A Survey of Research into Legacy System Migration

Jesus Bisbal, Deirdre Lawless,
Bing Wu, Jane Grimson, Vincent Wade,
Trinity College, Dublin, Ireland.

Ray Richardson, Donie O'Sullivan,
Broadcom Éireann Research,
Dublin, Ireland.

Abstract

Legacy information systems typically form the backbone of the information flow within an organisation and are the main vehicle for consolidating information about the business. As a solution to the problems these systems pose - brittleness, inflexibility, isolation, non-extensibility, lack of openness etc. - many companies are migrating their legacy systems to new environments which allow the information system to more easily adapt to new business requirements.

This paper presents a survey of research into Migration of Legacy Information Systems. The main problems that companies with legacy systems must face are analysed, and the challenges possible solutions must solve discussed. The paper provides an overview of the most important currently available solutions, and their main downsides are

1. INTRODUCTION 2

2. MIGRATION ISSUES 3

2.1. JUSTIFICATION 3
2.2. LEGACY SYSTEM UNDERSTANDING 4
2.3. TARGET SYSTEM DEVELOPMENT 6
2.4. TESTING 6
2.5. MIGRATION 8

3. LEGACY SYSTEM MIGRATION TOOL SUPPORT 9

3.1. JUSTIFICATION TOOLS 10
3.2. LEGACY SYSTEM UNDERSTANDING TOOLS 10
3.2.1. Program Understanding Tools 11
3.2.2. Database Understanding Tools 12
3.3. TARGET SYSTEM DEVELOPMENT TOOLS 13
3.3.1. Client/Server Development 13
3.3.2. Transaction Processing 15
3.3.3. Internet Based Technologies for Legacy Systems 15
3.4. TRANSITION SUPPORT TOOLS 16
3.4.1. Year 2000 Problem Tools 16
3.4.2. Integration Software 17
3.4.3. Consultancy Services 19
3.5. TESTING TOOLS 19
3.6. LIMITATIONS OF MIGRATION SUPPORT TOOLS 20

4. APPROACHES TO MIGRATION 20

4.1. THE BIG BANG APPROACH 21
4.2. THE DATABASE FIRST APPROACH 22
4.3. THE DATABASE LAST APPROACH 24
4.4. THE COMPOSITE DATABASE APPROACH 25
4.5. THE CHICKEN-LITTLE STRATEGY 26
4.6. THE BUTTERFLY METHODOLOGY 28
4.7. SCREEN SCRAPING 30

5. OPEN RESEARCH ISSUES 31

5.1. LEGACY SYSTEM UNDERSTANDING 31
5.2. TARGET SYSTEM DEVELOPMENT 32
5.3. DATA MIGRATION 33
5.4. MIGRATION TOOLS 34
5.5. MIGRATION APPROACHES 34
5.6. GENERAL MIGRATION ISSUES 35

6. SUMMARY 36

7. REFERENCES 37

APPENDIX I - ANALYSIS OF CURRENT MIGRATION APPROACHES 38

APPENDIX II - OPEN RESEARCH ISSUES 39

identified. Finally, it defines the stages involved in any migration process, and a set of tools to support each one are outlined.

Index Terms *- Legacy systems, migration methodologies, migration steps, migration tools, re-engineering.*

1. Introduction

Legacy information systems[1] typically form the backbone of the information flow within an organisation and are the main vehicle for consolidating information about the business. If one of these systems stops working, the business may grind to a halt. These mission critical legacy information systems are currently posing numerous and important problems to their host organisations. In particular,

- these systems usually run on obsolete hardware which is slow and expensive to maintain;
- maintenance of software is generally expensive; tracing faults is costly and time consuming due to the lack of documentation and a general lack of understanding of the internal workings of the system;
- integration with other systems is greatly hampered by the absence of clean interfaces;
- evolution of legacy systems to provide new functionality required by the organisation is virtually impossible.

The last point is particularly relevant in today's competitive environment where organisations must perform their operations in the most cost-effective and efficient way ([25], [48], [29], [55]). Organisations are constantly growing and changing business focus in order to remain competitive. Major changes in business practice inevitably require major changes to the supporting information systems. However, legacy systems are characterised as being very brittle with respect to change. Small modifications or enhancements can lead to unexpected system failures which are very difficult to trace in a largely undocumented environment.

Many organisations now wish to move their legacy systems to new environments which allow information systems to be more easily maintained and adapted to new business requirements. The essence of Legacy Information System Migration is to allow them to do this, retaining the functionality of existing information systems *without having to completely redevelop them.*

Of the currently available, and generally ad-hoc, methods for legacy system migration, few have had, limited, success ([11], [21]). In view of this, many organisations are reluctant to migrate their legacy

[1] A legacy information system can be defined as "any information system that significantly resists modification and evolution", [Brod95].

systems to newer technologies and now find themselves in a 'catch 22' situation: mission critical legacy systems which on the one hand provide life support to the organisation are also a major impediment to progress.

Thus there is an urgent need to provide tools, methodologies and techniques not only for accessing the data which is locked inside these systems, but also to provide a strategy which allows the migration of the systems to new platforms and architectures. The exigency of this requirement is all the more highlighted by the "Year 2000 problem"[2] which will render many legacy systems practically unusable. All these issues have caused legacy information system migration to become one of the major issues in both business and academic research ([7], [59]).

This paper presents an overview of the whole process of legacy system migration. It is divided into 6 sections. The next section discusses the issues involved in any migration project. Section 3 outlines the currently available tool support for migration. Section 4 details existing legacy system migration methodologies. A number of future research issues are presented in Section 5. The concluding section presents a summary of findings.

[2] Well known concept which refers to the problems that many (mission critical) information systems will suffer because the date arithmetic they perform will be invalid at the change of millennium, see section 4.4.1.

2. Migration Issues

Legacy system migration encompasses many research areas. A single migration project could, quite legitimately, address the areas of reverse engineering, business reengineering, schema mapping and translation, data transformation, application development, human computer-interaction, and testing. For the purposes of this paper, the following phases for a generic migration process have been identified:

- Phase 1: Justification
- Phase 2: Legacy System Understanding
- Phase 3: Target System Development
- Phase 4: Testing
- Phase 5: Migration

In this section each of these phases will be discussed with a view to providing a clear understanding of the issues involved in a migration. A general description of the objectives and expected outputs of each phase is provided. Section 3 focuses on particular tools available to support each phase, and gives more detailed descriptions of the expected results.

2.1. Justification

Legacy system migration is a very expensive procedure which carries a definite risk of failure.

Consequently before any decision to migrate is taken, an intensive study should be undertaken to quantify the risk and benefits and fully justify the redevelopment of the legacy system involved [49]. The primary outputs of this investigation should be a cost benefit analysis and an estimation of the possibility of failure. Software quality metrics can be used to estimate the level of technical difficulty involved. The legacy systems contribution to profit and the significance of its information should be used as measures of the systems business value. Its business value combined with estimations of its life expectancy and current maintenance costs could be used to arrive at a figure for the possible benefits redevelopment could bring. The size, decomposability, relation to other systems and relative stability of the systems functionality combined with the technical difficulty involved in its migration should give an estimation of the risk involved. Gaining a measure of how easy software is to maintain by using software maintainability metrics for example, can identify potential reusable components and provide a quantifiable method to determine components which must be completely rewritten ([14], [58]).

2.2. Legacy System Understanding

Legacy system migration can be viewed as a constrained problem solving activity [56], the major constraint being the legacy system. The system resulting from a migration must meet some business/user requirements but as the legacy system already partially meets these requirements, it is essential to the success of the migration to understand the functionality of the legacy system and how it interacts with its domain.

Due to the fact that many legacy systems have poor, if any, documentation, many research projects have chosen to focus on the area of recreating documentation from legacy code (see section 3.2). It has been found that this process cannot be fully automated [9]. In order to recover a relevant design and documentation, much interaction from a system expert is required. Once the design and documentation have been constructed, they still have to be understood. Reusable components and redundancies have to be identified before the requirements for the target system can be produced. A poor understanding of the legacy system will lead to an incorrect specification of requirements for the target system and ultimately a failed migration project.

Legacy system understanding can be aided by reverse engineering. Reverse engineering can be defined as identifying the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction [66].

As well as understanding legacy applications, the structure of the legacy data must also be uncovered. Many different data models have been proposed and implemented over the years including the hierarchical, network, relational, extended relational, and object-oriented. These data models are different ways of organising data. A general rule of thumb is that the older the data model the more difficult it is to rediscover the legacy data structure. The situation is much more complex for the still very prevalent standard file format, for which there is no centralised description of the data structures, but they are buried in the applications' code. Individual source programs must be examined in order to detect partial structures of files. Very often data structures are hidden, optimisation constructs are introduced (e.g. padding for address alignment or record splitting when the size is greater than the page size), and specifications are left to be procedurally encoded.

Both standard files and early data models were not powerful enough to model certain data characteristics and the application developers had to hand-code these features, (e.g. referential integrity constrains). This leaves migration engineers having to wade through complex code fragments in order to retrieve the legacy data structures. As with applications, once the legacy data structure is extracted it still has to be understood. A general lack of standardisation with regard to naming and structuring of data elements combined with implementation specific optimisation components makes the understanding process very difficult to automate. Once the structure has been understood redundancies have to be identified so that only necessary data will be migrated. This data will then have to be analysed and cleaned before migration. The adage 'Garbage In - Garbage Out' is most applicable to legacy system migration.

In addition, interactions between the legacy system and other information systems and resources must be identified. Failure to do so can result in expensive failures in related systems when a legacy system is migrated [11]. This interaction is often unclear from the code or documentation, adding another source of complexity to this phase.

Much of the existing research into the area of legacy system migration has focused on this understanding phase ([12], [17], [31], [33], [45], [64]). Although there are many commercial tools which claim to automate tasks in this phase (see section 3.2), most are specific to particular types of application and require a particular code or data structure and all require extensive input from the migration engineer.

2.3. Target System Development

Once the legacy system is understood a requirements specification can be prepared. A target system developed according to this specification will have the same functionality as the legacy system. Decisions have to be made with regard to the architecture should be chosen for the target system. This is a crucial stage of any migration project, the target environment chosen must support the target application requirements [43].

Most current systems are developed using a 3-tiered client server architecture ([36], [13]). A primary design intention of these architectures is to facilitate maintenance and extension in the future. Instead of developing single monolithic applications developers are encouraged to separate the interface, business logic and data components of an application; these represent the three tiers of 3-tier client/server computing (see section 3.3.1). Even within tiers autonomous components are identified and separated out. A common Interface Definition Language (IDL) can be used to describe the interfaces to the individually constructed components. Communication between components is only possible through these predefined interfaces and components do not need to be aware of how other components are implemented. This arrangement allows components to be modified and extended without affecting the operation of other components. This extends the idea of data independence, which is well known in the database world, to the world of component applications.

