Integrating Legacy Mainframe Systems: Architectural Issues and Solutions

An Investigation into Architectural issues raised by making the Mainframe a peer in a distributed network in the Financial Sector.

By

John Butler, B.Sc.

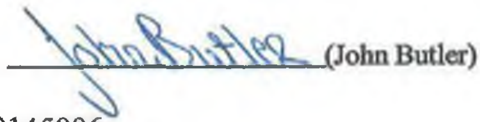
University:	Dublin City University
Supervisor:	Renaat Verbruggen
School :	Computer Applications

A dissertation presented in fulfilment of the requirements for the award of

M.Sc. in Computer Applications, February 2004

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Science in Computer Applications is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:  (John Butler)

(Candidate) ID No.: 99145006

Date: 1st February 2004

Acknowledgements

Firstly, I would like to thank Róisín for all her enthusiasm, support, and invaluable help with the proof reading.

I would like to give special thanks to my parents and family for their support and constant encouragement to keep going.

Sincere thanks is due to Renaat for all his advice, council and guidance throughout the research.

I would like to thank my employers ORBISM for facilitating this study in every possible way.

I am also indebted to my colleagues at Credit Suisse Zürich, for their help and knowledge sharing in all areas of CORBA and large-scale systems.

Last but not least, I would like to thank Colm for his help with the proof reading and being there to field any research related questions.

Integrating Legacy Mainframe Systems: Architectural Issues and Solutions

John Butler

Abstract

For more than 30 years, mainframe computers have been the backbone of computing systems throughout the world. Even today it is estimated that some 80% of the worlds' data is held on such machines. However, new business requirements and pressure from evolving technologies, such as the Internet is pushing these existing systems to their limits and they are reaching breaking point. The Banking and Financial Sectors in particular have been relying on mainframes for the longest time to do their business and as a result it is they that feel these pressures the most.

In recent years there have been various solutions for enabling a re-engineering of these legacy systems. It quickly became clear that to completely rewrite them was not possible so various integration strategies emerged.

Out of these new integration strategies, the CORBA standard by the Object Management Group emerged as the strongest, providing a standards based solution that enabled the mainframe applications become a peer in a distributed computing environment.

However, the requirements did not stop there. The mainframe systems were reliable, secure, scalable and fast, so any integration strategy had to ensure that the new distributed systems did not lose any of these benefits. Various patterns or general solutions to the problem of meeting these requirements have arisen and this research looks at applying some of these patterns to mainframe based CORBA applications.

The purpose of this research is to examine some of the issues involved with making mainframe-based legacy applications inter-operate with newer Object Oriented Technologies.

Table of Contents

DECLARATION II

ACKNOWLEDGEMENTS III

ABSTRACT IV

TABLE OF CONTENTS V

TABLE OF FIGURES XII

1 INTRODUCTION 1

1.1 MAINFRAMES AND EARLY TECHNOLOGY SOLUTIONS 1
1.1.1 Using Information Technology to conduct Business *1*
1.1.2 The Mainframe as the early platform of choice *1*
1.1.3 Living with these older applications today *1*
1.2 TECHNOLOGY REQUIREMENTS IN THE BANKING SECTOR 2
1.2.1 The Nature of the Banking Industry *2*
1.2.2 Use of Technology in the industry *2*
1.2.3 Further pressures from new Technology *2*
1.3 PROBLEMS ARISING 3
1.3.1 Why are these older applications a problem? *3*
1.3.2 How widespread are these older applications? *3*
1.3.3 What is a Legacy System? *3*
1.3.4 Who understands the application completely? *3*
1.3.5 How can the information be retrieved? *4*
1.3.6 Integrating the Legacy Systems *4*
1.3.7 The Real Challenge *4*
1.4 SOLUTIONS TO THESE PROBLEMS 5
1.4.1 Changing the way of thinking of Legacy Systems *5*
1.4.2 Standards Based Solutions *5*
1.5 FURTHER DIFFICULTIES AND THEIR SOLUTIONS 5
1.5.1 Example Standards Based Solutions *5*
1.5.2 Problems arising in using these Standards *5*
1.5.3 Sharing Solutions *6*
1.5.4 Patterns *6*
1.6 SUMMARY OF THE ISSUES AND RESEARCH AIMS 7
1.7 RESEARCH MOTIVATION 7
1.8 THE REST OF THE DOCUMENT 7

2 LEGACY APPLICATIONS IN THE FINANCIAL SECTOR 9

2.1 NATURE OF THE BANKING BUSINESS 9
2.1.1 Early Banking *9*
2.1.2 General Functions *9*
2.2 ARCHITECTURE OF THE MAINFRAME OPERATING SYSTEM 10
2.2.1 A Brief History of Computing *10*
2.2.2 Categories of Computers *10*
2.2.3 Early Compatibility Issues *11*

2.3 MAINFRAME TECHNOLOGY 11
2.3.1 Mainframe Overview *11*
2.3.2 From MVS to OS/390 ..... *11*
2.3.3 MVS Products *12*
2.3.4 TPF (Transaction Processing Facility) and VM *13*
2.3.5 Storage considerations *13*
2.3.6 Transactions *14*
2.3.7 Limitations and Benefits of the Mainframe Architecture *14*
2.4 FIRST BANKING APPLICATIONS 15
2.4.1 What was available? *15*
2.4.2 Applications performing Business operations *15*
2.4.3 Essentials of a Business Applications *15*
2.4.4 Composition of a Business Application *16*
2.5 NEW CUSTOMER DEMANDS AND PRESSURES ON THE SYSTEMS 16
2.5.1 Early Technology Requirements *16*
2.5.2 Changing Business Requirements *16*
2.5.3 Evolving Technologies *17*
2.5.4 New Opportunities *17*
2.6 CHALLENGES FOR THE FUTURE 18
2.7 CONCLUSION 18

3 APPROACHES TO INTEGRATING LEGACY SYSTEMS 20

3.1 INTRODUCTION 20
3.2 REPLACING THE ENTIRE BANKING SYSTEM 20
3.3 APPROACHES TO INTEGRATING THE LEGACY SYSTEMS 21
3.3.1 Level of intrusiveness *21*
3.3.2 Reusability and Robustness *22*
3.3.3 Complexity *23*
3.4 EMULATOR 23
3.4.1 TN3270 Protocols *23*
3.4.2 Using the Emulator Approach *23*
3.5 SCREEN SCRAPING (PRESENTATION LAYER INTEGRATION) 24
3.5.1 What is Screen Scraping *24*
3.5.2 Advantages and Disadvantages *24*
3.6 FUNCTIONAL LEVEL INTEGRATION 24
3.7 USING CORBA FOR FUNCTIONAL LEVEL INTEGRATION 25
3.7.1 CORBA and Screen Scraping *25*
3.7.2 CORBA Adapter Technology *25*
3.7.3 Full Integration *26*
3.7.4 Why CORBA *26*
3.7.5 Using J2EE Connectors *27*
3.8 DATA LEVEL INTEGRATION 28
3.8.1 XML *28*
3.9 CONCLUSION 28

4 INSIDE CORBA 30

4.1 HISTORY OF DISTRIBUTED TRANSACTIONAL COMPUTING 30
4.1.1 Centralised versus de-centralised *30*
4.1.2 GUIs and Smarter Clients *30*
4.1.3 The Client/Server Model *31*

4.1.4 3-tier Computing ... 32
4.1.5 Difficulties with 3-tier computing ... 32
4.1.6 Remote Procedure Call ... 32
4.1.7 Object Oriented n-tier computing ... 33
4.1.8 Internet Revolution ... 33
4.1.9 Different Connectivity Models ... 34
4.1.10 Component Applications ... 34
4.1.11 Improvements in Design ... 35
4.1.12 General benefits and problems with N-tier computing ... 35
4.2 INTRODUCTION TO CORBA ... 36
4.2.1 Difficulties of Distributed Object Programming ... 36
4.2.2 What is an ORB ... 36
4.2.3 The Object Management Architecture (OMA) ... 37
4.2.4 The Nature of CORBA Objects ... 37
4.2.5 The Structure of a CORBA Application ... 38
4.2.6 The Structure of a Dynamic CORBA Application ... 38
4.2.7 Dynamic Server Programming ... 38
4.2.8 Interoperability between ORBs ... 38
4.2.9 The CORBA Services ... 39
4.2.10 The CORBA Facilities ... 39
4.3 LOOKING INSIDE CORBA ... 39
4.3.1 The Basics ... 39
4.4.2 IDL Mappings ... 41
4.4.3 Finding CORBA Objects ... 41
4.4.3.1 The Naming Service ... 42
4.4.4 Exception Handling ... 42
4.4.5 ORB Interoperability ... 43
4.4 SERVICE-BASED ARCHITECTURE ... 43
4.5 CONCLUSION ... 44

5 PATTERNS ... 45

5.1 WHAT ARE PATTERNS ... 45
5.2 HOW DOES A PATTERN COME TO BE ... 46
5.3 DESIGN PATTERNS ... 48
5.4 WHAT AREN'T DESIGN PATTERNS ... 49
5.5 ARCHITECTURAL PATTERNS ... 50
5.6 CATEGORIES OF ARCHITECTURAL PATTERNS ... 51
5.6.1 From Mud to Structure ... 51
5.6.2 Distributed Systems ... 52
5.6.3 Interactive Systems ... 52
5.6.4 Adaptable Systems ... 52
5.7 IMPLEMENTING ARCHITECTURAL PATTERNS ... 52
5.8 QUALITIES OF PATTERNS ... 53
5.9 CONCLUSION AND FURTHER DEVELOPMENTS ... 54

6 CORBA PERFORMANCE ISSUES ... 56

6.1 INTRODUCTION ... 56
6.1.1 Performance of Distributed Systems ... 56
6.1.2 Other Performance Problems ... 57
6.1.3 Designing IDL for performance ... 58

6.2 GENERAL SOLUTIONS TO THE CORBA PERFORMANCE PROBLEMS 58
6.3 MINIMISE THE NUMBER OF REMOTE OPERATIONS 59
6.4 OPTIMISING THE TYPE OF DATA SENT OR RETURNED 60
6.4.1 Orders of Magnitude *60*
6.5 OPTIMISING THE AMOUNT OF DATA SENT OR RETURNED 61
6.6 ADDITIONAL CORBA PATTERNS 63
6.7 CONCLUSION 65

7 SECURITY ISSUES 66

7.1 MAINFRAME SECURITY 66
7.1.1 RACF (Resource Access Control Facility) *67*
7.1.2 RACF Record Keeping *67*
7.1.3 Mainframe Security in a Client/Server Model *68*
7.2 BANKING SECURITY REQUIREMENTS 69
7.3 THREATS TO BANKING SECURITY 70
7.3.1 Internal Network Security *70*
7.3.2 Internet and Extranet Security *70*
7.3.3 Security Threats *71*
7.3.3.1 Deliberate Security Threats *71*
7.3.3.2 Accidental Security Threats *72*
7.4 REQUIRED SECURITY SERVICES 72
7.4.1 Security Services *72*
7.4.2 Applying Security Services *73*
7.4.3 Addressing different System Areas *73*
7.5 FIREWALL TECHNOLOGY 74
7.5.1 Firewall Security *75*
7.5.2 Packet Filters *76*
7.5.3 Proxy Servers *76*
7.6 DE-MILITARISED ZONES 77
7.7 PUBLIC KEY INFRASTRUCTURE 79
7.7.1 Secure Sockets Layer *79*
7.7.2 X.509 Certificates *81*
7.7.3 Certificate Authority *82*
7.7.4 Other PKI Issues *82*
7.7.5 PKI Locations *83*
7.8 INTEGRATING WITH OTHER SECURITY MODELS 84
7.8.1 CORBA Security Service *84*
7.9 SAMPLE ARCHITECTURES 86
7.10 SECURITY PATTERNS 87
7.11 CONCLUSION 87

8 SCALABILITY 89

8.1 INTRODUCTION 89
8.2 MULTITHREADING 90
8.2.1 Concepts behind Multithreading *90*
8.2.2 Multi-processing *90*
8.2.3 Multithreaded languages *90*
8.2.4 Difference between threads and processes *90*
8.2.5 Choosing threads or processes *91*
8.2.6 Choosing Multithreading or Single Threading *91*

8.2.7 Using threads *92*
8.2.8 Dangers of Multithreading *92*
8.2.9 Managing Threads *93*
8.2.10 Structured Locking Techniques *93*
8.2.11 Threading Policies *94*
8.2.12 Limits *94*
8.2.13 CORBA Alternatives to Multithreading *94*
8.3 CONNECTION MANAGEMENT 95
8.3.1 Establishing Connections *95*
8.3.2 Reconnection *96*
8.3.3 IIOP Connection Features *96*
8.3.4 Callbacks *96*
8.3.5 Direct IIOP Connection *96*
8.3.6 CORBA Daemon *97*
8.3.7 Closing Connection *97*
8.3.8 Connection Limits *97*
8.3.9 Connection Patterns *97*
8.3.9.1 Client Disconnects *98*
8.3.9.2 Concentrator *98*
8.3.9.3 Server Disconnects *98*
8.3.9.4 Other Idioms *99*
8.4 SESSION MANAGEMENT 99
8.4.1 Sessions *99*
8.4.2 Session Management Issues *99*
8.4.3 Availability *99*
8.4.4 Termination *100*
8.4.5 Service Architecture *100*
8.4.5.1 Concurrent Clients *100*
8.4.5.2 Number of Requests *100*
8.5 CONCLUSION 101

9 AVAILABILITY (LOCATING CORBA SERVICES) 102

9.1 INTRODUCTION 102
9.2 LOCATING A SERVICE 102
9.3 PROVIDING AN OBJECT REFERENCE 103
9.4 INTEROPERABLE OBJECT REFERENCES 104
9.5 PROPRIETARY SOLUTIONS 104
9.6 GETTING THE OBJECT REFERENCE 104
9.7 THE CORBA NAMING SERVICE 105
9.7.1 Choosing a Naming Service Hierarchy *106*
9.7.2 Extensions to the Naming Service *107*
9.7.3 Naming Service Difficulties *108*
9.8 THE CORBA TRADER SERVICE 108
9.9 BOOTSTRAPPING 109
9.10 CUSTOM OBJECT LOCATION 109
9.11 PUBLISHING CERTAIN OBJECTS 110
9.12 LIFETIMES OF OBJECTS 111
9.13 CONCLUSION 111

10 AVAILABILITY (FAILOVER) 112

10.1 INTRODUCTION....112
10.2 MAINFRAME AVAILABILITY....112
10.3 FAILURES....113
10.4 EXCEPTION HANDLING....113
10.4.1 Introduction to Error Handling....*113*
10.4.2 Early Error Handling....*114*
10.4.3 Dealing with Exceptions....*114*
10.4.4 Distributed Exception Handling....*114*
10.4.5 CORBA Exception Handling....*115*
10.4.6 CORBA User Exceptions....*115*
10.5 FAULT TOLERANCE....116
10.5.1 Introduction....*116*
10.5.2 Realising the failure....*117*
10.5.3 Recoverable Servers....*117*
10.5.4 Server Monitors....*118*
10.5.5 Replicating Objects....*119*
10.5.5.1 Primary-Secondary Replication....*120*
10.5.5.2 Stateful Objects....*120*
10.5.6 Multicast....*121*
10.5.7 Fault Tolerance Patterns....*121*
10.6 LOAD BALANCING....123
10.6.1 What is Load Balancing....*123*
10.6.2 Requirements of a Load Balancing Policy....*124*
10.6.3 Benefits of Load Balancing....*125*
10.6.4 Dangers of implementing a load balancing policy....*125*
10.6.5 Real World Uses....*125*
10.6.6 Load Balancing Algorithms and Policies....*126*
10.6.7 Implementing Load Balancing using the CORBA Naming Service....*127*
10.6.8 Network Based Load Balancing....*127*
10.6.9 Operating System Load Balancing....*128*
10.6.10 Software approaches to implementing a Load Balancing Solution....*128*
10.7 CONCLUSION....128

11 IMPLEMENTATION AND RECOMMENDATIONS....130

11.1 IMPLEMENTATION INTRODUCTION....130
11.2 CREDIT SUISSE MAINFRAME ARCHITECTURE....130
11.3 CHOOSING A SUITABLE INTEGRATION ARCHITECTURE....131
11.4 BUILDING AN ARCHITECTURE BASED ON MANAGED EVOLUTION....133
11.4.1 Services-modules instead of components....*133*
11.4.2 Bottom-up approach for the existing system....*134*
11.5 PERFORMANCE CONCERNS....136
11.6 SECURITY CONSIDERATIONS....139
11.7 SCALABILITY....142
11.8 AVAILABILITY....143
11.9 FAILOVER....144
11.10 OTHER IDIOMS AND USEFUL SOLUTIONS....146
11.11 CREDIT SUISSE CORBA INFRASTRUCTURE....147
11.12 CONCLUSION....148

12 FUTURE....149

12.1 The Future of Mainframes 149
12.2 The Future of Legacy Applications 149
12.3 The Future of Distributed Computing 150

13 CONCLUSION AND FURTHER WORK 151

13.1 General Work for the future 151
13.2 Specific Research Possibilities 152

APPENDIX A: WEB RESOURCES 153

BIBLIOGRAPHY 154

Table of Figures

FIGURE 2.1: A TYPICAL EARLY ARCHITECTURE 12
FIGURE 2.2: EARLY MULTIPLE VIRTUAL STORAGE 13
FIGURE 4.1: THE CENTRALISED MODEL OF COMPUTING 30
FIGURE 4.2: A TIMELINE FOR DISTRIBUTED TRANSACTIONAL COMPUTING 30
FIGURE4.3: THE 2-TIER MODEL 31
FIGURE 4.4: THE 3-TIER OR N-TIER MODEL 32
FIGURE 4.5: BROWSER BASED CLIENT/SERVER 34
FIGURE 4.6: COMPONENT MODEL 35
FIGURE 6.1: INTERFACES DEFINE WHAT A CORBA SERVICE WILL PROVIDE 58
FIGURE 7.1: THE PRIVATE COMPANY NETWORK 66
FIGURE 7.2 : RESOURCE ACCESS CONTROL FACILITY 67
FIGURE 7.3: RACF SECURITY ADMINISTRATION 68
FIGURE 7.4: BANKING SYSTEMS REQUIREMENTS 69
FIGURE 7.5: INTERNAL NETWORK SECURITY REQUIREMENTS 70
FIGURE 7.6: INTERNET AND EXTRANET SECURITY REQUIREMENTS 71
FIGURE 7.7: PROTECTING THE BANKS RESOURCES 73
FIGURE 7.8: DIFFERENT SECURITY AREAS 74
FIGURE 7.9: FIREWALL TECHNOLOGY 75
FIGURE 7.10: CORBA PROXY SERVER 77
FIGURE 7.11: DE-MILITARISED ZONE 78
FIGURE 7.12: MULTIPLE PROXY SERVERS 79
FIGURE 7.13: SSL PROTOCOL LAYERED BETWEEN IIOP AND TCP/IP 80
FIGURE 7.14: PUBLIC KEY CRYPTOGRAPHY/ASYMMETRIC KEY CRYPTOGRAPHY 81
FIGURE 8.1: MAKING CORBA LEGACY APPLICATIONS SCALE 89
FIGURE 9.1: OBJECT REFERENCE 102
FIGURE 9.2: OBJECT LOCATION MODEL 103
FIGURE 9.3: COSNAMING IDL 105
FIGURE 9.4: NAMING STRUCTURE 106
FIGURE 10.1 : ADDING EXCEPTIONS TO IDL 116
FIGURE 10.2: KEEPING STATE FOR RECOVERABLE SERVERS 118
FIGURE 10.3: SERVER MONITOR 119
FIGURE 10.4: BEFORE A LOAD BALANCING SOLUTION 123
FIGURE 10.5: SIMPLE LOAD BALANCING SOLUTION 124
FIGURE 11.1 CREDIT SUISSE TECHNOLOGY DETAILS 130
FIGURE 11.2 SERVICE-MODULES 133
FIGURE 11.3 SERVICE ARCHITECTURE 134
FIGURE 11.4 CREDIT SUISSE SERVICE ARCHITECTURE OVERVIEW 135
FIGURE 11.5: USE SEQUENCES INSTEAD OF NETWORK CALLS 137
FIGURE 11.6 USE STRUCTURES INSTEAD OF MANY ATTRIBUTES 138
FIGURE 11.7: INITIAL BUSINESS REQUIREMENTS 139
FIGURE 11.8: SECURE THE PERIMETER 140
FIGURE 11.9: ADDING A DMZ 141
FIGURE 11.10: FINAL CORBA SECURITY INFRASTRUCTURE 142
FIGURE 11.11 CORBA NAMING SERVICE 144
FIGURE 11.12 REPLICATED NAMING SERVICE 145
FIGURE 11.13 LOAD BALANCED ENTRIES 146

1 Introduction

The integration of mainframe legacy applications and their architectures is the central focus of this dissertation. This chapter will provide a general overview of the topics investigated. Firstly there is an examination of mainframe systems and the Banking Sector with a focus on how the requirements of this sector are rapidly changing. There are various solutions available in the marketplace to help meet these requirements but even these solutions can cause further difficulties. A new way of thinking and sharing of experiences can help overcome these problems. Throughout the dissertation, examples of "real world" architectures and their integration strategies will be provided.

1.1 Mainframes and early Technology Solutions

1.1.1 Using Information Technology to conduct Business

At the start of the 21st century, all of the world's large companies have an Information Technology department and there are few that conduct business without technology. However, quite a number of industries, especially in Banking Sector have particularly old applications that they still use and rely on **[Jordan 1996]**. This sector of industry was among the first to make use of computers as part of their business model and as the first real use was over 30 years ago, some of these corporations still rely on code that is as old as this for mission critical tasks.

1.1.2 The Mainframe as the early platform of choice

Typically these early applications were implemented on the mainframe platform and most probably in a language such as COBOL or PL/I. These computers could reliably, and securely provide a central server that many terminals could run applications from. During those early days, the mainframe was comfortably able to meet the demands of the Banking Sector, in that it could provide the various applications needed to conduct the core business of the institution and could do so safely and reliably.

1.1.3 Living with these older applications today

Many Banking institutions that used the mainframe and its applications in these early days of information technology must now face up to the requirements of today's business, which has changed completely since these pioneering days. Such institutions have to decide what to do with the "legacy applications" and how they can reinvent or reengineer them to provide what today's customers now expect from a software solution.

1.2 Technology Requirements in the Banking Sector

1.2.1 The Nature of the Banking Industry

The role of a Bank in today's global economy is changing frequently. Traditionally, a Bank safeguarded money and valuables, provided loans, credit and payment services such as checking accounts, money orders, and cashier's checks **[Horswill 2000]**. As the banking industry is slowly deregulated, banks are also offering more investments and insurance products that they were once prohibited from selling.

1.2.2 Use of Technology in the industry

For quite some time now, the Banking Sector has relied quite heavily on Information Technology to conduct its business. Many routine banking services that once required a teller, such as making a withdrawal or deposit, are now available through Automated Teller Machines (ATM) that allow people to access their accounts 24-hours a day. Direct deposit facilities allow companies and governments to electronically transfer payments into various accounts. Debit cards and "smart cards" instantaneously deduct money from an account when a card is swiped across a machine at a store's cash register. Finally, electronic banking by phone and increasingly via the Internet allows bills to be paid and money transferred from one account to another.

1.2.3 Further pressures from new Technology

In today's ever demanding global environment, systems are required that are more accessible, have greater functionality and provide all of the services detailed above with increased performance and better security than before. In addition, Graphical User Interfaces (GUIs) are a must for any new application, and these require more computing power **[Froidevaux et al. 1999]**.

These increased demands, and the current high pace of process reengineering is stretching existing systems to their limits. There are also extra pressures to move away from older technologies coming with the arrival of Internet, Electronic-Commerce, Online Banking, Wireless Application Protocol (WAP), GUIs, and Object Oriented Analysis and Design (OOAD). Additionally, corporate mergers and company take-overs also often require the joining of the Information Technology (IT) systems. All of these combined are putting older systems in a position where they are simply not able to adapt and move forward with the changes required. **[Chen et al. 2000]** outline some of the risks to business of increased advances in the technology sector. These include

- Due to rapid changes in markets, information must be continuously updated. This adds an extra cost of market intelligence.
- There is an increased market risk in the selling of new technology products that may not be successful.

- Information Technology investment is a large fixed cost. Additionally, security, performance, interoperability, and equipment depreciation add to this cost. Finally, the capital base required for enterprise IT solutions can be massive.
- There is a human constraint introduced as technology interacts with the human user.

1.3 Problems arising

1.3.1 Why are these older applications a problem?

Just as it is difficult for us today to anticipate all the demands that competitive business pressures will place on an organisation 30 years from now, it was impossible for developers thirty years ago to imagine where technology would be today. Indeed **[Lauder, Kent 2000]** warn of such a lack of domain expertise. As will be seen, these architectures do not suit the development of new applications as all aspects of computing, development and computer architectures have changed considerably.

1.3.2 How widespread are these older applications?

To answer this question it is useful to look at some figures regarding the use of COBOL over the last few decades.

From the late 1960's through the 1980's, 75% of all business applications were written in COBOL **[Kolodziej 1987]**. Organisations spent half of their MIS budgets to produce approximately five billion lines of COBOL code. **[Parikh, Girish 1987]** Towards the end of the 1990's it was estimated that there were more than 30 billion lines of COBOL code in operation. **[Lawrence 1996]**

Coupled with the pressure for change, and given the widespread reliance on COBOL and older applications, maintaining these applications and providing continued access to their data is among the number one concerns for modern banking corporations. Such institutions are constantly seeking way to modernise their systems.

1.3.3 What is a Legacy System?

When considering this question and the modernisation currently being attempted by large many of the world's largest companies, it is useful to read one definition of Legacy Systems

> *A legacy system is one that significantly resists modification and evolution to meet new and constantly changing business requirements* **[Brodie Stonebraker 1995]**

1.3.4 Who understands the application completely?

It is clear as to why these legacy systems pose a problem in today's environment. Typically, many such systems have been maintained over time by a series of software developers. Each of these developers may not have completely understood the entire system or program. It is next to

impossible to properly rewrite systems that have been built over 2 or 3 decades without considerable expense. One of the difficulties faced throughout the software industry is that often people don't know exactly how the entire systems work. **[Morris, Isaksson 2002]** detail the different levels of intrusiveness with different integration strategies and each of these will be looked at in turn in later sections.

In addition, if a re-write was to take place of such an application today, it is likely that a considerable amount of the original programming team have long since left the organisation. Likewise, it is conceivable that all of the original documentation is not still available, and in fact, it is the case that many of these applications are largely undocumented. Over the years, many localised fixes are likely to have been applied to these systems (again as is in the nature of application development) to make a feature work to meet an immediate deadline or requirement. All of these factors have made the application "brittle" and requiring additional maintenance. Likewise, it has made any uniform upgrade of the architecture of the entire system, highly costly, and quite unlikely. **[Jiang 1998]** investigates the costs and benefits between Legacy extension and Legacy Integration.

1.3.5 How can the information be retrieved?

In essence, organisations are looking at how to best get at the knowledge and information already existing in legacy applications. Quite often, these organisations have millions of Euro invested in systems that were built in-house and although many were replaced at Year 2000 (Y2K) by packaged Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) systems, there are still enough legacy applications untouched to make it a highly discussed issue. The big challenge in today's IT industry is to integrate these legacy systems with today's web and Internet based technologies.

1.3.6 Integrating the Legacy Systems

Since the arrival of some modern operating systems and programming languages, there have been methodologies and approaches available to make these newer paradigms work together with their older predecessors. These approaches have varied from completely re-writing the legacy applications to work with the newer platforms, to less intrusive approaches such as „wrapping“ the legacy application or simply getting access to the data the legacy applications represented. **[Reddy 2002]** outlines a "Screen-Scraping" approach which is among the least intrusive approaches of integration while **[Froidevaux et al. 1999]** outline a full-integration approach.

Each of these approaches has been tried and tested and certainly all have worked with differing degrees of success depending on the application involved.

1.3.7 The Real Challenge

One of the greatest challenges with regard to legacy systems is to understand them completely. Second to this comes the challenge of integrating them with the new paradigm. Many organisations see the key requirements as being how to leverage what is currently there without having to go inside and document it by hand. Even still, almost all answers as to how to utilise these legacy systems are likely to require significant engineering expertise and monies. Even then

there is no guarantee of successful renovation and integration. However it what is widely agreed is that there is significantly less risk in revamping these systems than in re-writing them from scratch.

1.4 Solutions to these problems

1.4.1 Changing the way of thinking of Legacy Systems

The role of the mainframe is changing with the dawn of the new century. Companies have already spent a considerable amount readying their mainframe systems for the year 2000. This plus other investments are forcing companies to once again view mainframe technology as a core resource in their IT infrastructure.

1.4.2 Standards Based Solutions

Many Banking Institutions have opted for the "wrapping" approach as the method of choice to bring their legacy applications into synchronisation with the rest of their computing environment. Early attempts at this integration strategy used solutions proprietary to the organisation. This very quickly leads to further legacy applications.

At the start of the 1990's, the concept of standards based solutions started to appear on the market. These solutions involved using standards for inter-platform and inter-operating language communication and immediately it became clear that they could be used in this way to access the legacy systems from new technologies such as the Internet and Java. **[Murer 1999]** outlines the use of a standards-based solution in the Credit Suisse legacy integration project.

1.5 Further Difficulties and their Solutions

1.5.1 Example Standards Based Solutions

One of the most popular solutions available in the industry today is the Common Object Request Broker Architecture (CORBA) by the Object Management Group (OMG). Another is the Java 2 Enterprise Edition (J2EE) by Sun Microsystems (in particular the Connector Architecture specification). A third popular standard in the middleware industry is the DTP (Distributed Transaction Processing) specification by the X/Open Group.

1.5.2 Problems arising in using these Standards

Early pioneer projects that used CORBA or J2EE Connectors as their chosen Architecture solution for integrating their legacy applications soon ran into problems. The specifications needed market implementations and these implementations deviated from the specification from time to time. On other occasions the specification itself was simply not precise enough.

In addition, the implementations sometimes only provided the base architecture for integrating legacy systems and the difficult parts of object location, thread management, transaction support and security were left up to the individual organisations to implement themselves. **[Koch Murer 1999]** outline such problems seen in Credit Suisse.

1.5.3 Sharing Solutions

It quickly became apparent that throughout the banking sector the same problems were arising. It also became apparent that once an organisation or project came up with a solution for the problem that this solution could be useful to others trying to accomplish the same goals.

1.5.4 Patterns

One of the most useful additions to software development in recent times has been the concept of a pattern. A pattern is a solution to a well-known problem that can be applied in different instances to overcome this problem. One of the most influential books of the last 10 years has been "Design Patterns – Elements of Reusable Object Oriented Software" by Gamma, Helm, Johnson and Vlissides **[Gamma 1995]**. This book provides many patterns or well-known solutions to various problems encountered with software development using object-oriented principles. These patterns do not claim to have all of the answers or a complete solution, but the use of the principles inside can make the overall development process less error prone.

The concept of patterns in computing came from the field of Architecture and specifically **[Alexander 1977]** and **[Alexander 1979]** which were revolutionary in that field and, in turn, changed the way people though about problems in world of computing too.

A form of pattern that has started to become popular in recent times is the "Architectural Pattern". This type of pattern can be used in high-level architectures and Legacy integration projects to ensure that any solution will work at the enterprise level as is mandated in the requirements of any large scale project by today's Banking Sector.

In turn, people such as **[Mowbray, Malveau 1997]** researched and produced patterns for specific areas and solutions so that a project choosing one integration strategy could benefit from the experience of others that had been in the same situation previously.

1.6 Summary of the Issues and Research Aims

The following list comprises the research aims described in this document:

- Define the requirement of modern banking systems
- Investigate the approaches available for integration of the legacy systems
- Consider the critical problem areas and investigate solutions and patterns.

1.7 Research Motivation

Most of the large organisations within the Banking Sector have realised that today there is a problem with their Legacy Systems. This problem is simply that they cant live with them because of new technology and customer requirements, but they cant live without them for business reasons. Those who have decided not to replace these systems are going for various integration strategies to enable integration with newer technologies including the Internet and Java.

Among those who have already begun this process, there is a realisation that the task at hand is not trivial. It is clear that to get to where they need to be on budget and on time they are wise to use standards based approaches. To also look at "patterns" and solutions suggested by other projects attempting the same task would further help this process.

The main aim of this research is to find out the possible blockages to a smooth integration of legacy and object-oriented technologies. A second, more important aim is to find possible solutions to these problems and offer advanced and industry recognised solutions to further potential problems.

Throughout the research each of these goals and objectives will be re-examined to ensure that the real-world problems of legacy integration projects are addressed and to evaluate the approaches proposed in this context.

At the end of the research, it should be possible to compare any approaches recommended in terms of is benefits to the Banking Sector. Any new overheads or difficulties should also be reviewed, so that extra research could offer further solutions.

1.8 The Rest of the Document

The following topics will be considered moving through this thesis:

Section 2 looks into the requirements of information technology in the Banking Sector and the legacy systems that are already there. Section 3 considers the various approaches to integrating these systems.

Section 4 investigates one of the approaches (CORBA). Section 5 looks at the concept of Patterns and details the concept of architectural patterns in this context.

Sections 6,7,8 9 and 10 looks at some of the most difficult areas for integrating legacy systems. These areas are Performance, Security, Scalability and Availability. Section 11 details a real world implementation of such a legacy integration problem.

In the Conclusion there is a look at an overview of what has been investigated and other considerations for the future.

