_STATUS [In use=1945] [diff 6]
short ORDER_TYPE [In use=1945] [diff 2 values= [-1], [1]]
double ORDER_VALUE [In use=1945] [diff -1575]
double OVERALL_DISCT_PCT [In use=1945] [diff 1 value 0.0]
int PURCHASE_HEADER_REC [In use=1945] [diff 344]
String QC_IND [In use=0] [diff 0]
String REPEAT_ORDER [In use=824] [diff 1 value M]
String RETURN_NOTE_EXPECTED [In use=0] [diff 0]
String SHORT_NAME [In use=1945] [diff 358]
String SPECIAL_INS_1 [In use=234] [diff 32]
String SPECIAL_INS_2 [In use=100] [diff 45]
short SSD_ENTRY_REQUIRED [In use=1945] [diff 2 values= [0], [2]]
String SUPP_ADDRESS_NAME [In use=1665] [diff 360]
String SUPP_ADDRESS_LINE_1 [In use=1665] [diff 356]
String SUPP_ADDRESS_LINE_2 [In use=1664] [diff 331]
String SUPP_ADDRESS_LINE_3 [In use=1661] [diff 240]
String SUPP_ADDRESS_4 [In use=1422] [diff 327]
String THEIR_REFERENCE [In use=20] [diff 20]
String VAT_INCLUSIVE_PRICES [In use=0] [diff 0]
int WAITING_DELIVERY [In use=1945] [diff 18]
int WAITING_INVOICE [In use=1945] [diff 14]