Frameworks, such as CORBA, OLE/COM and DCE ([36],[37]), exist to facilitate the seamless distribution of components. Services within these frameworks allow components to discover each other and interoperate across networks. The traditional interoperability barriers of heterogeneous languages, operating systems, and networks are effectively overcome by this component based distributed framework architecture. An overview of these technologies and their relevance to systems re-engineering and migration can be found in [43]. There can be no guarantees that applications developed using this architecture will never become tomorrow's legacy systems. However, by choosing the most appropriate architecture and methods, target applications that facilitate change can be developed.

2.4. Testing

Testing is an ongoing process throughout the migration of a legacy system. Up to eighty percent of a migration engineer's time could quite legitimately be spent testing [18]. Due to the high risk involved in any migration project, increased when the legacy system is mission critical, it is imperative that there are no

inconsistencies between the output of a legacy system and that of its replacement system. By keeping the same functionality, a direct comparison of outputs is sufficient to determine the validity of the target system. Sommerville has termed this process *Back-to-back* testing [65], and is shown in Fig. 1. Basically, the

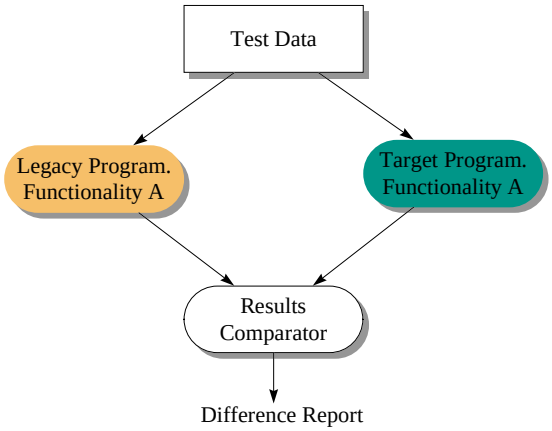

Fig. 1. Back-to-back Testing Strategy

existence of more than one version of a system, as is the case for legacy system migration, is exploited for testing. Selected test data is presented to both versions of the system, and differences in the outputs probably indicates the existence of software malfunctions. In the case of legacy migration, where the outputs of the target implementation do not correspond with its legacy equivalent, the target system needs to be carefully investigated.

The idea of back-to-back testing has also been reported and successfully implemented by Beizer [5]. Fig. 2 represents his framework for testing rehosted software, which can in fact be seen as a refinement of Fig. 1. An *adequate test suite* [5] is selected for testing the legacy software, and the outcomes of this test suite

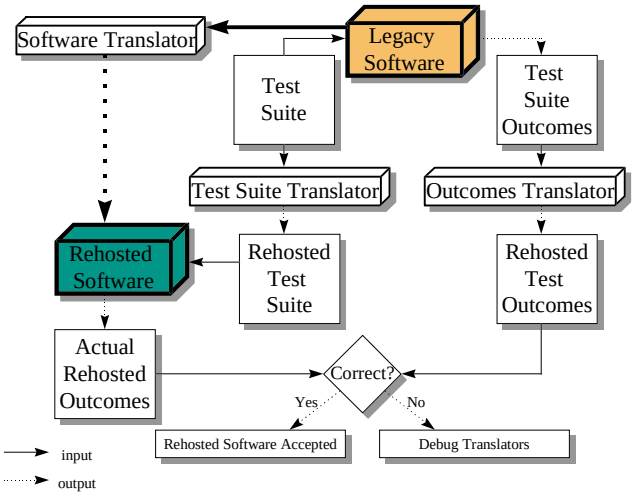

Fig. 2. Testing Framework for Rehosted Software

are recorded. Ideally, this test data should be that which was used when developing the legacy software, although this information would be rarely available. Following this framework, the legacy software is translated into the target environment. The test suite is also translated so that it can be used as input for the new rehosted software. Finally the test suite outcomes are translated as well, in order compare them with the outcomes of the rehosted software, named Actual Rehosted Outcomes in Fig. 2. If differences between actual and rehosted outcomes are detected, the translators (of software, test suite, or outcomes) must be

debugged, otherwise the software will be considered as being successfully rehosted.

It must be noted that this framework assumes that an adequate test suite for the legacy software will became an adequate test suite of the rehosted software (after translation), which is not always true. In fact, this test suite is the minimum test which should be performed against the target system during a migration project, in order to test its functionality. Additional testing may be required depending on the specific target environment. For example, additional performance testing will be imperative when migrating from a mainframe based to a client and server environment.

The back-to-back testing strategy works under the premise that both versions of the software being tested implement the same functionality. For this reason it is not advisable to introduce new functionality to the target system as part of the migration project ([11], [5]). However, in reality it is likely that in order to justify the expense of a migration project, the target system will be required to offer new functionality. In this case, the legacy system should be migrated, without any enhancement, first. New functionality can be incorporated into the target system after the initial migration has been performed.

2.5. Migration

The migration phase is concerned with the cut over from the legacy system to the target system. Dealing with mission-critical legacy systems means that this cut over must cause as little disruption to the business environment as possible. Therefore, a naive approach of simply switching off a legacy system and turning on a new feature-rich replacement is, in many cases, not a realistic option [11]. Also, cutting over to the target system in one single step, as show Fig. 3(a), represents too high a risk for many organisations as it results in the *whole* information flow would be managed by a system which has never been operational, and thus necessarily untrusted.

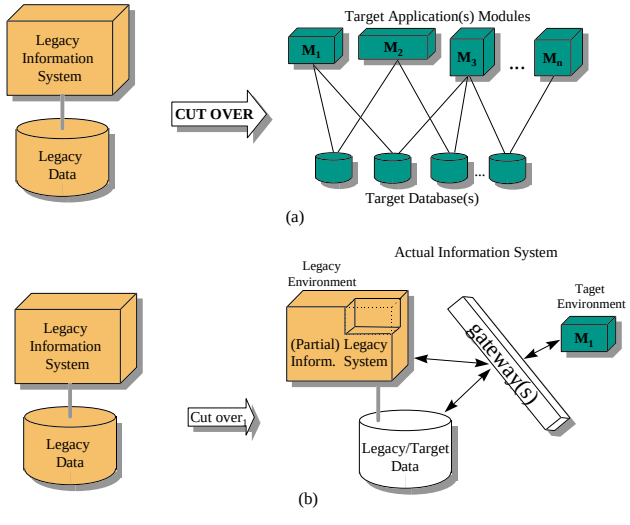

Fig. 3 Cutting Over the Legacy Information System

Ideally, to reduce the risk involved in this phase, the cut over must be performed incrementally, and by small steps. Each step should result in the replacement only a

few legacy components (applications or data) by corresponding target components. The example illustrated in Fig. 3(b) represents a step where only a small part of the legacy functionality has been cut over to the target, the rest remains in the legacy. An additional module has been introduced, termed gateway, to integrate both the target and the legacy environments, which together form the actual information system the organisation will use during migration. Further steps will be required to migrate more functionality from the legacy to the target systems. When the complete legacy system has been migrated to the target environment, a gateway will be no longer required.

The incremental migration described above is a potentially highly complex process. For this method to be successful, it must be possible to split the legacy applications in functionally separate modules. However, this is not always possible for legacy systems, the vast majority of which are badly structured. The same problem arises if the legacy data is also incrementally migrated. It could be difficult, if not impossible, to find out which portions of the data can be migrated independently. In addition, the management of the whole process would not be an easy task.

The construction of the gateway could also be extremely difficult as it may involve dealing with heterogeneous environments, distributed applications and distributed databases. Each of these fields is still an open research area issue and results may not be mature enough to be used in a mission-critical migration.

Migrating a legacy system in an incremental fashion is designed to reduce the risk of the migration phase. However, its inherent complexity carries with it a high risk why may actually result in increasing the risk involved in migration. These two sources of risk must be balanced if the migration phase is to succeed. This phase is central to every migration project and much research is still required in this area

Section 4 analyses different approaches to migration. Each one aims to meet a different trade-off between the risk introduced by its complexity (section 4.5) and the risk introduced if the migration is performed in a non-incremental fashion (section 4.1).

3. Legacy System Migration Tool Support

Migrating a legacy information system is a long, high-risk process, typically lasting five to ten years [11]. Migration tools can considerably reduce the duration of a migration project, helping the migration engineer in tedious, time-consuming and error-prone tasks. This section is not intended to be a catalogue of all tools, or types of tools, available, rather it presents some examples of the tools currently available to assist in the migration process, and illustrates what kind of support migration engineers can expect from such tools.

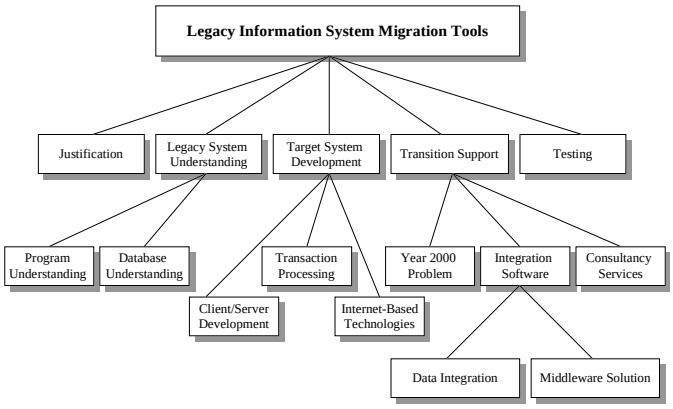

Fig. 4. Migration Tools Hierarchy

Following the migration issues discussed in section 2, each tool is classified according to which phase of the process it is intended to support, as shown in Fig. 4. Section 3.1 outlines some tools that help to justify the migration of a legacy system. Tools analysed in section 3.2 help provide a better understanding of a legacy system. Section 3.3 describes tools which provide automated support for engineers building open systems, the preferred target systems for legacy migration. Section 3.4 lists a miscellaneous set of services and tools which include partial solutions to migration, software to support enterprise wide applications and consultancy companies concerned with managing migration projects. Section 3.5 relates a migration project to the testing needed in any software engineering project, and lists some tools which can help in this process.

3.1. Justification Tools

The general justification process for migration projects can be related to the planning phase of any application development. CASE tools and software metrics [14] are thus readily available to support this process.

The RENAISSANCE project [42] started in 1996, plans to develop methods for assessing cost, risks and benefits of migration and will itself be supported by existing CASE tools. It will also produce CASE toolkit extensions to assist in reverse engineering the design of system families written in the C programming language.

3.2. Legacy System Understanding Tools

As discussed in section 2.2, understanding the legacy information system is essential to the success of a migration project. A growing number of tools are becoming available to aid migration engineers in the legacy system understanding phase. Although these tools can considerably reduce the amount of time needed to understand a migrating system, the automated understanding of a system's structure is still far from being achieved [9]. Legacy system understanding tools can clearly be subdivided into those which analyse the application's code and those that analyse the data structure. The following two subsections are based on this classification.

A more detailed list of tools to assist in this phase can be found in ([33], [64]).

3.2.1. Program Understanding Tools

The general aim of program understanding tools is to develop mental models of a software system's intended architecture, meaning and behaviour [33]. These models make understanding easier for the engineer.