2 Legacy Applications in the Financial Sector

2.1 Nature of the Banking Business

2.1.1 Early Banking

Today's banking industry is a cornerstone of countries' economy. It is interesting to learn however, that the concept of a bank pre-dates even the use of coinage in society. It is believed that banking first originated in Ancient Mesopotamia where the royal palaces and temples provided a secure safekeeping place for grain and other commodities. **[Glynn 1996]**

This early form of banking included the use of receipts for transfers to both the original depositors as well as to third parties. With the success of this banking, private houses also became involved in similar banking operations and laws regulating these banking practices were included in the code of Hammurabi, which was the rule of law in Mesopotamia.

The history of money and coinage goes back to very early times with increases in population groups and the use of barter as means of exchange. Money as an accepted form of payment began with the first finding of precious metals and eventually "token" money was accepted instead of a metal of equivalent value to the product being bought.

Today, money serves various purposes. Among its concrete functions, it should be an acceptable medium of exchange in trades for goods and services. It should be a standard of value for a good or service, and should be a store of value so that it can be saved and used in the future **[Glynn 1996]**

2.1.2 General Functions

The primary function of a modern bank is to safeguard money and valuables. Other functions include the providing of loans, credit and payment services, such as checking accounts, money orders, and cashier's checks. Modern banks also offer insurance and investment products.

As **[Jordan 1996]** indicates, the functions of modern retail banks include

- Conducting exchange: clearing and settling claims
- Funding large-scale enterprises: resource pooling
- Transferring purchasing power across time and distance
- Providing risk management: hedging, diversification, and insurance
- Monitoring performance of borrowers: mitigating adverse incentives
- Providing information about the relative supply and demand for credit

Several types of banks exist in this marketplace that differ in the number of services they provide and the clientele they serve. Commercial banks that dominate the industry typically offer a full range of services for individuals, businesses, and governments. These banks come in a wide range of sizes from large "money centre" banks to regional and community banks. Such banks tend to

get their primary revenue from the interest on loans to private or commercial parties. Of course, they can use the money deposited in savings accounts to fund these loans and will give a lower rate of interest on this money than the interest they charge on the loans, ensuring profitability.

Money centre banks are often located in major financial centres and are usually involved in international lending and foreign currency, in addition to the more typical banking services. Regional banks are usually concentrated in a geographical area and their numerous branches and Automated Teller Machine (ATM) locations appeal to individuals. Community banks are based locally and offer more personal attention that small businesses prefer.

There are also Savings banks catering to the saving and lending needs of individuals, Credit Unions with members often from the same organisation or company, Federal Reserve Banks which are Government agencies performing various financial services on behalf of the Government.

2.2 Architecture of the Mainframe Operating System

2.2.1 A Brief History of Computing

The mainframe operating system was among the first commercially available operating systems to be used in the banking industry. The modern computer followed on from Charles Babbages' first experiments with numerical machines. Some 70 years later, the first electronic computers were created.

In some banks, use was made of electromechanical relays that served as on and off switches. It could handle 23 decimal places and four arithmetic operations. It could also run special programs to handle logarithms and trigonometric functions. These new machines could complete tasks that would take a man six months in one day.

An example of such technology was the Mark I project from Harvard University in 1941. The Mark I machine was soon replaced by the ENIAC (Electronic Numerical Integrator and Computer). For full details on this technology see **[Weik 1961]**

The widespread use of computer in banks started in earnest with the era of the big or “super” computers that began in the early 1960’s. There were three general categories of computers:

2.2.2 Categories of Computers

The Scientific computers were a family of computers that were primarily designed to perform calculations with large numbers. Floating point arithmetic is the term for large number processing and allows large or small numbers to be represented and manipulated. This type of machine was popular with universities and large companies.

Decimal Computers were primarily designed to perform calculations with currencies. Decimal arithmetic is the term for this type of processing and businesses wanting to process financial data were the big customers.

Character Computers were character oriented and were more general purpose. Character oriented computers were designed to address character strings such as names and addresses.

2.2.3 Early Compatibility Issues

All of these families of computers could perform decimal, number and character manipulation. The architecture, however, was aimed at one type or the other and if an organisation wanted to move more towards a different family, their application programs would require large changes.

Each family used different instructions to perform the same functions. The instructions were different and conversion from one software system to another was traumatic for designers and developers alike. Every application had to be completely rewritten from scratch to use another hardware. **[Johnson 1989]**

The actual programming of such machines was also not a trivial task. This can be seen from a programmer of such systems

"*At first, programmers were given an IBM memory chart on which were printed 200 rows and 10 columns. The theory was that as you wrote your code, you would place your instructions and data in an optimal location and then mark that memory location off on the chart. This sounded great until you actually started modifying a large program that nearly filled the drum. You soon took what you could get*" **[Kugel 2001]**

2.3 Mainframe Technology

2.3.1 Mainframe Overview

With mainframe software architectures all intelligence is within the central host computer. Users interact with the host through a terminal that captures keystrokes and sends that information to the host.

MVS (Multiple Virtual Storage) is the operating system from IBM that is installed on most of the mainframes and large server computers in the world. MVS is a generic name for specific products that includes MVS/SP (MVS/System Product), MVS/XA (MVS/Extended Architecture), and MVS/ESA (MVS/Enterprise Systems Architecture).

2.3.2 From MVS to OS/390

Historically, MVS evolved from OS/360, the operating system for the System/360, which was released in 1964. It later became the OS/370 and the System/370. OS/370 evolved into the OS/VS, OS/MFT, OS/MVT, OS/MVS, MVS/SP, MVS/XA, MVS/ESA, and finally OS/390 and the newer Z/OS available in the last few years.

Throughout this evolution, application programs written for any operating system of this type have always been able to run in any of the later operating systems due to IBM's guarantee of forward compatibility between operating systems.

2.3.3 MVS Products

An MVS system is a set of basic products and a set of optional products. This allows a customer to choose the set of functions they need and exclude the rest. The main user interface in MVS systems is TSO (Time-Sharing Option).

The Interactive System Productivity Facility is a set of menus for compiling and managing programs and for configuring the system. This product also provides for versioning, auditing, promoting code and configuration management to track all application components.

The main work management system is either Job Entry Subsystem 2 or 3 (JES2 or JES3). Storage Direct Access Storage Device (DASD) management is performed by DFSMS (Distributed File Storage Management Subsystem).

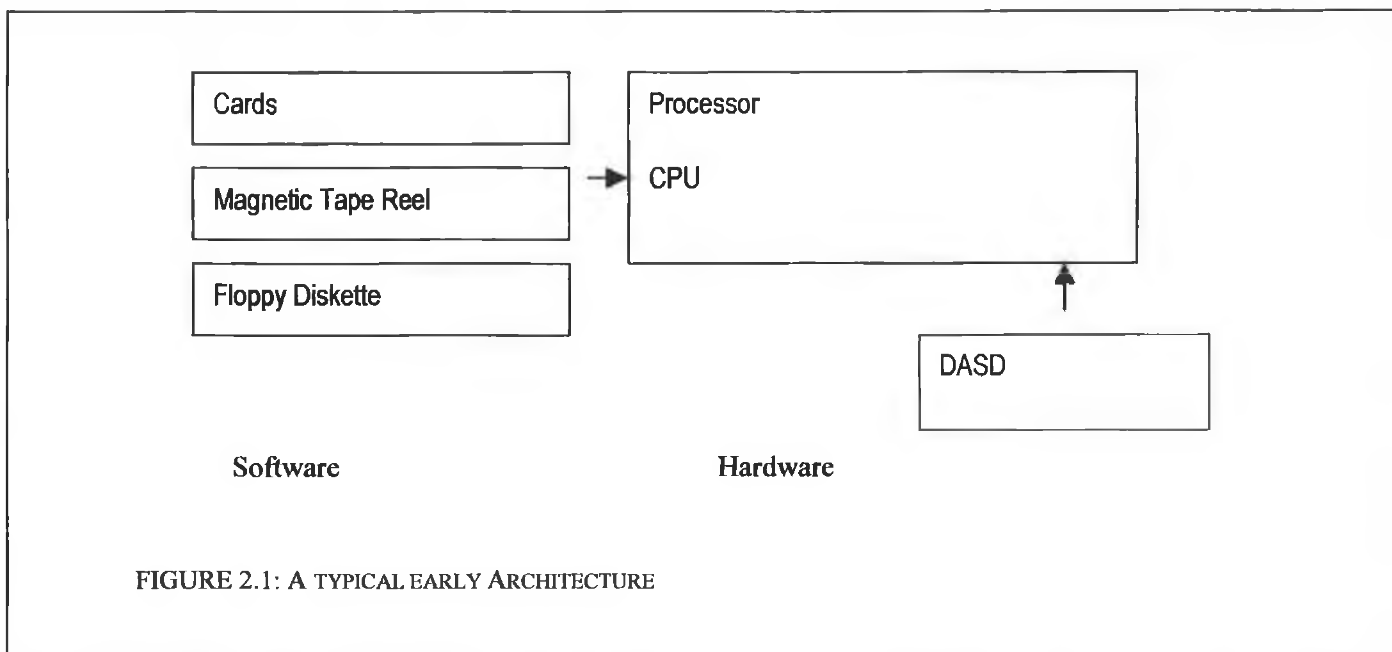


FIGURE 2.1: A TYPICAL EARLY ARCHITECTURE

The *Virtual Storage* in MVS refers to the use of virtual memory in the operating system. Virtual storage or memory allows a program to have access to the maximum amount of memory in a system even though this memory is actually being shared among more than one application program. The operating system translates the program's *virtual* address into the real physical memory address where the data is actually located. The *Multiple* in MVS indicates that a separate virtual memory is maintained for each of multiple task partitions.

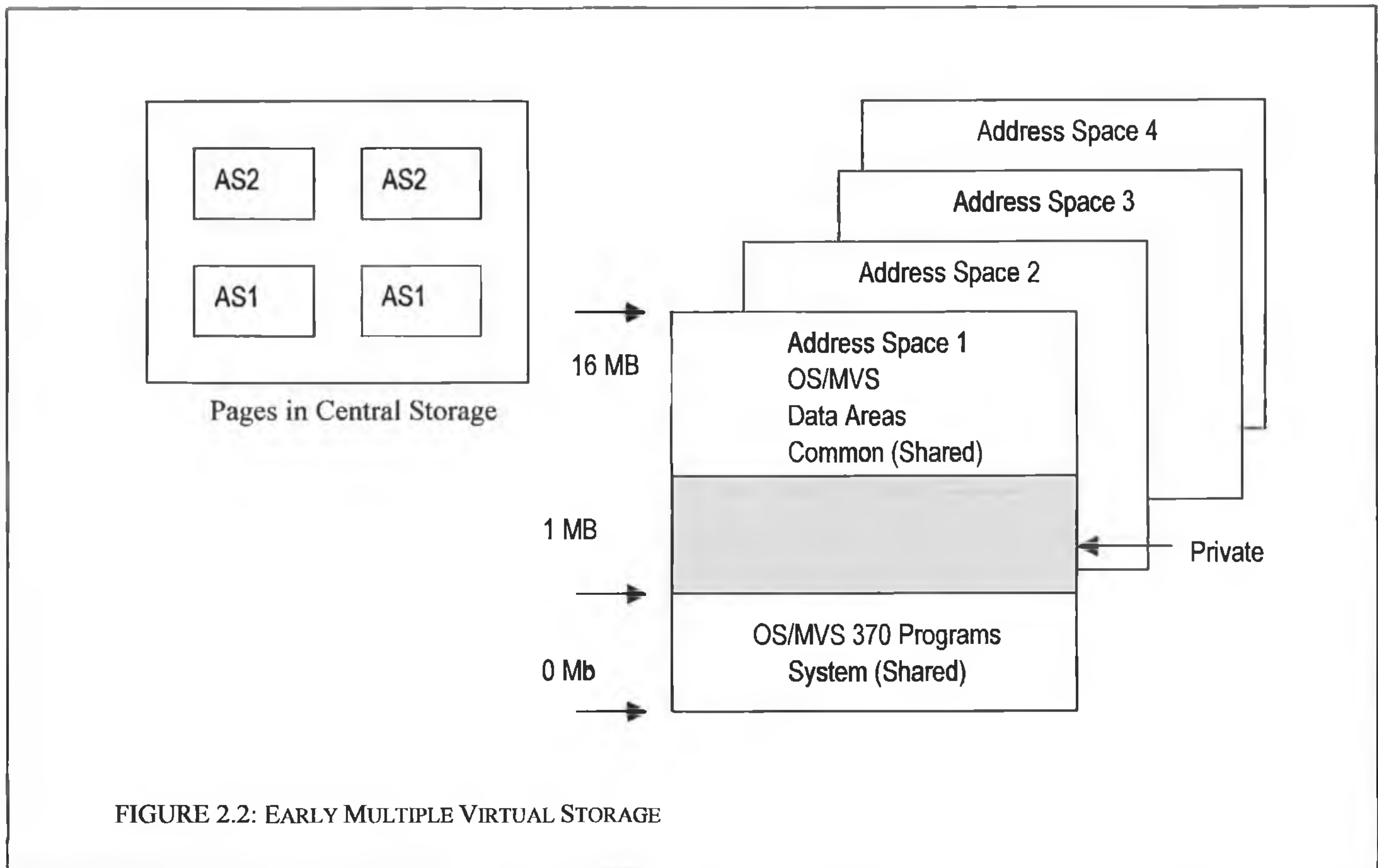


FIGURE 2.2: EARLY MULTIPLE VIRTUAL STORAGE

The MVS operating system is considerably more complex and requires much more education and experience to operate than smaller server and personal computer operating systems.

2.3.4 TPF (Transaction Processing Facility) and VM

Other IBM operating systems for their larger computers include or have included: the Transaction Processing Facility (TPF), used in some major airline reservation systems, and VM, an operating system designed to serve many interactive users at the same time.

The TPF platform is especially useful for business-critical systems. Airline reservation systems as well as many systems for railroads, hotels, government as well as financial institutions rely and trust this operating system.

2.3.5 Storage considerations

Back in these early days, central storage upon which the central processor processed operations was a costly and scarce resource. System/360 system control software was characterised by techniques for managing central storage:

- PCP dedicated it to one application
- MFT split it into several fixed pieces or partitions
- MVT varied the number and size of these pieces, calling them "regions"

With the advent of virtual storage, systems were freed from the constraints of Central Storage by supplementing them with less expensive, though slower, external software devices in a manner transparent to most software by employing Direct Address Translation (DAT) hardware. This allowed the migration of existing Central Storage operating systems into a virtual storage environment. **[Johnson 1989]**

2.3.6 Transactions

The main purpose of most mainframe systems is the provision of different services to a multitude of concurrent users. In a common scenario several thousand users interact concurrently with the same mainframe system.

Computer memory is relatively cheap today, however twenty years ago, providing an individual process for each user was not an option. To manage this large number of processes and store the status information for each in memory would have been too expensive so the solution to this problem involved each user sending requests for execution of a function. **[Koch, Murer 1999]**

This solution to the problem is what is known as a *transaction*. A transaction request contains all required information in a set of parameters. The function itself is a stateless programming entity (at least from the users point of view) and may be removed from memory as soon as the execution is finished.

Many banking applications were built as transactions under popular mainframe transaction monitors such as CICS and IMS for the very reasons outlined above. It is these applications that will be considered when investigating possible integration strategies with modern technologies.

2.3.7 Limitations and Benefits of the Mainframe Architecture

These early mainframe software architectures were popular for the following reasons.

- They were reliable.
- They were secure.
- They scaled well.
- They met the early business requirements of the banks.

As technology advanced however, certain limitations appeared.

- They did not easily support graphical user interfaces(GUI) or access to multiple databases
- Produces substantial network traffic
- Requires a complex operating System
- Expensive to maintain

2.4 First Banking Applications

2.4.1 What was available?

Older systems running these applications were typically first developed in COBOL. Other programming languages available included PL/I and FORTRAN and Assembler. These applications used the Virtual Storage Access Method access method for file management and Virtual Telecommunications Access Method for telecommunication with users. IBM's primary relational database management system RDBMS is DB2. Typical applications that were required included payroll, accounts receivable, transaction processing, database management, and other programs.

In the past, the banks' applications were almost exclusively realised as large program blocks executed on the host computer. Almost all applications were developed in-house, apart from elementary services such as databases and transaction monitors

2.4.2 Applications performing Business operations

Business operations applications perform the business transactions on behalf of the bank. These applications are crucial to companies. Typically these applications would include any or all of the following **[Horswill 2000]**

- Cash for goods transactions
- Any buyer/supplier transactions than can be translated into digital format
- Internal business processes dealing with company resources
- Credit Card Transactions
- Cash Transactions from a banks ATM or cash dispensers
- Stock market transactions for a stock exchange or brokerage
- Information Transactions for collecting, collating and distributing news.
- Payroll Transactions
- Logistics transactions (scheduling of vehicles)
- Voice application transactions
- Sales transaction

2.4.3 Essentials of a Business Applications

These Business Applications typically had the same features as each other. For example, in early business applications, there was a computer at the centre with people being the focus of the application.

The application's purpose was to keep accurate, up-to-date, secure operational business information and deliver rapidly to end-users. It needed to be fast, accurate, secure and auditable and information must be up to date and available to multiple users across a company, its suppliers, customers and business partners

The Responsibilities of application should be divided and there should be support for the various lifecycle requirements including Design, Development, Test and Update. There were certain technical requirements that had to met including Accessibility, Availability, Communication, Manageability, Prioritised use of hardware, Rapid Response, Reliability, Recoverability, Scalability and Security

2.4.4 Composition of a Business Application

A typical business application had code separated into components that managed the complex IT system. There were different responsibilities within the overall application. **[Horswill 2000]**

These responsibilities have traditionally been broken into two types of components. (Business Logic and Presentation Logic Components)

Business Logic Components typically had functions such as validating Input Data, searching the database, Cross-validating input data and database data, updating data (including additions and deletions) and log activities. These were all done according to the business rules.

Presentation Logic Components had as their responsibility, the presentation of data to the end user and the receiving data from the end user. They invoked general-purpose presentation management code controlling layout of data on screen or output device. They validated input, handled interactions in correct sequence, confirmed completeness and invoked business logic as needed

2.5 New Customer Demands and Pressures on the Systems

2.5.1 Early Technology Requirements

The first pressures for banking Technology to change came with the advent of Automatic Telling Machines (ATM) and Smart Cards that could be used to electronically pay for goods.

Routine bank services that once required a teller are now available through these ATM that allow people to access their accounts 24 hours a day.

Since the start of recent technology advances, the Internet and e-business have arrived and gone past being just buzzwords in the industry. They are now real requirements. There are pressures for change of existing banking applications coming from two major directions.

2.5.2 Changing Business Requirements

The fast pace of business process reengineering coupled with banking and corporate restructuring and mergers has stretched the flexibility of existing system to their limits. **[Froidevaux et al. 1999]**

In addition, requirements for new Services and Real Time availability have left this older technology creaking under the pressure.

2.5.3 Evolving Technologies

With the old 3270 "Green Screens" being replaced with Graphical User Interfaces came a requirement for more computing power. In addition the very nature of programming has changed with the arrival of Object Oriented Analysis and Design and the necessity to have an e-presence have also added to the pressures for change.

Specialisation and automation in banking have lead to dozens of special purpose systems (ATM, phone banking, trading systems, etc.) with hundreds of specialised interfaces among each other.

The software required today is being developed based on logical components and managers are placing emphasis on producing code that is transportable across the layers of n-tier architecture and that is reusable among business applications. Application software development, which is based on the business-component factory model strongly, supports business objectives of cost efficiency enhanced customer service, and quick time-to-market. **[Summers 2000]**

The previous chapter outlined some of the other concerns about how legacy systems are coping with these pressures for change. **[Kudrass et al. 1996]** outline a real-world example where, because of a lack of documentation and in-house know-how concerning the legacy MIS it has to be analysed and documented before being replaced.

In this case, the documentation is supposed to act as an online help system for the MIS administrator. The documentation should further include an analysis of the system's interfaces in order to maintain these more efficiently.

In this scenario, another well-known problem can be seen, where, as a result of multiple modifications over time the data structure of the MIS is messy and the data is partially inconsistent.

All of these concerns, when combined place a very real pressure on today's banks to either replace or upgrade their existing applications.

2.5.4 New Opportunities

However, banks do not have to simply be reactive and wait for difficulties to rise, rather, information technology provides some new opportunities. **[Chen et al. 2000]**

They can expand their product range using new technology and the resulting product innovation and diversification. This will expand the earning base and increase earning stability. In addition, automation of services becomes easier, an expanded customer becomes a target and the cost of market entry is lowered by this technology. Examples of this can be seen in the new purely-internet based online banks of the dot-com period.

2.6 Challenges for the future

As outlined in **[Murer 1999]**, there are conflicting challenges in the new role of IT in the banking sector. On one hand, new distribution channels require new technologies but a high reliability requirement along with enhanced security and performance, adds extra risk to the introduction of this new technology. There is further trade-offs between short project cycles required for the integration of standard applications compared with possible long project cycles due to cost restraints.

It is clear as discussed in Chapter 1 that as it is difficult for us to know how IT systems will look in 30 years time, it was equally as difficult for mainframe developers of 30 years ago to foresee where technology would be today. As a result these older architectures do not suit the development of new application that meet today's requirements.

Having said that it is also accepted that the mainframe is still at the core of many of the World's systems. It is also accepted that the full expertise as to how these systems work does not exist, but there is still a need to get at the information contained within.

Banks will be motivated to overcome obstacles such as systems incompatibility and consumer privacy concerns, to achieve greater operating efficiencies and to protect their valuable payments franchise. In addition, on-line purchases of sales and securities by individuals will also continue to increase, providing a growing source of commission for banks and financial institutions **[Mc Donough 1999]**

As shall be seen in the next chapter, there now exists a catch-22 situation where it is agreed that there is a need to modify these legacy applications but that there could be large risks involved. As a result several strategies for integration or reengineering are required.

The extra complexity of these new banking applications is also cause for some concern. Operating risk, in particular, has attracted more attention in recent years, partly because improvements in technology and data storage permit institutions to retain and analyse more data and also because the increased volume and complexity of bank transactions have, arguably, increased this risk for many banks. **[Ferguson 2002]**

2.7 Conclusion

In this section it has been discussed how the nature of the banking sector relies on technology to service its customers. A short look has been taken at early mainframe architectures and it can be seen how these are no longer adequate for today's business and technological requirements. In the next section there will be an investigation about various methods for integrating these older systems with today's technology.

As outlined in the aims and objectives of this research, a key focus is to see possible difficulties with integrating older mainframe technology with modern object-oriented technologies. This chapter gave an overview of the mainframe operating system plus the applications that run on it. Some of the pressures from business and technology that are forcing industry to integrate these different technologies were seen.

It is possible to see that there is really no alternative but to integrate or replace the legacy systems and the next step in the research will be to see what options are available today to facilitate this migration.

These results make the objectives of this research somewhat clearer. There are pressures on modern Banking IT Systems to adapt to newer technologies. However, the existing systems in place are not easily replaced. The next step of the research is to examine some integration strategies available today.

3 Approaches to Integrating Legacy Systems

3.1 Introduction

This chapter introduces the various approaches that are available in today's market place for integrating legacy systems. Each of the approaches is being used in real world projects but some are more suited to different kinds of projects.

There are currently many definitions of legacy systems. As outlined in Chapter 1, the definition by **[Brodie Stonebraker 1995]** best matches the problem facing the Financial Sector today. **[Bennett 1995]** extends this to state that legacy systems are those that cannot be easily changed with but that are vital to the organisation. A narrower definition for our purposes, and as defined by **[Juric et al. 1999]**, states that

> *A legacy system has existing code, is useful today as well as in use today and the architecture of the system it resides on is not distributed and not object oriented.*

This is commonplace with old mainframe based IMS and CICS transaction. This definition is a somewhat limiting, as a legacy system does not have to be an old system. However, as this thesis focuses on legacy systems as those running on the mainframe platform, it is a valid definition.

Another view of these systems is outlined in **[Lauder, Kent 2000]** where it can be seen why organisations are afraid of replacing their legacy systems, as it is a significant drain on the organisation's resources. Another reason is that they have undergone years of debugging effort and truly reflects the workings of the business.

This chapter will look at exactly these of types of legacy systems and consider how best to bring them into synchronisation with the newest technologies and the changing business requirements.

3.2 Replacing the entire banking system

In principal, the simplest solution to the problem of outdated banking applications would be to replace the whole system and start from scratch. Looking a little deeper into this solution however, one can quickly see how difficult this would be in reality. As mentioned in the previous chapter, many banking applications have had up to 30 years of continuous development and have reached a degree of integration and sophistication that would be extremely difficult to replace without major effort.

[Lauder, Kent 2000] warn how the legacy system itself is often the only source of recorded domain expertise. The question that arises is whether the new systems will really contain all the functionality of the older systems. The danger of building such a new system would be that it just contains the knowledge of which the business is aware and misses much of the implicit knowledge added into the older systems over decades. Missing this knowledge could result in further updates required to the new systems, which are costly in development terms and customer trust terms.

However this solution was found also to be not financially viable **[Erlikh Goldbaum 2001]** and could take many years to implement at a cost of many millions and even then it is particularly difficult to describe the requirements for a banking system five years into the future. **[Koch Murer 1999]** outline how a first estimate for Credit Suisse showed that it would cost roughly 800 million Euro over a period of 5 to 7 years to replace their existing mainframe solutions.

Another approach would be to adapt the system from another organisation that had already migrated their legacy technology. This would be a way of getting a mature system quickly. But it has been found that this has only really worked historically in merger situations and has not solved the problem for large organisations **[Koch, Murer 1999].** It was also found to be not possible to buy a complete solution, as there simply is not a large scale, complete application that matches all of the requirements of a modern bank in existence.

For many organisations, the only solution is to try and re-engineer their existing systems so that they can work side by side with the newer technologies.

3.3 Approaches to Integrating the Legacy Systems

Some obvious choices for integrating our mainframe applications with new technologies include emulation, screen scraping, using adapters to pass information, and full integration between the two paradigms.

Generally, there are two possible solutions to modifying a legacy system **[Jiang 1998]**

- Legacy extension
- Legacy integration.

Legacy extension is a means of matching the short-term requirements placed on an existing system by fixing system deficiencies and adding enhancements.

Legacy integration however, attempts to reuse the legacy system to implement a new architecture. The idea is to hide these systems behind consistent interfaces that hide the implementation details. It is a solution that tries not to propagate the weaknesses of past design and development methods. Unlike legacy extension, this is a long-term solution as the use of interfaces allows for changing or replacing the implementation at a later date without affecting other systems.

Each project will need to decide on an integration strategy that best suits the needs and requirements of that project. This strategy would have to be based upon different criteria that will be investigated throughout the rest of this section but include obvious topics such as cost, risk involved, levels of intrusiveness etc

3.3.1 Level of intrusiveness

As mentioned in an earlier chapter, one of the biggest headaches in re-engineering or integrating legacy code is to really understand the older system. If a company has full access to all the documentation and all the source code it might be relatively painless, but if there is not full knowledge of how the older system worked, this might be impossible.

This knowledge of the existing system can be crucial in determining the level of invasiveness of an integration strategy. **[Morris, Isaksson 2002]** compare the problems associated with the different levels of invasiveness. When a solution actually intrudes upon the functional legacy code there is a risk of tampering with the programming as the code is often old, may have been extensively modified and the original developer may not be available. A non-invasive approach usually involves working from the legacy green screens or source files.

If a more invasive approach in chosen, and attempts are made to change this original code, there is a risk of compromising the integrity of the enterprise data. There are some benefits to the less invasive approach:

- The Business logic has been tuned and refined over time
- Developers do not have to change mainframe application logic
- Strategic business processes remain intact
- Risks of reuse are minimal and returns typically high
- Development is rapid and costs are comparatively low

A common non-invasive approach is integrating purely at the presentation level, i.e. screen-scraping. This involves an Object Oriented client navigating its way through the legacy application screens and submits and retrieves results through screen fields. The advantages above are clear but there is a negative side. These negative points include:

- System is still not truly understood
- Future changes difficult
- Inefficient

A Company could opt for integration at the functional level. Using this approach would involve

1. Modularising and componentising the existing legacy applications
2. Wrapping these components with an Object Oriented wrapper
3. Integrating these Object Oriented components.

The benefit of choosing this approach above the others would be that it is now possible to share functionality as well as data in a peer to peer relationship.

3.3.2 Reusability and Robustness

Any integration strategy needs to be considered in terms of its Reusability. When any of these approaches are used and the system requirements change, the question will be posed as to how easy will it be to implement or update the existing infrastructure.

Also in question will be the robustness of the system, if a screen-scraping approach is taken. Compare this robustness to that of a system with full integration between the legacy system and the rest of the system. A reason for this comparison is that projects that opt for a screen-scraping solution run the risk of having a less robust solution because the internals of the application are not modified and there is less chance to adapt the application to the new architecture. This is an important area that must be investigated at an early stage.

There is a fundamental problem in that legacy applications do not look like distributed objects and this makes it difficult to reuse these systems **[Juric et al. 1999]**. Attention must be focused on the object interfaces, so that legacy applications are encapsulated in order that collectively they implement reusable, virtual distributed objects.

Enterprise Systems require an infrastructure that contains the various components of the system. Certain approaches require less work building such an infrastructure and this point can be linked to the intrusiveness of a given approach.

3.3.3 Complexity

When an integration approach has been decided upon and is implemented, it must be estimated how complex will it be to actually use this system. Some of the less intrusive approaches require very little extra effort in terms of building an infrastructure but then to actually use the approach can cause some headaches.

[Reddy 2002] outlines an example where the green-screen custom application that provided all the information for a bank's employee to do their jobs is replaced by a GUI enterprise package application. In order for the employee to do their job, they must now navigate 14 different screens as opposed to just one screen in the older customised application. In an example like this, no amount of GUI or computing power will replace the ease of use of the older application.

3.4 Emulator

3.4.1 TN3270 Protocols

TCP/IP based mainframe connectivity is most often furnished by the TN3270 protocol **[TN3270 2001]** which enable TCP/IP nodes to emulate 3270 terminal sessions and deliver 3270 functionality to the desktop over TCP/IP networks. This protocol is one of the most popular means of desktop to mainframe connectivity with an estimated 23.1 million clients in use in 2000. The hierarchical structure SNA (Systems Network Architecture) devised by IBM for mainframe connectivity is being abandoned in favour of TCP/IP – a protocol noted for its openness, extensibility, manageability and real world functionality. These features make it ideal for enterprise connectivity.

3.4.2 Using the Emulator Approach

Using an emulator simply brings 3270 screens to the desktop with no intrusiveness into the legacy application and no GUI functionality as well. There is little overhead but there is no real gain in terms of new GUI and OO technologies.

The original design of 3270 devices deliberately favoured high performance over user interfaces. From the early 1970s, 3270 devices brought transaction processing to the end user. A key design feature was to provide useful data to the end user over the network in less than one second. This was to be completed with a bandwidth that would be considered to be abnormally narrow today. **[Horswill 2000]**

3.5 Screen Scraping (Presentation Layer Integration)

3.5.1 What is Screen Scraping

Screen Scraping is the presentation layer integration approach where an OO client navigates it's way through the legacy application screens and requests and results are submitted and retrieved through screen fields.

[Froidevaux et al. 1999] outline how this is achieved with traditional legacy systems. In the case of IMS and CICS, transactions are terminal oriented. The transaction receives a request from a terminal, processes the request and sends the response back to the terminal. There is a terminal session manager that is responsible for managing terminal sessions. Terminals handle form-oriented user interfaces. Forms contain constant text and a set of input and output fields. The user fills in the input fields and after completion transmits the form. Input fields are transmitted to the terminal session manager who sends the request on (to the TP monitor scheduler in a TP monitor such as IMS). The TP monitor handles the request.

[Stroulia et al. 2000] outline an experiment to integrate a legacy system that uses a block-mode transfer protocol between the mainframe application host and the user terminals such as the 3270 protocol. The idea behind the project is that during its interaction with a user, the legacy application goes through a sequence of distinct behavioural states, which correspond to the distinct screens that it forwards to the user. The implication then is, that identifying the distinct screens that an application may forward to its users corresponds to identifying the behavioural states that it goes through, during its interaction with the users. This is an example of a project successfully using screen scraping to access their data.

3.5.2 Advantages and Disadvantages

The Screen Scraping approach to integrating legacy systems is known for being "quick and dirty". This approach is relatively easy to implement and to get access to the legacy's applications core via the screens is relatively straightforward. However, there is still no real insight as to how the application works and any functionality changes that are required by new business requirements cannot easily be met.

With the focus of screen scraping on the user interface rather than modifying core process models, screen scraping offers the least benefits to business **[Erlikh, Goldbaum 2001].**

3.6 Functional Level Integration

Another approach involves componentising the existing legacy system and wrapping these components with an OO wrapper. This can then be integrated with OO components and functionality and data are shared. The result of this approach is a peer to peer relationship between OO and Legacy Systems.

Integration with wrapping makes legacy systems look like distributed objects. **[Juric et al. 1999]** These wrapped objects are like any other object in the distributed system and clients do not have to know about the implementation details

This is essentially the approach that adopts the middle ground. Intrusiveness is kept to a minimum in as much as possible but there is also have a chance to get access to the functionality of the legacy systems.

3.7 Using CORBA for Functional Level Integration

The CORBA framework allows us to define interfaces for components and provides a transport mechanism for requests between components. Appropriate components must be identified and their interfaces defined. The next step is to design the newly integrated system to use these components.

Even within the CORBA world, there are differing degrees of invasiveness.

3.7.1 CORBA and Screen Scraping

There is a possibility with CORBA that allows our legacy applications to communicate with real distributed objects in the system but not being wrapped themselves. This is a halfway solution but still has all of the problems of traditional screen scraping.