Legacy systems typically have poor if any documentation (see section 2.2). The research prototype called Rigi ([60], [45]), is an example of a tool which to assist in the reconstruction of system documentation. Rigi claims to parse legacy code and produce useful documentation on how it performs its functionality. Other such tools are offered by companies like Computer Associates, IBM, Compuware, Intersolv, Microfocus, and Bachman. All these companies offer tools to isolate the data information in COBOL applications and help separate the code into more useful and readable segments.

Reasoning Systems offers Software Refinery a reverse engineering tools generator [41], one of the most mature reengineering products known to the authors. Using a grammar of the language to be analysed, this tool creates a parser that builds a high level representation of the structure of the legacy program. The user can manipulate this tree-style representation instead of using the code itself. Software Refinery provides symbolic computations which ease the reengineering process. This company also provides some tools[3] constructed using Software Refinery that support the key tasks when working with legacy systems implemented using specific languages. They help in tasks such as: understanding code structure, analysing the impact of changes, generating documentation, and reengineering.

The DECODE research project [12] aims to develop a Co-operative Program Understanding Environment. The main idea is to use an algorithm to automatically extract part of the program design and then co-operate with the user to improve and extend this design. The user can create a hierarchy of design components. Then it is possible to link operations to components, and code to the operation that it implements. These hierarchies abstract the internal structure of the code and allow the user to navigate through it.

A different approach was followed in [31] to construct COGEN, a knowledge-based system for reengineering a legacy system. It involved restructuring the user interface, re-writing database transactions, and translating language features. The knowledge base is ad-hoc built, and its quality depends on the complexity of the legacy system. However, it is estimated that when

translating general language features the automated conversion was nearly 100%. The goal of this system was not to increase the understanding of the legacy application, but to translate it into the target system in a way as automated as possible. Although this objective is different from the general aim of creating models of the legacy program, this approach should be considered for some migration projects. The main disadvantage of COGEN is that it is highly specific to the legacy system it works with, but at the same time this results in a highly automated process, its strongest feature. The knowledge base is only applicable for one specific legacy environment (DG COBOL) and one specific target environment (IBM's CICS). Thus its lack of generality is obvious, even if some rules could be common or quite similar for other environments. However, as program understanding, translation, etc. are very difficult tasks, high automation levels will only be achieved using heuristic methods and ad-hoc solutions. COGEN exploits techniques from artificial intelligence (AI) and applies them to legacy system understanding.

Finally, a novel approach is proposed in [44] whereby Case Based Reasoning and domain specific ontologies are employed to understand legacy systems and to learn from experience. This is another example of a marriage between AI and legacy understanding.

3.2.2. Database Understanding Tools

Other legacy system understanding tools include those for database reverse engineering. The Bachman Re-Engineering Product Set [3] and the DB-MAIN CASE tool [24] both concentrate on recapturing the semantics of physical database designs. Little support is provided for extracting the data structure but once this is available, both tools provide considerable support in comprehending the structure.

A research prototype called SeeData [2] employs computer graphics to produce very elegant and easy to understand representations of an underlying relational legacy database structure.

Finally, the INCASE Software Code Interviewer (SCI) ([46], [47]) is a static analysis reverse engineering tool that examines an application's source code. SCI aims to discover an application's data model from existing COBOL source code and Job Control Language statements. This software parses the application's code and loads the SCI Acquisition Database (ADB). The ADB represents application components that are needed in the discovery process, as for example COBOL source code, COBOL statements and statements arguments, etc. Once the ADB contains this low-level information,

[3] Refinery/Cobol, Refinery/Ada, Refinery/Fortran, and Refinery/C.

pre-defined and user-defined rules are used to search the database for major entities and attributes of entities which will constitute the data model.

3.3. Target System Development Tools

The main goal in a migration project is to build a fully operative, functionally equivalent system into a target open environment. An essential requirement of legacy system migration is that the newly developed target systems should not become tomorrow's legacy systems. Currently, it is believed that client/server architectures lead to open applications which will not suffer the same deficiencies that current legacy applications (see section 1), making these the desired target architectures. Also, due to the mission critical nature of many legacy systems, transaction support will be a basic requirement for the target, enterprise-wide applications. These applications seek a secure and distributed environment, where many users can access simultaneously and efficiently a diversity of data sources and applications. Finally, giving the growing importance that the World Wide Web has in the way companies do business, the target architecture should facilitate WWW's integration within the enterprise wide information system. The following subsections provide an overview of tools that provide support for developing applications which fulfil these requirements. A more detailed discussion can be found in [43].

3.3.1. Client/Server Development

Client/Server computing is the current development paradigm. It is to this technology that many existing legacy systems will be migrated. This section examines tools available to aid in the development of client/server applications, focusing, in particular, on the support these tools provide for migrating/integrating legacy applications.

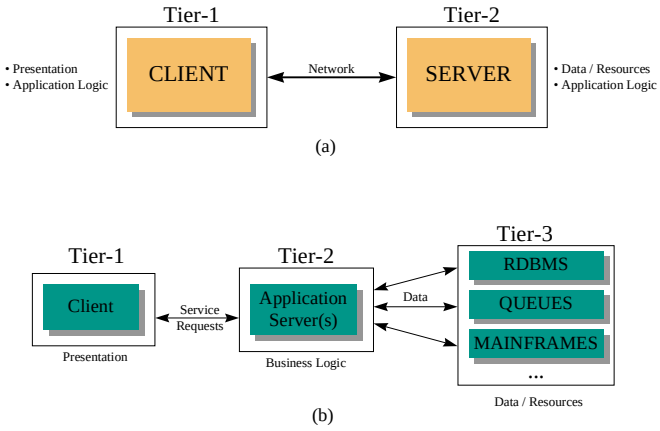

Fig. 5. Two-Tier(a) and Three-Tier(b) Client/Server Architectures

The initial generation of client/server computing was motivated by the proliferation of cheap PC computing power. Users demanded that information systems make use of this cheaper computing power. This resulted in user-friendly interfaces being incorporated into information systems with much more local processing and control, see Fig. 5(a). A primary downside of this

swing from the centralised mainframe to the PC was maintenance problems. Business logic now resided on the PC clients, referred to as "Fat Clients" [36], so any updates or extensions had to be replicated on all clients resulting in enormous versioning and operational headaches. The next generation of client server computing, referred to as 3-tier client/server, aims to eliminate these problems by abstracting the business logic into a third 'middleware' layer. The three layers are thus the interface layer, the application logic layer, and the resource servers layer [36], see Fig. 5(b).

First generation (or two-tier) client server application development features two types of tool :

- Graphical tools that focus on client side database access. These automate database access but can lead to the "Fat Client" syndrome. Examples of these tools include PowerSoft's PowerBuilder, Semantics Enterprise Developer, and Compare's Uniface.
- Tools that combine visual screen builders with traditional third generation languages. These include Microsoft's Visual Basic and Visual C++, Borland's Delphi, etc.

The greatest and most cost effective benefit of both types of tool lies in developing small, uncomplicated applications that do not require broad deployment and high volumes of users. Unfortunately, the typical legacy system is by definition complex, used by large numbers of users, and deployed across the entire enterprise. Besides this, first generation tools are not developed to handle complex development functions like transaction management, locking models, and multiple database access. These functions are central to the operation of any legacy system. Situations where first generation tools find application in the migration process can be seen in the area of "screen scraping", i.e. the replacement of character based legacy system front ends with graphical user interfaces (see section 4.7).

3-tier client/server tools address application scalability for high volume transaction support and expanding volumes of users. The main players in this arena are Visions Unify product, Forté from Forté Software Inc., and Dynasty from Dynasty Technologies. The main feature that distinguishes these products from others is their automatic partitioning mechanism. This automatically analyses an application and decides which tier of the 3-tier client server architecture various application components should reside on. This removes any business functions from the client and ensures bulky unprocessed data is no longer shipped across an overloaded network.

3.3.2. Transaction Processing

Most mission critical systems are data intensive and require substantial transaction processing support. This type of support was traditionally only associated with large scale mainframe based DBMSs. Desktop transaction processing support through triggers and stored procedures is regarded as “TP lite” and not suitable for mission critical enterprise wide applications. However, new support for transactions on the desktop is now provided in the form of Transaction Processing (TP) monitors.

Examples of such tools are Transarc’s Encina [27], Tandem’s Tuxedo TP monitor [54] and NCR’s TOP END [34]. They provide Distributed Transaction Processing (DTP) and also aim to help in some aspects of 3-tiered application development. Other services commonly provided by these kinds of tools are load balancing and application replication management.

3.3.3. Internet Based Technologies for Legacy Systems

The rapid growth in use of Internet and Intranet software in the mid nineties has mainly been due to the prevalence of World Wide Web (WWW) technology. The main motivation for this technology’s deep penetration into the distributed software market has principally been the ability to deliver and render graphics, hyper-text information, audio, video and animation on client machines networked to servers (via HTTP over IP protocol) and providing simple information retrieval mechanisms [8].

Perrochon outlines how WWW technology could be used in accessing legacy information systems [39]. Gateways, built using Common Gateway Interface(CGI) or Server Side Include(SSI) scripts, are used to access the legacy data from web based clients. Using the Web in this way provides a unified interface to legacy systems for a potentially unlimited audience. As with screen scraping (see section 4.7), the issue of legacy migration is not addressed. Instead the emphasis is on providing access to legacy data stored in obsolete formats and locked inside closed systems. The use of CGI and SSI to build these gateways could prove to be quite difficult, especially if any serious data translation/manipulation is required. A more suitable tool for the task might be the Java programming language from Sun, ([22], [23]).

Developed by Sun Microsystems Inc, Java is the name of a specific technology used to create and run active documents and downloadable programs. It consists of the Java programming language (which can be used much as a conventional Object Oriented Language as well as for writing active WWW based documents called applets), Runtime Environment which

provides the facilities needed to run a Java program, and Class library which makes Java applets easier to write [15]. There are several development environments which aid implementation of Java programs namely Sun Microsystem's Java Development Environment (JDE) and Semantec's Café. Java technology has been applied to providing distributed access to databases [26], [50].

WWW products and technologies are not yet widely used in legacy systems migration. However the above approaches are being used to allow distributed access to existing information systems and as such are candidates for either interim/target platform environments on which to migrate systems.

3.4. Transition Support Tools

Legacy system migration is a very rapidly expanding commercial field. For this reason many companies claim to offer 'legacy migration services' although they may not support the complete migration process. The services available can be roughly classified as follows:

- Addressing the Year 2000 problem
- Integrating different software that builds the global information system of a company
- Providing consultancy during a migration project.

This section outlines these kinds of services.

3.4.1. Year 2000 Problem Tools

As the year 2000 approaches many areas of the computer industry face the potential failure of the standard date format: MM/DD/YY. To save storage space in the past, and perhaps reduce the number of keystrokes necessary to enter a year, many IS groups have only allocated two digits to the year. For example, "1996" is stored as "96" in the data files, and "2000" will be stored as "00". This two-digit date affects data manipulation, primarily subtractions and comparisons.

Many software systems may either crash or produce garbage with the advent of the year 2000 [32]. The cost of modifying these legacy systems is enormous, the whole problem relates back to the undocumented nature of legacy information systems, discussed in section 2.2. The simple act of determining how many systems in an organisation will be affected can easily require many man months of effort alone. The deadline for commencing reengineering projects to deal with the year 2000 problem is thus fast approaching for many large organisations. Unlike other motivating factors for system reengineering such as the requirement for Business Process Reengineering, or faster processing, there are no doubts about the consequences of inaction for the year 2000 problem.

In response to the Year 2000 problem a large number of companies have produced products to assist. An

extense and very comprehensive guide of tools which assist on solving this problem can be found in [63].

3.4.2. Integration Software

Some companies focus more on integrating legacy systems with newer technology applications rather than actually migrating legacy systems to new architectures. Not all the integration processes have the same purpose. Some set out to integrate their data sources, or incorporate new data management technology. A more complex approach is to create a framework where all applications in the company, legacy and newly developed, are mutually accessible. These options are analysed in the next two subsections.

3.4.2.1. Data Integration

This section outlines some tools focused on integrating data sources or incorporating new management technology.

Acucobol is a company which concentrates on bringing COBOL applications into the 'Open-Systems' market. One of its products, Acu4GL [1], claims to provide a seamless interface from COBOL to relational database management systems (RDBMS). This product executes COBOL Input/Output (I/O) operations by automatically generating SQL statements, so that applications do not have to be modified and users do not need to learn SQL. This is possible because all Acucobol I/O passes through a generic file handler, and whenever it encounters an input or output to a file that must be managed by a RDBMS, the request is passed to the Acu4GL interface which in turn accesses the database.

Persistence Software [40], in contrast, does not provide access to new data management but aims to interface object-oriented developed applications with relational database management systems, i.e. new applications accessing legacy data. The development of an application with Persistence Software starts by specifying the applications' data model. Persistence then generates database objects which manage the mapping to relational tables. Another component of Persistence provides an object-oriented interface between application objects and relational data. Finally, a different component is concerned with retrieving the data, ensuring integrity and optimising performance.

UniData [57] was originally a relational database vendor. It has now branched out into the area of legacy system migration. The angle of its particular relational RDBMS is in the storage, retrieval, and manipulation of nested data and repeating groups as well as traditional data. This feature greatly reduces the complexity of migrating data from network and hierarchical databases to a relational database format. This provided UniData with an instant advantage in the fast growing world of

legacy data migration. They subsequently developed a suite of tools to address the larger problem of legacy system migration, such as a screen scraper (see section 4.7), a COBOL code analyser to extract application's data model, and a fourth generation language for developing client/server applications. UniData concentrates on providing a relational database solution to legacy systems. It addresses the requirements of those legacy systems whose information needs to be leveraged on the desktop but who are not themselves in urgent need of migration.

Apertus' Enterprise/Integrator [20] is an example of a tool designed to integrate the different data sources of an enterprise. It offers a rich set of features that address the full life cycle of the data integration process. Some notable facilities provided by Enterprise/Integrator include support for both the detection of redundant data and what is known as value conflict resolution. Rules can be defined by the user to identify logically redundant data. These rules might inspect combination of attribute values to determine if two objects are logically equivalent (redundant data). The need for value conflict resolution arises because it is not uncommon that information representing the same real world entity, coming from different sources, have conflicting values. A separate set of rules, Property Value Conflict Resolution (PVCR) rules, aim to solve these kinds of conflicts, deciding which value to use to populate the integrated data store.

3.4.2.2. Middleware Solution

In the past, different departments in a company have built their information systems independently of each other. There is a need to integrate these disparate information systems, to preserve the investment in legacy systems, and to incorporate new technology. This situation has led many companies to develop products to support integration of heterogeneous environments that will result in what is called *Enterprise-wide Information Systems*.

Open Horizon's Connection [35] is an example of this kind of product. It could best be described as a proprietary Object Request Broker (ORB) [36]. The particular goal of Connection is to provide a secure connection for any end user application to the database server tier and the application server tier in a 3-tier client server application. The product provides many of the facilities proposed by CORBA, a distributed communication infrastructure, security services, directory services, and dynamic binding. It achieves most of this by being built on top of the OSF's DCE. In addition, OpenHorizon allows for a single sign on so that users only have to present their credentials once

and are then free to access all database servers on the network.

Enterprise/Access [19] is another such product. It is a middleware tool that enables the deployment of second generation (see section 3.3.1), enterprise-strength client/server applications and provides a controlled, cost-effective migration path off legacy systems.

3.4.3. Consultancy Services

This section outlines the services provided by some companies that provide support for the whole process of migrating a legacy information system.

I-Cube [28] is a consultancy firm which provides advice and contracting services for managing a migration project. No specific migration tools are produced by I-Cube.

LexiBridge [30] in contrast provides a toolkit for the migration process. The toolkit is made up of three components : the Workbench, the Repository, and the Databridge. The Workbench divides the legacy system into four layers: the User Interface, the Process Model, the Data model, and the Physical Model. PowerBuilder, refer to section 3.3.1, windows are automatically generated to replace the character based user interface, (effectively an internal screen scraper). LexiBridge relies on the use of triggers and stored procedures to replace the original legacy I/O. Legacy data is converted into Sybase, Oracle, or DB2 formats. Scrubbing mechanisms are provided for cleaning data to eliminate inconsistencies and redundancies. Finally a code optimiser is used to eliminate dead code, restructure weak logic, and rescope program variables.

Sector 7 [52] has developed a methodology for migration based on five steps: assessment, planning, porting, validation, and productization. It specialises in VMS to Unix and NT system migration. Sector 7 also offers a wide range of tools to support most stages of a migration project. Sector 7's methodology is too general to be used in any particular migration project.

3.5. Testing Tools

As mentioned in section 2.4, it is important that functionality is not added to the target system as part of the migration project [11]. If the functionality does not change, commercial tools can be used to automatically produce test environments to systematically validate the correctness of the target system. The general testing process of a migration project can be related to the testing phase of any software engineering project. Tools are thus readily available when testing a migration process.

Sector 7 and Performance Software [53] are examples of companies that provide support for automated testing. Another company, Cyrano [16], offers a set of products, known collectively as Cyrano

Suite, specialised in testing client/server based applications.

3.6. Limitations of Migration Support Tools

From the previous sections, it is clear that there is no complete solution or even agreed approach to the migration problem. Those commercial/research products that are available tend to address the issue from a number of varying angles. Some of them focus on a very narrow area of the process (database or code understanding, integration, etc.), and require a high level of user involvement.

Others address the overall legacy migration issue, but they offer a too general methodology which does not address the specifics of particular migration projects.

Another limitation of the majority of commercial solutions is that they focus almost exclusively on legacy systems written in COBOL. The most widely supported platform is IBM running MVS and using the IMS hierarchical database management system. A large number of legacy systems in the computer industry as a whole do operate with this configuration. However, there are still a lot of legacy systems written in other languages, (Fortran, C, Plex), running on different platforms, (Digital, Bull, Data General),and using different database management systems, (standard file, network, relational). The migration or integration of these systems is largely unsupported.

4. Approaches to Migration

Given the scale, complexity and risk of failure of legacy system migration projects, it is clear that a well-defined, detailed approach that can be easily implemented is essential to their success. In this section currently available legacy system migration approaches are discussed. Although legacy information system migration is a major research issue, there are few comprehensive migration methodologies available. Those that are documented are either so general that they omit many of the specifics or, they are centred around particular tools and are so specific to a particular phase that users might be in danger of overlooking other significant phases. This section presents 6 of those currently available :

- Big Bang Approach
- Database First Approach
- Database Last Approach
- Composite Database Approach
- Chicken Little Strategy
- Butterfly Methodology

Each approach has its advantages and disadvantages and some are more suitable for use in one particular migration project than in another. When considering

legacy system migration, an intensive study needs to be undertaken to find the most appropriate method of solving the problems it poses.

Section 4.7 briefly discusses a common approach to 'migration' adopted in industry.

A summary of how each of these approaches fulfil the areas a migration project must address (see section 2) is presented in Appendix I.

4.1. The Big Bang Approach

The Big Bang approach [4], also referred to as the Cold Turkey Strategy [10], involves redeveloping a legacy system from scratch using a modern architecture, tools and databases, running on a new hardware platform. For any reasonably sized system it is clear that this is a huge undertaking. In reality, the risk of failure is usually too great for this approach to be seriously contemplated. In order to justify adopting this approach, it is usually necessary to guarantee that the redeveloped system will include not only all the functionality provided by the original legacy system but also many new additional features. This adds greatly to the complexity of the migration and further increases the risk of failure.

Before migration can start, it is necessary to understand the legacy system fully. Often documentation for the legacy system at worst does not exist or, at best is incomplete or is out of date. Thus the functionality must be extracted from the code and the underlying data structures and understood before redevelopment begins. This adds both to the duration and the complexity of the project and can greatly increase the risk of failure if this process is flawed. The situation is further complicated by the fact that legacy systems do not generally operate in isolation. They often interact with other legacy systems and resources. In practice this interaction is often not clear from either the code or documentation. A decision to redevelop one legacy system from scratch could trigger failures in other dependent information systems.

Apart from the failure risks, another very real concern arises from the constantly changing technology and business requirements. Any project of this scale could take several years to complete. While the legacy system redevelopment proceeds, technology will continue to evolve and, more significantly, an organisation's business focus could change. Thus organisations could find themselves in a position where the redeveloped system no longer meets their business needs and the technology used is already out-of-date before it ever becomes operational.

It seems clear that it is not advisable to use the Big Bang approach for all legacy system migrations. However, where legacy systems have a well defined,

stable functionality, are not mission critical and are relatively small in size this approach could be used.

4.2. The Database First Approach

The Database First approach [4], also called the Forward Migration Method [10], involves the initial migration of legacy data to a modern, probably relational, Database Management System (DBMS) and then incrementally migrating the legacy applications and interfaces.

While legacy applications and interfaces are being redeveloped, the legacy system remains operable. This methodology falls within a group of methodologies which allow for the interoperability between both the legacy and target systems (sections 2.2 to 2.5). This interoperability is provided by a module known as Gateway: a software module introduced between components to mediate between them [11]. Gateways can play several roles in migration, insulating certain components from changes being made to others, translating requests and data between components or co-ordinating queries and updates between components..

The concrete gateway used by the Database First approach is called Forward Gateway. It enables the legacy applications to access the database environment in the target side of the migration process, as shown in Fig. 6. This gateway translates and redirects these calls forward to the new database service. Results returned by the new database service are similarly translated for used by legacy applications.