Terminal oriented transactions can be re-used by simulating a terminal. A form is modelled by a two-dimensional 25*80 character array. The CORBA client fills in the input-fields into the array and submits the request to the terminal session manager. The response is received as a character array and the CORBA client extracts the output fields.

IONA Technologies provide a static adapter for this purpose. This adapter is a standard CORBA server that implements a fixed set of CORBA interfaces for mainframe communication. The implementation of an operation might call a transaction for example. CORBA input parameters would be serialised for the receiving transaction (in a string). The request is parsed and returned as CORBA out parameters. Static adapters can be automatically generated from a CORBA Interface Definition Language (IDL). If the adapter contains application logic, manually written hooks (extra application code) have to be written.

3.7.2 CORBA Adapter Technology

The CORBA Dynamic Adapter permits existing transactions to be exposed in IDL and thus connected into the wider CORBA-based computing environment. An example of a dynamic adapter is the Orbix IMS Dynamic Adapter from IONA Technologies.

From the CORBA perspective, strictly speaking, the Orbix IMS Adapter is a conventional CORBA server. From the client perspective, it simply presents a set of client-callable IDL interfaces. In practice the Adapter, upon receiving a request from an Orbix client, consults a pre-defined mapping repository, looks up the appropriate IMS transaction name keyed on the incoming interface and operation names, and submits the request to the relevant IMS transaction.

When the IMS transaction receives control via the normal IMS dispatching process, it uses a set of Orbix-provided services to read in the operation's arguments and to return a result. The adapter will then return the results of the operation to the client.

A dynamic adapter performs the same operations as a static adapter but implements a dynamic set of interfaces. Dynamic adapters can be built analogously to inter-ORB bridges. They are implemented as CORBA dynamic skeleton interface (DSI) servers or as a GIOP (generic inter ORB protocol) routers. Because the adapter does all the necessary marshalling and demarshalling the request can be handled in the transaction without stub and skeleton code. **[Froidevaux et al. 1999]**.

In the case of the Orbix IMS Dynamic Adapter, an adapter is a running CORBA process. It imports its set of interfaces from an Interface Repository (IFR) and has a mapping file which maps each interface to an IMS or CICS Transaction. When a client request reaches this adapter, the request will be re-directed to the correct Transaction based on the interface name.

CORBA supports the use of this approach rather well. DSI objects allow the routing of requests of any interface available in the Interface Repository (IFR).

3.7.3 Full Integration

Due to the nature of existing CICS and IMS transactions, it is difficult to have these as true CORBA objects in a distributed system. However, there are industry solutions available that allow them to be seen as CORBA objects without the use of Adapter technology.

Take for example a legacy CICS transaction. A full integration with CORBA can allow a transaction to receive CORBA IIOP (Internet Interoperability Protocol) requests without the use of screen scraping and adapter technologies. This requires having a full CORBA implementation existing inside CICS and due to the short running nature of traditional transactions this is still not a perfected technology.

After deciding to adopt this approach the legacy system and our OO system must co-exist and co-operate. There must be transparency of implementation and technology. There must be consistency of interfaces and the system must act as peers. As has been seen earlier, CORBA provides all of these and more and consequently is the approach best suited to such integration.

Another popular method of having CORBA on the mainframe is by building CORBA objects on Unix System Services (a Unix Shell on OS/390), or as BATCH Jobs. It is possible to build C++ (or Java) CORBA objects on this platform that are true CORBA objects. Also possible is the use of Linux for OS/390 to hold these CORBA objects.

3.7.4 Why CORBA

One of the worries for banks before introducing a CORBA system is the perceived complexity of CORBA, particularly for the mainframe developers to understand the concepts of Object Oriented distributed programming.

However, the benefits of using CORBA to integrate legacy applications usually outweigh these fears. **[Murer 1999]** outlines just why Credit Suisse chose CORBA to integrate their mainframe applications.

- There is clean construction of heterogeneous distributed systems
- CORBA is a clean model with complete middleware functionality
- There are clean interfaces for application integration
- CORBA supports bridges to other important component standards that may be available on the market. This enables integration into standard software
- PL1 and COBOL CICS and IMS Transactions can be accessed from Java and Smalltalk clients
- There is a decoupling of interfaces from technology
- CORBA has mature implementations on many platforms (including OS/390)
- CORBA is a standards based solution

CORBA is becoming a mature and trusted technology that can be used for integrating legacy systems. However, technology is moving forward at a faster pace than ever at already there are alternatives on the market to CORBA that could be used in its place.

Two of these alternatives, namely XML (Extended Markup Language) and J2EE (Java 2 Enterprise Edition) Connectors could, when fully mature reduce even further the complexity of integrating Legacy Applications.

However, it should be noted that large corporations, especially banks are wary of non-mature technology and are usually among the last to "jump on the bandwagon" of advances that will "solve all their problems". The two technologies mentioned below are still in their infancies and until they are proven in enterprise solutions, they will be only regarded from a distance by large banks.

3.7.5 Using J2EE Connectors

[Hermansson, Akerlund 1997] outline the use of J2EE Connectors which are a part of the J2EE specification and manage integration with existing Enterprise Information Systems, EIS.

The J2EE Connector architecture provides a Java solution to the problem of connectivity between the many application servers and Enterprise Integration Systems already in existence. Using the J2EE Connector architecture, EIS vendors no longer need to customise their product for each application server.

This connector architecture is gaining popularity in the industry. It is also a standards based solution, but does not make legacy CICS and IMS transactions available as distributed objects. It is a Java based solution and does not completely suit the PL/I and COBOL based applications that require integration. Finally, there are industry concerns about the performance of Java and Java based applications within enterprise systems **[Chang 2000]**

3.8 Data Level Integration

This is another integration strategy that exists. In this approach, the OO and Legacy Systems share access to a Database Management System (DBMS). All existing DBMS's are accessed via a Meta DBMS.

With this approach there is only shared data, therefore the OO System cannot access the functionality of the legacy system and there is still a need to rewrite the business functionality on the client.

Using this form of integration has many of the same positives and negatives as screen scraping. It can be relatively easy to get at the legacy data but again, any new business requirements that are placed on the system will be problematic.

A newer form of Legacy Integration comes with the introduction of XML to the mainframe world.

3.8.1 XML

XML is a World Wide Web Consortium (W3C) standard for representing complex data with human readable labels and structuring.

XML allows complex data to be published with both context and structure preserved. This complements the closed, encapsulated data structures of CORBA and OO languages.

XML is encouraging a new generation of data centric legacy integration solutions. These solutions no longer need object wrappers or proprietary Enterprise Application Integration (EAI) solutions. Any applications that use XML to communicate will need either to understand XML or be wrapped in some form of translator.

XML essentially frees data from its dependence on software infrastructures. Its hierarchical data model makes it well suited for storage in object-oriented databases, and database vendors have been quick to leverage their edge by adding XML storage and query capabilities to their product offerings. **[Coyle 2000]**

In terms of its relationship to a middleware technology such as CORBA, XML provides a standard way to represent complex, object-oriented data without forcing all software components to use the same underlying OO or relational database. This type of database independence translates into greater flexibility when using a middleware technology such as CORBA to integrate diverse types of components into a single distributed application.

3.9 Conclusion

In this section there has been a discussion of some of the approaches available on the marketplace today for integrating legacy applications with object oriented solutions. These approaches vary from "quick and dirty" solutions to full integration. Moving forward, after examining the factors

affecting an integration strategy, there will be an investigation of the best approaches for integrating sample legacy applications using CORBA.

There was a brief overview of some new technologies that are gaining momentum as integration strategies. These however are immature technologies and the banking industry will not be prepared to jump in until they are tried and tested.

As pointed out by **[Harding 2001]** finding people to program the older applications is becoming increasingly difficult so by making these systems available with CORBA in the distributed world can take away from some of these headaches.

At the moment CORBA offers the best solution for long term success while still maintaining the power of the legacy applications.

Alternatives to CORBA include messaging systems from large IT companies, such as IBM's MQ Series product. Typically if an organisation wants asynchronous communication they will chose MQ Series, but if synchronous communication is required, CORBA is the favourite.

As outlined in the previous sections, the first aim of this research is to investigate possible strategies for integrating the mainframe based legacy systems with modern Object Oriented and GUI-based technologies. The approach that is recommended by this research is the CORBA standard by the OMG.

The next step in the research is to examine this CORBA standard a little more closely to see if actually provides all of the advantages outlined in this section. One of the aims of this research was to provide a solution for integrating mainframe based applications and it would appear that the CORBA standard is the best approach that provides this.

4 Inside CORBA

4.1 History of Distributed Transactional Computing

4.1.1 Centralised versus de-centralised

The history of the computing industry has switched between a centralised model of computing and a de-centralised model. The early banking systems investigated previously were very much oriented towards a central server with many clients [**FIGURE 4.1**].

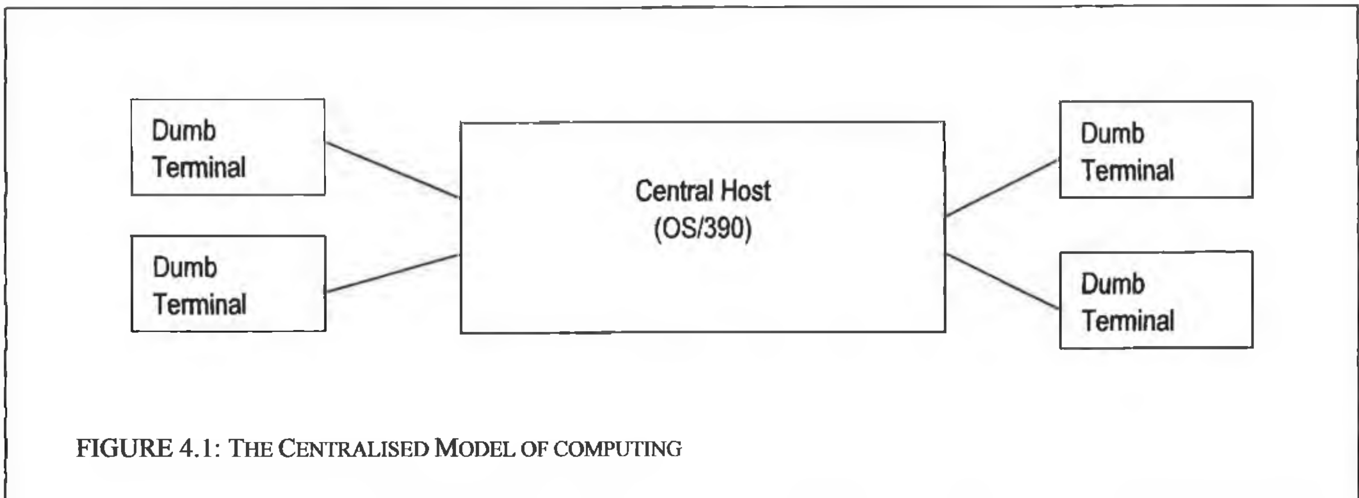


FIGURE 4.1: THE CENTRALISED MODEL OF COMPUTING

In recent times there has been a definite move away from this centralised model to a scenario where de-central platforms can perform their own computing.

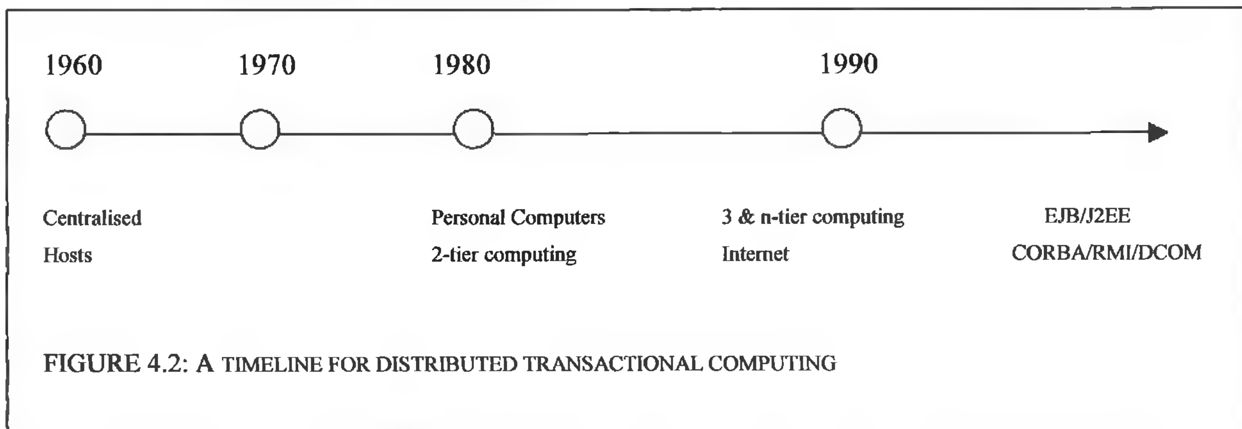


FIGURE 4.2: A TIMELINE FOR DISTRIBUTED TRANSACTIONAL COMPUTING

4.1.2 GUIs and Smarter Clients

The dominance of the central platform began to change in the early 1980's when client/server computing began to appear.

This movement started around the same time as the arrival of the Personal Computer and immediately proved popular due to the possibility of stylish GUIs (Graphical User Interfaces).

Some of the computing power and business logic was also moved to the client side and these clients became "smarter".

The technical roots of object technology led to the development of several other object languages, in particular C++ **[Stroustrup 1991] [Stroustrup 1992]** and Smalltalk **[Goldberg, Robson 1989]**. Only later was object technology applied in other areas such as databases, operating systems and the analysis and design of information systems.

4.1.3 The Client/Server Model

As more power went into the clients, the industry saw a shift further towards the decentralised model. Some immediate benefits were felt as PC development environments quickly became available and evolved to make client side GUI development a popular role.

These early Client/Server systems were very much 2-tier models. Such a model would include a "Fat" PC client connecting to a database server. These "Fat" clients would contain both GUI presentation logic and business logic as well as the code to access the database. The client would use database APIs for transactions.

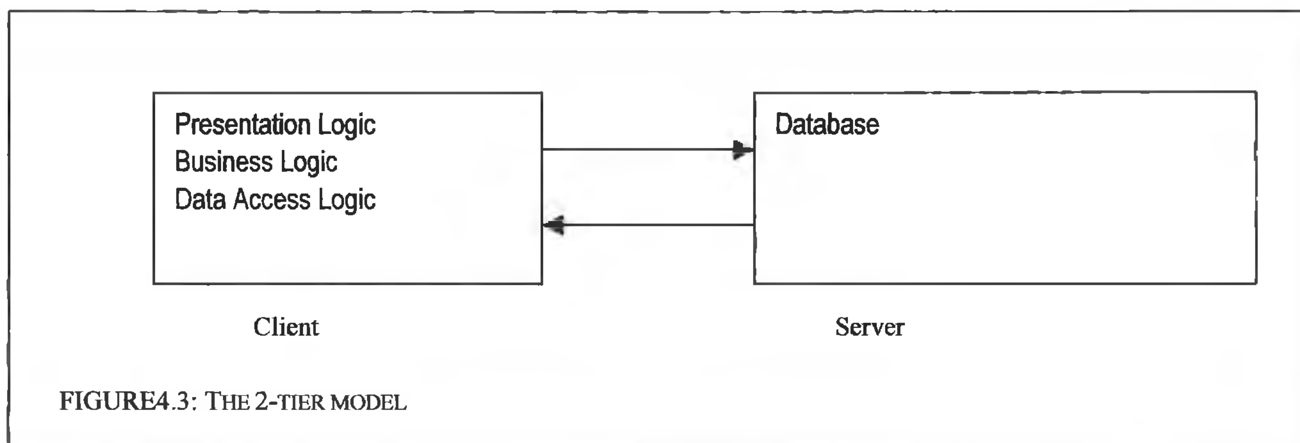


FIGURE4.3: THE 2-TIER MODEL

The immediate disadvantages of this 2-tier model were apparent very quickly. When presentation logic, business logic and data access logic are all mixed together, reuse, and performance immediately lose out.

Database servers suffered frequently from bottlenecks and, as a result, the scalability of the system suffered. This was heightened by the fact that a database connection for every client caused even more scalability difficulties.

Maintenance was difficult as application and database drivers needed to be installed and configured on every client and so some versioning or deployment strategy was required but was not easy, as the entire software would need to be reinstalled for upgrades.

The fat clients make maintenance and reuse difficult. Update and maintenance costs are high, because changes have to be re-deployed on every client. Any change in the logic must be redistributed to all clients. They are also difficult to use because applications are tightly bound to data schema since the client contains SQL queries.

They can also cause high network traffic because data is transferred for processing at the client. Finally, database connection costs are high because there is one database connection per user with no connection pooling or multiplexing.

The advantages of such systems included fancy GUIs and nice user interaction, especially when compared to the 3270 Screens many operators were used to.

4.1.4 3-tier Computing

Given the problems with the 2-tier approach, it did not take very long for a new approach to come on the scene. This was the 3-tier model of computing.

The difference between the 2-tier and 3-tier model was quite simple: separate the presentation logic from the business logic

Each of these tiers would be implemented on a separate machine. The middle tier would take care of the business logic and the business methods would run on the server.

The client would then make "requests" that the server would execute these methods. The client and server use a protocol that represents a conversation at the level of business transactions, instead of at the level of SQL.

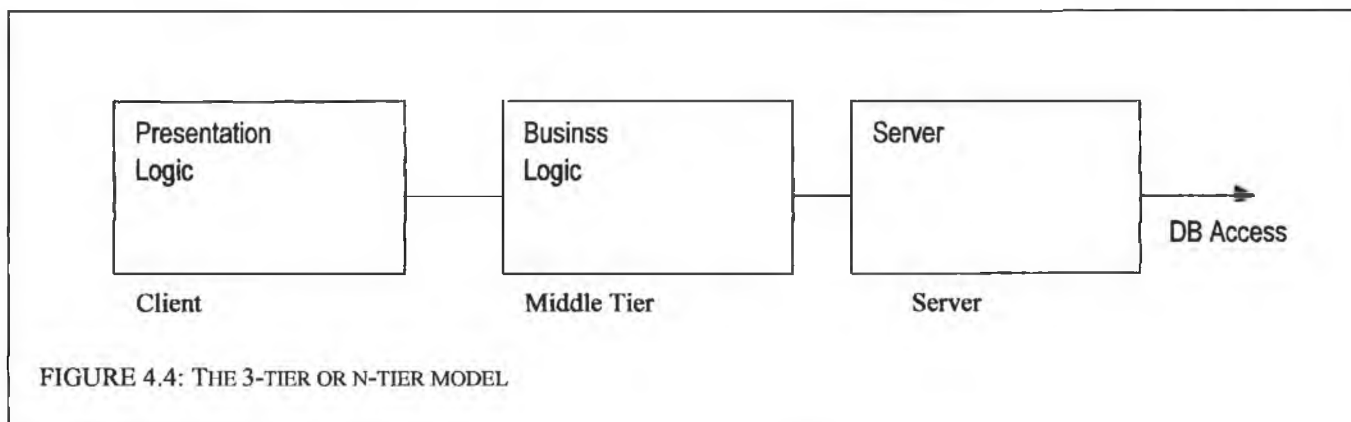


FIGURE 4.4: THE 3-TIER OR N-TIER MODEL

4.1.5 Difficulties with 3-tier computing.

The 3-tier and n-tier models solved many of the problems of 2-tier computing. With this model, clients were far easier to keep up to date as they separated presentation logic from business logic and re-use became a possibility.

Various new problems became apparent however. For example, applications became significantly more difficult to program as programmers needed to manage multithreaded concurrency, transactions and security themselves.

4.1.6 Remote Procedure Call

Remote Procedure Call came along with the advent of n-tier computing. RPC is a standard and transparent way to call procedures remotely. This is a type of protocol that allows a program on

one computer to execute a program on a server computer. Using RPC, a system developer need not develop specific procedures for the server. The client program sends a message to the server with appropriate arguments and the server returns a message containing the results of the program executed.

4.1.7 Object Oriented n-tier computing

RPC was not object-oriented and the next phase in the distributed computing world was 3 tier computing using objects. Here there was more than one choice.

CORBA: Object Request Broker is equivalent to object oriented RPC. It allows clients to communicate with remote objects. Interoperability with a wide variety of software follows. The communication protocol IIOP is an inter-orb protocol so the Financial Institution is not tied to a specific implementation.
RMI: This is the Java Object Request Broker from Sun Microsystems. It is also like an object oriented RPC. It is simpler to use than CORBA but only for Java to Java communication. There is now RMI over IIOP allowing Java to non-Java communication.
DCOM: This Microsoft product allows modules to communicate remotely, like a binary RPC. But distributed computing, with multiplexing, connection pooling, concurrency and multithreading is difficult to program.

Only the first of these three (CORBA) was going to prove useful for interacting with legacy systems as most of these older systems were not written in Java or C++ and were not Microsoft based. CORBA was the only option that really allowed communication with any older platform and language.

This does not just apply to the Banking Industry, for example **[Sang et al. 1999]** outline how many scientific applications in aerodynamics and solid mechanics are written in Fortran and that refitting this legacy Fortran code with CORBA objects can increase the code reusability.

4.1.8 Internet Revolution

The next revolution to come to the industry that everyone wanted to be a part of was the Internet. This really changed the way companies did business and as was seen, the banking sector needed to be a part of this change if they wanted to stay ahead.

Geographical barriers of enterprise LAN no longer became an issue and the format of clients started to change to much "thinner" browser based clients.

However, the quick pace towards decentralisation started to go into reverse. With the arrival of HTTP, HTML, Java, Applets, JSP etc, the concept of the "fat" clients became redundant and large number of "thin" clients became popular as they could have the same functionality as their large counterparts.

Because of this defined, but limited, set of functionality, this created the trend of centralising business logic on the server. The server in this context was no longer a mainframe or mini-computer, but any machine running a web server.

4.1.9 Different Connectivity Models

Applications that were starting to be based on this new model with an Internet Browser based client immediately had to choose between one of many connectivity models. Examples of these included **CGI (Common Gateway Interface), NSAPI (Netscape Application Programming Interface),** and **ISAPI (Internet Server Application Programming Interface)** to name a few.

The web server concept also caused headaches as it tried to provide the connectivity between the browser-based client and each individual enterprise applications. Maintenance was once again very difficult, as was the programming of these applications.

Using the Internet was obviously a big advantage in terms of business requirements, as the application became available to any user with a desktop and browser.

Disadvantages include the fact that mission critical transaction oriented applications can not be done easily and maintain-ably with CGI and Perl.

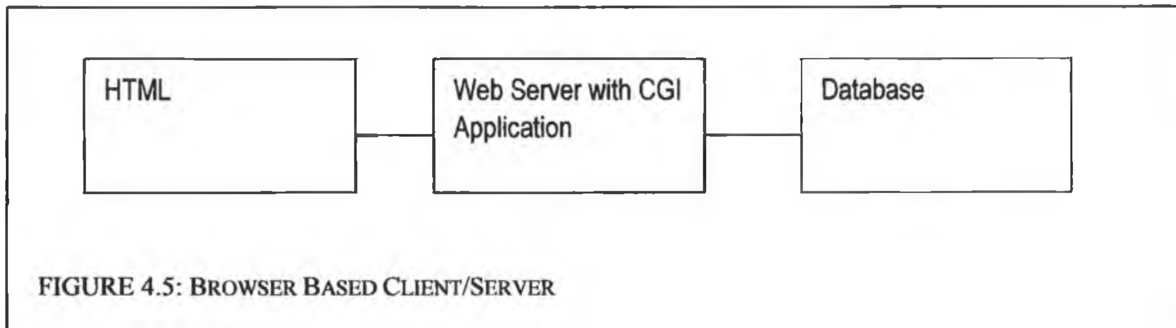


FIGURE 4.5: BROWSER BASED CLIENT/SERVER

In most of these cases a monolithic application was created on client and server sides. This monolithic application was typically composed of one binary file. Any changes meant recompiling and redeploying the application. This made it difficult to maintain because the requirements and environment may change and updates may be needed.

4.1.10 Component Applications

This changed for the better with the advent of **component applications.** These software components can be changed or updated without recompiling and replacing the entire application, like hardware components can be changed. They simplify the deployment of updates. Upgrades and bug fixes are easier to make.

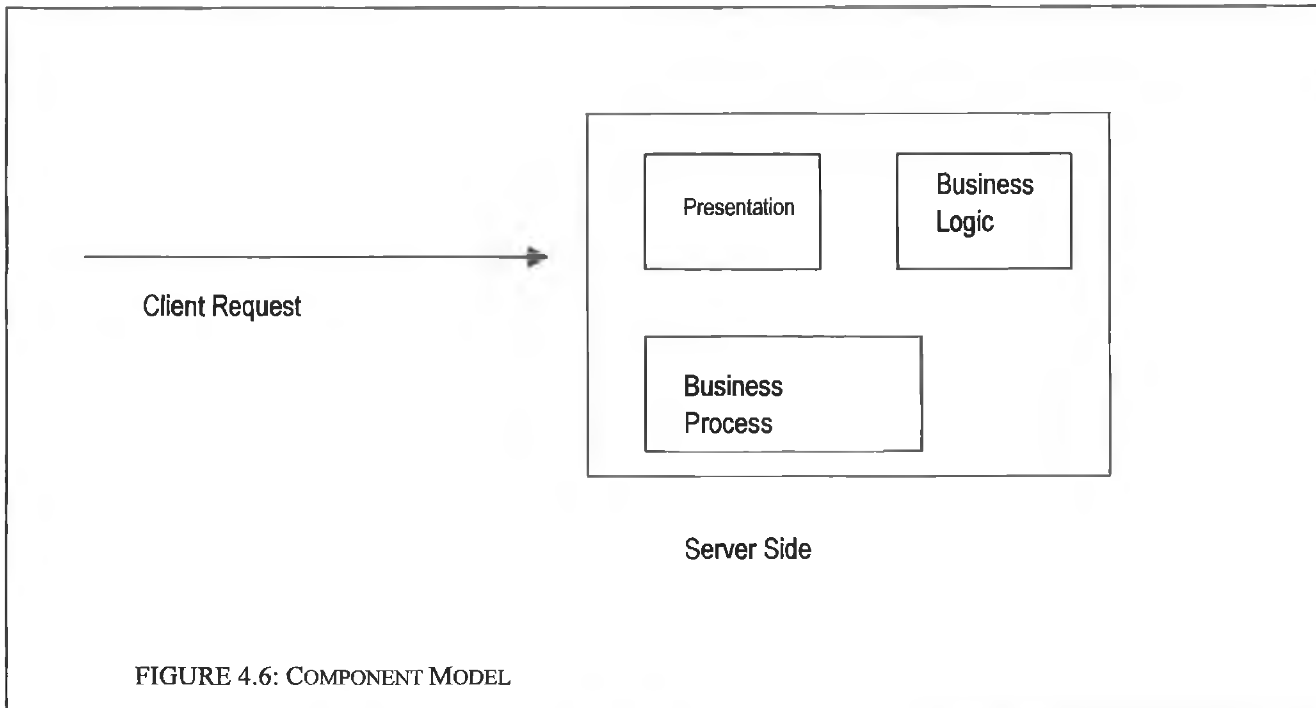


FIGURE 4.6: COMPONENT MODEL

With components, the application is separated into logical separate pluggable parts like presentation parts, business process parts, business logic etc. Component programming allows building an application using pre-built components with well-defined interfaces. Implementation is separated from the interface, meaning that the implementation can be changed without changing the interface and the other components using the interface. This reduces complexity, because other developers do not have to understand how it is implemented, just how to use it.

4.1.11 Improvements in Design

Dividing an application into components improves the application's design. Component programming forces developers to define the application in terms of well-structured objects, which have well defined interfaces so that they can interact properly with each other. Business logic can be reused; multiple instances of the same component can be used in multiple applications. Updates are easier because it allows the changing of a part without changing the whole. Development is easier because it allows the testing and building of small parts incrementally, and to divide development into smaller parts developed by different people.

4.1.12 General benefits and problems with N-tier computing

N-tier computing has various advantages. One tier can be changed without changing the rest, there will be lower development costs and maintenance costs are a reality. Resources can be pooled and re-used. Applications become more flexible, scalable and performant. Thin clients can be made available on the Internet.

It does however match these advantages with a new level of complexity when programming multi-user sessions. Thin-client multi-tiered applications are hard to write because they involve a

lot of complex code to handle transaction management, multithreading, database connection and resource pooling, performance issues etc. Developing distributed applications is difficult and requires highly skilled and experienced people

4.2 Introduction to CORBA

4.2.1 Difficulties of Distributed Object Programming

Today's IT Systems have become even more diverse that their counterparts of the early days of computing. This is especially true in the world of network programming where the diversity of the networks needing to be integrated makes the task quite a challenge. Coupled with this, there are many programming languages available and in-use today. Sometimes even these languages will perform differently on different operating systems.

In the area of object technology, much work is being done to make objects interoperate in a heterogeneous networked environment. The standardisation efforts of the Object Management Group lead to a general API for distributed objects, the Common Object Request Broker Architecture. The most recent version of this specification is the CORBA 3.0 Specification.

As per this OMG Specification, CORBA (Common Object Request Broker Architecture) is a standard that defines a framework for developing object-oriented distributed applications.

Using the CORBA architecture makes network programming easier by allowing a developer to create distributed applications that interact as though they were implemented in a single programming language on one computer.

CORBA enables the distributed applications to be developed in an object-oriented manner. It allows us to design a distributed application as a set of co-operating objects and to reuse existing objects in new applications.

4.2.2 What is an ORB

An ORB (Object Request Broker) is a software component that mediates the transfer of messages from a program to an object located on a remote network host. CORBA defines a standard architecture for ORBs.

Essentially the role of the ORB is to hide the underlying complexity of network communications from the programmer.

An ORB allows you to create standard software objects whose methods can be invoked by client programs located anywhere in your network. A program that contains instances of CORBA objects is often known as a server.

When a client invokes a member method on a CORBA object, the ORB intercepts the method call. The ORB then redirects the method call across the network to the target object. The ORB then collects results from the method call and returns these to the client.

4.2.3 The Object Management Architecture (OMA)

An ORB is one component of the OMG's Object Management Architecture (OMA). This architecture defines a framework for communication between distributed objects. The OMA includes the following elements.

- Application objects
- The ORB
- The CORBAServices
- The CORBAFacilities

Application objects are objects that implement programmer-defined IDL interfaces. These objects communicate with each other, and with the CORBAServices and CORBAFacilities, through the ORB. The CORBAServices and CORBAFacilities are sets of objects that implement IDL interfaces defined by CORBA and provide useful services for somc distributed applications.

4.2.4 The Nature of CORBA Objects

CORBA objects are standard software objects implemented in any supported programming language. CORBA supports several languages, including C++, Java, COBOL and PL/I.

By making calls to an ORBs' application-programming interface (API), it is possible to make CORBA objects available to client programs in the network.

Clients can be written in any supported programming language and can invoke the member methods of a CORBA object using the normal programming language syntax.

Although CORBA objects are implemented using standard programming languages, each CORBA object has a clearly defined interface, specified in the CORBA Interface Definition Language (IDL).

The interface definition specifies what member methods are available to a client, without making any assumptions about the implementation of the object.

To invoke member methods on a CORBA object, a client needs only the object's IDL definition. The client does not need to know details such as the programming language used to implement the object, the location of the object in the network, or the operating system on which the object runs.

The separation between an object's interface and its implementation has advantages. For example, it allows you to change the programming language in which an object is implemented without changing clients that access the object.

It also allows you to make existing objects available across the network.

4.2.5 The Structure of a CORBA Application

To start developing a CORBA application, you must define the interface to the objects in your system using CORBA IDL. These interfaces should then be compiled using an IDL compiler.

For example, an IDL compiler can generate Java or COBOL from IDL definitions. This code includes client stub code, which allows you to develop client programs, and server skeleton code, which allows you to implement CORBA objects.

When a client calls a member method on a CORBA object, the call is transferred through the client stub code to the ORB. If the client has not accessed the object before, the ORB refers to a database known as the Implementation Repository, to determine exactly which object should receive the method call. The ORB then passes the method call through the server skeleton code to the target object.

4.2.6 The Structure of a Dynamic CORBA Application

One difficulty with normal CORBA programming is that you have to compile the IDL associated with your objects and use the generated code in your applications. This means that your client programs can only invoke member methods on objects whose interfaces are known at compile time. If a client wishes to obtain information about an object's IDL interface at runtime, it needs an alternative, dynamic approach to CORBA programming.

The CORBA Interface Repository is a database that stores information about the IDL interfaces implemented by objects in your network. A client program can query this database at runtime to get information about those interfaces. The client can then call member methods on objects using a component of the ORB called the DII (Dynamic Invocation Interface).

4.2.7 Dynamic Server Programming

CORBA also supports dynamic server programming. A CORBA program can receive method calls through IDL interfaces for which no CORBA object exists. Using an ORB component called the Dynamic Skeleton Interface, the server can then examine the structure of these method calls and implement them at runtime.

4.2.8 Interoperability between ORBs

The components of an ORB make the distribution of programs transparent to network programmers. To achieve this, the ORB components must communicate with each other across the network.

In many networks, several ORB implementations coexist and programs developed with one ORB implementation must communicate with those developed with another. To ensure that this happens, CORBA specifies that ORB components must communicate using a standard network protocol called the Internet Interoperability Protocol.

4.2.9 The CORBA Services

The CORBAServices define a set of low-level services that allow application objects to communicate in a standard way. These services include the following:

The Naming Service: Before using a CORBA object, a client program must get an identifier for the object, known as an object reference. This service allows a client to locate object references based on abstract, programmer-defined object names.

The Trading Service: This service allows a client to locate object references based on the desired properties of an object.

The Object Transaction Service: This service allows CORBA programs to interact using transactional processing models.

The Security Service: This Service allows CORBA programs to interact using secure communications.