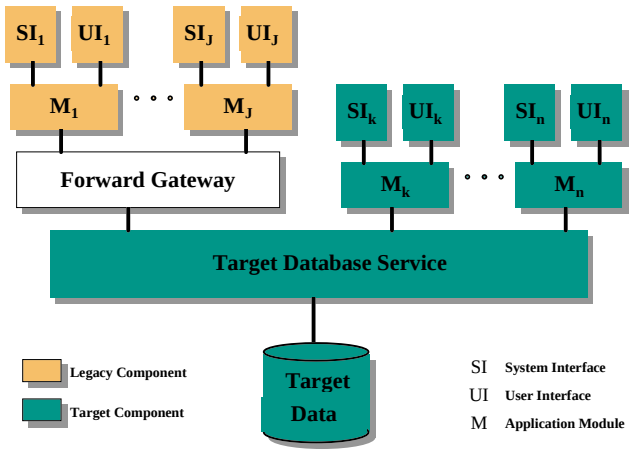

Fig. 6. Database First Approach

The main advantage of this approach is that once the legacy data has been migrated, the latest fourth generation language and reporting tools can be used to access the data providing immediate productivity benefits. The legacy system can remain operational while legacy applications and interfaces are rebuilt and migrated to the target system one-by-one. When the migration is complete, the gateway will no longer be required and can be decommissioned as the old legacy system is shut down.

There are several disadvantages to this approach, in particular, it is only applicable to fully decomposable

legacy systems[4] where a clean interface to the legacy database service exists. Also, before migration can start, the new database structure must be defined. The major risk with this activity is that the structure of the legacy database may adversely influence the structure of the new database. The Forward Gateway employed can be very difficult, and sometimes even impossible, to construct due to the differences between the source and the target in technology, in database structure, constraints etc..

Overall this is a rather simplistic approach to legacy system migration. The migration of the legacy data may take a significant amount of time during which the legacy system will be inaccessible. When dealing with mission critical information systems this may be unacceptable.

An enhancement to this method is proposed by Menhoudj and Ou-Halima [67]. Using this method, migration is carried out through several small migration steps. At each step one or more files are migrated, following a predefined order. An application module is migrated only when all the files it accesses have already been migrated. Therefore, there will never be a module in the target system which needs to access a file stored in the legacy system. Modules in the legacy system may access files (tables, once migrated) in the target system, thus a forward database gateway is required.

An example of this migration is shown in Fig. 7. Fig. 7(a) represents the set of modules (M_i) and data files (F_i) that constitute the legacy system. The assumption is that the migration sequence defined by the method states that firstly F_1 must be migrated, then F_2, and finally F_3. Once file F_1 has been migrated into table T_1, module M_1 only accesses data stored in the target system, thus it must be also migrated. This situation is shown in Fig. 7(b). Then file F_2 is migrated, as shown in Fig. 7(c), after that module M_2 is also migrated, Fig. 7(d).

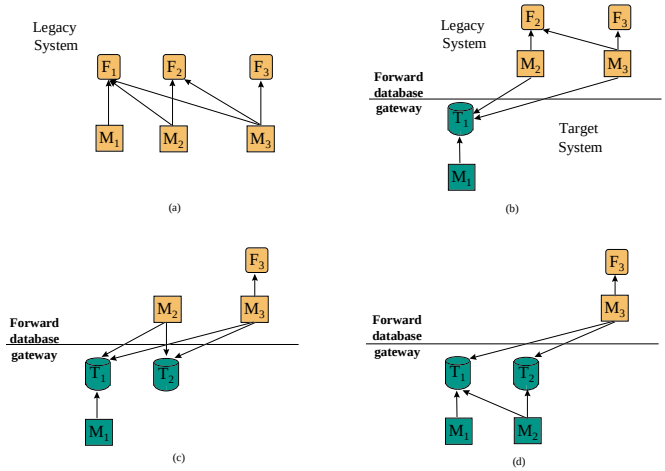

Fig. 7. Intermediate Steps of a Migration Process

[4] A fully decomposable IS is one where applications, interfaces and databases are considered to be distinct components with clearly defined interfaces. Applications must be independent of each other and interact only with the database service.[Brod95]

The key of the method relies on the way in which the migration sequence is determined. It is based on the principle of minimizing the changes required to the legacy modules when some files or modules are migrated. The main idea is to analyse the interdependencies between legacy data files, which leads to a partial order between these files. Due to space limitation is not possible to detail the whole process, for more details refer to [67].

The method could be thought as being excessively simplistic, for example, the assumptions made regarding the type of legacy system target systems, (file based and relational based, respectively), and the possibility of clearly uncovering the set of interdependencies between data files, do not always hold when facing a generic legacy problem. The method, however, offers a most important contribution to the area of legacy system migration in that it addresses the problem from a very practical (and necessarily specific) point of view. Most of the available methodologies for legacy migration are defined at a very abstract level, and do not address many of the practical problems to face when an actual migration is being implemented.

4.3. The Database Last Approach

The Database Last approach [4], also called the Reverse Migration Method [10], is based on a similar concept to the Database First approach and is also suitable only for fully decomposable legacy systems. Legacy applications are gradually migrated to the target platform while the legacy database remains on the original platform. The legacy database migration is the final step of the migration process. As with the Database First approach, a gateway is used to allow for the interoperability of both information systems. In this case a Reverse Gateway enables target applications to access the legacy data management environment. It is employed to convert calls from the newly created applications and redirect them to the legacy database service, as shown in Fig. 8.

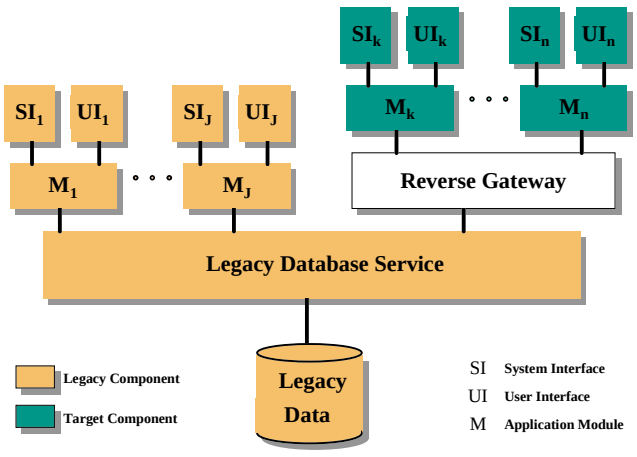

Fig. 8. Database Last Migration Approach

If the legacy database service is to be migrated to a relational database management service, then the target applications will be developed completely with SQL calls to the data service. It is these calls that are captured by the Reverse Gateway and converted to the equivalent legacy calls. The Database Last approach has a lot in common with the client/server paradigm. The legacy database takes on the role of a database server with the target applications operating as clients. There are commercial products available which effectively act as reverse gateways, e.g. Sybase's OmniConnect and DirectConnect products.

The Database Last approach is not without its problems however. Performance issues can be raised with regard to the gateway. The Reverse Gateway will be responsible for mapping the target database schema to the legacy database. This mapping can be complex and slow which will affect the new applications. Also many of the complex features found in relational databases (integrity, consistency constraints, triggers etc.), may not be found in the archaic legacy database, and hence cannot be exploited by the new application.

This approach is probably more commercially acceptable than the Database First approach as legacy applications can continue to operate normally while being redeveloped. However, the migration of the legacy data will still require that the legacy system be inaccessible for a significant amount of time. When dealing with mission critical information systems, this may be unacceptable.

4.4. The Composite Database Approach

The Composite Database approach outlined in [10] is applicable to fully decomposable, semi-decomposable[5] and non-decomposable[6] legacy systems. In reality, few legacy systems fit easily into a single category. Most legacy systems have some decomposable components, some which are semi-decomposable and others which are non-decomposable, i.e. what is known as a Hybrid Information System architecture.

In Composite Database approach, the legacy information system and its target information system are operated in parallel throughout the migration project. The target applications are gradually rebuilt on the target platform using modern tools and technology. Initially the target system will be quite small but will grow as the migration progresses. Eventually the target system should perform all the functionality of the legacy system and the old legacy system can be retired.

During the migration, the old legacy system and its target system form a composite information system, as

[5] A semi-decomposable IS is one where only the user and system interfaces are separate components. The applications and database service are not separable.[Brod95]

[6] A non-decomposable IS is one where no functional components are separable.[Brod95]

shown in Fig. 9 (modified from [10]), employing a combination of forward and reverse gateways. The approach may involve data being duplicated across both the legacy database and the target database. To maintain data integrity, a Transaction Co-ordinator is employed which intercepts all update requests, from legacy or target applications, and processes them to identify whether they refer to data replicated in both databases. If they do, the update is propagated to both databases using a two-phase commit protocol as for distributed database systems [6].

Analysing non-decomposable legacy components can be very difficult. In the worst case the component must be treated as a black box. The best that can be achieved is to discover its functionality and try to elicit as much legacy data as possible. Sometimes using existing legacy applications, (e.g., database query, report generation, and access routines), is the only way to extract the legacy data. Once the functionality has been ascertained, the component can be re-developed from scratch. It can often be very difficult to identify when legacy data or functions are independent; in many cases they may simply have to be replicated and target copies co-ordinated until the entire legacy system can be safely retired.

The Composite Database approach eliminates the need for a single large migration of legacy data as

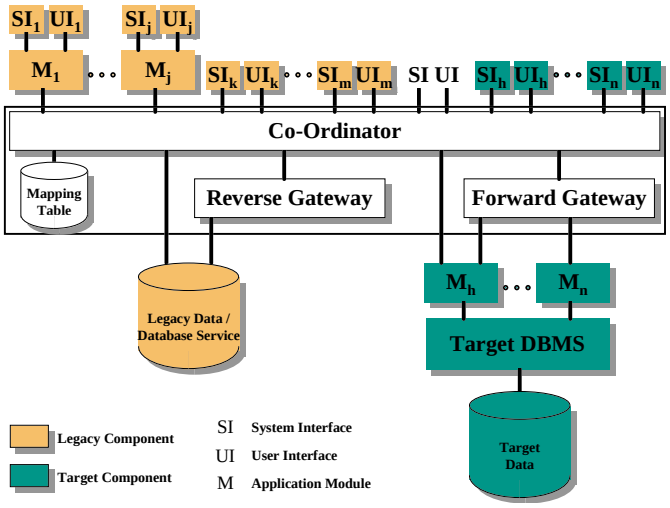

Fig. 9. Composite Database Migration Approach

required in the Database First and Database Last approaches. This is significant in a mission critical environment. However, this approach suffers from the overhead not only of the other two approaches but also the added complexity introduced by the co-ordinator.

4.5. The Chicken-Little Strategy

The Chicken Little strategy outlined in [11] is a refinement of the Composite Database approach. Chicken Little proposes migration solutions for fully-, semi- and non-decomposable legacy systems by using a variety of gateways. The difference between these kinds of gateways relies on where in the system they are placed, and on the amount of functionality they provide. All of them have the same goal, however, i.e. to mediate between operation software components, as said in section 4.2. The types of gateways referred to in

sections 4.2 and 4.3 will here be called *database gateways* to distinguish them from the others.

For a fully decomposable legacy system a *database gateway*, either forward or reverse, is used and is positioned between the application modules and the database service. An *application gateway* is used for a semi-decomposable legacy information systems. This gateway takes the form of the gateway illustrated in Fig. 9 and is positioned between the separable user and system interfaces and the legacy database service. For non-decomposable systems an *information system gateway* is positioned between the end-user and other information systems and the legacy information system. This gateway also takes the form shown in Fig. 9, but it is expected to be much more complex than an application gateway. An information system gateway has to encapsulate the whole functionality of the legacy system, while an application gateway encapsulates only from application modules down.

As well as database, application, and information system gateways, the concept of an *interface gateway*, as shown in Fig. 10, is also proposed for non-decomposable legacy systems. The idea is to insulate end users from all underlying processes in the migration. The interface gateway captures user and system interface calls to some applications and redirects them to others. It also accepts the corresponding responses and translates, integrates and directs them to the calling interface.

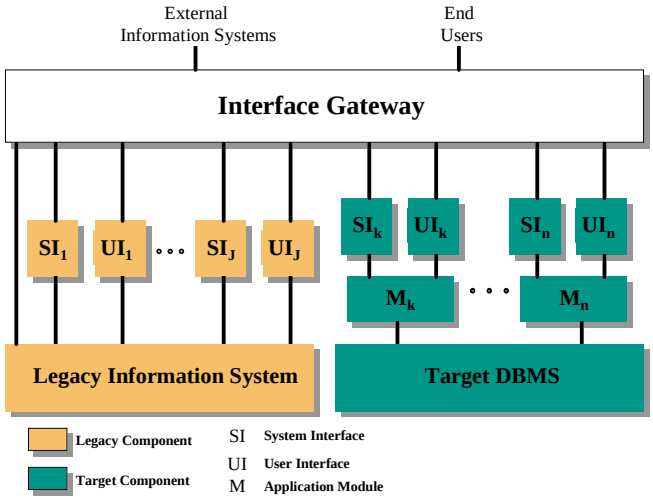

Fig. 10. Interface Gateway

Chicken Little also proposes an 11 step plan to be followed in any migration project, shown in Fig. 11. Each step handles a particular aspect of migration, e.g. migrating the database or migrating the application. The method can be adapted to fit individual legacy systems

Step 1 : Incrementally analyse the legacy information system
Step 2 : Incrementally decompose the legacy information system structure
Step 3 : Incrementally design the target Interfaces
Step 4 : Incrementally design the target applications
Step 5 : Incrementally design the target database
Step 6 : Incrementally install the target environment
Step 7 : Incrementally create and install the necessary gateways
Step 8 : Incrementally migrate the legacy databases
Step 9 : Incrementally migrate the legacy applications
Step 10 : Incrementally migrate the legacy interfaces
Step 11 : Incrementally cut over to the target information system.

Fig. 11. Chicken Little Migration Steps

migration requirements. Steps do not have to be performed in sequence and several steps can be performed in parallel.

The method is designed to be incremental i.e. the legacy system(s) are migrated to the target system(s) one component at a time. Gateways are then used to allow the legacy and target systems to interoperate. Using *Chicken Little* data is stored in both the migrating legacy and the growing target system. In most cases, gateway co-ordinators have to be introduced to maintain data consistency. As Brodie and Stonebraker themselves point out "update consistency across heterogeneous information systems is a much more complex technical problem with no general solution yet advised, and it is still an open research challenge" [11]. Thus it seems that to apply *Chicken Little* approach would be a big challenge to any migration engineer. In addition, the strategy does not include a testing-step (see section 2.4), which is clearly essential and a vital part of the process before cutting on the target information system.

4.6. The Butterfly Methodology

The Butterfly Methodology is based on the assumption that the data of a legacy system is logically the most important part of the system and that, from the viewpoint of the target system development it is not the ever-changing legacy data that is crucial, but rather its semantics or schema(s). Thus, the Butterfly Methodology separates the target system development and data migration phases, thereby eliminating the need for gateways.

Using the Butterfly Methodology, when the legacy data migration begins, the legacy datastore is frozen to become a read-only store. All manipulations on the legacy data are redirected by a module called the *Data-Access-Allocator (DAA)*, see Fig. 12. The results of these manipulations are stored in a series of auxiliary datastores: *TempStores (TS)*. The *DAA* effectively stores the results of manipulations in the latest TempStore.

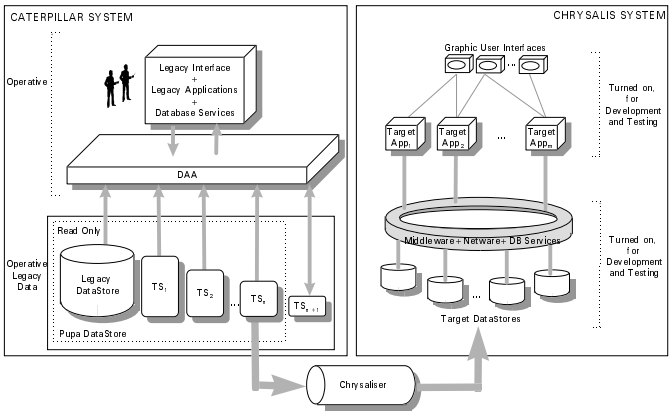

Fig. 12. Butterfly Methodology, Migrating the Data in TempStore TSn

When legacy applications access data, DAA retrieves data from the correct source, e.g. the legacy datastore or the correct TempStore.

A Data-Transformer, named Chrysaliser in Fig. 12, is employed to migrate, in turn, the data in the legacy system as well as in the TempStores to the target system. While Chrysaliser is migrating the legacy datastore all manipulations are stored in the first TempStore (TS_1); when migrating data in TS_1, all manipulations are stored in TS_2; and so on.

If the time taken to migrate a TempStore is faster than that taken to build the next one, the size of each TempStore will decrease at each iteration. When the amount of data in TS_n is sufficiently small, the legacy system can be brought down and the data in the last TempStore migrated to the target system, without causing any serious inconvenience to the core business. This will result in an up-to-date target database and the target system can be made operative. Thus using the Butterfly Methodology, at no time during the migration process will the legacy system by inaccessible for a significant amount of time.

Fig. 12 shows a scenario during the legacy data migration. It can be seen that the combination of the DAA and Chrysaliser serve as a legacy data migration engine.

Using the Butterfly methodology, Target SampleData, which is based upon the target system data model, is stored in a Sample DataStore. Target SampleData is transformed from Legacy SampleData, a representative subset of the data in the legacy data store. The Sample DataStore is employed to support the initial development and testing of all components (except for data) of the target system.

The Butterfly methodology is a novel approach to handling legacy system migration. Target applications can be exhaustively tested against actual data held in the Sample Datastore. Each step of the Butterfly methodology can be completely tested and the legacy database can be rolled back at any stage. The legacy system can continue to operate as normal throughout the migration until the last TempStore has reached the pre-determined size.

From a pragmatic point of view, the main factor which will determine whether or not this methodology is usable, is the value of $\frac{v}{u}$ (where u is the speed of Chrysaliser transforming the data, and v is the speed of the DAA building up new TempStores). If v = 0 (i.e. the DAA does not build the TempStores), the methodology reverts to a 'Big Bang' migration (see section 4.1). If v > u (i.e. the sizes of the TempStores do not decrease at each iteration), then the migration process will never finish.

Factors relevant to the success of the methodology include:

- a thorough understanding of the legacy and target systems

- an accurate and concise sample datastore
- a fast chrysaliser
- an efficient Data-Access-Allocator.

4.7. Screen Scraping

Few organisations have as yet attempted a full-scale migration project. Many attempt to implement solution which allows them to gain some of the benefits of new technology without having to interrupt their mission-critical legacy system. One particularly popular approach is *Screen Scraping*.

Screen scraping is the process of replacing the character based front end of a legacy information system with a PC client based graphical user interface. Putting a graphical user interface onto a legacy system is a cheap and effective method of leveraging legacy data on the desktop. Users are free to use the graphical data manipulation and input tools common on the desktop to input data and process the system output.

At present there are a large number of products available to perform screen scraping. Some of the better known include MultiSoft's WCL/QFG toolkit, Client Server Technology's GUISys, and Co*STAR from ClearView. These products depend on the use of a terminal emulator for the communications link to the host mainframe legacy system. Most include their own emulators but some rely on third party vendors such as Systems Synchronous Incorporated. The user builds the graphical user interface in a structured interactive session. The processing logic for the host screen is then specified. The resulting PC client application is usually in the form of some automatically generated first generation client server tool, (in nearly all cases either Microsoft's Visual Basic or PowerSoft's PowerBuilder).

Despite the commercial success of screen scraping it is still very much a short term solution. Placing a graphical interface onto a legacy system does not address many of the serious problems faced by legacy systems. Problems of overloading, inability to evolve to provide new functions, and inordinately high maintenance costs are all ignored. Screen scraping simply provides an easy to use interface for the legacy system. At best it reduces training costs for new employees and allows an interface to the legacy system on the desktop. There is no recoding of the legacy system so no new functionality is provided, also all processing still takes place on the mainframe thus not easing the burden on the overloaded mainframe or reducing operational costs by using the cheaper computing power of the desktop.

In many cases screen scraping not only fails to provide an adequate solution but actually serves to compound an organisations maintenance problems. Wrapping legacy system access with screen scraping

software adds a functionally superfluous layer to the legacy system, which itself will have to be maintained in future systems maintenance procedures.

5. Open Research Issues

While some phases of migration, such as legacy system understanding, have been the subject of research for a number of years, others, such as migration, have remained relatively untouched until recently. This section provides a brief outline of some of the major areas in which research is still required. These issues are summarised in Appendix II.

5.1. Legacy System Understanding

This area has been the subject of much research in recent years and has resulted in a number of tools and techniques to assist in both program and data understanding tasks, see section 3.2. However, to a large extent the problems encountered in this phase have remained unsolved and are worthy of further research.

- **Application Understanding**

The majority of research in this area has concentrated on legacy systems written on COBOL. Although this is the language of a lot, perhaps even the majority, of legacy systems, it is by no means the only language used. Tools and techniques for understanding systems written in other languages are required. In addition, as poor programming practice, and documentation, is not limited to legacy systems, research need not be limited to typical legacy languages such as COBOL and FORTRAN, languages such as C, and even C++ and Java, should also be considered. Within this area, the effect of which version of a language was used to implement systems also needs to be investigated . For example, does COBOL 85 need a completely different tool than COBOL 74.

- **Data Understanding**

Research in this area has mainly concentrated on providing tools and techniques to graphically represent and understand database/file schemas, see section 3.2.2. Little research has concentrated on extracting the data structure from existing code and data, one of the most difficult tasks in legacy system migration. Again the majority of research has focused on legacy systems written in COBOL using flat files, relational or hierarchical databases. Many legacy systems use other relational databases for which tools are required. As with application understanding, the effect of which version of a database was used to implement systems also requires investigation.

- **Legacy System Understanding**

Many legacy systems solve common problems. The majority of large organisations would have legacy systems which deal with personnel and payroll issues

and it is logical to assume that there is a lot of common ground in most. Investigating the subject areas of legacy systems could aid understanding. For example, it seems logical all medical administration systems should have a concept of a patient record, patient history etc. Identifying a set of key concepts to search for could be useful in both application and data understanding.