The Event Service: This service allows object to communicate using decoupled, event-based semantics, instead of the basic CORBA function-call semantics

4.2.10 The CORBA Facilities

The CORBAFacilities define a set of high-level services that applications frequently require when manipulating distributed objects. The CORBAFacilities are divided into two categories.

The horizontal CORBAFacilities consist of user interface, information management, systems management, and task management facilities.

The vertical CORBAFacilities standardise IDL specification for market sectors such as healthcare and telecommunications.

4.3 Looking inside CORBA

4.3.1 The Basics

IDL is a part of the OMG's CORBA specification and it is an ISO (International Organisation for Standardisation) standard. However it is not a programming language and in fact it enables interfaces to be developed independently of the languages used to implement these interfaces.

An interface definition provides all of the information needed to develop clients that use the interface. Essentially it provides a description of the functionality provided by the CORBA objects.

Interfaces are the basic unit in IDL and define the interface to a service/object. An interface is composed of operations and attributes.

Conceptually, attributes correspond to the variables that a component implements. Attributes indicate that these variables are available in a component and that clients can read or write their values.

Attributes normally map to a pair of functions in the programming language used to implement the component. These functions allow client applications to read or write the attribute values. If proceeded by the keyword readonly, clients can only read the attribute value.

IDL operations define the format of methods that clients use to access the functionality of a component. An IDL operation can take parameters and return a value, using any of the available IDL data types.

CORBA must know the direction in which a parameter is being passed in order to manage these parameters. There are three modes for parameter passing

- In: The parameter is passed from the client to server
- Out: The parameter is passed from the server to client
- Inout: The parameter is passed both in both directions

An interface can be defined within a module; this allows interfaces and other IDL type definitions to be grouped into logical units.

Names defined within a module do not clash with names defined outside the module. Essentially a module defines a naming scope within an IDL file.

An IDL operation may raise an exception indicating that an error has occurred. This will be investigated in more detail in a later chapter.

Inheritance in IDL is a method for using the properties of an existing interface in a new interface.

IDL provides preprocessing directives that allow macro substitution, conditional compilation and source file inclusion. The IDL preprocessor is based on the C++ preprocessor. For example, the #include directive allows an IDL file to be included in other files.

A typedef declaration can be used to define a meaningful or a simpler name for a basic or user-defined type. This definition can be used to make code easier to read.

An interface must be declared before it is referenced. A forward declaration declares the name of an interface without defining it. This allows the definition of interfaces that mutually reference each other. The interface definition must appear later in the specification.

4.4.2 IDL Mappings

Mapping for basic types include short, long, unsigned short, unsigned long, float, double, char, boolean, octet, any, string.

An IDL string is a one-dimensional array of characters with a variable length

Mapping for constructed types include struct, union, enum

A struct is an IDL data type that can package a set of named members of various types.

A union provides a space saving type whereby the amount of storage required is the amount necessary to store the largest possible element. Only one element will be held.

An enumerated type allows the members of a set of values to be depicted by identifiers.

Other mappings are provided for arrays and template types such as sequence.

An IDL sequence is a one-dimensional array with a variable length.

4.4.3 Finding CORBA Objects

A common problem is distributed programming is enabling a client to find the correct object. The purpose of CORBA is to enable a client to use remote objects if they were local. Local object references acts as a proxy for the remote object. Every time operation is invoked on a local object, the ORB passes the request onto the remote server.

The client (or client proxy) needs details of where the service is located in order to forward the request.

The information needed includes the interface name, machine name or IP address and the port number

A CORBA Object Reference (OR) or an Interoperable Object Reference (IOR) hold this information. An IOR is an OMG specified string that uniquely defines the location of a CORBA object but also adheres to the CORBA 2.0 specification that introduces interoperability between ORBs

Real-world distributed object computing requires much more than a communication mechanism; it requires infrastructure. **[Curtis 1997]**

- Applications need to find objects that are migrating about the network

- Objects that the applications need may be dormant and require activation

- Applications need to obtain services based on general property descriptions rather than specific identities

- Applications need transactional integrity among groups of distributed objects

- The software components that constitute a distributed system need to be administered and managed through standard interfaces

- The underlying mechanisms that support communication, location, and other basic services must be reliable, able to recover from errors and re-configure themselves as necessary to provide high availability

These requirements are met by a distributed computing infrastructure, an architecture of underlying mechanisms and basic services that provide a stable, powerful platform upon which applications can be built. The OMG Object Management Architecture (OMA) (see section 4.2.3) provides this platform, including the core CORBA ORB specification and a set of object services (called CORBAServices).

4.4.3.1 The Naming Service

The Naming Service is a simply another CORBA server. It maps IORs to a humanly readable name. A server program can then publish its IOR in the Naming Service database and a client program can retrieve the IOR using the provided name.

The names that are used to identify objects in the Naming Service are made up of contexts and application objects. Application objects are actual objects that you can invoke operations upon. Contexts hold the application objects. A single context can hold multiple application objects. The root context is the primary context (or first point of contact). All names must start from the root context.

The typical sequence of events that takes place is the server starts and its name will be registered with the NS. The client then starts and resolves to the root context (top of the naming structure) of the NS and then resolves the name and retrieves the IOR for the object. It can then use this IOR to invoke a remote operation.

4.4.4 Exception Handling

IDL operations can raise exceptions to indicate the occurrence of an error. CORBA defines two types of exceptions

System Exceptions are a set of standard exceptions defined by CORBA

User-defined Exceptions are exceptions that you define in your IDL specification.

All IDL operations can implicitly raise any of the CORBA system exceptions. No reference to system exceptions appears in an IDL specification.

To specify that an operation can raise a user-defined exception, first define the exception structure and then add an IDL raises clause to the operation definition. An IDL exception is a data structure that contains member fields.

4.4.5 ORB Interoperability

Since its inception in 1991, CORBA has proven itself as a solid basis for heterogeneous object-oriented distributed systems. Like all technologies, however, CORBA must evolve in order to remain viable. **[Vinoski 1998].** It must allow for different ORB implementation to co-exist and co-operate within the same company network.

ORB Interoperability allows communication between independently developed implementations of the CORBA standard. ORB interoperability enables a client of one ORB to invoke operations on an object in a different ORB via an agreed protocol. Thus, invocations between client and server object are independent of whether they are on the same or different ORBs. The OMG has specified two standard protocols to allow ORB interoperability, GIOP and IIOP

The OMG-agreed protocol for ORB Interoperability is called the General Inter-ORB Protocol (GIOP). GIOP defines on-the-wire data representation and message formats. It assumes that the transport lawyer is connection oriented. The GIOP specification aims to allow different ORB implementations to communicate without restricting ORB implementation flexibility.

The Internet Inter-ORB Protocol (IIOP) is an OMG defined specialisation of GIOP that uses TCP/IP as the transport layer. Specialised protocols for different transports such as OSI, NetWare, IPX) or for new features, such as security can also be defined by the OMG.

4.4 Service-Based Architecture

[Koch, Murer 1999] outline how that analysing the characteristics of large-scale systems lead to the concept of a managed evolution and a service architecture. Service Architecture is based on the idea of using large-grained Services instead of fine-grained objects to represent the objects in the banking system. This leads to an architecture where fine-grained components, like customers or accounts, reside within large grained components. There will be further discussion of this approach in chapters on Performance [Chapter 6] and Scalability [Chapter 8].

The benefits of using a Service-Based Architecture include

- Simplify evolution by decomposing systems into services
- Encapsulation of Data
- Having well defined interfaces
- Useable independent of technology (Implementation Independence)
- Renewable (easier to upgrade) without affecting other parts of the system, reducing risk
- Leads to more IT efficiency when building new applications
- Better reuse of Services
- Less risk for complete system, due to isolation by service interfaces

CORBA is a suitable choice for a Service-Based Architecture for the following reasons:

- Clean Model for construction of heterogeneous distributed systems
- Potentially complete middleware functionality
- Clean interfaces for application integration
- Bridges to other important standards available, therefore integration into standard software possible.
- Provides a technical bridge - i.e. Java Client to PL/I IMS Transaction
- Decouples interfaces from technology

4.5 Conclusion

In this chapter there has been a closer examination of a standard known as CORBA (Common Object Request Broker Architecture) and how applications can be built using this standard. It is clear from the previous chapter why CORBA is a popular solution for integrating legacy systems and in the next chapter there will be an investigation into how a real legacy application can be integrated with modern technologies using CORBA.

The OMG is a very active consortium. Its many task forces and special interest groups cover nearly the entire spectrum of topics related to distributed computing, including real-time computing, Internet, telecommunications, financial systems, medical systems, object analysis and design, electronic commerce, security database systems, and programming languages.

As a result, Request for Proposals (RFPs) and technology adoptions in almost all of these areas have either already occurred or soon will

In the last chapter, CORBA was put forward as one possible solution to the integration problems that modern Banking Sector companies are facing. This chapter focused on looking deeper inside CORBA to see some of the extra benefits that could be used. One interesting usage of CORBA, the Service-Based Architecture, was discovered. This approach meets the requirements of an integration strategy but also provides for larger and enterprise scale banking systems.

The use of a Service-Based CORBA Architecture is now the preferred solution moving forward in this research. The next few chapters will assume the use of this approach and look at potential problems using it will cause to such large-scale integration projects.

Before this however, there will be a brief look at the concept of Patterns. These are a way of providing well-known solutions to every day problems and this research will be looking for such solutions to the potential problems with large-scale integration projects.

5 Patterns

5.1 What are Patterns

The concept of a pattern in software development has arisen over the last few decades. Essentially a pattern is a solution to a well-known or recurring problem that occurs during the software development cycle.

As software development has evolved over the years, teams of developers have come across the same problems over and over again. Often a development house would provide a Frequently Asked Questions (FAQ) list of common problems for other developers in the company.

However, in the last twenty years there has been a movement towards sharing this knowledge with all developers who could use this information. Experienced programmers began to recognise the similarity of new problems to problems they had solved before. With a little more experience, they realised that the solutions for similar problems follow recurring patterns. As these programmers become used to the concept of a pattern, they can learn when to apply this solution to a situation without having to stop and analyse the problem and investigate possible strategies.

This concept of sharing solutions to well known problems originally came from the field of architecture. In the late 1970's there were two revolutionary books published by Christopher Alexander. These books "A Pattern Language, Towns, Buildings, Construction" (Oxford University Press, 1977) **[Alexander 1977]** and the "Time Timeless Way of Building" (Oxford University Press 1979) **[Alexander 1979]** described patterns in building architecture and urban planning. These patterns could be applied again and again in different areas of architecture and building. Taking one of these patterns, "Pedestrian Street" for example:

Context: The simple social intercourse created when people rub shoulders in public is one of the most essential kinds of social "glue" in society.
Problem: This glue is largely missing, in part because much of the process of movement is taking place in indoor corridors and lobbies.
Solution: Pedestrian street. Arrange buildings so that they form pedestrian streets with many entrances and open stairs directly from the upper storey to the street, so that even movement between rooms is outdoors, not just movement between buildings.

The field of software development was quick to see the usefulness of this concept and in 1987, Ward Cunningham and Kent Beck **[Beck Cunningham 1987]** used some of Alexander's ideas to develop five patterns for User Interface Design. This paper "Using Pattern Languages for Object-Oriented Programs" was published at the OOPSLA-87 conference.

Following on from this was the book "Design Patterns" by Erich Gamma, Richard Helm, John Vlissides and Ralph Johnson **[Gamma 1995]**. This book was first published in 1984 and is considered one of the major advances in software development in the last 20 years.

Most of the developers in today's IT projects will know some or all of these and other patterns. Experienced developers will know when and where to use some or all of them and in which circumstances they do not add to the solution but simply add more complexity.

There are many benefits to having various patterns in your "toolbox" for use when programming IT solutions, just as any builder would have similar solutions when building a house but there are also drawbacks. In some way patterns have been over-hyped and there are solutions available today that are not really patterns but rather idioms or rules-of-thumb. The trick to being a good developer is to know which patterns are useful but more importantly in which circumstance they apply.

This "toolbox" mentioned comes in the form of a "Pattern Language" where a number of patterns are used together (and are even designed to facilitate each other) to solve various problems that arise in the system as a whole. These problems can occur either at an application design level or in a larger scale at the system architecture and design level.

For example, Enterprise systems are often developed without security in mind, as applications programmers are more focused on trying to learn the domain than worrying about how to protect the system. In response to this requirement **[Yoder, Baraclow 1997]** define a collection of patterns to be used when dealing with application security.

5.2 How does a pattern come to be

As outlined above, programmers come across many problems in their daily work effort, the trick is to find patterns that can be applied to similar problems in different environments. A recognised way of doing this is to articulate the problem/solution pair in words. One this can be done, the pattern can be discussed among programmers who know the pattern to collaborate on the details.

When the pattern is in words, it can be explained to others who are not familiar with the problem. It can be fine-tuned and changed as the discussion develops so that what is there at the finish is a solution that can be applied not to one problem domain but to many situations in different projects.

[Levine, Schmidt 2000] offer a fascinating slant on using patterns within software design and architecture by comparing it to becoming a Chess Master.

To how become a Chess-Master involves following some steps

- First learn the rules and physical requirements such as names of pieces, legal movements, chessboard geometry and orientation etc.
- Secondly learn the principles such as the relative value of certain pieces, strategic value of centre squares and power of a threat etc.
- To become a master of chess, one must study the games of other masters. These games contain patterns that must be understood, memorised and applied repeatedly. There are hundreds of these patterns.

In the same way, to become a software design master has certain steps

- Firstly, learn the rules such as the algorithms, data structures and languages of software

- Secondly, lean the principles such as structured programming, modular programming, object oriented programming, generic programming etc.
- To truly master software design, one must study the design of other masters. These designs contain patterns that must be understood, memorised, and applied repeatedly.

There is a recognised form for a pattern definition to take. This written form will contain the following details:

- Description of the pattern
- Concrete example
- Specific solution for this example
- Summary of issues involved in the initial formulation of the general solution
- The General Solution
- Any consequences of using this pattern
- Pros and Cons of the pattern
- List of related patterns

This list, in its complete form is designed to give any developer who has not previously come across this problem an insight into the pattern that can be applied. One common form of pattern definition presents these details using the following headings:

PATTERN NAME

Name, Bibliography reference (where did pattern come from)

SYNOPSIS

Description of the pattern that conveys the essence of the solution provided by the pattern. This is directed at experienced programmers who may recognise them.

CONTEXT

The problem the pattern addresses as part of a concrete example and a suggested design solution.

FORCES

Summarise the considerations that lead to the general solution presented in the solution section.

SOLUTION

This is the core of the pattern. Describes the general-purpose solution to the problem the pattern addresses.

CONSEQUENCES

Explains the implications – both good and bad of using the solution.

IMPLEMENTATION

Describes the important considerations to be aware of when executing the solution. It may also describe some common variations of simplifications of the solution.

CODE EXAMPLE

Contains a code example showing sample information for a design that uses the pattern.

RELATED PATTERNS

List of patterns related to the one described

5.3 Design Patterns

A distinction should be made between the various types of patterns that are popular in industry today. Typically the common flavours are Design Patterns and Architectural Patterns. These differ in mainly in scale. Design patterns are medium scale patterns that are used to organise subsystem functionality in an application domain independent way whereas Architectural patterns typically are used to define the structure and high-level architecture on a larger scale.

There are also other solutions to these problems that do not fit the definition of a pattern. A **framework** for example is a set of co-operating classes that makes up a reusable design for a specific class of software **[Gamma 1995]**. These frameworks can be used in specific cases but are not general solutions to industry-wide problems.

Such a framework does however provide architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customises the framework to a particular application by sub-classing and composing instances of framework classes.

One definition of a design pattern is that is describes a commonly recurring structure of communicating components that solve a general design problem in a particular context.

Design patterns facilitate architectural level reuse by providing "blueprints" that guide the definition, composition, and evaluation of key components in a software system. In general a large amount of experience reuse is possible at the architectural level. However, reusing design patterns does not necessarily result in direct reuse of algorithms, detailed designs, interfaces, or implementations. **[Schmidt Stephenson 1995]**

Examples of Design Patterns

It is possible to group design patterns into categories of related patterns. These groupings can look like

- Structural Decomposition
- Organisation of Work
- Access Control
- Management
- Communication

There are quite a number of Design Patterns that fit into each of these categories. Certain ones such as *Proxy, Facade* and *Iterator* which fit into the Access Control group apply in the case of integrating CORBA and Legacy Solutions and these will be looked at later. Likewise the Communication Group contains patterns such as *Forwarder-Receiver, Client-Dispatcher-Server*, and *Publisher-Subscriber* that are inherent in any CORBA solution.

5.4 What aren't Design Patterns

There are also many solutions to well-known problems in the software world that do not make good design patterns. Typically these are known as Idioms whereas others that can have more of a negative effect on a systems are known as anti-patterns.

Idioms can be thought of as low-level patterns specific to an particular implementation problem. They describe how to implement particular aspects of components or the relationships between them using features of a given programming language. These can include naming conventions, source code formats, memory management rules etc.

The easiest way to distinguish between design patterns and idioms is that idioms are less portable implementations of design patterns. They cannot be easily adapted to other similar solutions but should not be completely forgotten about as they can also help in problem solving and training of new team members.

Every software-architecture typically builds on certain principles. The understanding and acceptance of these principles is crucial in understanding the architecture because they guide architectural decisions. The most important difference between a principle such as Information Hiding and a pattern is that patterns define structure and interactions. Principles, on the other hand, do not have a specific structure, they are more guidelines, or rationale, for the structure of specific patterns.

[Völter 2000] outline how principles can also be regarded as high-level goals, which we want to reach by applying the patterns. Many patterns will reference the principles in order to explain why a pattern has some specific structure.

[Foote Yoder 1997] document various "anti-patterns" that detail real-world architecture problems as seen in the design and growth of major cities over time. An anti-pattern outlines both what should not be done in a system but also what does happen in the real world. Many of these are applicable to software architecture and particularly to the mission-critical mainframe systems that have evolved over recent decades.

BIG BALL OF MUD

When a system has reached a state equivalent to BIG BALL OF MUD, it is already in a state that makes change or adaptation difficult or impossible. Such systems can continue to function in their current state but any attempt to change it will be resisted.

THROWAWAY CODE

This pattern details a fairly common problem in software development. When the developers or designers are producing the proof-of-concept or prototype for a system, they do not worry about the elegance or efficiency of that system. Rather they are concerned to show that it works and will be re-written correctly at a later stage. The danger comes when this prototype is chosen in its current state for release. This can easily happen for time and budget reasons and will lead to further problems at a later stage.

PIECEMEAL GROWTH

Many mainframe applications have fallen victim to the PIECMEAL GROWTH situation. Essentially they began their life as small, simple applications that provided certain functionality. As time went on, extra features were continuously added and these in themselves may have required extra features. Like many cities of today, these applications became massive, system critical enterprise systems even though they were not planned to be.

KEEP IT WORKING

This is one of the most common situations in a company that relies on mainframe systems. The business has come to rely on these systems and the data behind it. If the system was to become unavailable it would cause chaos in the day-to-day business. These systems are mission critical and must be kept running at all costs. So many systems throughout the world are in exactly this state.

SWEEPING IT UNDER THE RUG

Another common situation with mainframe systems is hiding the mess. As with the run-down areas of a large city, the system can be made to look cleaner by putting cleaner, newer modules that hide the mess and complexity of some of the other code and modules.

RECONSTRUCTION

This situation is somewhat less common in the mainframe world. In large cities, there comes a time when the only solution to a problem building is to demolish and start again. This can happen when the existing building simply cannot be extended or enhanced to meet the requirements. In the mainframe world the costs of such an approach and be prohibitive and there is the problem of what to do while the newer system is being built.

5.5 Architectural Patterns

While software patterns are useful for developers writing code to solve business requirements, our interests lie in a slightly different place. There also exists the concept of an architectural pattern, which is used to specify the fundamental structure of a software system.

An architectural pattern does not only express a fundamental structural organisation schema for software systems. It also provides a set of predefined subsystems, specifies their responsibilities and includes rules and guidelines for organising the relationships between them. It can be

considered as a high-level strategy that concerns large-scale components and the global properties and mechanisms of a system.

[Keshav, Gamble 1998] define an integration architecture to be the software architecture description, using integration elements, of a solution to interoperability problems between at least two interacting component systems. In this regard, we will look at various architectural patterns that are already available and used in the industry.

Such type of pattern is particularly useful for us in the CORBA-Legacy Integration architecture design. Our predecessors will already have come across the problems of ensuring that our integrated solution is secure, scalable, reliable, available and meets the performance requirements we need.

These patterns can have a large impact and implications, which affect the overall structure and architecture of such a system and as a result have received much attention over the last few years.

The architecture of a software system is unique. It depends on the context in which it is developed and on various aspects: Expected lifetime, cost of development, foreseen evolutions, experience of architects and developers etc.

For a particular problem, there does not exist **one** optimal architecture but rather an architecture adapted to a given context. **[Gueheneuc Juissen 2001]**. Therefore, we focus on general and context-independent architectural problems.

Software and Design patterns often simply provide a "better" way to code an application so that a developer will avoid various pitfalls that are inherent with this process. The Architectural patterns that we will consider are aimed at getting the System Architecture correct in the first place.

In this research into CORBA and Integration approaches, we have already come across two possible Architectural Patterns

<u>Managed Evolution</u> [Section 4.4] involved an evolutionary approach when migrating to a new technology.
<u>Standard-Based Solutions</u> [Section 1.4.2] are solutions with a industry-wide consensus coming together to provide standards on a certain technology

5.6 Categories of Architectural Patterns

Architectural Patterns can be broken down into the following subcategories according to their properties.

5.6.1 From Mud to Structure

This type of pattern is used in the creation of the initial system. Some well-known patterns that fit into this category are *Layers, Pipes and Filters, Blackboard.*

The Layers pattern for example, structures the system into groups of subtasks working on a particular level of abstraction. This pattern can be useful when re-using the different layers and supports standardisation. In addition the various dependencies are kept local. On the negative side there can be changing behaviour, lower efficiency, unnecessary work and difficulty in establishing the correct granularity. Examples of this pattern include OSI and the Internet Protocol Suite.

5.6.2 Distributed Systems

Distributed Systems patterns are the type of Architectural patterns that have special interest for those of us trying to implement distributed solutions in an enterprise situation. The types of pattern that can be applied in these cases include *Broker, Pipes and Filters, Microkernel.*

The Broker pattern is used to co-ordinate communication between distributed software systems in order to enable remote use of services. CORBA, OLE and Active X are examples of this pattern in use.

5.6.3 Interactive Systems

Interactive Systems patterns are very much geared towards user interaction and human involvement in an application or architecture. Such patterns include *Model-View-Controller, Presentation-Abstraction-Control.*

The *Presentation-Abstraction-Control* pattern (PAC) defines a structure for interactive software systems in the form of a hierarchy of co-operating agents. Every agent is responsible for a specific aspect of the application's functionality and consists of three components: presentation, abstraction, and control. This subdivision separates the human-computer interaction aspects of the agent from its functional core and its communication with other agents.

5.6.4 Adaptable Systems

Finally we have Adaptable Systems patterns which will need to be used in systems that can be easily adapted. These include *Reflection, Microkernel.*

The *Microkernel* pattern applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and customer-specific parts. The Microkernel also serves as a socket for plugging in these extensions and co-ordinating their collaboration.

5.7 Implementing Architectural Patterns

When Architects and System Designers come together to design the outline of an enterprise project or evolving an existing one, they do not want to start from scratch. They will be aware of

patterns that exist and will use these in conjunction with experience and convention to lead them to apply common ways to solve common problems they will encounter.

Architectural patterns aid developers in resolving coarse-grained integration problems among components. These patterns are assembled from functionality slices that resolve various communications problems between applications. However little attention has been paid to how interoperability problems and their resolution are embodied in these patterns.

[Davis Gamble 2001] notice that mapping these problems to specific functionality promises insight into composing integration architectures by illuminating the consistent, high-level solutions that resolve individual conflicts.

For example, in the case of some attempting to design a user-intensive system, one proven way to organise the abstractions is to use a model-view-controller pattern, in which you clearly separate objects (the model) from their presentation (the view) and the agents that keep the two in sync (the controller).

Another example would be, if you are building a system for solving cryptograms, one proven way to organise your system is to use a blackboard architecture, which is well suited to attacking intractable problems in opportunistic ways.

Object-oriented patterns and frameworks describe architectural aspects of software. Both are based on the class/object concepts which are available in object oriented analysis and design methodologies and in object oriented programming languages **[Jacobsen et al. 1997]**

It is clear to see that patterns can help us to visualise, specify, construct, and document the artefacts of a software-intensive system. In addition, we can forward engineer a system by selecting an appropriate set of patterns and applying them to the abstractions specific to our domain.

It is also possible to reverse engineer a system by discovering the patterns it embodies, however this is not really an elegant way to engineer such a system.

A better approach would be, when we deliver a system, we can specify the patterns it embodies so that when someone later tries to reuse or adapt that system, its patterns will be clearly manifest.

5.8 Qualities of Patterns

As patterns as a concept are almost twenty years old, there is now somewhat of a broad consensus as to what qualities a pattern should have to make it useful and not just a rule-of-thumb. **[Lea 1993]** outlines some of these qualities:

- Encapsulation and Abstraction

Each pattern encapsulates a well-defined problem and its solution in a particular domain. Patterns should provide clear boundaries that help crystallise the problem space and the solution space.

- Openness and Variability

Each pattern should be open for extension or parameterisation by other patterns so that they may work together to solve a larger problem. A pattern solution should be also capable of being realised by an infinite variety of implementations (in isolation, as well as in conjunction with other patterns).

- Generativity and Composability

Each pattern, once applied, generates a resulting context, which matches the initial context of one or more other patterns in a pattern language. These subsequent patterns may then be applied to progress further toward the final goal of generating a "whole" or complete overall solution.

- Equilibrium

Each pattern must realise some balance among its forces and constraints. This may be due to one or more invariants or heuristics that are used to minimise conflict within the solution space. The invariants often typify an underlying problem solving principle or philosophy for the particular domain, and provide a rationale for each step/rule in the pattern.

5.9 Conclusion and further developments

In this chapter we have seen some definitions for the concept of Architectural Patterns. We shall see various uses for some or all of these patterns throughout our investigation on Integrating Legacy Systems and CORBA. Many of the patterns we see will have been used similar projects throughout the world-wide banking industry and others will be little more than idioms or rules-of-thumb that can help with a particular implementation solution.

As discussed in Chapter 3, one of the newest competitors for CORBA in the legacy integration area is Enterprise Java Beans. Although Enterprise JavaBeans provide simple APIs for relatively complex tasks, designing and implementing a scalable, maintainable and reasonably fast application based on EJB is not trivial. Over time, a set of proven patterns has emerged and these can be seen in **[Wolff, Schmid, Völter 2001]**

[Quinot et al. 2001] outline DROOPI (Distributed Reusable Object-Oriented Polymorphic Infrastructure) , which is a novel middleware that will allow interoperability of distributed object-oriented applications across distributed platforms. This paper outlines the completed first step of this project, which consists of the definition of a generic middleware architecture. This architecture integrates and extends several aspects of existing middleware and is an interesting view of where the future of middleware integration projects could be headed.

In the context of this research, Patterns are an approach we can use at a later point when encountering difficulties and blockages with large-scale integration projects. Specifically, the next few sections deal with some of major areas where such difficulties are usually encountered. Knowing that we can use "well-known" solutions of others to overcome these difficulties will make the task of this research significantly easier.

In addition, this research does not aim to find new patterns but to apply existing patterns related to CORBA and Integration to any of issues arising in enterprise integration projects. To start this

process we must discover what problems are being faced when migrating legacy systems to CORBA.

For example, **[Kim Bieman 2000]** discuss how we must solve the following problems:

- **Variety of the interfaces to legacy systems:** There are many interfacing styles in legacy systems, they have different implementations from each other and are also dedicated. This makes it difficult for server-side application developers to implement wrapper objects for legacy systems even though they understand some of the interfaces to the legacy systems.

- **Representation of interfaces to legacy systems:** To generate wrappers automatically, a server-side developer should submit interfacing information for legacy systems to an automatic wrapper generator. Thus, some representations are required to describe easily the interfaces to legacy systems.

6 CORBA Performance Issues

6.1 Introduction

Another of the areas of uncertainty that is introduced into a Banking environment when older systems are reengineered as peers in a distributed computing environment is performance.

Traditional mainframe applications could rely on good performance and as we shall see, when we introduce distributed applications to this platform, there are various performance overheads that will be associated.

What we need to look at is various ways of minimising this overhead so that access to critical data can still be retrieved quickly, whether from a 3270 Terminal or a Java Applet.

[Kahkipuro 1999] details some of the ways in which the performance of CORBA based applications can be compromised. These include

- Distribution transparencies
- Marshalling and demarshalling of parameters
- Invocation routing
- Network bandwidth and latency
- Use of Network connections
- Server Contention

As noted in **[Gokhale, Schmidt 1998]**, the success of CORBA in mission-critical distributed computing is dependant on the ability of the Object Request Broker to provide the necessary quality of service (QoS) to applications. These quality of service requirements include high bandwidth, low latency, and scalability of endsystems and distributed systems. We will investigate the first two of these in this chapter and consider the latter in a later chapter.

It is important to note that for this discussion we will focus on what can be done in the CORBA environment and specifically in terms of IDL and Service granularity to improve performance. We can assume that a large organisation is using the latest hardware and software technology and this does not require a discussion.

Various other issues such as the Availability and Scalability of the system will also have real affects on the systems performance and in later chapters we will look at these topics and at various solutions that can be applied from these areas to the performance of the system

6.1.1 Performance of Distributed Systems

The very first thing we can assume with distributed systems is that there will be a performance hit due to network latency. It is accepted that a call across the network will always be substantially slower than a local call. In reality, this factor may be thousands of times slower. **[Koch, Murer 1999]**.

Even the modern advances in technology and various CORBAServices have not overcome this problem. Extending this problem, it is also likely that inter-process calls on the same physical machine will be slower than calls that execute in the same process.

Naturally this performance hit that comes with distributed systems is going to be of big concern in a CORBA environment where there can be direct communication between many objects.

[Gokhale, Schmidt 1997] detail how the QoS requirements for delay-sensitive applications include an absolute need for low-latency. Modern banking high-speed networks (such as ATMs) support quality of service in terms of bandwidth and latency. Using CORBA means significant performance overhead in such applications, which have to be overcome or lower level communications would be preferred such as sockets, which do not have the other benefits of CORBA such as reliability, flexibility and reusability.

In fact, **[Rackl 2000]** suggests that these requirements are heading in two distinct directions. He maintains that on one hand, optimised platforms for specific application domains like high-performance computing are being developed but on the other hand, integration solutions like CORBA have improved interoperability as their primary goal, allowing communication between different middleware products and the integration of legacy applications.

However, as seen from earlier chapters, there is in fact many large organisations that require both improved interoperability and high performance and this is the target we need to reach.

6.1.2 Other Performance Problems

The performance degradation of an application due to network latency is not the only issue we need to consider in the CORBA world. **[Slama et al. 1999]** outline how we must consider the type of data being passed between CORBA objects, and the amount of this data being passed.

[Silva et al. 2000] also outline how there is a performance cost due to the creation and deletion of CORBA objects, specifically with the start-up of the Java virtual machine when this is the operating system of choice for CORBA servers. This additional cost is added for the start-up of each server.

6.1.3 Designing IDL for performance

The Interface Definition Language defined by the OMG for defines CORBA interfaces have been outlined in a previous chapter and it can be seen how IDL is a flexible way of defining our interfaces and other associated elements.

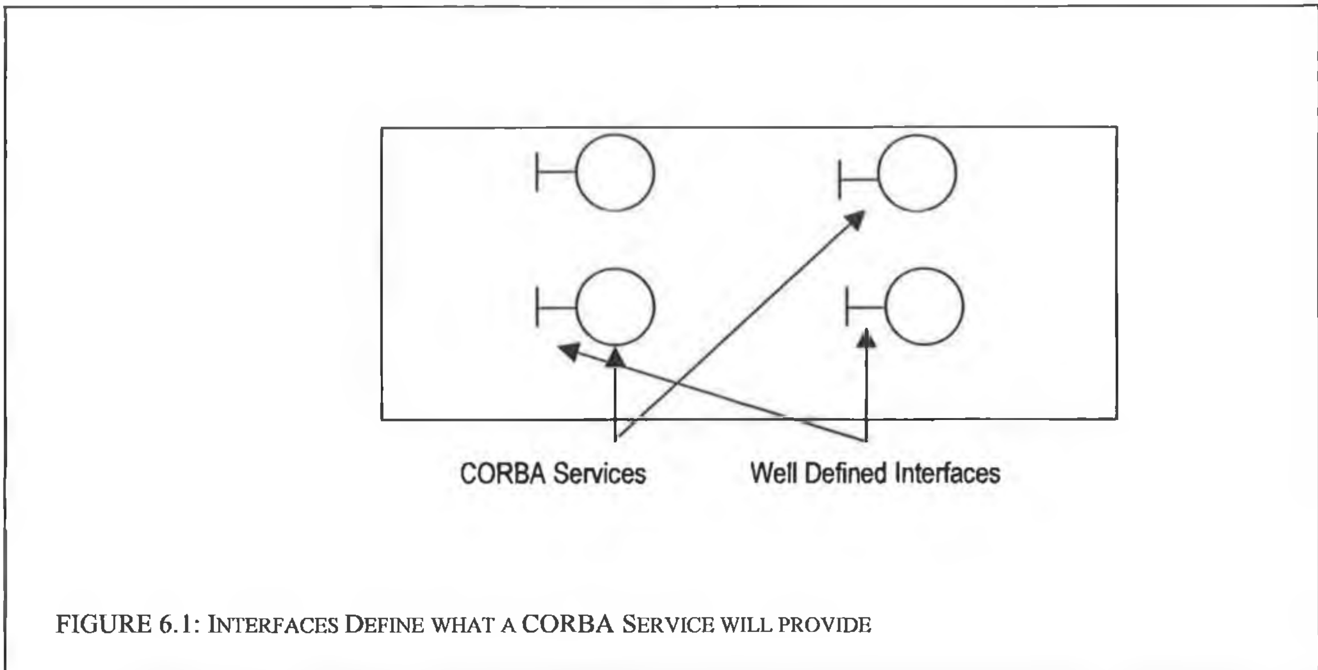


FIGURE 6.1: INTERFACES DEFINE WHAT A CORBA SERVICE WILL PROVIDE

However IDL is more than a useful tool. It is the key to designing our CORBA Services and can have a massive impact later on if not thought out properly in the early phase. One key component to remember is that IDL is for designing interfaces for the Service and not the implementation. This is especially true when dealing with Legacy Applications or Legacy Data. We must design the IDL irrespective of how it is or will be implemented.