- **AI Support for Understanding**

The majority of understanding tools developed to date, see section 3.2, require extensive input from an expert in the migrating legacy system. To a large extent the available tools automate the understanding of relatively straight-forward aspects of legacy systems, e.g. identifying access to files/tables, identifying separation of functionality in the form of functions or procedures, and rely on the legacy system expert to provide direction and information to overcome more difficult problems, e.g. identifying the full database schema, identifying redundant data. The knowledge of a system expert is crucial. The application of AI research in this area could be most useful. For example, a novel approach, worthy of more serious investigation, has been proposed by Richardson et al [44], in which Case Based Reasoning is used to understand legacy systems and learn from experience of maintaining legacy systems.

5.2. Target System Development

When considering legacy system migration, developing the replacement target system would appear to be one of the simplest tasks. There are, however, significant difficulties which must be overcome, see section 2.3, and require research.

- **Validating Target Functionality**

The new target system must exploit the target architecture to offer new benefits while retaining the tried and trusted functionality of the legacy system. In addition, it must do so as soon as cut-over has occurred. Testing the target system is therefore more crucial for a migration project than for a traditional systems development project. Ensuring that the target system produces the same results as the legacy can be extremely difficult. As with any systems development project errors and omissions may not be found the target system is operational. How to correct such errors is extremely significant in a migration project where, depending on the methodology used, part of mission-critical information system may be operating in the target environment and part in the legacy environment.

Techniques are required to correctly elicit the core legacy system functionality and to validate that this has been incorporated successfully into the target system. The Butterfly methodology, see section 4.6, has proposed using a sample datastore containing a sample

of actual legacy data to test the newly developed target system before it is deployed. Techniques for constructing and ensuring the correctness of this data are required.

- **Enterprise Computing**

In the past, individual departments have developed their computer systems independently. This has been recognised as a mistake and the aim for many organisations is to develop an enterprise-wide computing culture. With this in mind, a migrating system cannot be considered in isolation. While it is migrating, other systems may require access to it. Once it has been migrated, it may require access to systems which are undergoing migration. Investigation is needed into how migrating systems can interoperate with others within the enterprise.

- **Methods for Reuse**

A primary reason for migrating a legacy system rather than simply redeveloping it from scratch is to retain legacy system functionality in the new target system. There are few methods available to identify potential candidates. Similarly, the primary method for reusing a component has been to wrap it in an additional layer of software which allows it to be used in its new environment [51]. Research is required into other techniques for reuse.

5.3. Data Migration

Legacy data is perhaps the most important aspect of any legacy system yet the issues involved in migrating it to a new environment have not been widely considered.

- **Efficient Real-time Data Migration Engine**

As yet there are few approaches for migrating mission-critical legacy data. Allowing the target and legacy systems to interoperate via a gateway, see section 4.2 to 4.5, offers an alternative to migrating legacy data in a one massive step. However, using gateways can result in a complex target architecture in which gateways can never be eliminated. As mentioned in section 4.5, maintaining data consistency across heterogeneous systems is a highly complex technical problem and into which research is required.

The Butterfly methodology, see section 4.6, offers an alternative approach to migrating legacy data using a series of temporary stores to store results of manipulations on legacy data while this is being migrated to the target system. This approach offers an alternative to the bulk of existing research into this area, but it has yet to be tried in practice.

- **Determining Data for Migration**

Deciding exactly what legacy data to migrate is also an area requiring research. For example, one migration project found that 80% of accesses to data in a particular legacy system were read-only [11]. In such a case, it may not be necessary to migrate all the legacy

data at one time. Research into identifying which legacy data is crucial for the initial target system operation could significantly reduce migration timescales. Methods for handling exceptions within the target system, such as requests to unmigrated data, are also required.

- **Dirty Data**

The condition of the legacy data is also an important issue. Not all legacy data can be classified as good quality data [38]. Methods for deciding what data is dirty and for cleaning legacy data prior to, or after, migration are required.

5.4. Migration Tools

As discussed in section 3, numerous tools to assist in the migration process have already been developed. There is however, the potential for more research into this area.

- **General Tool-kit Framework**

The majority of tools which could be used in a migration project are developed in isolation and are not generally designed to co-operate. Developing tool-kits or support environments for software development projects has been an area of research for a number of years. Investigating the support a migration project requires and existing research into software development environments could be applied to migration would be useful.

- **Individual Migration Tool Development**

Although numerous tools have already been developed, as has already been discussed in this section and section 3, there are still many areas in which tools are required. In particular the areas of target system development, testing and migration have been largely unsupported to date.

5.5. Migration Approaches

To date few migration approaches have been proposed, see section 4. Those that have been proposed have few, if any, practical results to support them. A safe, comprehensive migration approach is a necessity.

- **General Migration Approach**

The few approaches that have been developed, see section 4, have adopted widely varying solutions to the problems of migration. It is unlikely that a single migration approach, be suitable for every possible type of legacy system will emerge. The migration requirements of all types of information system, legacy or otherwise, need to be investigated and approaches developed. Some aspects of migration, such as developing and testing the target system, the sequence in which tasks should be performed, should be sufficiently similar for all projects to allow a model of migration, such as that proposed in section 2, to be developed.

- **Refinement of Current Approaches**

Existing approaches have generally been presented at a very high level. Each proposed step encompasses many important tasks. For example, The Chicken Little approach includes a step "Incrementally design the target database". This is clearly not a simple task and needs serious investigation.

5.6. General Migration Issues

The previous sections have presented issues which clearly fall within particular areas. This section outlines some more general issues for migration.

- **Managing Migration**

The management of software development projects has long been the subject of research. A migration project is in many respects similar to a software development project and therefore it could be considered that the management issues are similar. It could also be considered that the issues unique to migration, such as the reuse of legacy components, migration of data, provide a much more challenging prospect. Expanding existing migration approaches and building a general model of migration should provide a clearer view of exactly what a migration project involves and from this a management perspective can be derived. In addition, the management of the system resulting from migration must ensure that as few of the problems which effected the legacy system are experienced in the future.

- **Justifying Migration**

As mentioned in section 2.1, a migration project represents a huge undertaking for any organisation. It is a very expensive undertaking and carries a serious risk of failure. Methods to identify factors which affect the level of risk and to quantify this risk are required. Metrics have been developed to identify the risk involved in software development projects ([14], [58]), similar metrics are required for migration.

- **Target Environment Selection**

A major pre-requisite for justifying the commitment of the necessary time and resources a migration project requires must be that the target environment offer substantial immediate and long-term benefits. In today's competitive and fast-changing business and technological world, no organisation should be willing to expend a large amount of resources on migrating to a target platform which will become obsolete in a relatively short time.

Although it is impossible to predict the technological advances the future will bring, it can be predicted with reasonable certainty that choosing the most applicable architecture for an application can help lessen, or even avert, many of the problems currently being experienced by legacy system users. There a numerous potential target architectures and tools which could be used to develop the target system, see section 3.

Research into guidelines for choosing appropriate architectures would be most useful.

- **Managing Human Aspects of Migration**

The development of the target system is a crucial aspect of migration. It may be best to use existing staff or employ new staff for this specific task. The skills required by those responsible for developing such a system need to be investigated.

It must also be remembered that the target system will be deployed in a 'legacy' culture. Those responsible for maintaining the legacy may also be responsible for maintaining the target. This will, perhaps, involve a complete change of working practice. Research into how best manage this change is required.

- **Practical Experience Reports**

Few organisations have attempted migration projects. Much of the research into the area of legacy system migration is thus unsupported by practical results. As the year 2000 approaches it is to be hoped that many organisations may take the opportunity of migrating their legacy systems rather than enhancing them to cope with the year 2000.

6. Summary

Legacy Information Systems Migration is currently the focus of much attention from both business and research communities. The problems posed by legacy systems are a roadblock to progress for many organisations. Legacy systems are in danger of reducing their host organisation's competitiveness. This paper has presented an overview of the problems posed by legacy system and the challenges that possible solutions must address to overcome them.

The most significant proposed approaches to legacy migration have been discussed. It has been concluded that actual methodologies are either too general to be applied in practice or too specific to guide a complete migration project and are supported by few practical results. Current approaches fail even to agree in what phases must involve a generic migration process. This paper has outlined a set of phases any successful migration process should include. Each phase has been detailed and the challenges to overcome and the expected outputs of each have been clearly identified.

A set of tools to support each phase of migration has been identified. From the discussion of these tools, it has been found that most available tools are those needed in a any software engineering process (i.e. (target) system development, and testing). Specific tools for legacy migration are still to come (i.e. justification, target system cut-over, and understanding).

Legacy information system is becoming an area of increasing importance in both industry and academia and requires a lot more research.

7. References

[1] Acucobol, "Acu4GL: Interfaces to Relational Database Management Systems",http://www.acucobol.com/Products/Acu4GL/A4GL11genWP.html, 1996

[2] J. M. Antis, S. G. Eick and J.D. Pyrce, "Visualising The Structure of Large Relational Databases", IEEE Software, pp. 72-79, Jan. 1996

[3] Bachmann, "A CASE for Reverse Engineering", Datamation, pp. 49-56, July 1988

[4] A. Bateman and J. Murphy, "Migration of Legacy Systems", School of Computer Applications, Dublin City University, Working Paper CA-2894, http://www.compapp.dcu.ie/CA_Working_Papers, 1994

[5] B. Beizier, "Software Testing Techniques", Second Edition, Van Nostrand Reinhold, New York, 1990

[6] D. Bell and J. Grimson, "Distributed Database Systems", Addisson-Wesley, 1992.

[7] K. Bennet, "Legacy Systems: Coping with success", IEEE Software, pp. 19-22, Jan. 1995

[8] H. Berghel, "The Client's side of the World Wide Web", Communications of the ACM, 39(1), pp. 30-40, Jan. 1996

[9] T. J. Biggerstaff, "Design Recovery for Maintenance and Reuse", IEEE Software, pp. 36-49, July 1989

[10] M. Brodie and M. Stonebraker, "DARWIN: On the Incremental Migration of Legacy Information Systems", TR-022-10-92-165 GTE Labs Inc., http://info.gte.com/ftp/doc/tech-reports/tech-reports.html, Mar. 1993

[11] M. Brodie and M. Stonebraker, "Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach", Morgan Kaufmann. USA, 1995

[12] D. N. Chin and A. Quilici, "DECODE: A Co-operative Program Understanding Environment", Journal of Software Maintenance 8(1); pp. 3-34, 1996.