[Smith, Williams 1998] also recommend considering performance issues early in the development process to ensure optimal performance of an application. They outline various problems with applications using the "fix-it-later" approach.

6.2 General Solutions to the CORBA Performance Problems

[Grahn, Holgersson 2002] and **[Slama et al. 1999]** provide some general performance guidelines that can help to improve the performance of a CORBA based system. These are

- Reducing the number of remote operations
- Optimising the amount of data passed
- Optimising the IDLs types used.

For the reasons outlined above, the single best solution to performance problems in a distributed system is to reduce the number of remote operations. The cost of network communication can be significantly reduced if each invocation deals with a reasonable amount of work.

Another performance improvement can be made in recognising that different IDL datatypes cost more to marshal and unmarshall than others and to design the IDL with this in mind can reap big benefits. In the same manner, IDL can be designed so that the number of remote operations is kept to a minimum. We shall see how this is done later.

Finally, another common solution to improving system wide performance is to implement a load-balancing policy. This policy is a way of ensuring that the load on the system is spread evenly so that queues and bottlenecks are kept to a minimum.
We shall look at some of these general solutions to ensuring that the addition of CORBA to the mainframe environment does not have catastrophic effects on performance.

6.3 Minimise the number of remote operations

We should offset network latency at an early stage of the applications development lifecycle. As we have just seen, distributed computing adds significant overhead due to slower communication than the traditional centralised mainframe model.

[Kahkipuro 1999] outlines some well-known techniques, such as caching and pre-fetching to reduce network latency and **[Gokhale, Schmidt 1998]** outline some system level options that can help. These include

- Changing socket queue size
- Turning on the TCP "No Delay" option
- Modifying the data buffer size
- Changing the number of servants on the host side.

These will of course help reduce existing network latency but to reduce the actual amount of network communication will lead to the increased performance benefits. **[Koch, Murer 1999]** outline two general rules to help us.

1. Use sequences whenever several calls to the same operation may occur

A network call is the most expensive part of distributed communications. To minimise the number of network calls will reduce the overall performance costs. Instead of a client calling a remote operation n-times, it should create a sequence of n entries and allow the operation to deal with all of these at once. The size of the in and out requests will be much larger but there will be just one network call.

2. Communicate structures that contain all or at least several attributes of an object

Another approach to reduce the amount of network calls is to group attributes inside a structure and allow the server to set or get their values at the same time rather than individually.

This type of architecture known as "Service-based Architecture" results in smaller grained objects such as customers or accounts residing within larger grained components. **[McCauley 1999]**. The users of this system will then access a "Service" rather than a small grained object.

Such an architecture is particularly suitable for the mainframe environment where existing CICS and IMS Transactions can be offered as services on an interface. This makes the integration or wrapping of such legacy transactions as CORBA interfaces rather simple.

6.4 Optimising the type of data sent or returned

Another well-known solution to CORBA performance reduction is to consider the types of data that are sent in remote invocations. **[Gokhale, Schmidt 1998]** The different CORBA data types require different marshalling and unmarshalling times depending on their complexity as some are more expensive to marshal than others.

[Slama et al. 1999] detail how that every time a distributed invocation takes place, the data being passed has to be copied from variables into a buffer by the sender and extracted from the buffer into variables by the receiver. These IDL data types map to different constructs depending on the programming language and thus have different costs associated with this marshalling and unmarshalling.

This is another CORBA optimisation that can take place in the IDL design phase of the application development lifecycle as we can chose which datatypes to use in our interface definition.

However, there is also some good news when using CORBA as outlined in **[Khandker et al. 1995]**. The performance metrics for DCE, Java/RMI and several CORBA vendor products show CORBA in a favourable light. In fact, their results show an improvement of at least five-fold over DCE for complex data types when using Java and CORBA.

6.4.1 Orders of Magnitude

As outlined above, different data types map to different constructs depending on the programming language but there are general guidelines outlining which of the CORBA data types are more or less expensive to marshal.

The "simple" datatypes are the easiest to marshall and unmarshall. An IDL short for example will be relatively fast to marshal and unmarshall since it is small and of fixed size and maps to a native data type. An octet can be even faster to unmarshall as it never has to undergo any character conversion.

An IDL string however will be more expensive since it is of variable length and in JAVA maps to an instance of class String and in PL/I maps to a pointer. An ANY is even more expensive because not only does it hold variable length data, it also has to hold typecode information detailing the type of the data held in the ANY.

Object References are the most expensive to manage. They are of variable length as the size of an object reference depends on the length of the interface, the size of the ORB-specific object key and the number of additional profiles associated with the IOR.

To unmarshall an IOR within a receiving process involves more than simply extracting data from a buffer. A proxy object must be instantiated and initialised, which involves some further overhead

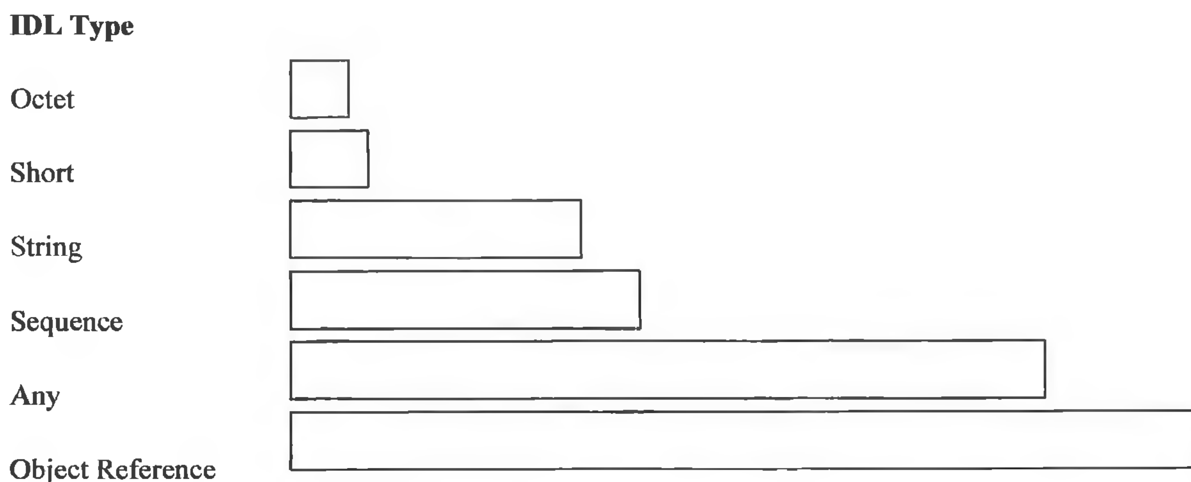


FIGURE 6.4: Marshalling Costs of the Various IDL DataTypes **[Slama et al. 1999]**

6.5 Optimising the amount of data sent or returned

Another area where we can reduce performance costs is when deciding how much data will be passed with each remote invocation. One result of our using a Service based architecture is that we will now tend to pass more information per request that a non-optimised CORBA system.

The amount of data passed can affect a system performance as, simply put, it takes longer to send a lot of data than it does to send a little data. The more data being sent the longer it will take.

The graph of Throughput versus Message Size Looks like :

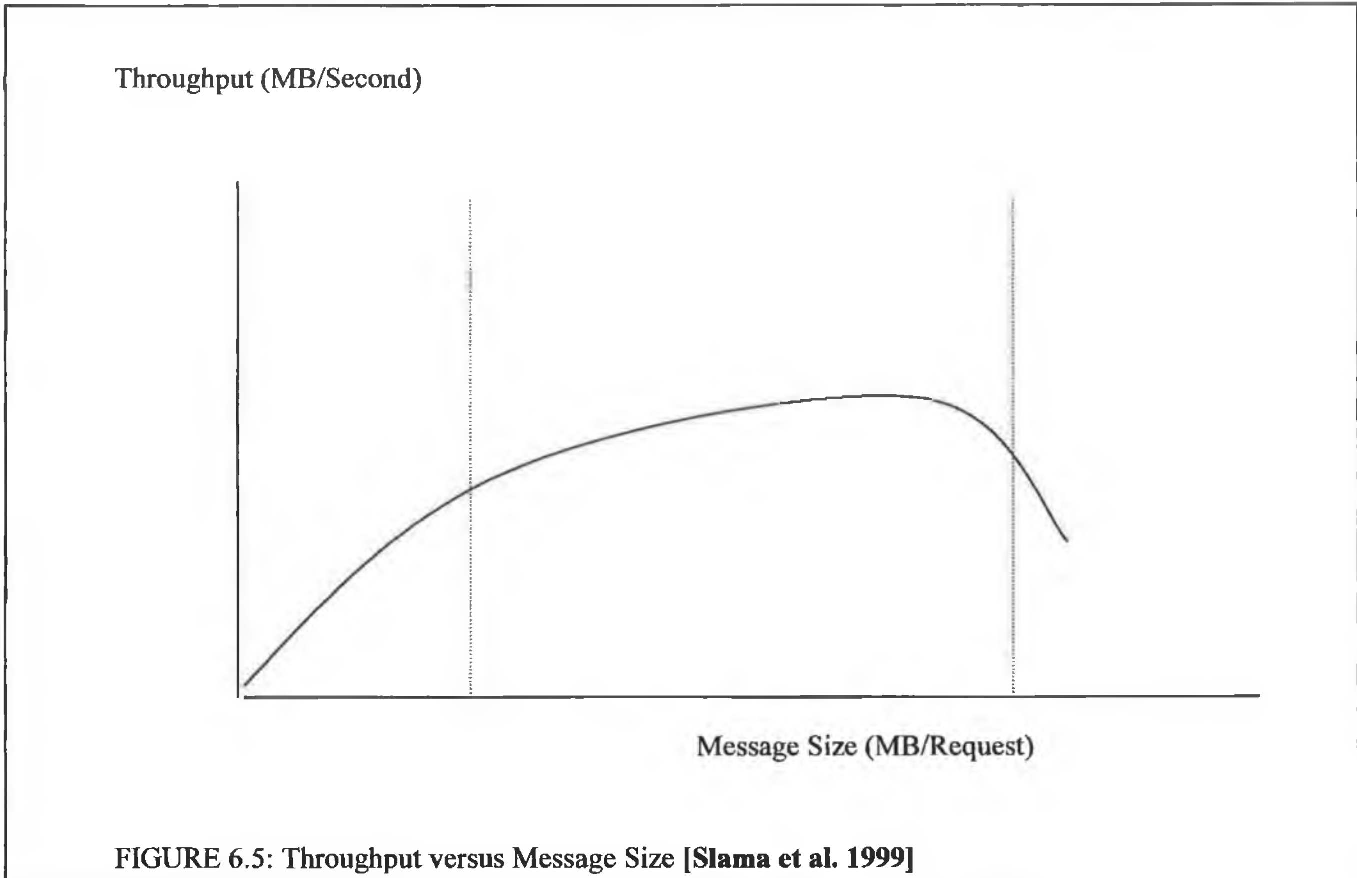


FIGURE 6.5: Throughput versus Message Size **[Slama et al. 1999]**

However, studies by **[Grahn, Holgersson 2002]** and **[Slama et al. 1999]** have shown that this in not a linear graph. Just by increasing the amount of data passed per request does not always increase throughput of the system.

The graph above outlines the approximate results of a study on throughput of CORBA requests. Naturally, this graph is different depending on platform, operating system and language but generally it tends to look like this.

With this in mind, we can once again decide at design time, just how much data we expect to send with each request and can design our IDL accordingly.

In a real world study of Audio/Video streaming using CORBA, **[Mungee, Surendran, Schmidt 1999]** found that when smaller buffer sizes were used in their experiments, there was the largest disparity between CORBA and TCP or A/V streaming implementations, with about 50% worse performance with the CORBA option. However, when the buffer sizes were increased, the ORB performance improved considerably and attained nearly the same throughput as the TCP and A/V streaming options.

Some general solutions to this problem involve reducing the amount of data sent across the wire. If the server were to pre-process the data for example and send these results to the client there could be an improvement.

There is an Iterator design pattern that is quite common and rather powerful but should be considered carefully in the performance context. **[Frölich, Gal, Franz 2002]** note that certain programming languages offer an iterator construct to traverse encapsulated data structures in a modular manner. The Iterator design pattern uses this concept when there is a large amount of data to be passed with a CORBA request/reply. Rather than returning the entire result set to the caller, a CORBA server will return an initial chunk of the data plus an iterator.

This iterator is an object-reference which the client can then use to make on to obtain further chunks of data. If the client uses a subset of the data this can be a very beneficial solution. However, if the client is always requiring the full set of data, an iterator can have a negative performance impact by adding more network calls.

When and where to use iterators very much depend on individual applications. If there will be significantly more network calls as a result then they may not provide the benefit expected. However, when the data being sent really is large, this pattern can ensure that the ORB is not overwhelmed.

6.6 Additional CORBA Patterns

There are also various well-known CORBA Patterns that can also be applied in general cases to enhance the performance of a system. We have already seen how a Service Based architecture is probably the most ideal for a mainframe based system but some or call of the others can also be considered.

Distributed Callback Pattern [Mowbray, Malveau 1997]

This pattern is useful when a client process needs the result of a service, but cannot afford to wait during processing.

If a client is using synchronous messaging, server-side processing can entail significant delays. For example, a client application needs to guarantee a reasonable response time to a user. A user interface program must have continual awareness of user events and respond accordingly.

In CORBA terms this pattern often entails the use callbacks so that the client does not have to wait. We will see some more of this in the section on Scalability.

Fine Grained Framework Pattern [Mowbray, Malveau 1997]

This pattern is used to define and use fine-grained objects in a distributed system without incurring prohibitive costs in terms of performance and system complexity.

In a Service based architecture, coarse-grained objects are typically preferred but this pattern can also be considered.

Independent Objects Pattern [Mowbray, Malveau 1997]

This pattern is used to resolve processing bottlenecks due to tight coupling of implementations, such as separating the factory implementation from the objects that it creates.

Instant Reference Pattern [Mowbray, Malveau 1997]

The Instant Reference Pattern is used to optimise performance of object instances through shared sever implementations. It provides a mechanism of mapping from the implementation of an object's interface to a specific object instance.

Library Skeleton Pattern [Mowbray, Malveau 1997]

The Library Skeleton Pattern can be used to limit the amount of network calls by collocation of clients and object implementations. Again, any attempt to reduce the amount of network calls in a system will provide immediate performance improvements.

Partial Processing Pattern [Mowbray, Malveau 1997]

The Partial Processing Pattern can be used to improve the performance of a CORBA-based application by optimising the amount of parallelism.

Replication Pattern [Mowbray, Malveau 1997]

This pattern shows how to provide improved performance and reliability by replicating an object in multiple distributed locations.

Load Balancing is the technique of spreading the work of a server over many servers that support the same implementation. Replication is most often used to implement a load-balancing policy. We will investigate Load Balancing and Replication in a later chapter.

Flyweight [Gamma 1995]

The Flyweight pattern outlines a design pattern that can be uses sharing to support large numbers of fine-grained object efficiently. In a Service-based architecture there will typically be a few large-grained objects that contain many finer-grained objects. As a result, there is not a need to manage large numbers of fine-grained objects.

Interceptor [*from the OMG CORBA Specification*]

The interceptor pattern enables the transparent adding of services to a framework and for them to be triggered automatically when certain events occur. As per the CORBA specification, there is a request interceptor that is designed to intercept the flow of a request/reply sequence through the ORB allowing a service to transfer context information between clients and servers.

6.7 Conclusion

We have seen in this section that the largest factor that affects the performance in a distributed system is network latency. What we have provided here are various approaches to reducing this network latency through the design of performance enhancing IDL and by adopting a Service-Based Coarse grained Architecture.

[DSRG 1999] point out in their report, that there is also a different in performance between different ORB implementations and an organisation should be aware of these differences before committing to a product.

In the Sections on Scalability and Availability there are further approaches and patterns that can be used to further enhance the performance of the network by balancing the load on the system, by managing the connections in the system, by managing the sessions and by providing multi-threading.

Unfortunately the addition of Security can impose performance overheads too but we want to reach a status quo where the CORBA Service only adds a marginal performance overhead when compared to that of the legacy application that preceded it.

The CORBA specification does provide enhancements such as the interceptor pattern to further assist system designers, but **[Schmidt et al. 1997]** reckon that CORBA in general is not well suited for performance-sensitive real-time applications due to lack of standard quality of service policies and mechanisms.

They also argue that CORBA has a lack of real-time features as well as a lack of performance optimisations. In addition **[Wang, Schmidt, Levine 2000]** outline a possible extension to the ORB specification defining a *local* keyword to the IDL syntax that would support locality constrained object interfaces. This keyword would allow developers to define and use their own locality-constrained objects to avoid unnecessary traffic and marshalling/demarshalling operations. This solution is also an example of the optimising principle pattern *Avoiding gratuitous waste*.

There is an OMG Realtime Special Interest Group to consider all such extensions to the core specification.

Among the initial goals of the research was to find areas such as system performance, which could prevent the enterprise integration project being successful. We have seen some industry standard solutions in for improving performance in a distributed environment. In a later section we will see the application of the approaches recommended here and how they can be used in real-world situations.

7 Security Issues

7.1 Mainframe Security

Previously, the mainframe was typically inside the private company network and so the chance of attack from the outside was often not of considerable concern for System Administrators. However, as we are now making Legacy Applications available to the Intranet and to external Internet users requiring banking functions, the industry needs to completely re-evaluate its security measures.

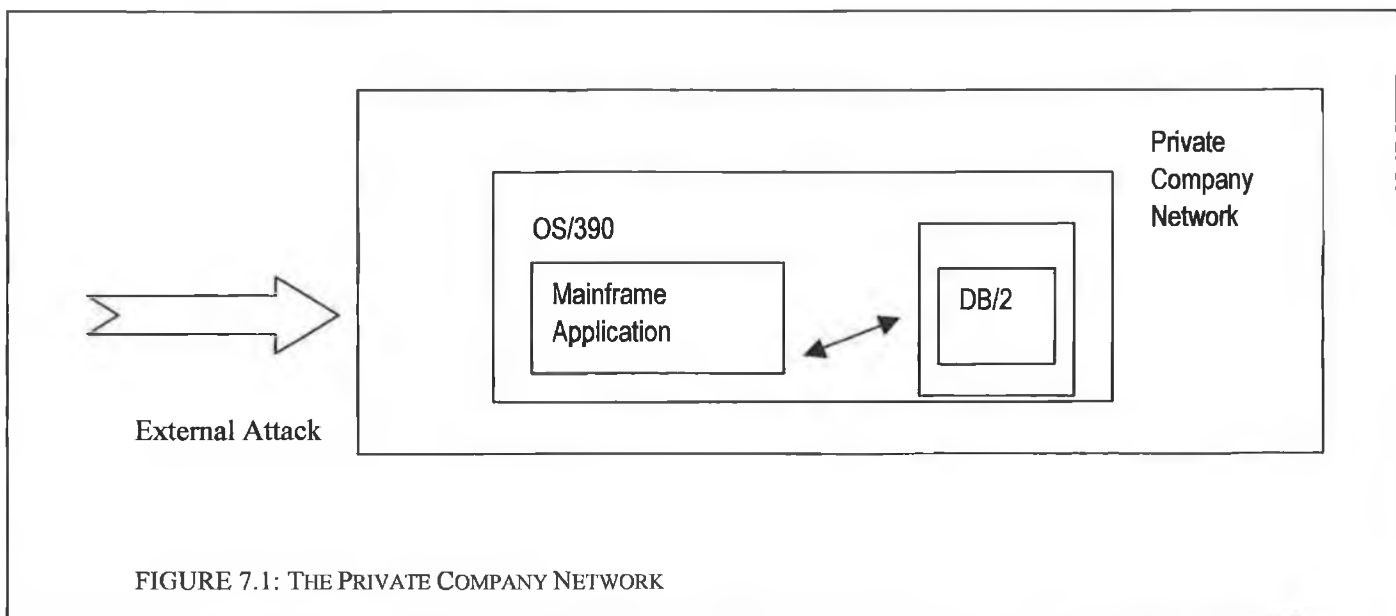


FIGURE 7.1: THE PRIVATE COMPANY NETWORK

Typically, mainframe security utilised the IBM RACF or Computer Associates ACF2 on OS390/MVS. These access control facility models are a centralised and well-proven security model. They work by controlling access to mainframe resources by legitimate users.

Where once it was the case that no requests from outside the Private Company Network should get through (with some exceptions but over secure lines), we now need to support requests from a variety of different clients (some of whom will be external to the network). These clients will end up accessing secure data.

As noted in **[Slama et al. 1999]**, enterprise systems are moving increasingly into the Internet environment. This typically involves building Java front ends to existing systems, or developing new Internet-focused systems. With this, come requirements for thin client models, and therefore lightweight security libraries not typically available from, or suitable to, traditional DCE/RACF security solutions.

However, even with the traditional security mechanisms, security is never absolute. **[Johnson 1989]** states that the probability of a compromise may approach, but is never, zero. Securing a system is the act of moving that probability closer to zero than it was before

7.1.1 RACF (Resource Access Control Facility)

The RACF component of OS/390 works together with the system to protect critical data in the enterprise and give only authorised users to this data.

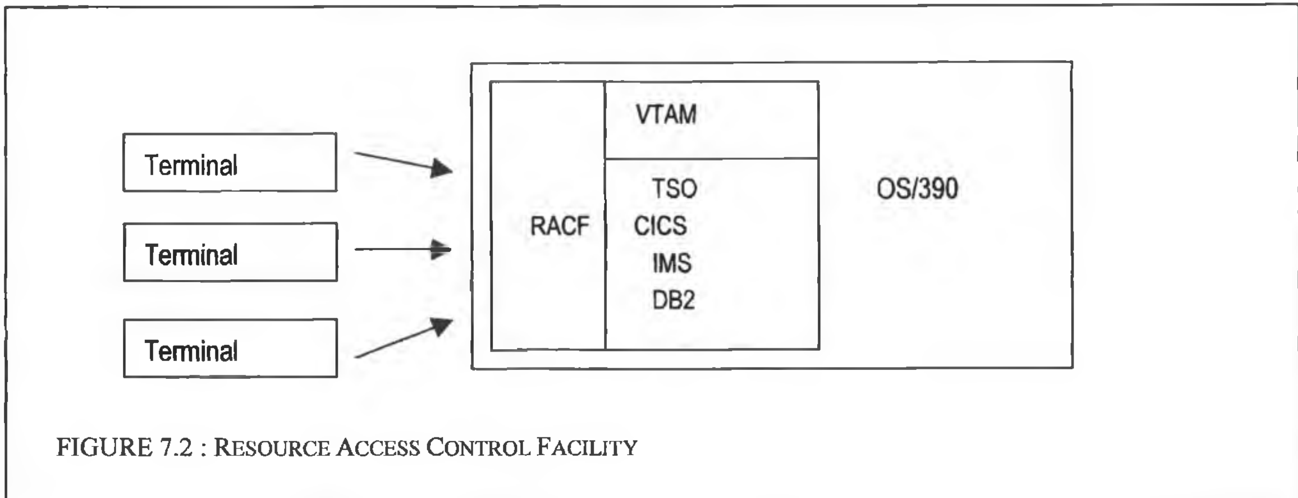


FIGURE 7.2 : RESOURCE ACCESS CONTROL FACILITY

RACF uses a unique user ID to identify each person trying to access the system and a password to authenticate that identity. After the system has been accessed, RACF then controls the level of access authority for each resource a user tries to get at. It will check the security classification of both the user and the data and give access if the required conditions are met.

RACF can be tailored to interact with the bank's operating environment and adapt to its changing security needs. The RACF remote sharing facility gives you the flexibility to move work from one system to another and administer several systems from a central database.

7.1.2 RACF Record Keeping

One of the important features of a security model as we shall see in modern requirements is some form security logging.

RACF keeps statistical information, such as the date, time, and number of times a user enters a system, and the number of times a specific resource was accessed by any one user.

RACF also writes security log records to help you verify the security of the system. And, RACF provides utilities that create reports from this data to help you detect possible security exposures.

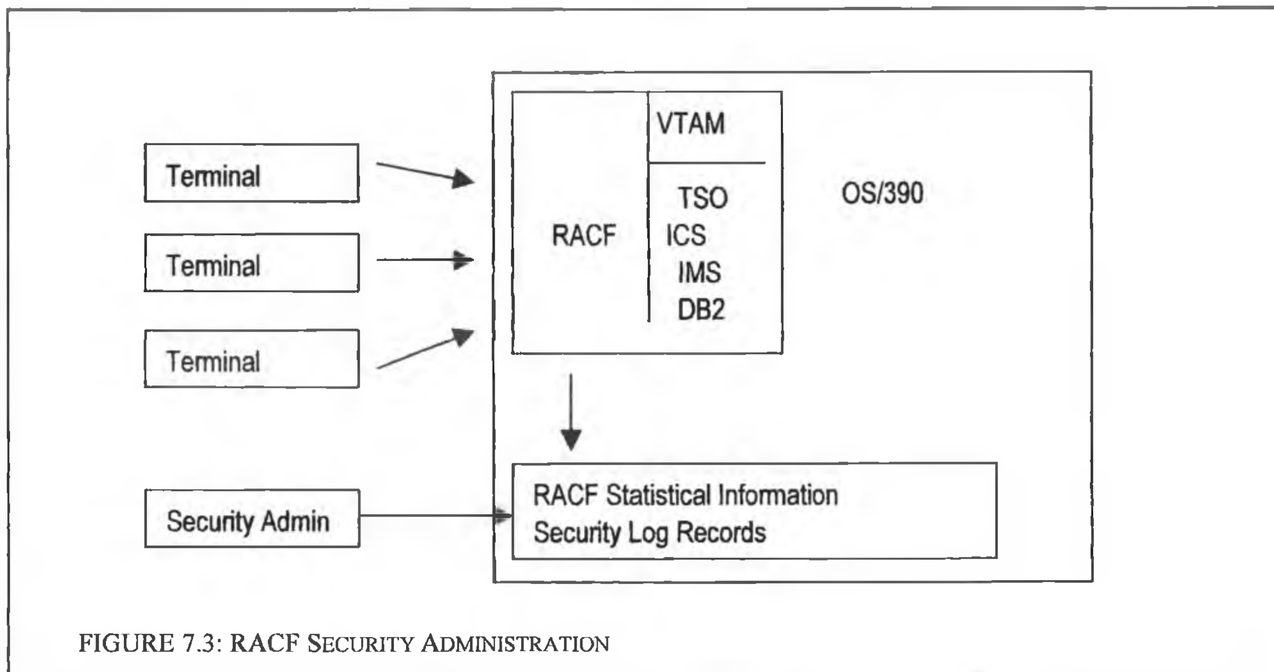


FIGURE 7.3: RACF SECURITY ADMINISTRATION

7.1.3 Mainframe Security in a Client/Server Model

It must be noted that the traditional mainframe security model has been tried and tested and is regarded as an adequate security mechanism when inside the private company network. Opening this network to external clients in addition to internal distributed systems with the mainframe as a peer, lead to immediate questions as to the effectiveness of this model.

[Mowbray Malveau 1999] rate security as an important aspect of management of IT resources. The secure control of information and services is becoming more important as systems become increasingly networked and Interoperable

As **[Lang 1997]** points out, it is more difficult to establish a basis of trust in distributed systems. More than one global systems mechanism needs to be trusted. The degree of trust in elements of distributed object systems may change over time whereas in mainframe systems trustworthiness is typically static.

In modern object-oriented systems, it is also true that objects cannot be trusted to enforce their own security, since application developers may lack the necessary security knowledge or may simply not want any security-related performance overhead.

As we will see in later sections, PKI (Public Key Infrastructure) and SSL (Secure Sockets Layer) are among the most popular security systems used in decentral computing systems. If we can use a combination of PKI, SSL and traditional mainframe security to protect the key data we can go a long way towards satisfying the most nervous of Managers that the critical data is safe.

RACF has been designed however so that it can be integrated into a system that uses public key SSL technology and this will make our task somewhat easier. It has various defined mechanisms to allow it use SSL integration with its own access control facility.

7.2 Banking Security Requirements

In Chapter 2 we discussed and formulated the business requirements of modern Banks and we saw some of the I.T. systems that were required to meet these requirements. A modern bank will require many different types of functionality in their systems but will also require additional aspects such as mail servers to handle e-mail, web servers to handle middle-tier CORBA and EJB servers and internal applications to administer the company.

However **[Slama et al. 1999]** rightly point out that with these additional requirements, more possible attack points are created and protection of non-public information from unauthorised users is as critical as ever before. Essentially, the biggest challenge facing developers of secure systems today is trying to find a comprehensive solution that can address all of the features required of the system, while still providing room for the system to evolve and grow.

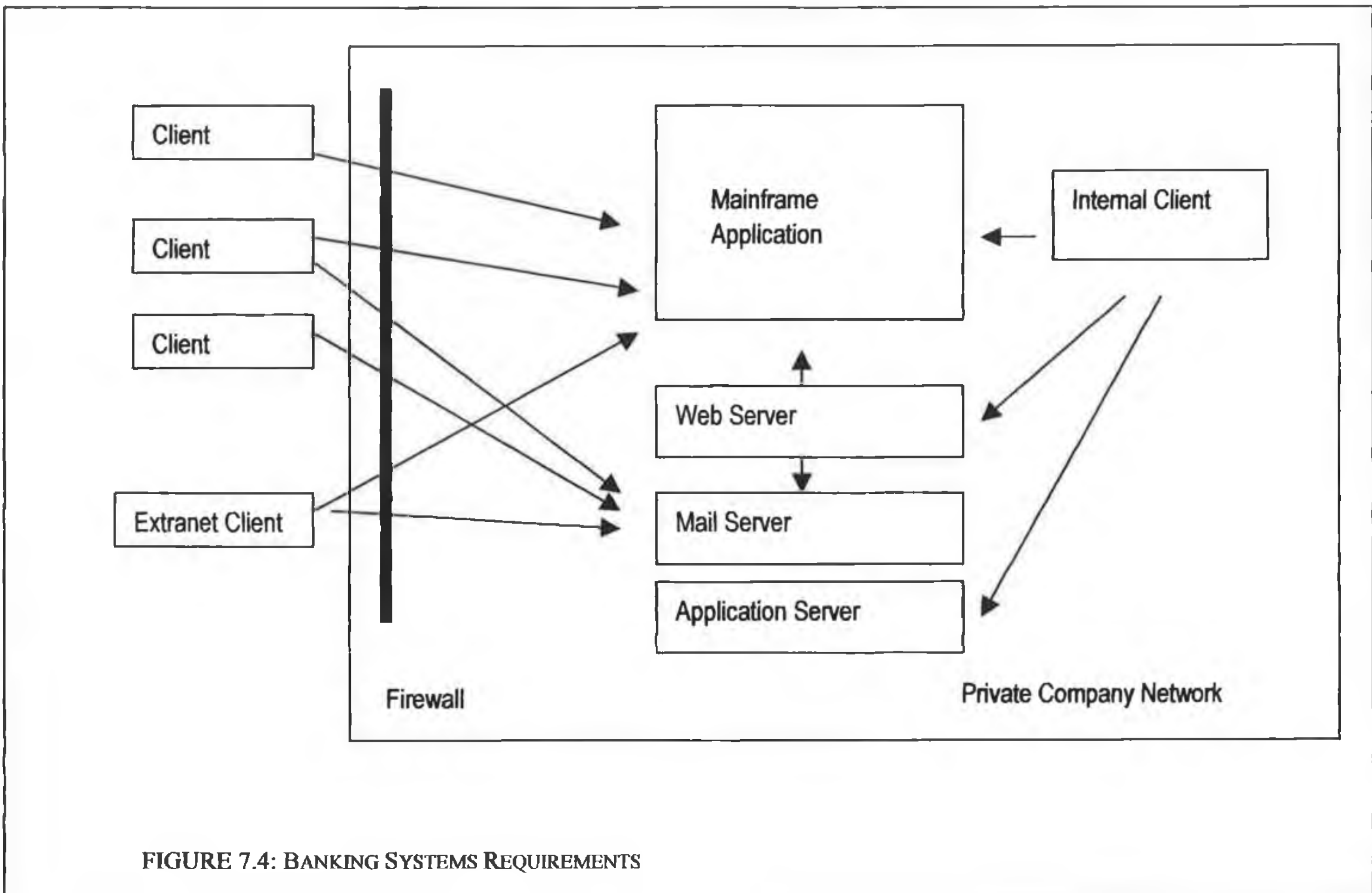


FIGURE 7.4: BANKING SYSTEMS REQUIREMENTS

It is easy to see that the diagram above is flawed and would be difficult to implement. For example would we really want an external client to have direct access to a mainframe application? Even if this was to be allowed, how could we guarantee the security of this connection. Rather, this diagram should be viewed in terms of the requirements of the banking systems. It is the case that external clients might want access to their account information residing on a mainframe but in reality, they would not be allowed direct access.

7.3 Threats to Banking Security

For very obvious reasons, the security of ones money in a bank is taken for granted by the average consumer. It would be unacceptable to log-on to ones account one morning to find that "someone else had been there first". From the Banks perspective, this complete faith from their customers that their assets are safe must be met without question.

From a systems point of view, there are various threats that apply to each part of a bank's systems and these must be broken down and considered.

7.3.1 Internal Network Security

The internal network security that a bank must provide includes authenticating users to ensure they say who they say they are. There must be control over who has access to resources. These security requirements apply right across the internal enterprise.

In addition, the security model that was used on the mainframe, RACF for example must be able to be integrated with different security models such as SSL and PKI.

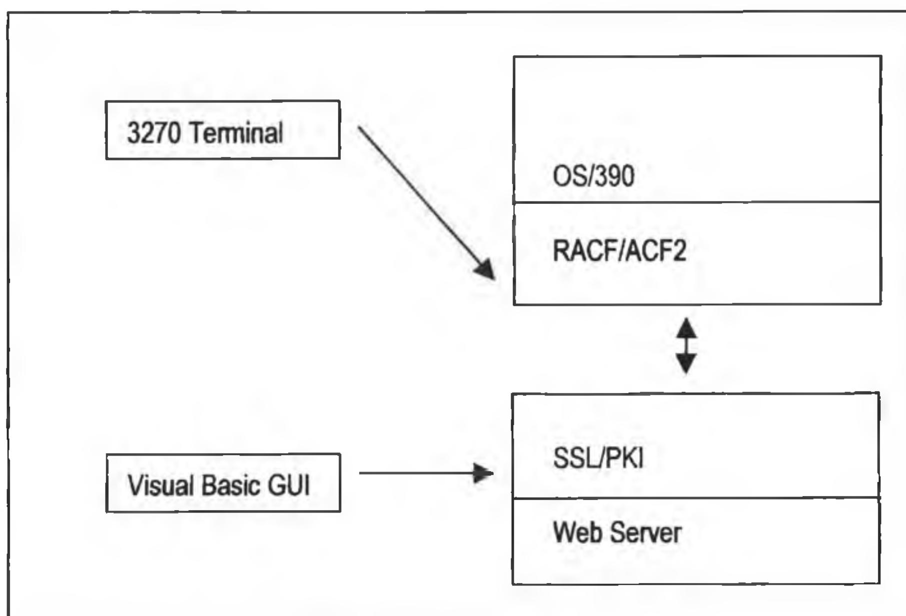


FIGURE 7.5: INTERNAL NETWORK SECURITY REQUIREMENTS

7.3.2 Internet and Extranet Security

Most banks have as business requirements public internet access. This access is typically provided to provide easy access for a Banks' customers but at the same time there is a clear requirement to protect the resources and to monitor unauthorised intrusions and actual attacks.

Likewise for Extranet users there is a requirement to authenticate access and to provide access control for authorised users whilst also securing communications and ensuring confidentiality.

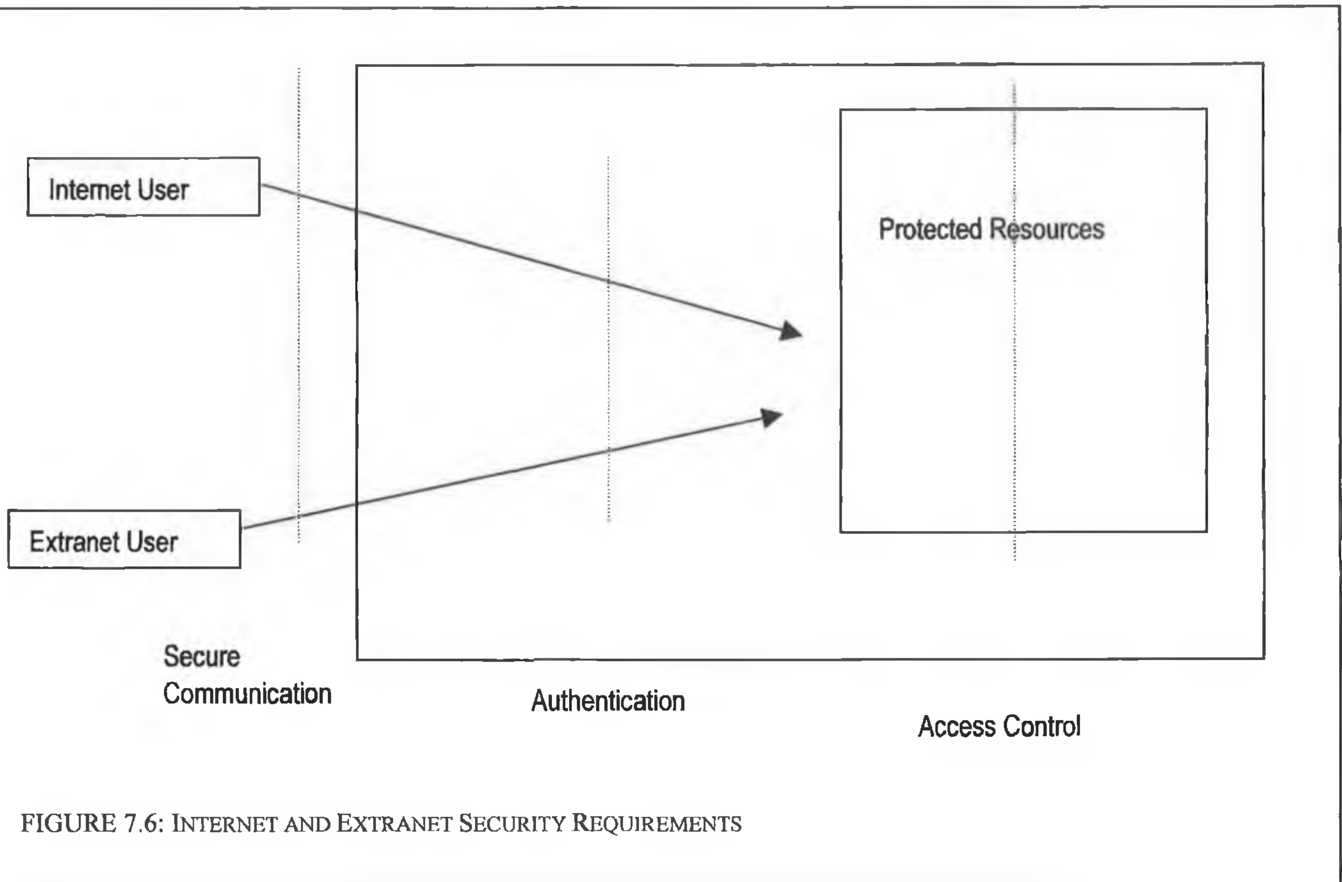


FIGURE 7.6: INTERNET AND EXTRANET SECURITY REQUIREMENTS

7.3.3 Security Threats

There are different categories of Security Threats in terms of seriousness for the banks and likewise there are different causes of Security threats and these can be broken down into deliberate or accidental.

7.3.3.1 Deliberate Security Threats

Deliberate Security threats typically come from malicious individuals or groups outside the private company network but increasingly such attacks are coming from disgruntled or agitated employees from inside the network.

Examples of such attacks include brute fore attacks where all possible combinations of a password to a protected resource are tried, usually via a piece of software or mechanical tool. Another popular attack from outside hackers are web spoofing attacks where a false web page might be used and the users of the bank might be tricked into giving their information.

Other individuals attempt to gain access to the protected resources by falsifying IP addresses so that a malicious packet can get through the firewall or router that guard the entrance to the private network. The firewall or router believes the IP address to be valid and allows the packet through.

Another attack on large organisations that has been known is disabling a particular service on a network or the entire network for example by using a specially constructed packet to crash the network or continuously sending packets until you flood and crash the server. This form of attack is known as denial of service.

Attacks from inside the company might include deliberately deleting protected resources so that the system as a whole fails. Giving protected passwords to outside parties is another way of compromising security from the inside as is modifying data such as account balances and overdraft limits.

All of these attacks along with the regular viruses and worms that can cause chaos with the worlds computers are very real threats and need to be taken seriously. Any security policy that a bank may implement will take a look at each of these in turn.

7.3.3.2 Accidental Security Threats

There is also a threat to a Banks resources from accidental attacks on the protected resource. These usually happen when an individual modifies or deletes restricted resources by mistake and may not even realise the consequences of their actions.

Large organisations have to be especially careful as to how gets access to the inner restricted resources so that only those with knowledge of what they are doing can modify or delete such critical information.

7.4 Required Security Services

There are certain security services that every bank needs to protect its resources. The most basic of these need to be supported by any security policy implemented by the bank. These include:

7.4.1 Security Services

Authentication - Proving that someone is who they say they are.

Access Control - Giving a user access to different parts of the system, also known as authorisation.

Integrity - Proving that a message sent from a user has not been tampered or altered in any way.

Confidentiality - Maintaining the privacy of messages sent from a user

Auditing - Logging of user interactions and tracking of these actions for reporting purposes. We want to know who did what and when.

Non-Repudiation - Users must take responsibility for their actions. We must ensure that someone cannot do something and then claim they did not.

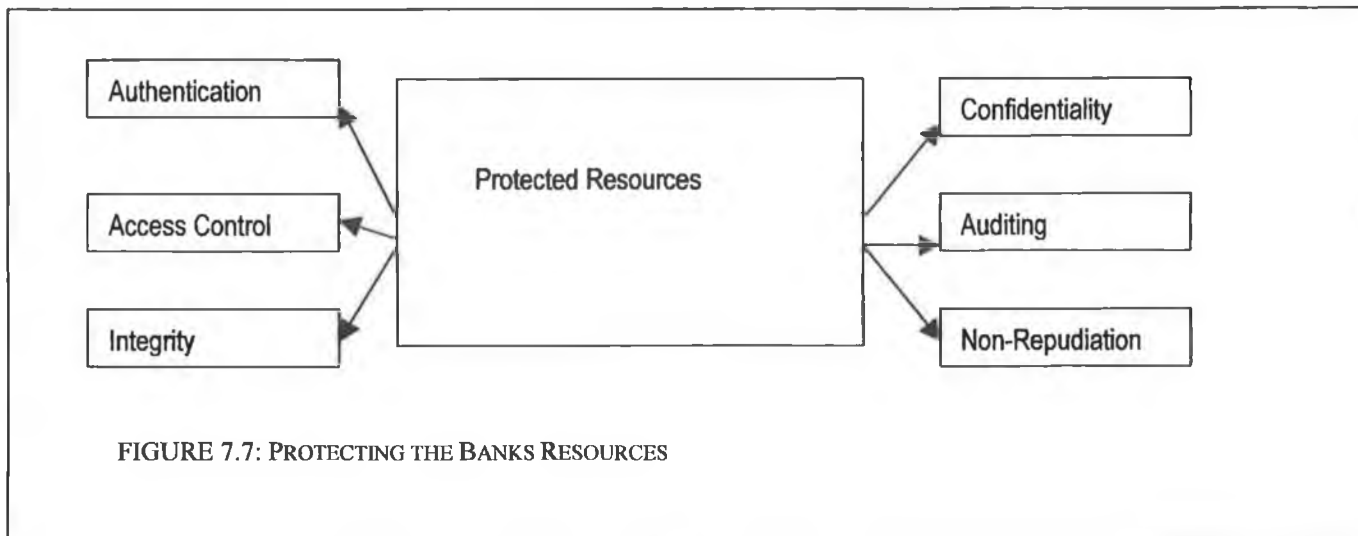


FIGURE 7.7: PROTECTING THE BANKS RESOURCES

7.4.2 Applying Security Services

Each of these services detailed above has a very real place in modern banking systems. What we need to do is find the difference in the services offered and work out where they should be applied in the different places in our system.

Any security policy needs to be enforceable both technically and organisationally and needs to address all aspects of security. To address these aspects there must be a certain amount of risk management applied - i.e. a bank must decide what needs to be protected and how well it needs to be protected.

There must also be an investigation into the cost of security, there is no benefit in spending large amounts on a security system that is protecting resources which if compromised will only cost the bank a fraction of this amount in losses.

One of the major problems in finding an adequate security solution is outlined by **[Slama et al. 1999]**. They note that the today's market is showing the strains of having emerged quickly from a DCE- and RACF- dominated world into an Internet arena that is focused on more lightweight technologies than the established DCE or RACF solutions.

There is a myriad of emerging standards, technologies, and products that solve one piece of the security puzzle or another, but do not yet interoperate well.

7.4.3 Addressing different System Areas

There are different areas of a banks' IT system that need to be addressed. These include the procedural, the physical and logical security of the system.

The **Procedural security** is the administration of the system. For a bank this would include new user accounts, disaster planning, backup procedures etc. The **Physical Security** includes access to buildings, rooms, machines, disk drives, printers, etc.
The **Logical Security** includes authentication, access control, integrity, confidentiality, non-repudiation and auditing.

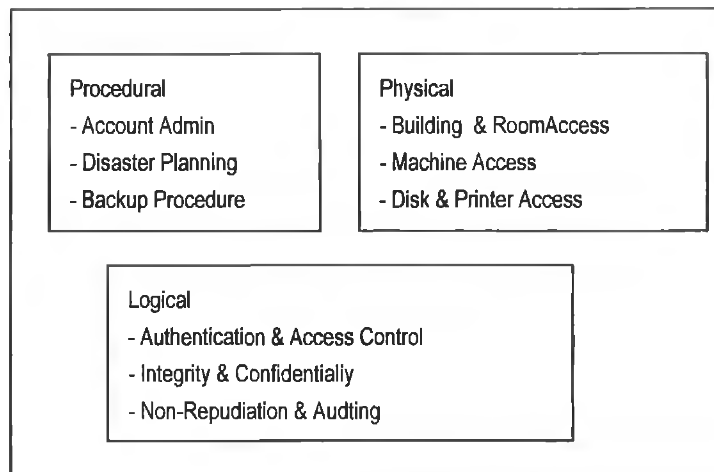


FIGURE 7.8: DIFFERENT SECURITY AREAS

There are further design and architectural issues to be considered when implementing any security policy. A decision must be made whether to deny access to all services and only allow those that are required. Perhaps known security vulnerabilities such as services and ports should be restricted and maybe access to and from certain hosts should also be restricted. The question of how much monitoring is required should also be looked into.

Someone needs to decide how much time and money needs to be spent on the security system both in terms of money and in terms of hours spent on implementation and design.

7.5 Firewall Technology

In a previous section we have looked at traditional mainframe security that provided adequate security for earlier distributed systems. However, as was mentioned, since we now required heterogeneous and distributed computing both inside and outside the private company network, new solutions have to be considered.

Firewalls are a technology than can be used to protect the perimeter as well as providing access control and auditing to protect the systems resources. Firewalls are in essence like a brick wall in a building that prevents a fire from spreading. An Internet Firewall is a facility to secure a web-sites perimeter.

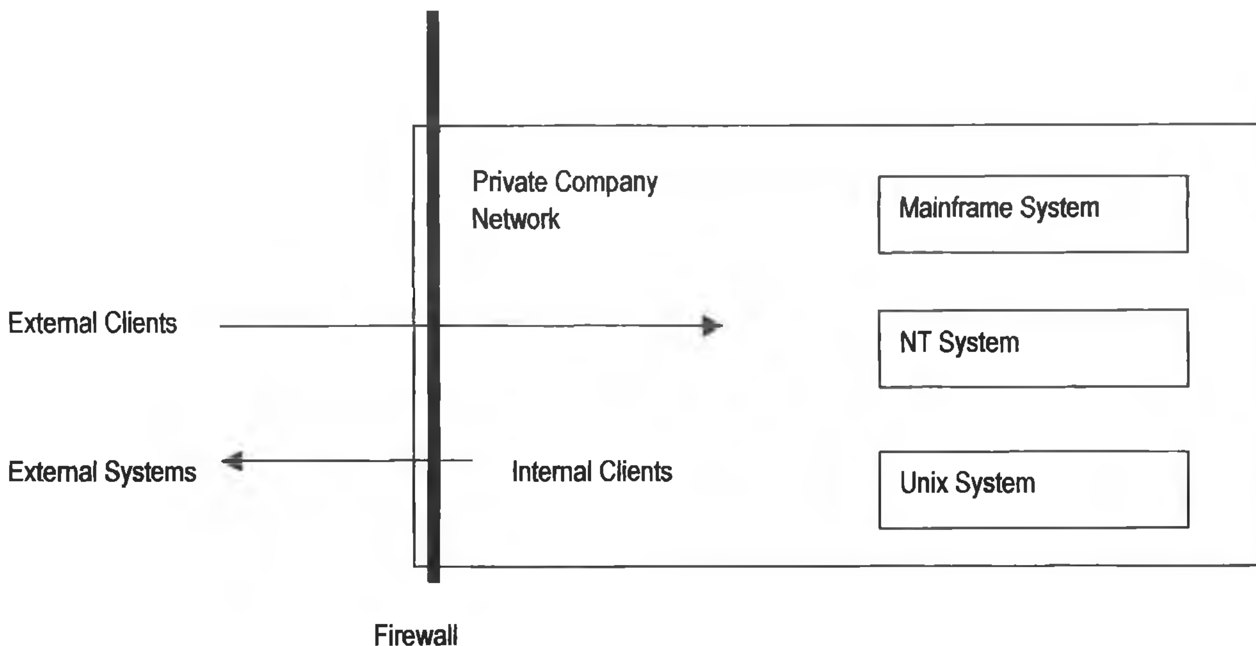


FIGURE 7.9: FIREWALL TECHNOLOGY

7.5.1 Firewall Security

Firewall Technology can be added at the boundary of a private company network and it can be used to control access to and from the private network. Essentially it does this by providing a single entry point to the system at which point security authorisations can be added and auditing procedures implemented.

The addition of this authorisation will restrict the services that can be accessed and reduces security vulnerabilities. Likewise, a firewall can stop unauthenticated interactive logins from the Internet to stop unwelcome individuals accessing the system resources. However, a firewall will not protect against viruses and other forms of external attack as detailed previously.

A firewall can also work in the other direction and can be used to restrict internal employees accessing external systems. This technology will still not protect against disgruntled employees and those on the inside wishing to do damage.

It should be noted that a firewall by itself will not provide all the security required as there are many other ways to compromise a banks security, other measures will be required and we can see these in the following sections.

There are different types of firewalls include Packet Filter which are simple packet routers that can make basic access decisions (sometimes called Network Level). Proxy servers understand protocols and can filter based on this (sometimes-called Application level).

[Slama et al. 1999] point out how, in the Internet-oriented world, firewall technology is understood and established, but does not solve all the issues around system and data protection. .

The OMG provide now provide a CORBA Firewall specification. This details how, in a CORBA environment, firewalls are used to protect objects from clients in other networks or sub-networks. A firewall will either permit access from another network to a particular object or will prevent it.

When access through a firewall is permitted this may be at various levels of granularity. For example, access could be permitted to some objects behind the firewall, or access could be restricted to certain operations on particular objects.

An enclave is a group of objects protected by a firewall. The firewall protects the enclave's network (or sub-net) by separating it from other enclaves and/or the Internet at large. The separation is the result of the fact that all communication between the enclave and the outside must pass through the enclave firewall (or one of its firewalls, if there are several). Firewalls have two distinct duties: inbound protection and outbound protection. Inbound protections are used to control external access to internal resources. Outbound protections are used to limit the outside resources that can be accessed from within the enclave.

For a real-world example, the Xtradyne Domain Boundary Controller (DBC) is a CORBA Firewall (application layer firewall) that securely transmits CORBA requests and replies across the domain boundaries including packet filter firewalls and NAT Routers. Acting as a CORBA Firewall the DBC checks the correctness of IIOP messages (or RMI/IIOP messages respectively) and filters out hostile and destructive messages.

7.5.2 Packet Filters

Packet Filters are routers that can base access decisions on source destination addresses and ports in the IP packets. There are generally mounted on a single secured host, a bastion host and separated from the internal network. They provide a single point of access and are usually transparent to users. There are some more sophisticated products that hold internal info about the state of connections.

7.5.3 Proxy Servers

Proxy Servers are pieces of application component software on a firewall that understand the application protocol. These are especially useful when using a standard such as CORBA as there exists on the marketplace such Proxy Servers that understand IIOP.

These Proxy Servers can make more detailed access decisions based on the protocol message and only allow certain types of messages to pass in either direction.

There are varying levels of sophistication and some Proxy Servers can perform user authentication and various auditing procedures.

In the CORBA world, use of these Proxy Servers mean that every IIOP request passing in or out of the private company network would pass through and this can impact performance. They are

also typically protocol specific and if a new application or protocol were added to the system, then a new Proxy Server would need to be added.

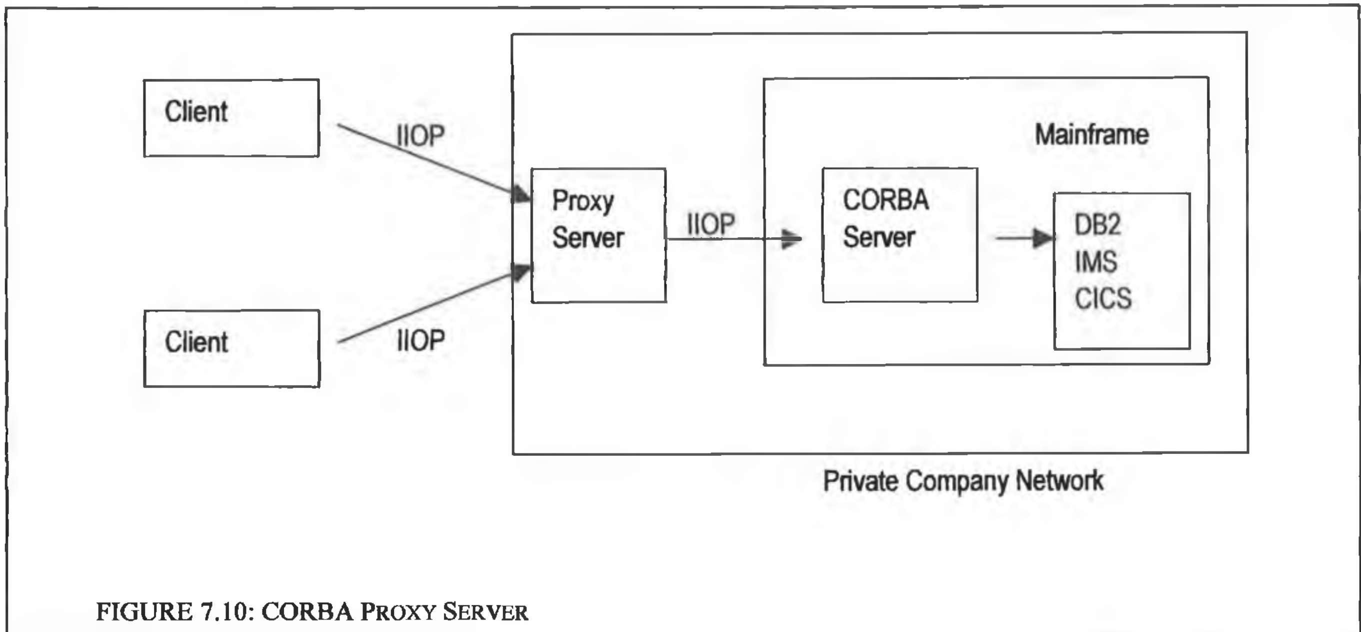


FIGURE 7.10: CORBA PROXY SERVER

In the diagram above, an IIOP Proxy Server sits at the entrance to the private company network. Valid IIOP requests are allowed to pass through to the CORBA server in the network and valid IIOP results are allowed to pass back in the other direction.

7.6 De-Militarised Zones

The concept of a De-Militarised Zone (DMZ) is an additional way of providing more security when using firewalls. It completely isolates the internal network from the external internet by placing an intermediate/buffer network in between them and this buffer network is the De-Militarised Zone (DMZ).

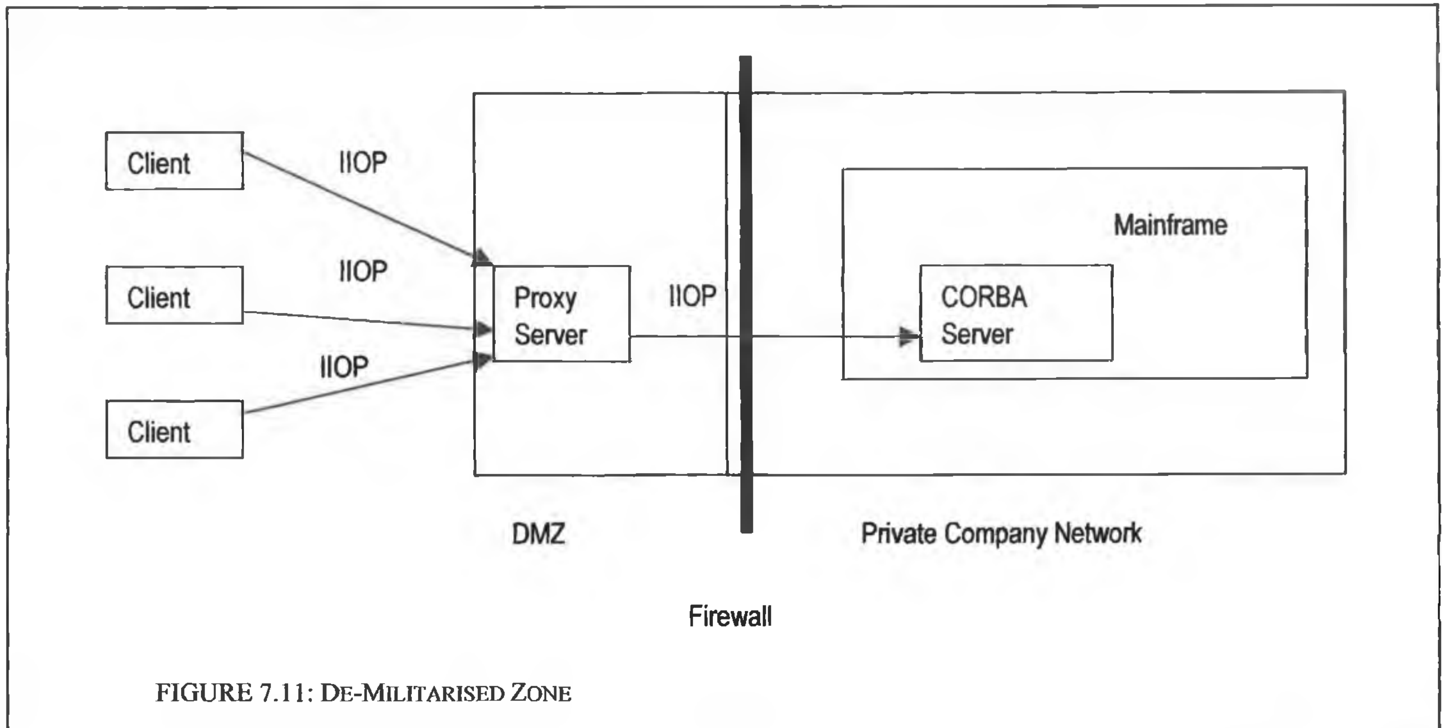


FIGURE 7.11: DE-MILITARISED ZONE

It can be arranged to have multiple subnets in the DMZ each with their own security level. This provides another level of security. The reason you might want to have such multiple subnets is because often breaking into an unsecured host/subnet can then allow a hacker to abuse or utilise trust relationships with more secure host/subnets. So by isolating hosts of different security levels from each other, you are providing another barrier to the hacker.

A DMZ can also act as a Network Address Translator (NAT). NAT is also known as IP masquerading because it basically hides internal hosts from the outside world. It hides IP address by converting them to the address of the firewall, and so it hides all TCP/IP-level information from hackers.

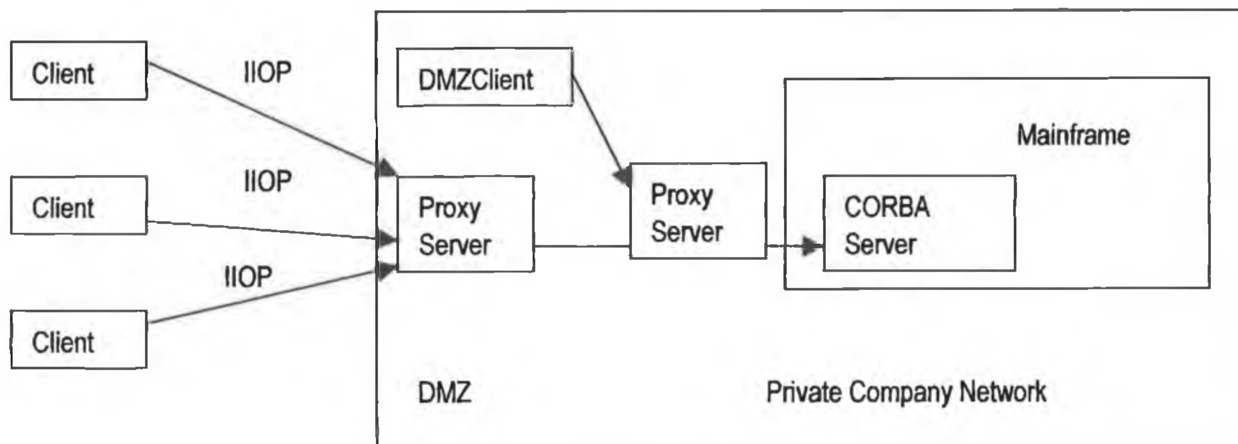


FIGURE 7.12: MULTIPLE PROXY SERVERS

There can be multiple proxy-servers acting as firewalls. In the diagram above there is a firewall between the Internet and the DMZ and another firewall between the DM and the internal network. In this case an external client never has direct access to internal hosts (even Proxy Servers on the internal network)

7.7 Public Key Infrastructure

Public Key Infrastructure is one of the more popular solutions available on the market today. Its role is essentially to support public key technologies that provide authentication, integrity and confidentiality.

Public Key Technologies make use of the SSL protocol and X.509 to provide the required elements of a security model that can provide secure communications by ensuring integrity and confidentiality. Such a solution is an absolute requirement for banks that are looking at expanding their mainframe based legacy systems to work as true peers in a heterogeneous distributed environment.

7.7.1 Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol was originally defined by Netscape and is a transport layer security protocol layered between application protocols and TCP/IP. It is a lightweight Internet security solution but is very useful for our requirements as it fits nicely between IIOP and TCP/IP to provide a security layer. The SSL protocol uses RSA (Rivest Shamir Adlemann) public key cryptography for authentication.

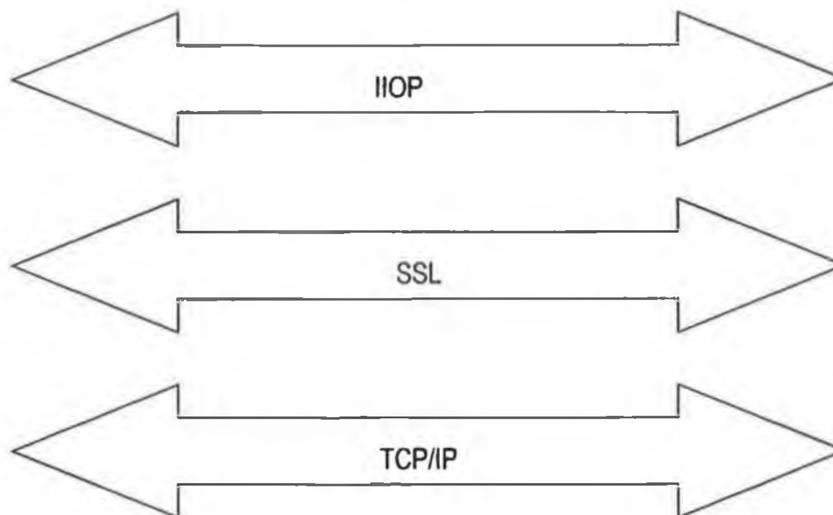


FIGURE 7.13: SSL PROTOCOL LAYERED BETWEEN IIOP AND TCP/IP

SSL authentication is based upon the use of Certificates. These Certificates are signed messages specifying a name, a public key and the name of the Issuing Certification Authority. We will see the full details in the next section on X.509 certificates.

Commonly known as "credit card security" as defined by **[Slama et al. 1999]**. SSL is extremely well suited to the security needs of Internet commerce systems. Fast, lightweight, but providing a robust and strong level of security, it is ideally suited to the Internet where public servers are common, but protection of sensitive information such as credit card details in transit is of extreme importance.

There are many successful products on the market that deal with these issues, providing full support for SSL and X509 security standards, and that can be used with the CORBA security services to manage a large-scale secure distributed system.

The thinking behind SSL Authentication involves the use of public key cryptography. In public key cryptography, each application has an associated asymmetric public key and private key pair. Client messages are encrypted using the server's public key and the server decrypts the message using its private key.

When sending a reply, the server encrypts this with its own private key and the client decrypts it again using the server's public key. This Public Key Cryptography is also known as asymmetric key cryptography. There is an overhead, in that the authentication handshake adds an extra 5-20% performance overhead, but this is reduced for the remainder of the connection.

Symmetric (or secret key) cryptography relies on the client and server sharing a single key, which is used to both encrypt and decrypt a message. Symmetric key cryptography is faster and more efficient than asymmetric key cryptography. The most widely known and used symmetric key algorithm is the Data Encryption Standard (DES) and its more secure derivative Triple-DES (3DES).