[13] S. Clinton, "Developing for Multi-Tier Distributed Computing Architectures with Delphi Client/Server Suite 2.0", http://netserv.borland.com/delphi/papers/, 1997

[14] D. Coleman, "Using Metrics to Evaluate Software System Maintainability", IEEE Computer, pp. 44-49, Aug. 1994

[15] D. Comer, "Computer Network and Internets", Prentice Hall, ISBN 0135990106, 1997

[16] Cyrano, "CYRANO's Automated Software Quality Products", http://www.pstest.com/, Dec. 1996

[17] Computer Science Department - The University of Namur, "DB-MAIN: A R&D Programme in Database Applications Engineering and Case Technology", http://www.info.fundp.ac.be/~dbm/, Feb. 1996

[18] G. Dedene and J. De Vreese, "Realities of Off-Shore Reengineering", IEEE Software, pp. 35-45, Jan. 1995

[19] Apertus Technologies Inc., "Enterprise/Access White Paper", http://www.apertus.com/prod/access/whitePaper.html, Mar. 1997

[20] Apertus Technolgies Inc., "Enterprise/Integrator White Paper", http://www.apertus.co.uk/app.rods/esg/eiwp.htm, March 1997

[21] P. Fingar and J. Stikeleather, "Next Generation Computing: Distributed Objects for Business", SIGs Books & Multimedia New York, 1996

[22] D. Flanagan, "Java in a Nutshell - A Desktop Reference for Java Programmers", O' Reilly & Associates Inc., 1996

[23] J. Gosling and H. McGilton, "The Java Language Environment: A White Paper", http://java.sun.com:80/doc/language_environment/, May 1996

[24] J-L. Hainaut, J. Henrard, J-M. Hick, D. Roland and V. Englebert, "Database Design Recovery", Proc. 8th Conf. on Advance Information Systems Engineering, CAiSE'96 Springer-Verlag pp. 463-480, 1996

[25] M. Hammer and Champy J, "Re-Engineering the Corporation - A manifesto for Business Revolution", Nicholas Brealey Publishing, 1993

[26] A. Hemrajani, "Networking with JAVA", Dr Dobb's Sourcebook, pp.34, Sept/Oct 1996

[27] P. J. Houston, "Introduction to DCE and Encina", http://www.transarc.com/afs/transarc.com/public/www/Public/ProdServ/Product/Whitepapers/, Nov. 1996

[28] Int'l Integration Inc., "I Cube", http://199.34.33.188/compdesc.htm, Dec. 1996

[29] I. Jacobson, M. Ericson and A. Jacobson, "The Object Advantage: Business Process Reengineering with Object Technology", Addison-Wesley New York, 1995

[30] LexiBridge, "The premier solution for migrating legacy systems to client/server", http://www.lexibridge.com/, July 1996

[31] Z-Y Liu, M. Ballantyne and L. Seward, "An Assistant for Re-Engineering Legacy Systems", Proc. 6th Innovative Applications of Atrificial Intelligence Conf. pp 95-102, AAAI, Seattle, WA,http://www.spo.eds.com:80/edsr/papers/asstreeng.html, Aug. 1994

[32] R. Martin, "Dealing with Dates: Solutions for the Year 2000", IEEE Computer, Mar. 1997, 30(3), pp. 44-51

[33] Dr. H. A. Muller, "Understanding Software Systems Using Reverse Engineering Technologies Research & Practice", Tutorial presented at Int. Conf. on Software Engineering 18, http://tara.uvic.ca/UVicRevTut/UVicRevTut.html, Mar. 25-29 1996

[34] NCR, "NCR TOP END: Robust Middleware For Transaction Processing", http://www.ncr.com/product/integrated/software/p3.topend.html, Mar. 1997

[35] OpenHorizon, "OpenHorizon - 3-Tier Client/Server Application", http://www.openhorizon.com, Mar. 1997

[36] R. Orfali, D. Harkey and J. Edwards, "Essential Client/Server Survival Guide", John Wiley, 1994

[37] R. Orfali, D. Harkey and J. Edwards, "The Essential Distributed Objects Survival Guide", John Wiley 1996.

[38] R. Orli, "Data Quality Methods", http://www.kismeta.com/cleand1.html, 1996

[39] L. Perrochon, "On the Integration of Legacy Systems and the World Wide Web", Presented at 4th Int'l World Wide

Web, Boston, MA, http://www.inf.ethz.ch/department/IS /ea/publications/ 4www95.html, Dec. 1995
[40] Persistence Software Inc., "Persistence Sofware: Enabling the Integration of Object Applications with Relational Databases", http://www.persistence.com/, July 1996
[41] Reasoning, "Reasoning Systems - Reengineering Solutions", http://www.reasoning.com, Mar. 1997
[42] ESPRIT Project, "RENAISSANCE Project - Methods & Tools for the evolution and reengineering of legacy systems", http://www.comp.lancs.ac.uk/computing/rese arch/cseg/projects/renaissance, Nov. 1996
[43] RENAISSANCE Project, "D5.1c Technology selection", http://www.comp.lancs.ac.uk/computing/research/cseg/pr ojects/renaissance/D5.1C_introduction.html, July 1997
[44] R. Richardson, D. O'Sullivan, B. Wu, J. Grimson, D. Lawless, J. Bisbal J, "Application of Case Based Reasoning to Legacy Systems Migration", Proc. 5th German Workshop on Case-Based Reasoning Foundations, Systems, and Applications, pp. 225-234, Mar. 1997
[45] Dr. H. Muller, "RIGI Project - An Extensible System for Retargetable Reverse Engineering", University of Victoria, Canada, http://tara.uvic.ca, Nov.1996
[46] T. Sample and T. Hill, "The Architecture of a Reverse engineering Data Model Discovery process", EDS Technical Journal, 7(1), 1993
[47] K. Sedota, J. Corley, J. Niemann James and T. Hill, "The INCASE Source Code Interviewer", EDS Technical Journal, 4(4), 1990.
[48] R. Shelton, "Business Objects and BPR", Data Management Review 4(11) pp. 6-20, Nov. 1994
[49] H. M. Sneed, "Planning the Re-engineering of Legacy Systems", IEEE Software, pp. 24-34, Jan. 1995
[50] Sun Microsystems, "Joe : Developing Client/Server Applications for the Web", Sun White Paper, 1997
[51] Systems Techniques Inc., "Wrapping Legacy Systems for Reuse : Repackaging v Rebuilding", http://www.systecinc.com/white/whitewrp.html, 1996
[52] Z. Tabakman and D. Pikcilingis, "Performing a Managed Migration", http://www.sector7.com/index.htm, 1995
[53] Z. Tabakman, "Successful Migration Through Automated Software Testing ", http://www.sector7.com/, 1996
[54] Tandem, "Non-stop Tuxedo: Open TP Monitor for Distributed Transaction Processing", http://www.tandem .com//INFOCTR/HTML/BRFS_WPS/NSTUXOTB.html, 1996
[55] D. Taylor, "Business Reengineering with Object Technology", John Wiley & Sons, New York 1995
[56] S. R. Tilley and D. B. Smith, "Perspectives on Legacy System Reengineering", http://www.sei.cmu.edu/, 1996
[57] UniData, "UniData - Data management that works", http://www.unidata.com/, July 1996
[58] K. D. Welker and Dr. P. W. Oman, "Software Maintainability Metrics Models in Practice", CrossTalk, Nov./Dec. 1995 8(1), 1995.
[59] P. Winsberg, "What About Legacy Systems ?", Database Programming and Design, 7(3), 1994
[60] K. Wong, S. Tilley, H. Muller, M. Storey, "Structural Redocumentation: A Case Study", IEEE Software, pp. 46-53, Jan. 1995
[61] B. Wu, D. Lawless, J. Bisbal, J. Grimson and R. Richardson, D. O'Sullivan, "The Butterfly Methodology : A Gateway-free Approach for Migrating Legacy Information Systems", in Proc. 3rd IEEE Conf. on Engineering of Complex Computer Systems (ICECCS '97), Villa Olmo, Como, Italy, Sep. 8-12 1997.
[62] B. Wu, D. Lawless, J. Bisbal, J. Grimson and R. Richardson, D. O'Sullivan, "Legacy System Migration : A Legacy Data Migration Engine", in Proc. 17th Int'l Database Conf., Brno, Czech Republic, Oct. 12 - 14, 1997.
[63] N. Zvegintzov, "A Resource Guide to Year 2000 Tools", IEEE Computer, 30(3), pp. 58-63, Mar. 1997
[64] M. Olsem, "Reengineering Technology Report Vol.1", Software Technology Support Centre(STSC, Oct. 1995
[65] I. Sommerville, "Software Engineering", Addison-Wesley, 1995
[66] E. J. Chikofsty and J. H.Cross II, 'Reverse Engineering and Design Recovery: A Taxonomy', IEEE Software, 7(1), January 1990, pp. 13-17
[67] K. Menhoudj and M. Ou-Halima, 'Migrating Data-Oriented Applications to a Relational Database Management System', Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS'96), Moscow Sept. 1996

Appendix I - Analysis of Current Migration Approaches

The following table is an analysis of the major migration approaches described in sections 4 and illustrates the extent to which each methodology addresses the five major areas of migration discussed in section 2. The main tasks to be performed by each methodology for each area is also outlined.

	Justification	Legacy Understanding	Target Development	Testing	Migration
Big Bang	Difficult due to huge cost and development time involved.	No legacy component reuse at all, so the legacy system must be completely understood.	Target technology could be out-of-date when project finishes, due to long development time.	Needs to be exhaustive due to the risk involved	Not contemplated. It is reduced to switching off the legacy and turning on the target
Database First	Data is the most valuable asset of a company. Migrating it as a first step could represent a risk that few companies are willing to take	Legacy data must be fully understood. Reuse is possible. Some components of the system can be treated as a black-box (i.e. not fully understood)	Could be incremental, which allows the system for adapting to business changes	Incremental implementation allows for exhaustive testing of each migrated component, before the new component is migrated.	Data must be migrated in one go, and during this time the information system is not operational
Database Last	Reverse database gateway tools available, leads to a quicker implementation. New target applications cannot exploit new database features until the migration has been finished	Same as *Database First*	Same as *Database First*	Same as *Database First*	Same as *Database First*
Composite Database **Chicken Little**	Both data and applications can be incrementally migrated. The massive complexity involved could be difficult to justify	Legacy data must be fully understood. Some components can be treated as black-box. However, need to understand (and manage) whether a component accesses legacy or target data (or both)	Same as *Database First* and *Database Last*	Same as *Database First* and *Database Last*	The data can be migrated in an incremental fashion. However, the management of the complexity involved to do so will be a serious challenge
Butterfly Methodology	Smooth migration step and controlled complexity	Legacy must be fully understood, and mapping between data models discovered. Reuse explicitly recommended	Supported by the Sample DataStore.	Explicitly supported by the Sample DataStore. In addition during migration the developing system can be tested against the already migrated data. Users can also be trained with this data	The legacy system is operation at all times. Also, the complexity is considerably reduced when compared to *Chicken Little*

Appendix II - Open Research Issues

The following table summarises the open research issues discussed in section 5.

General Migration Issues	Migration Methodologies	Legacy System Understanding	Target System Development	Data Migration	Migration Tools
Managing Migration	Developing a General Migration Approach	Language Understanding Tools	Validating the target system against the legacy system	An efficient real-time data migration engine	General migration toolkits
Justifying Migration Methods for evaluating risk, migration metrics		Data understanding Tools	Interoperation of information systems with migrating systems	Determining Data to be migrated	
Target Environment Selection	Refining Existing Migration Approaches	Effects of implementation versions on understanding			Developing individual migration tools
Managing human Aspects of Migration		Subject matter AI Support for understanding	Methods for component reuse	Cleaning dirty data	