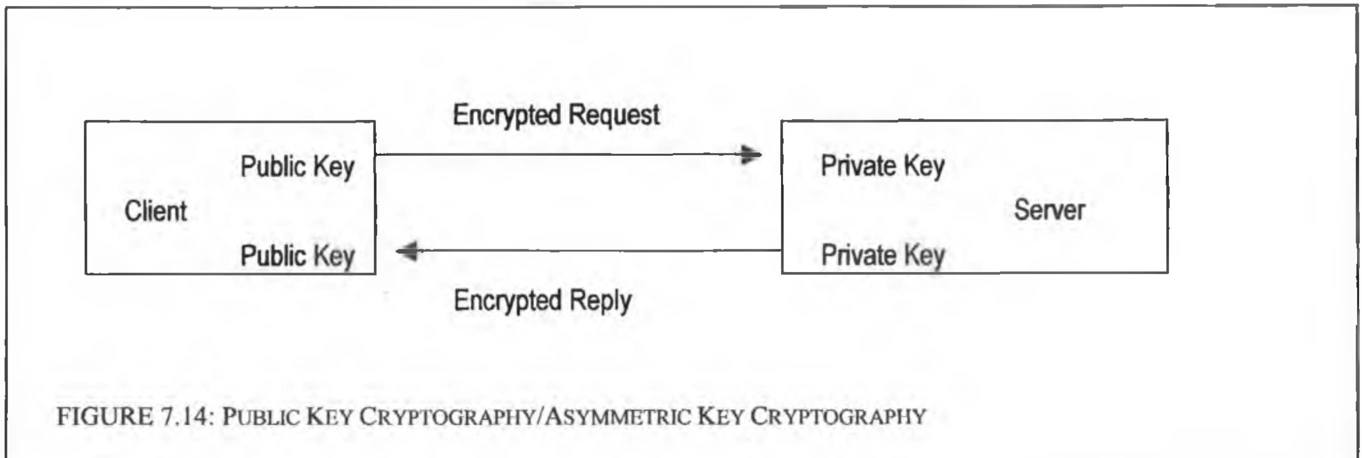


FIGURE 7.14: PUBLIC KEY CRYPTOGRAPHY/ASYMMETRIC KEY CRYPTOGRAPHY

SSL also defines cipher-suites. These cipher-suites are groups of mechanisms that the protocol uses. It allows client and server to negotiate and agree common mechanisms by selection from a list of supported cipher-suites. Some cipher-suites are intrinsically more secure than others because the mechanisms they used are deemed cryptographically stronger meaning that they are harder to break.

Another of the more beneficial features SSL provides is confidentiality. After authentication, SSL client application sends a once off encoded data value to the server - the "session key". A Session Key is a key to a secret key cryptographic algorithm, chosen for efficiency. Communications between client and server are then encoded using the agreed secret key cryptographic algorithm. This key is called a session key because it is only used once for single session between the client and server.

SSL provides integrity as it adds a Message Authentication Code (MAC) to each message. This MAC is like a hash value/checksum for the message.

7.7.2 X.509 Certificates

X.509 certificates are ASCII files that match public key and name. The X.509 is standard format for certificates defined by the International Telecommunications Union (ITU). An X.509 certificate includes information such as the security name of the entity identified by the certificate, the public key of the entity and the name of the Certification Authority that issued the certificate. This CA that is used for signing certificates can be either a private or commercial CA).

The Security Name part of the X.509 certificate also contains further fields such as the Common Name, the Organisation Unit, the Organisation, the Region and the Country as well as an expiry date.

The Public Key is publicly available, allowing users to encrypt messages to the owner of this key. Only the holder of the inverse Private Key (password protected) can decrypt the message.

7.7.3 Certificate Authority

A Certificate Authority (CA) is the authority that is primarily concerned with proving the identity of users and generation of X.509 certificates for those users. Trust of these certificates is only possible because they are signed by the CA, and this CA is trusted.

The CA can be trusted because their public key is so widely known or easily available that it can usually be accessed from several sources and thus easily verified. The selection of a CA used with SSL is a very important first deployment decision. The CA under-pins the security of your network and can also aid in certificate management.

The CA is used during the authentication phase to introduce unknown processes to each other by acting as a trusted third party. At runtime authenticated users are those users that have a valid X.509 certificate signed by the CA chosen by an organisation. In a CORBA environment for example, the public key of the trusted CA must be distributed to all secure CORBA application.

Some companies choose to use a **private CA**. This entails having a trusted node that they themselves set up to use and sign certificates. Securing the CA itself is also vital and in the case of a private CA, a bank has to ensure to take extra steps to ensure its security. Obvious measures are not putting it on the network, providing physical security to the machine etc.

Other issues with using a private CA include the large overhead that comes with acting as CA. The bank itself is now responsible for signing and deployment and this latter can be quite a task for large-scale deployments. There must also be a plan in case CA keys get compromised. The bank then needs to re-issue all certificates signed by the compromised CA

When using a **commercial CA** there is a company that signs certificates for many other companies e.g. Verisign, Entrust, RSA. The benefits of Commercial CAs' include a reduced responsibility with regard to certificate management. The CA can be globally recognised and therefore easier to extend the user base across the Internet or with other partnering companies. Commercial CAs' usually also provide tools to help with the PKI/Certificate Management. There is a downside to using a commercial CA and that is there is still a need to re-issue certificates if the CA is compromised and issuing is outside your control

7.7.4 Other PKI Issues

They are various other certificate management issues that arise when a bank chooses the use of Public Key Certificates as a security model.

When a certificate is no longer legitimate and needs to be revoked, a **certificate revocation** process can be a useful addition to the security model. The revocation process holds a list of revoked certificates and will check any certificate it is processing against this list to ensure they are still valid.

If a user loses their private keys they can be in a certain amount of difficulty as they will then be unable to operate within the Public Key Infrastructure. For example they might not be able to access documents they had encrypted with their key. A **key backup** mechanism is a solution to re-issue the private key to the user.

Of course there are further security issues arising from this. Where should this backup information be held securely while still protecting the user's private key as the private key usually remains in the sole possession of that user?

A **central repository** of certificates would be an advantage for holding public keys to be accessible to everyone. In a banking environment, the options available might include LDAP, DNS, or the corporate database.

Automatic Key Update is a way of automating the process of updating certificates that reach their expiry date. This can be done in a way transparent to the user.

Using **Key History Management**, we have a way of keeping a history of expired certificates plus the private keys as a user may have encrypted certain data with this older key and as a result this cannot be decrypted with a new key. Key History Management should be automatic so that a user does not have to intervene when trying to decrypt old documents.

Throughout this discussion on PKI we have only considered one PKI in one system. In reality there are many PKIs on different systems throughout the world and these need to interoperate. **Cross-Certification** is a way of ensuring this interoperability.

As in most IT systems, timestamps provide a very useful record of seeing what happened and when. In secure systems, **secure timestamps** are a key element in providing non-repudiation. They can show when something happened and thus can be used to decide whether an action should be allowed to occur.

Repudiation is the denial of having done something. **Non-Repudiation support** in security terms means we can ensure that if a user has completed an action, they cannot deny in the future that they were responsible. A PKI will have the evidence to prove the user completed the action.

7.7.5 PKI Locations

The use of PKI raises further issues as we have seen above. However some of these issues are more important in different systems and with different integration models in mind. We should have a closer look at these systems.

Internet PKI is essentially the security that is required when sending emails between friends or when browsing the web securely. This type of PKI essentially only requires the four basic components for SSL. These are Authentication, Integrity, Confidentiality and Certification Authority.

Extranet PKI is required when Extranet support is added to a system, as extra security needs to be provided also. A secure browser is still used but because users may come from outside the company more emphasis will be placed on validating certificates and ensuring that they have not been revoked.

Therefore the security components required for extranet security include Authentication, Integrity, Confidentiality, Certification Authority, Certificate Revocation and Key Backup

In the case of inter-enterprise signed transactions Cross-Certification will be used as two secure systems are interacting. Non-repudiation and Secure Time Stamping are required to ensure that all actions are accountable in between the two enterprises. Authentication and Authorisation as always real issues. This is known as **Inter-Enterprise Signed Transactions PKI**

Inter-Enterprise Signed Transactions require the most basic PKI components plus some additional features. The list includes Authentication, Integrity, Confidentiality, Certification Authority, Certificate Revocation, Key Backup, Cross-Certification, Non-Repudiation Support, Automatic Key Update and Secure Time-Stamping.

7.8 Integrating with other Security Models

We have taken a look at Firewall technology plus Public Key Infrastructure Technology and have seen how these can address the Internet side of banking systems but not how to provide internal security.

This internal security can be provided by RACF in the mainframe world but in general a comprehensive solution is required to provide internal access control to properly controlled resources and to provide internal auditing to aid intrusion detection and alarm and to provide management and administration facilities.

To recall, the RACF (Resource Access Control Facility) on OS/390 is a centralised and well-proven security model for OS/390. It controls access to mainframe resources by legitimate users. It also has defined mechanisms to allow it use SSL integration with its own access control facility.

Other security models that exist are often operating system based models. For example NT Security uses Domain, User Roles, File Permissions and Logging. The problem with these solutions is that the problem then becomes too platform specific.

There are various Distributed Systems Models available. These include COM+, which can inherit a security infrastructure from NT. Authentication, Authorisation (roles), privileges (provided by NT). The problem is that COM+ is really NT based.

Another Distributed model available is DCE (Distributed Computing Environment) which has a Kerberos based solution that uses Principals, ACLs (Access Control Lists), and tickets. This is a good security solution but in reality is too heavy weight for most users. There is a large footprint and considerable administration

A Banking System needs a solution that can cover a distributed heterogeneous system. In the Banking Systems we have been looking at there is typically an ORB used for the middleware, and these need to be secured. CORBA Security can do this for us.

7.8.1 CORBA Security Service

There are different levels of the CORBA Security Service (CORBASec) available from the specification. These include Level 0 (Internet Security), Level 1 (Security-unaware), applications, Level 2. (Security-aware applications)

As noted by **[Lang 1997]** CORBA Security builds on the underlying mechanisms and adds additional features which make it possible to use the mechanisms in a complex large distributed object systems environment.

The CORBA Security Specification provides security via Authentication, Access Control, Integrity, Delegation of Credentials, and Auditing. Some of the features it has available to provide these services include X.509 Certificates, User ID's with password login, Secure ID tokens and use of SSL over TCP/IP. Often there will be a Master Security Server (MSS) which is the central security server with runtimes for the clients and servers.

When Securing the Internal Network we need an MSS to be a central point of administration which can be accessed by the server runtime to verify policy information. The Security runtimes can be installed on both clients and servers and will facilitate security services in conjunction with the MSS

CORBA integrates all these Security Technologies using the CORBA Security Service Specification, the CORBA Firewall Specification and the CORBA/SSL Specification. It is possible to use all of these technologies and still work within a CORBA environment whilst still having a standards based solution and working within a single, but distributed security model.

[Alireza et al. 2000] outline some of the many problems that exist with the CORBA Security specification but also offer some guidance:

- Take into account the security of the entire system, not just the CORBASec components. It is always necessary to look at the system as a whole and at the interplay of its various components.
- Detect and solve weaknesses of CORBASec. (For example the management of users or domains).
- Develop creative solutions when needed, such as making the firewall ORB-friendly when it isn't
- Ignore absurd issues in the specification, such as the predefinition of the TCP ports for IIOP/SSLIIOP

The CORBA security model is security technology neutral. For example, interfaces specified for security of client-target object invocations hide the security mechanisms used from both the application objects and ORB (except for some security administrative functions). It is possible to implement CORBA security on a wide variety of existing systems, reusing the security mechanisms and protocols native to those systems.

The CORBA security service can control access to an application object without it being aware of security, so it can be ported to environments that enforce different security policies and use different security mechanisms. However, if an object requires application level security, the security attributes must be delegated and made available to the application for access control

In addition to the CORBA Security Specification from the OMG, there are now various other security related options.

The OMG Common Secure Interoperability Specification, version 2 (CSIv2) defines the Security Attribute Service that enables interoperable authentication, delegation, and privileges. The SAS

protocol is designed to exchange its protocol elements in the service context of GIOP request and reply messages that are communicated over a connection-based transport. The protocol is intended to be used in environments where transport layer security, such as that available via SSL/TLS or SECIOP, is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication.

The OMG Resource Access Decision Facility (RAD) provides a uniform way for application systems to enforce resource-oriented access control policies. By standardising this service, we enable the enterprise to define and administer an Enterprise Security Policy for use by all their software components - and allow these components "plug-in" to the enterprise security.

The OMG The Authorisation Token Layer Acquisition Service Specification (ATLAS) specification describes the service needed to acquire authorisation tokens to access a target system using the CSIv2 protocol. This design defines a single interface with which a client acquires an authorisation token. This token may be pushed, using the CSIv2 protocol in order to gain access to a CORBA invocation on the target.

7.9 Sample Architectures

In this section we will look at a few "real-world" examples and see how we can combine the technologies detailed above to provide the type of security that a bank would need.

[Lang 1997] points out that since all requests and responses in the CORBA model are inevitably sent through the ORB, and since objects cannot locate or call target implementations without ORB services, security enforcement is guaranteed. This removes the responsibility for security enforcement from potentially many application objects, which minimises the code responsible for security policy enforcement

The fact that a high level of security can be provided for applications completely unaware of security is one of CORBA's top security features. It is possible to put objects in domains where certain policies are automatically enforced during invocation and some security management is done, even if the object was not even designed to run on a secure system.

[Beznosov, Deng, Blakely 1999] present an approach in decoupling authorisation logic from application logic for those CORBA based application systems, which resort to application level access control in order to achieve fine granularity of protection or to use factors specific to the application domain in authorisation decisions or both.

They describe the design of an authorisation service that allows any level of access control granularity, applying authorisation policies of different types and from different authorities, as well as providing application domain-specific factors for evaluating such policies.

Finally, **[Bennett, Kannenberg 1996]** describe a project where by migrating its student administrative system from the mainframe to the Web, Stanford University provides functionality for students in an easy to learn and use format.

This Web-based system allows students to register, apply for housing, see grades, file study lists, update addresses and more. It is accessible day and night and provides a platform for increased functionality in the future.

A study in the real world example showed that security issues could not be solved by a single solution but rather by a combination of approaches. (1) establishing an authentication and authorisation approach (2) keeping data secure as it travels across the lines and (3) preventing the misuse of the Web access.

7.10 Security Patterns

[Yoder Baraclow 1997] outline some patterns for providing security

Single Access Point	Providing a security module and a way to log into the system
Check Point	Organising security checks and their repercussions
Roles	Organising users with similar security privileges
Session	Localising global information in a multi user environment
Full View with Errors	Provide a full view to users, showing exceptions when needed
Limited View	Allowing users to only see what they have access to
Secure Access Layer	Integrating application security with low level security

7.11 Conclusion

As we have seen there are many options available for a Security solution. The key is in finding optimal solution for particular needs. Therefore a bank needs to assess its risk, define a security policy and implemented the policy using proven technologies. It needs to provide a single, cohesive architecture than can be easily administered.

CORBA offers security solutions for distributed, heterogeneous systems and RACF provides the various access control and authentication required by mainframe applications. The key is to make us of each technology in its place.

[Koch, Murer 1999] confirm that many CORBA products are now mature enough to be used in an enterprise environment and that Necessary features like integration into a systems management framework or logging facilities for accounting and security can be integrated with reasonable effort

[Slama et al. 1999] also agree that for large-scale systems, while CORBA has comprehensively addressed many of the functional aspects for providing security in the system, manageability and scalability issues have not been fully addressed.

Various issues have yet been addressed by CORBA, and there are no plans for these issues to be supported. These include distributing and updating user authentication certificates or authorisation credentials, maintaining records of users that may not be permitted access to systems under any circumstances, and the implementation details of the storage of the authentication information (whether to use Smart Cards, secure tokens, or login and password). CORBA deals with functionality specification rather than implementation details, so these issues must be solved separately outside of the CORBA security area.

As per the previous section on Performance, this section continues the theme of this research of finding weakness areas in the enterprise integration systems that need to be reviewed. Again, we can apply some well-known patterns and industry solutions where these weaknesses arise.

At a later point in the research we shall apply these solutions to see how they can ensure the security of an enterprise integration project can be as reliable as that of its mainframe predecessor.

Given that security is of large concern to any financial institution, the original objectives of the thesis, which aim to find any weaknesses are still being reached. Only after we apply these new found solutions can we ascertain whether the advantages will outweigh any disadvantages.

8 Scalability

8.1 Introduction

Another important issue that needs to be considered when integrating legacy systems using CORBA is the area of scalability. It is relatively easy to wrap a legacy CICS or IMS transaction using CORBA and make this service available to a number of CORBA Java Clients within the system. It is an entirely different matter however to ensure that the CORBA service scales to supporting many thousands of client requests per day, per hour or even per minute.

As outlined in the **[ORBOS 1998]**, scalability can be thought of in terms of the number of users and/or objects that can be supported on either a single node or collectively on all nodes in a system. It mentions how the exact methods for measuring scalability have been widely debated but that measuring the throughput or capacity of the system is one good definition. With this proposal, there is a discussion about how scalability can be increased by adding additional memory, or processing power.

If we really expect to replace or reengineer legacy systems as CORBA Services we **must** ensure that these new Services scale as well as their predecessors and do so with no significant performance degradation.

[LaLiberte Braverman 1999] defines scalability in a distributed system like CORBA as meaning being able to meet the requirements of clients of the services being provided even when the number of different variables describing the size of the system vary, sometimes dramatically

There are a number of tools at our disposal that help us to ensure that these Service scale. These are the concepts of *Multithreading*, *Session Management* and *Connection Management* and we will look at each in turn.

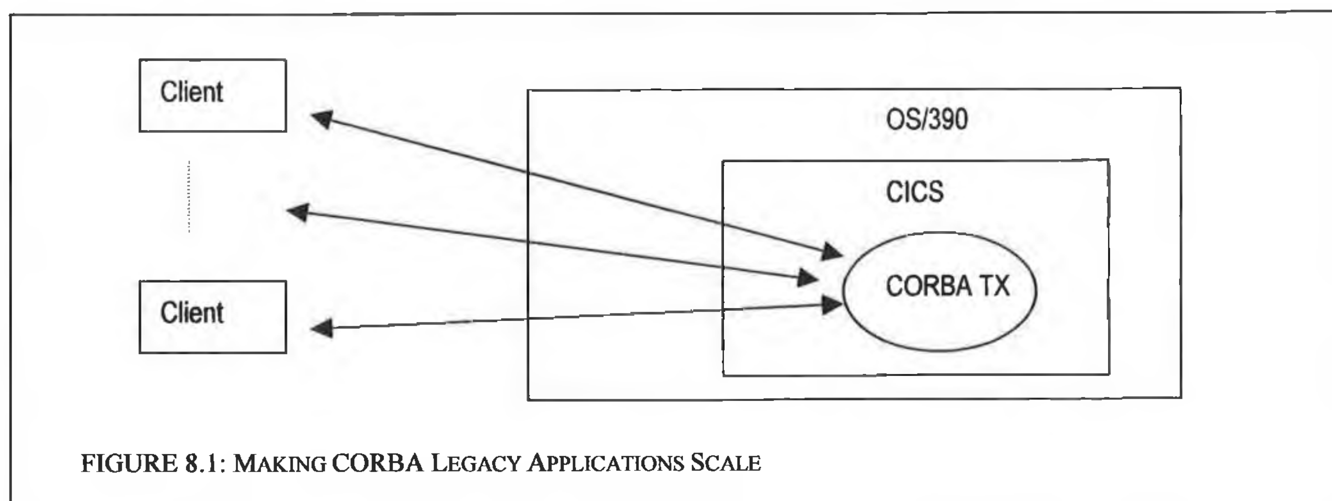


FIGURE 8.1: MAKING CORBA LEGACY APPLICATIONS SCALE

Multithreading is a technique used at the application level to ensure that the application can perform several different tasks at once. Connection management and Session management can be applied throughout the enterprise to ensure that connections and sessions are only "alive" for as

long as is necessary. All three combined can be a powerful tool that can be applied to ensure that we can support as many client requests simultaneously and at peak times as is mandated by the business requirements.

8.2 Multithreading

8.2.1 Concepts behind Multithreading

Threads are concurrent paths of execution within a process. Each thread has code, which it executes and is usually provided as a function. Each thread also has its own stack and registers and all the threads share the address space of the process they run in.

[Shultz 2001] outlines how the use of threads can significantly improve an applications structure and make its development more intuitive by delegating specific tasks to threads which otherwise would have required sophisticated mechanisms to integrate them into a single execution flow.

8.2.2 Multi-processing

In today's operating systems, it is common place to support multi-processing whereby the operating system automatically arranges for slices of CPU time to be allocated to multiple processes to create the illusion that they are all running concurrently. The concurrency may be real on a multiprocessor machine or time-sliced on a single processor.

8.2.3 Multithreaded languages

A lot of modern operating systems and languages are also multi-threaded. In this case within each process there may be multiple threads of execution which run concurrently, allowing an application to be doing several things "at the same time". The concurrency of threads, like that of processes, is achieved by time slicing the threads to be run on the available processor(s).

In multi-processor machines there may be genuine concurrency when two threads are running on separate processors – but since there are usually more threads than processors, time slicing still occurs.

8.2.4 Difference between threads and processes.

Both threads and processes are scheduled by the operating system on the available CPUs to execute some piece of code.

- Threads exist within a process whereas processes may contain threads.
- Threads share an address space whereas processes have their own address space

- Threads communicate easily via memory whereas processes only communicate via costly inter process calls.
- Threads are intimately interrelated where as processes have a limited ability to affect each other.
- Threads must use thread synchronisation to cooperate where as processes may use process synchronisation to cooperate
- Finally it is relatively cheap to start or switch a thread whereas this is relatively expensive with processes.

8.2.5 Choosing threads or processes

It can be more prudent to choose multiple threads for concurrent activities when there is a lot of sharing of information that can be kept in memory. Also when there is a need to communicate very efficiently between each other and there is tight coupling that depends heavily on each other's results, threads are the most efficient. Finally when the activities are developed together a thread can be better.

It is usually wise to use multiple processes for concurrent activities that do not share much information or can share via an external repository such as a database or CORBA server. Activities that need to be isolated from each other should be implemented as processes such as activities being developed by different teams or that use incompatible libraries. Finally activities that need to be run on separate hosts are other classic process material.

Popular example for thread usage are asynchronous blocking I/O, possibly long running or blocking code sequences like database queries, sorting of large amounts of data or number crunching. Multi-threading also allows to make full use of the CPU resources provided in multi-processor systems **[Schultz 2001]**

8.2.6 Choosing Multithreading or Single Threading

There are various costs and overheads that are associated with multithreading. These include the synchronisation costs that significantly complicate coding and debugging. The runtime overhead of thread creation and synchronisation can be another stumbling block and the overall performance can decrease the performance of compute-bound code on a single processor.

As **[O'Ryan et al. 1999]** outline, highly scalable systems may want to use a pool of threads to dispatch events, thereby taking advantage of advanced hardware overlapping and I/O computation

The benefits of multithreading include increased server or GUI responsiveness while long running computations are handled. There is additional deadlock avoidance in re-entrant servers for example and the throughput of IO bound applications such as CORBA calls can be increased. Finally, the harnessing of multi-processors gives an obvious advantage.

The performance of multithreading depends on several factors: **[Chan 1998]**

- Whether there is enough parallelism in the application such that a ready-to-run thread is always available upon remote operation
- Whether the overhead of context switching is high relative to the latency of a remote operation and
- Whether the locality effects of sharing the same local portion of the memory hierarchy are positive or negative

8.2.7 Using threads

When a process starts running, it contains a single thread of execution called the main thread. This thread may choose to start additional threads that will run concurrently with the main thread. The application terminates when all threads have completed.

Thread packages provide a call to start a new thread. This call typically takes a function pointer and some data or in object oriented APIs a thread object may provide a 'start' function to start a thread. Once a thread is started, it executes concurrently with the remaining code in the thread that started it.

A thread can be 'joined'. This is where one thread can wait for the completion of another thread. The join statement blocks the calling thread until the requested thread has finished executing, so statements following the join are guaranteed to occur only after the joined thread has completed.

8.2.8 Dangers of Multithreading

Threads share an address space, and can potentially read or write any memory in that address space. This allows very efficient communication between threads, since they can read and write to normal memory (without having to map it to the file system as processes must when they share memory)

It is possible as a result for threads to badly damage each other by attempting to use common memory at the same time. Access to shared data structures by concurrent threads must be carefully synchronised.

A **race condition** occurs when threads attempt to access the same data, and because they are both running concurrently they produce invalid results. This can happen if the operations involve several steps, and the steps being performed by one thread are interleaved with the steps being performed by another in such a way that the threads operate with inconsistent temporary results.

A **critical section** is a section of code which accesses shared data and could cause a race. To prevent races we need to ensure that the operations in two critical sections do not occur concurrently, i.e. in two threads. We need to ensure that critical sections are executed atomically.

Implementing an Object Adapter in the CORBA world that works correctly and efficiently in a multi-threaded environment is hard. **[Pyarali et al. 2000]** show how there are many opportunities for deadlock, unduly reduced concurrency, and priority inversion that may arise from recursive

calls to an Object Adapter while it is dispatching requests. Likewise, excessive synchronisation overhead may arise from locking performed on a dispatching table.

8.2.9 Managing Threads

A **mutex** is the simplest thread synchronisation primitive. It is a mechanism that allows one thread to get exclusive access to a resource and block other threads until it is finished. There are two operations in a mutex, lock and unlock. Only one thread may hold a mutex at a time, other threads which to try to lock the mutex are blocked until the holder releases it with unlock. We must be extremely careful of deadlock. This means holding two mutexes at once or holding a mutex while waiting for a signal.

The **Reader-Writer Lock** is a variation on the simple mutex that allows two different kinds of lock. Many threads may simultaneously hold a "reader" lock but on only one thread can hold a "writer" lock.

A Semaphore is a 'resource counter' A thread that needs a resource waits on this semaphore. This decreases the counter, unless the counter is 0, in which case the thread is blocked. A thread that makes a resource available can post to the semaphore. This increases the counter by one or allows one of the waiting threads to proceed continue.

Code that can be correctly executed by several threads concurrently is called **thread safe**. All functions that use only local variables are thread safe as each thread has its own stack. Functions that access shared data must be synchronised. They can use Mutexes to serialise access to the sensitive resources or use conditions/semaphores/Events to wait for other threads.

Overall however, using threading in a CORBA server improves overall throughput. **[McCauley 1999]** show how using threads in a CORBA environments can raise issues where slow IDL operations "hog" the system. It also shows how CORBA provides good support for assigning threads to incoming requests via a filter. In this example, they use a thread pool and reader/writer locks to allow multiple concurrent read access to servants.

8.2.10 Structured Locking Techniques

There are different approaches to locking. For example we could provide **code locking** where there is a mutex per group of related functions. We would put a mutex around critical sections that access shared data.

We could chose a **data locking** approach where there is a mutex per data item. This involves placing a lock mutex around all accesses of data.

Object locking is an object-oriented combination of data and code locking. In this case there is a mutex per object. There is no public data and a lock mutex is placed around critical sections in member functions that access private data.

There is a pattern called *The Strategised Locking Pattern* addresses some of the challenges associated with developing efficient, predictable, scalable, and flexible dispatching components **[Schmidt 1999]**

8.2.11 Threading Policies

Developers of enterprise level applications need a method to estimate the resources required to scale their applications to support thousands of users across hundreds or thousands of servers located on multiple platforms.

They need to either benchmark their own applications and/or use existing commercial benchmarks in this effort. For example, the threading policy available with an ORB product and the efficiency of its implementation would greatly influence different benchmarks and hence, determine which products were appropriate for a given application.

There are different policies that are commonly used when implementing a threading policy in an enterprise banking system.

Thread per operation involves creating a new thread for every operation. With this method there is maximum concurrency but it does not scale well as a solution.

A **thread pool** is similar to thread-per operation but the total number of threads is limited. A pool of threads is created at start-up and the first available thread in the pool handles each request.

A **thread per object** policy is less frequently used but can provide concurrency between objects. A final approach is thread per client, which again is not that common but can provide concurrency between clients.

There are variations that can be applied within each of these policies and they can be combined to introduce additional constraints. For example special operations can get preferential treatment by adding operation priorities to the policy. There is a technique known as **overload control**. A server imposes a limit on its maximum load and will start to reject requests when load becomes too high. This stops the server using all available memory and crashing.

8.2.12 Limits

Concurrency can improve throughput of an IO bound application but single processor concurrency cannot speed up computation and in fact context switching and synchronisation slow it down somewhat and all we can do is spread the waiting a little more evenly.

A sever must always be able to handle its average throughput. Queuing requests helps with occasional bursts of load but the server must be able to catch up at some point. Overload control just moves the problem to the client.

8.2.13 CORBA Alternatives to Multithreading

CORBA provides various alternatives to Multithreading. **IDL Oneways** indicate that the method implementation will not block the caller. Once the result is sent the caller will continue processing. This means however that no results can be returned from a oneway call. Asynchrony

can possibly be achieved using a pair of oneways where invocation results would be returned to the caller via a callback object. This can be successful as long as the work being performed is suited to partial processing, suspension and later resumption.

Deferred Synchronous and **DII** can also be used with CORBA. CORBA offers this mechanism for making non-blocking calls, even with non-oneway operations. If a client makes use of the Dynamic Invocation Interface, the ORB will allow it make an invocation and later retrieve the results from the ORB.

This DII approach has lots of difficulties in practice and a simpler alternative is the use of **Asynchronous Method Invocations (AIM)** which are part of the CORBA 3.0 Messaging Specification. With this, client can make asynchronous calls to normal two-way IDL operations. The use of asynchronous calls is transparent to the server and the IDL compiler generates the client-side support.

[Henning 1999] shows that the CORBA POA specification also permits a server application to arrange for a callback if the server-side run time cannot locate a servant in the active object map. This mechanism allows a server to bring objects into memory on demand instead of permanently having all servants in memory.

8.3 Connection Management

For a fixed system with a given throughput (e.g., a single node), there is an inverse relationship between the response time and the numbers of clients. The more clients submitting requests the longer the delays. **[ORBOS 1998]**

Extending this, because client connections use resources (i.e., memory and cycles), the more client connections (even if they are collectively not submitting requests more requests) the less throughput and hence the less scalable the system.

Similarly, while adding additional servers/objects to the system could initially improve throughput (e.g., if there were less servers than processors), at some point the throughput declines as the instantiation of the servers/objects increases.

Some other factors that cause scalability problems are outlined by **[Luomala 2000].** These include

- Growth of user base and hence service requests and hence network traffic
- Size of the data objects moving in the network
- Amount of accessible data in the system. (handling becomes more complex)
- Non-uniform distribution of user requests (time, geography)

8.3.1 Establishing Connections

With CORBA we need to decide on the number of connections that exist. Connections are established when a client calls a remote operation on a reference to a server (IOR) which it is not already connected to. This is basically the first time a client invokes on an object in a server. We

should note that it is only created when invoking on an object reference, not when the reference is created.

8.3.2 Reconnection

When a connection is closed it can be re-established by the ORB. This can happen when a server times out while a client is connected and re-establishing the connection will re-launch the server.

8.3.3 IIOP Connection Features

IIOP object references contains host and port for the server and allows TCP connection. There is an **object key** that identifies the object instance and this is passed to the server. The IIOP **location forward** reply can be returned in response to any remote invocation and this means that the requested object is not available on this connection but provides a new object reference for where it can be found.

The IIOP **locate request** probes for existence of an object on a connection. The IIOP **locate reply** may provide the actual location is its not there.

8.3.4 Callbacks

With Normal IIOP there will be one listening socket on each client. There will be additional connections per server that invokes any callbacks. Certain ORB implementations support bi-directional IIOP in which case there will be one listening socket on the client and that's all. The client-server connection will be reused for callbacks.

[O'Ryan et al. 1999] outline how, in many applications, only a small fraction of the consumers are interested in a particular event. To reduce the time required to dispatch an event by reducing the set of consumers tested would improve scalability.

8.3.5 Direct IIOP Connection

The server can embed its own port directly in a reference. The client using this reference can connect directly to the server. The server must run persistently on a well-known port.

Another major source of scaling problems comes are outlined in **[Ballintijn et al. 1999].** These come from the limitations of services that form part of middleware. The research shows how using a naming service allows different users to find, access, and share distributed resources. Consequently, if scaling the implementation of the naming service fails, it hardly makes sense to put any effort in attempting to scale other parts of the middleware system.

Another major problem is with object references with location information encoded within. Once the object moves to another location, the reference becomes invalid **[Ballintijn et al. 1999].**

8.3.6 CORBA Daemon

Some ORBS use the concept of a daemon to act as the middleman between clients and servers. These daemons are CORBA servers in their own right and can accept all incoming client requests and redirect them to the relevant server implementation.

By embedding the daemon port in object reference, the daemon is able to launch servers even for client of other CORBA compliant ORBs. References can be exported to foreign clients using the naming service or object_to_string()

If the transient port is embedded in a reference, then the server must be already running and listening on the correct port to it needs to be launched persistently

The decision to use a daemon or a fixed port is completely up to the individual project and simply boils down to whether or not the project wants to use a daemon to manage its connections. The dangers to this approach are that the daemon is a single point of failure. On the plus side with a daemon we do not have to worry about managing lists of ports for the servers.

8.3.7 Closing Connection

Connections are only closed when TCP closes them. This is automatic when a client or server exits or crashes. When a network fails or a host crashes it will only be detected if TCP keepalive is enabled. Applications can explicitly close the connection using various implementation specific approaches.

8.3.8 Connection Limits

There is a per process limit count on clients and servers that is mandated by the ORB limit of connections per process or the operating system limit on file descriptors. It should be noted that not all descriptors are available for connections. For example, sdtin/out/err, open files, other IPC libraries used by application etc also use FDs. All servers must live within these defined limits.

There are also per host limits to be considered. A daemon typically has the same connection limits as a normal server. The sum of clients connecting to a host plus the persistent numbers running on the host all count towards the limit. TCP may impose a per-host limitation.

8.3.9 Connection Patterns

Three useful patterns that will help in connection management are outlined in **[Slama et al. 1999]**

- Client Disconnects
- Concentrator
- Server Disconnects

8.3.9.1 Client Disconnects

The idea here is that clients would disconnect when they are finished using a connection. The TCP connection will close so the server file descriptor will be closed also. In principle the client will know when it is finished with a connection.

The issues that exist are that the server depends on polite clients and there is no defence against faulty clients. In addition, the client must know what collections of objects are sharing a connection. There is ORB functions that give the descriptor associated with a proxy. The connection is re-established if the client uses objects again.

Many clients fit this pattern. This is especially true of servers that are used by clients at application start-up such as the naming service or the daemon. Once a client has initialised, it no longer needs this server and can safely close the connection. This helps central servers such as the naming service scale up and be able to service large numbers of clients.

8.3.9.2 Concentrator

A concentrator server is a forwarding server that simply passes on operation calls to some application server. Concentrators can reduce the number of connections to any one process in the system or the overall number of connections in the system as a whole.

Concentrators can be on different hosts to avoid host connection limits. There are certain implementation issues that need to be looked into. There may be a need for multiple threads. If a static concentrator is used then it may be necessary to rebuild the funnel if new interfaces are involved. A dynamic concentrator can be difficult to implement.

There will be an obvious performance overhead on each call but it can make for more scalable systems. Security and Resource Management issues can also be overcome.

8.3.9.3 Server Disconnects

A server can close connections from client using ORB functions. Client connections can be detected using other ORB functions in a per-process filter inRequest filter point. An IO callback is called for every connection (listening socket, daemon, to other servers) not just client connections. Using an inRequest filter point will detect only client connections, and will also detect every use of a client connection that allows connections to be time stamped for idle time calculations.

To time-out idle connections, one could check the last used time in a filter point or periodically in a separate thread. Instead of timing out connections, a server could close connections when the total number of connections passes some threshold. An IO callback can count the total connections and initiate a cleanup of old or idle connections when the count gets too high.

Closing a connection while there are still requests being processed mean the reply cannot be sent to the client, and will be discarded by the server.

8.3.9.4 Other Idioms

Connection limits are not typically a large problem. In practice systems with large numbers of clients will already need to be distributed across multiple server processes and/or hosts for load balancing and fault tolerance purposes.

Centralised services such as the Naming Service or a daemon may need connection management.

8.4 Session Management

8.4.1 Sessions

The concept of a session needs to be defined, as there are many interpretations. A Session can be any of a login, an extended unit of work, an instance of a client process.

We must decide how long a session should last, and how can it be terminated. Finally we need to look if there is always a human user involved in the session. In reality the final definition of a session will depend on the enterprise banking application that is being architected.

Construction of scalable components in CORBA requires a solution of well-known trade-off. This is between simplicity of navigation in a large collection of objects on the one hand and a system time of reaction which is a major scalability factor on the other. **[Szymaszek, et al. 1998]**

8.4.2 Session Management Issues

A common requirement of a banking system is that it maintains session information on behalf of the user. In this case we can define a session as a login, usually of short duration and usually tied to an instance of a client application.

Long-lived application work in progress is not a session as we have defined it – it is workflow. Sessions are typically associated with server resource allocation and cleanup. Client may make invocations on multiple servers within a session.

8.4.3 Availability

We must look at where in the system the session is available. In a single available session the session is only available through the server process with which the client initiated the session. In this case other server instances will not recognise the session. In fact the session may be implicitly tied to a TCP connection to the client or to an object instance.

In the case of multiply available sessions, the session is available through any server process supporting a given set of interfaces. A session key may be replicated across servers or managed centrally as in the case of a Security Service.

8.4.4 Termination

Sessions must be terminated at some point for business as well as resource reasons. Explicit termination is a common approach where the client explicitly terminates the session via a logoff() invocation. The server will immediately free any resources allocated to the session.

In the case of implicit termination, which is also a common approach, the server will terminate the session and free any allocated resources. There is potential for losing information but server robustness will require this. This approach is used in backup mechanisms as well as session timeout or idle detection mechanisms

8.4.5 Service Architecture

In terms of a Service based Architecture, the scalability of the system can be broken into two types of scalability. The first is the number of concurrent clients required and the second is the number of process requests per second.

8.4.5.1 Concurrent Clients

In a legacy application, the system could be expected to deal with many thousands of concurrent clients. When we make these legacy applications CORBA compliant, they must still be available to many thousands of clients. The concentrator pattern seen with in the performance section can also be adopted to ensure that a maximum number of concurrent connections are never reached.

8.4.5.2 Number of Requests

Legacy applications, especially those implemented using CICS or IMS can rely on these highly scalable transaction-processing monitors. Any CORBA application using these technologies must scale to the limits of CICS or IMS as the overhead of CORBA is low compared with the application logic.

This means that in large banking enterprises we can expect to reach several hundred or even several thousand requests per second.

[Froidevaux et al. 1999] show how we can differentiate between scalability of the number of concurrent clients and the scalability of the number of processed requests per second.

[Ezhilchelvan et al. 2001] outline a real-world application where the applications with large and geographically dispersed client bases are currently supported in a centralised manner:

Client requests are sent (over the Internet) to systems located in a central place for processing. This centralised approach has serious scalability problems. A customer who is close to the central server can have faster server access than a remote client, and thus may have an unfair advantage over the latter. As the number of simultaneously arriving client-requests increases, the server load increases - resulting in performance degradation.

The solution involves describing a hierarchic architecture to satisfy the quality of Service and reliability requirements of a large number of geographically dispersed clients (of an auction system).

Individual projects must make decisions about the importance of scalability as detailed in **[Carzaniga et al. 1999]** . They envision a wide-area event service as an effective platform for the integration of distributed heterogeneous objects. The realisation of such an infrastructure sees two major conflicting challenges, namely scalability and expressiveness. Some systems offer rich selection mechanisms but with a centralised architecture, others adopt a more scalable distributed architecture, but they give scarce accuracy in filtering events.

8.5 Conclusion

We have seen a few approaches to providing the scalability required of CORBA Services. As mentioned this is especially true of Services that wrap or reengineer Legacy Applications because these applications will have provided the scalability required as per the mission critical business requirements.

The ability to run distributed applications over a set of diverse platforms is crucial for achieving scalability as well as gracefully handling the evolution in hardware and platform design. **[Shen et al. 2000]**

Providing Multithreading, Connection and Session Management within a large enterprise system will go a long way towards reach this goal.

[Vinoski 2000] sums up the state of scalability in CORBA systems. He states how many areas of the middleware and the application affect middleware scalability. These include server implementation, persistent storage, connection management, *object location techniques, binding techniques*, configurability, installation, versioning etc. Our job is to apply some of the tools and patterns available to us to ensure that we maintain a high level of system scalability.

Given that scalability is of high importance in any integration project, it automatically became one of the objectives of this research. In this section we have seen some modern techniques and patterns that can be used to overcome this possible area of weakness in Object-Oriented based systems.

At this point in the research we have seen three of the major areas of weakness for an enterprise integration project. The next area we must investigate is the availability of CORBA Services in a Service-Based Architecture.

9 Availability (Locating CORBA Services)

9.1 Introduction

In the context of enterprise CORBA systems, the Availability of these systems is of utmost importance. Essentially it can be broken down into two separate areas. "How does a client find a Service?" and "How does a client continue to find a Service?"

In the banking world, the integration of legacy systems with Java and OMG CORBA has led to a variety of Internet applications. These applications can experience communication and node failures on occasions, which affect both the performance and consistency of the service being provided. Such failures in commercial services can result in a loss of both revenue and credibility **[Little 1999]**

A second related issue for banking applications is that problem of initially obtaining the location of, or reference to, an object in a distributed system. This task, and that of maintaining the reference is often solved by naming an object and then making that name known to other potential clients via some repository. **[Falkner 2000]**

These two areas that when combined together make banking applications "available" are of utmost importance to the success the application. This chapter will investigate the first of these two topics, i.e. making a service available to clients wishing to use it, with the latter being investigated in the next chapter on Failover.

9.2 Locating a Service

In any client server communication, the client must first be able to find the server that contains the object it wishes to invoke upon and this is especially true in a CORBA environment. The client needs a reference to the object it wishes to invoke upon and this is known as an *object reference*.

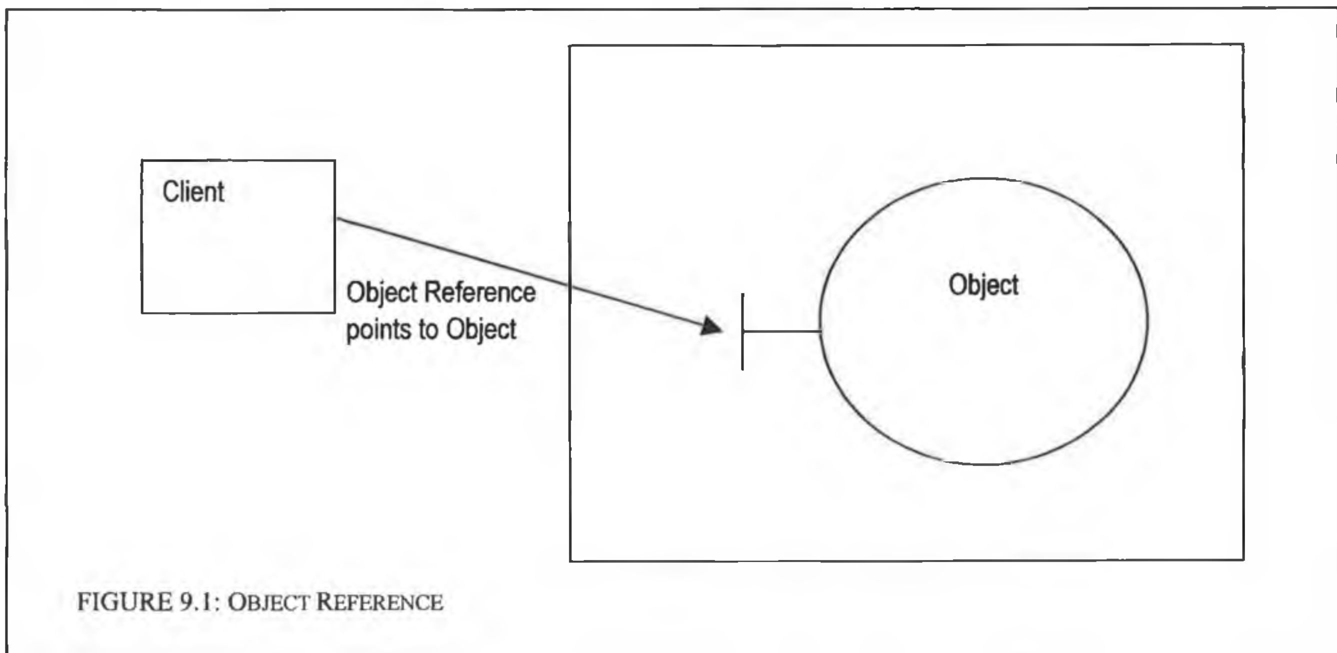


FIGURE 9.1: OBJECT REFERENCE

This Object Reference is defined in the CORBA Specification by the OMG as the information needed to specify an object within an ORB. The representation of an object reference handed to a client is only valid for the lifetime of that client. **[OMG CORBA 2001]**

This concept of the object reference maintains some of the principles of CORBA. If a service is moved to another machine, or implemented in a different language, or completely re-written in the same language, the client does not need to know. All that is required is that the client would be given the new object reference and it can invoke on the object it needs. In such a case when the server details have changed, so long as the IDL remains the same, there is no need for a client to be recompiled.

9.3 Providing an Object Reference

The usual way to provide an object reference to a remote object is when the server itself publishes its own object reference. However, a CORBA object is useless if no one knows where it is, and therefore cannot access it [**Claesson 2001**].

A solution to this is to store the Object Reference in some central repository of Object References so that when a client is looking for a particular object, it will do a lookup in this Repository and get the OR it requires. It can then use it to invoke on the remote object.

CORBA provides various solutions for each of the steps outlined in this object location model. These include the CORBA Naming Service and the CORBA Trader Service.

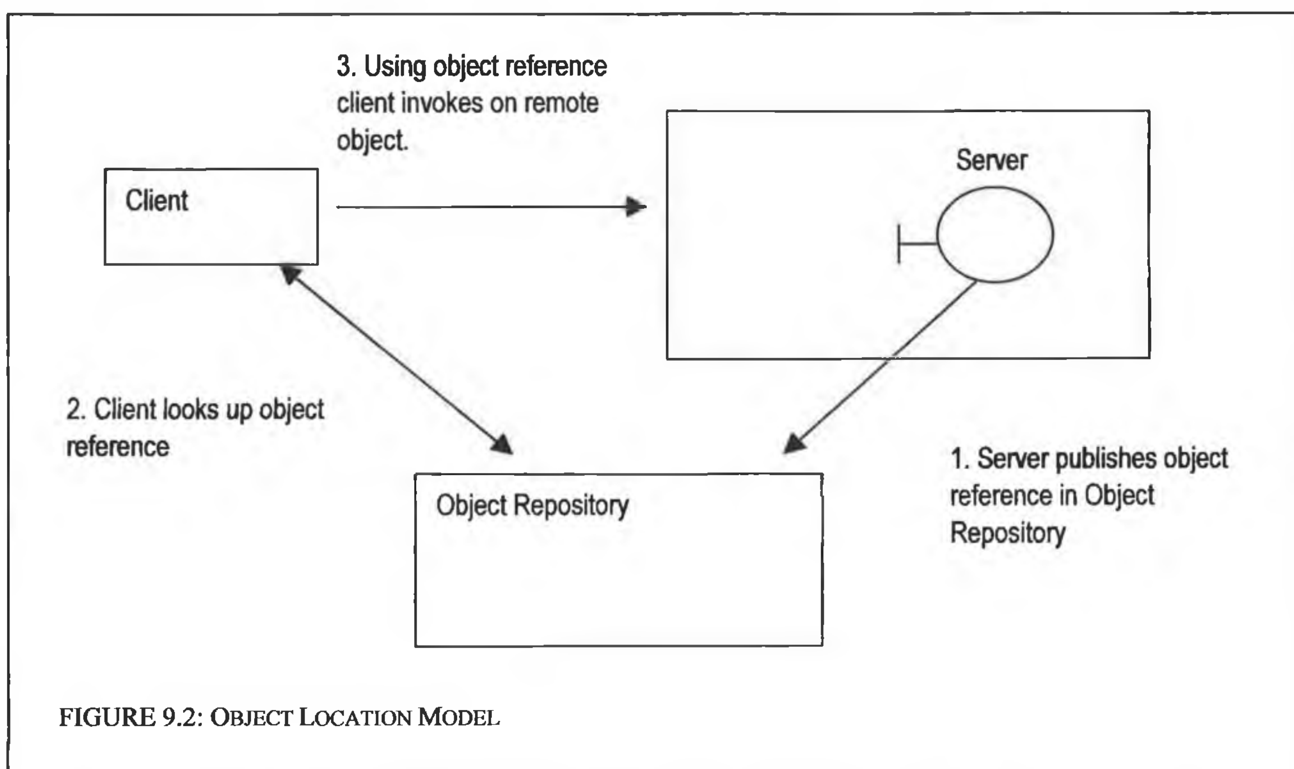


FIGURE 9.2: OBJECT LOCATION MODEL

9.4 Interoperable Object References

There was however, a fundamental problem with Object References that needed to be addressed. Prior to the CORBA 2.0 specification, any ORB vendor was allowed to use proprietary methods for distributed communication. This met the criteria of the CORBA specification but it meant that ORBs from different vendors were unable to communicate with each other.

To put this right, the CORBA 2.0 specification introduced the IIOP (Internet interoperability) Protocol for distributed CORBA communication and also a standard for Object References called Interoperable Object References (IORs)

An Internet Inter-ORB Protocol (IIOP) defines a transfer syntax and message formats (described independently as the General Inter-ORB Protocol), and defines how to transfer messages via TCP/IP connections. The IIOP can be supported natively or via a half-bridge. **[OMG CORBA 2001]**

This IOR is a data structure in a standard format containing information clients use to establish connections to servers and information servers use to identify target objects. The main elements of an IOR include.

The Port Number which is the TCP/IP port number that a connection will be opened on, the hostname or IP address of the host on which the object is implemented and the Object Key which is opaque to clients which servers use to uniquely identify objects within their domain.

9.5 Proprietary Solutions

Prior to the CORBA 2.0 specification, an ORB vendor could implement their own transport protocol and object reference as outlined above. In addition, many vendors added functionality within the ORB for object location outside of the CORBA services.

One example of this is the POOP (Plain Old Orbix Protocol) used by Orbix from IONA Technologies prior to the Orbix 2.3-product version. This proprietary solution had to be re-written when a CORBA compliant protocol became available with the CORBA 2.0 specification.

Typically these solutions were easy to implement but were flawed in that they required clients to have some knowledge of the server location, registration information or internal object keys.

9.6 Getting the Object Reference

Once an object comes into existence (is instantiated) it is then possible for a client who has its object reference to invoke upon this object. As outlined in the object location model above, the trick was how to get this object reference to the client.

The most trivial solution in this case would be to take the IOR and copy this by hand into the client code **[Claesson 2001]** but this of course requires human intervention and is not particularly scalable.

For example, looking at the information system of the UNICIBLE Data Centre in Switzerland, where the heart of their OSIRIS system resides on OS/390 and is made up of some 37,000 programs that could become CORBA objects **[Clerc 1999]**. It is easy to see that any time these object references changed it would be impossible to update all the clients of theses systems by hand.

9.7 The CORBA Naming Service

There is an OMG provided CORBA solution that is also a well-known Architectural Pattern called The Naming Service. [Mowbray, Malveau 1997]

The CORBA Naming Service is one of the CORBAServices defined by the Object Management Group in the CORBA Specification and that implements the Naming Service IDL (CosNaming). (This CORBA Service has a well-defined interface, as would any CORBA Server. This interface is used by clients and servers to store and retrieve object references from the naming service database.)

```
module CosNaming {

struct NameComponent {
        Istring id;
        Istring kind;
};

typedef sequence<NameComponent> Name;

interface NamingContext {
        void bind (in Name n, in Object o);
        Object resolve(in Name n);
};
};
```

FIGURE 9.3: COSNAMING IDL

All the IDL components are defined with the module CosNaming that defines a structure NameComponent to hold name components. This NameComponent structure is made up of a string defining the id and a string defining the kind.

Essentially its function is to provide a repository for object references whereby an object reference can be mapped to a readable name so that clients wishing to invoke using a particular object reference can do a lookup of this name and hence get the object reference they require

The names are organised hierarchically like a file system and the names are human-readable. The Naming Service concept is based on a telephone directory where each telephone number is

mapped to a name and address, well with the Naming Service each IOR is mapped to a readable name.

The Names are stored in a hierarchical format and both object references and naming contexts are stored. A Naming Context can contain object references and is equivalent to a high level qualifier for names. An example Naming Hierarchy might look like

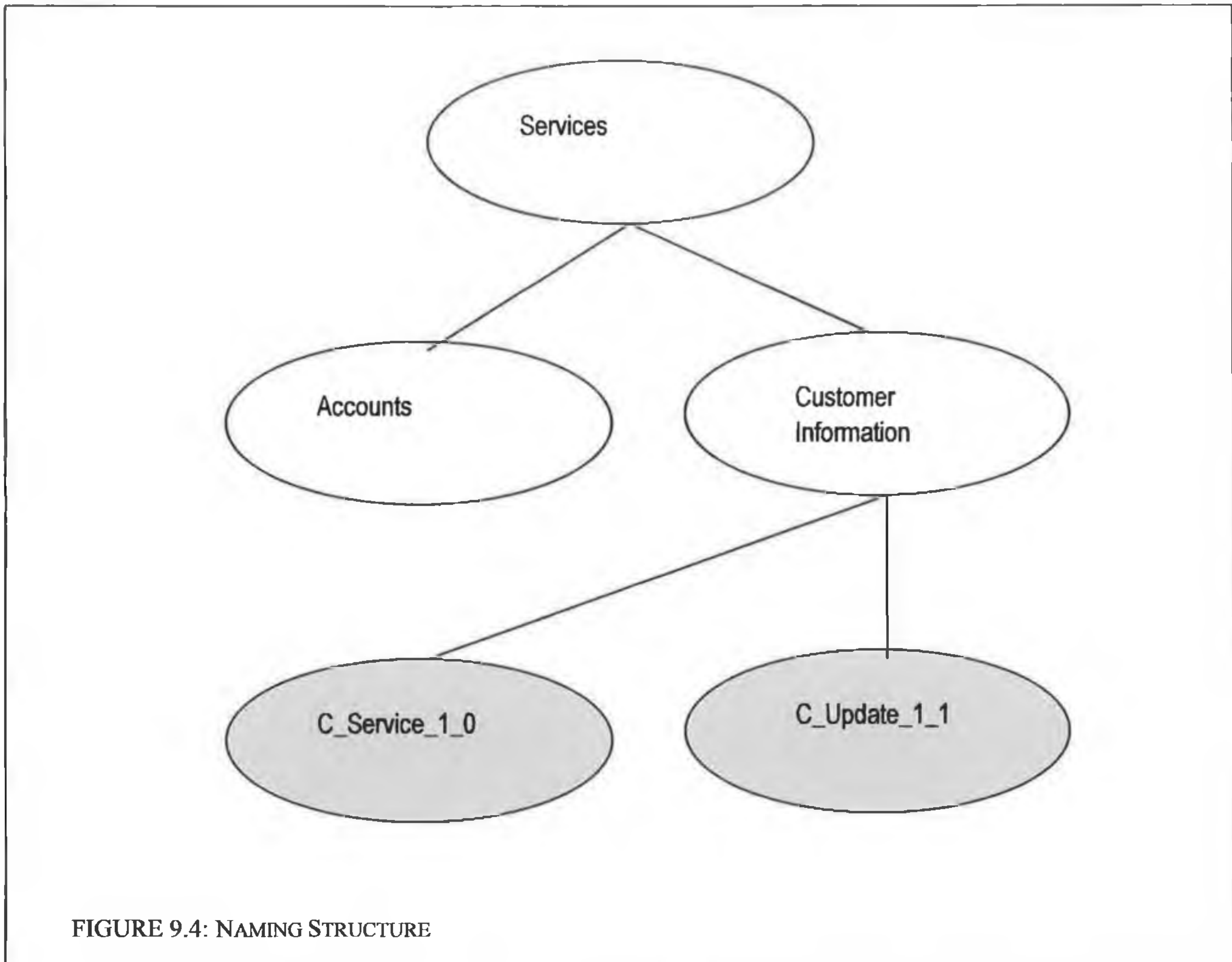


FIGURE 9.4: NAMING STRUCTURE

In the above diagram Services, Accounts, and Customer Information are all naming contexts while C_Service_1_0 and C_Update_1_1 are object references.

9.7.1 Choosing a Naming Service Hierarchy

The Naming Service is a particularly useful solution when clients look up objects based on a fixed and consistent set of criteria or when clients only want a single object reference returned as the lookup properties have static values.

There are different ways to structure a naming service and these very much depend on how the naming service is to be used and the structure of applications within the enterprise. The conventions adopted in a large-scale project will have significant impact as to how the naming

structure is defined - i.e. whether a descriptive or compact hierarchy is preferred or how deep or flat the Naming hierarchy need be.

Using a descriptive Naming-Service Hierarchy results in a user-friendly method of finding object references. The names are well defined and easy for users to access via a GUI for example. With a Compact Naming Service Hierarchy however, the names have very little meaning to a human user but rather suit applications that have proxies, helper classes or concentrators doing the naming lookup.
We may also choose between a flat Naming Service Hierarchy that only allows for the flat level of Naming Context and where objects are identified by only their name and their id and kind fields are unique. Alternatively there is a Deep Naming Service Hierarchy where objects are identified by their positions as well as their name. Id and kind fields can have the same names but within different contexts. This is the more common form of Naming hierarchy.

In a Service-Based architecture however, the naming service context structure can reflect the organisational structure of the bank. In fact, preferably the organisational structure should be implemented as an explicit service, but then we face the problem, that resources and processes are parts of the organisation and not beside it. **[Koch, Murer 1999]**

In with such a naming context structure, it is up to the bank to decide on a descriptive or compact naming hierarchy but a deep naming hierarchy would be more common than a flat naming hierarchy.

9.7.2 Extensions to the Naming Service

There are various extensions to the CORBA Naming Service that further enhance its usability in enterprise solutions. The first of these is an extension that standardises a number of elements in the specification. This is known as the Interoperable Naming Service.

The definition of this Interoperable Naming Service as defined in the CORBA specification outlines a Service that defines one URL-format object reference, corbaloc, that can be typed into a program to reach defined services at a remote location, including the Naming Service. A second URL format, corbaname, actually invokes the remote Naming Service using the name that the user appends to the URL, and retrieves the IOR of the named object.

This first feature includes a standard string representation of names. This feature uses a "/" as context separator and a "." as id-kind separator so that a name might look like Bank/Accounting.Customer/C_Update_1_0.

There is also a URL format for these names (both with IOR and stringified names), a standard configuration of returning a single initial naming context to all clients via a resolve_intial_references() and various other clarifications and enhancements to the specification.

Another extension to the CORBA Naming Service outlines the concept of an object group. A typical CORBA Naming Service entry is strictly one to one where a name can only be bound to a single object reference and names within a naming context must be unique. The notion of an object group extends this by allowing a one-to-many relationship between names and object references: in essence, it allows many servant objects to be registered with the same name in the

naming service. Resolution of the name to an object reference is mediated with the aid of location policies such as round robin and random. **[Slama et al. 1999]**

There is another concept known as a federated naming hierarchy. Within the CORBA Naming Service, a naming context can contain object references as well as other naming contexts. These naming contexts can be remote as well as local allowing the federation of naming hierarchies together.

9.7.3 Naming Service Difficulties

The Naming Service can be a single point of failure for an enterprise banking system so we need to allocate resources to ensure that it is always available. The next chapter will outlines various methods of ensuring this and thus enabling continuous access to up-to-date object references for interested clients.

Another issue that must be considered when using the CORBA Naming Service is ensuring that it is kept up to date. **[Felber 1998]** outlines one solution to keeping the information up to date by doing an update each time a view change occurs. However, even this solution does not guarantee up-to-date information.

Each enterprise solution will have to consider a framework for keeping the Naming Service up to date but typically some form of automated tool or script that updates the information periodically as well as removing out of date references is ideal.

9.8 The CORBA Trader Service

There is another CORBAService that works in a similar way to the CORBA Naming Service and that is the CORBA Trading Service. Again, there is a Trading Service IDL definition specified in the CORBA specification. **[Mowbray, Malveau 1997]** outline the Trader Service pattern.

The concept behind in the Trading Service is marginally different from the Naming Service. The Trading Service stores Object References as before, but the object reference is referred to as an offer and must follow an offer type, where an offer type is a definition of the number and type of properties of the object.

The idea is that instead of returning one object reference as per the Naming Service, a client lookup would return a group of object references categorised into groups of logically related object references in the manner that a Telephone Yellow Pages returns groups of numbers based on the Service required.

Values are assigned to the properties when the object instances are published and the objects are retrieved based on queries that are evaluated against published object properties of the specified type

The Trading Service is better suited than the Naming Service when clients look up objects based on a varying set of criteria or clients want multiple object references returned for further examination. In this case the lookup properties can have static or dynamic values.

As noted in **[Slama et al. 1999]** the trader's object directory is not structured in any formal way (as the naming hierarchy is). Rather, the trader service is based on the concept of a service type that contains an IDL interface identifier plus some data defining attributes associated with this type.

In **[Modi 2000]** we see a real-world example of the CORBA Trader services where an object installs itself with its name, persistence and filter properties. Interested applications and processes can find the object using these properties. An application that requires filtering but not persistence would indicate these requirements to the Trader Service, which will provide the application a list of references matching these requirements. The application may then choose one from this list depending on other properties it requires.

9.9 Bootstrapping

One of the problems with using the Naming or Trading Services is that they are useful when a client performs a lookup on a name and retrieves and Object Reference or set of Object References. However, there is still the problem of how a client locates the Locating Service (i.e. the Naming or Trading Service). This problem is known as bootstrapping.

CORBA provides a well-known solution to this in the specification by ensuring that every ORB implementation supports the function:

```
Object resolve_initial_references(in string serviceName)
```

An initial reference to an instance of the CORBA Naming or Trader Service can be obtained by calling this resolve_initial_references with the ObjectID of the required service

This function works when a client supplies a service name (i.e. *NameService* or *TradingService*) and they receive an object reference for that service.

Another approach to bootstrapping would be to obtain an object reference for the desired service, in string form. This stringified object reference could either be output by the service itself or generated by an IOR creation tool. CORBA has standard APIs such as object_to_string() and string_to_object() to make IORs easy to pass around and convert to a stringified form. Once a client has obtained this string they could call CORBA::ORB::string_to_object() to create a reference for the object. **[Slama et al. 1999]**

9.10 Custom Object Location

As we have seen, the use of a CORBA specified object location Service can very much help making objects and Services available across the enterprise. However, sometimes the architects of such a system will prefer to use customised location servers to meet their requirements. These can include helper servers and/or concentrators

Implementing or extending the standard CORBAIDL in a customised manner is an appealing combination as an enterprise can combine its own object location model that might provide performance or availability enhancements. Helper server or helper classes can be written to remove any of the complexity of using the Naming or Trader Service away from the developers.

There is a concentrator pattern that outlines a concentrator or funnel server, which is a process that sits between the clients and servers. The clients connect to the concentrator process. For every service a client uses, the concentrator must implement that object (from the client's perspective). In reality it delegates the call to the real implementation of the service. This can be also used to avoid connection limits and in load balancing. **[Slama et al. 1999]**

The customised solutions can then be integrated with the standard solution if required, for example a client might still use the Naming Service for bootstrapping but then use the customised solutions.

As we will see later on, the standard Object location models can be single points of failure, and so certain customised solutions can be implemented to provide a level of fault tolerance and Failover in critical cases.

9.11 Publishing Certain Objects

Once we have an object location model in place we then have to decide which objects we want to publish and make available using this model. Typically not all CORBA objects will need to be published and some enterprises will want to publish a smaller number of entry points instead.

As we have seen in the section on Performance, the world of mainframe integration often opts for a Service-based architecture where the Services are large grained objects rather than many fine-grained objects. In such as scenario there will be fewer Services in the system but performing more work and by their very essence each of these Services can be published in an object location Service.

Where finer grained objects are used in large-scale systems, factories or entry points can be considered as good candidates for publishing. However, when we only publish a subset of the objects available in the system, we then must provide "find" and "lookup" methods for the other objects, which can lead to complicated systems.

There is a factory pattern that can be useful for dynamically created or dynamically activated objects. Using a factory is like using a distributed constructor for object creation.

[Slama et al. 1999] outline a factory object pattern is any object that returns a reference to another object as a result of a method invocation. In an enterprise system, rather than publishing references to all the servant objects, the server can publish just a few factory objects, which the clients use to obtain references to the remaining objects in needs.

9.12 Lifetimes of Objects

There are various states in an object's lifetime including creation, activation, deactivation, deletion and there is even a CORBA Service called the CORBA LifeCycle Service specified to help manage CORBA objects throughout their lifecycles.

There is an Object Lifecycle Manager Pattern **[Levine, Gill, Schmidt 2000]** that can be used to govern the entire lifecycle of objects, from creating them prior to their first use to ensuring they are destroyed properly at program termination. In addition, this pattern can be used to replace static object creation (and destruction) with dynamic object pre-allocation (and de-allocation) that occurs automatically during application initialisation (and termination)

However, in the CORBA-mainframe integration projects that are based on a Service-Architecture we are typically talking about stateless objects (an object/service with no state) that often resides inside CICS or IMS and thus has a fixed lifetime. **[Koch, Murer 1999]** These transactions can be offered as services on an interface.

There are exceptions such as long running CICS transactions or IMS Wait-for-input transactions. However, in the majority of cases the lifetime of the Service is short (i.e. by its very nature its role to is provide a Service and then end). As a result we do not need to be overly concerned with the creation, activation, deactivation and deletion This is especially the case when dealing with TP Monitors such as CICS or IMS as these are taken care of for us by the TP monitor

In non-mainframe based CORBA applications there sometimes needs to be decisions made as to how long the object will stay alive for - i.e. until no further clients are connected, or indefinitely or only while the object. The Evictor Pattern **[Henning, Vinoski 1999]** can be used in this case to enable a server to deactivate the objects based on a policy for choosing when to deactivate an object. The policy can be based on Least-Recently-Used, Least-Frequently-Used, oldest-object etc. However, this is a very comply area of CORBA and distributed programming in general and care should be taken.

9.13 Conclusion

In this section we have seen some issues that can arise in an organisations enterprise solution. In other sections we can see how to provide CORBA Services to the rest of the enterprise but in this section the important question is "How do clients get access to these Services ?".

As we have seen there are various OMG Defined CORBAServices that can assist us in this goal and there are also provisions made for those who require additional features in their system.

An investigation into the concept of making a Service available meets yet another of the requirements of this research. In this section we continued the investigation into areas of critical importance and possible danger for enterprise integration projects. This topic certainly fits into that criteria.

In the next section we will investigate another area related to the Availability of a system and that is Failover. As noted in the introduction, it is one thing to make Service available but it is an entirely bigger task to keep those Services available.

10 Availability (Failover)

10.1 Introduction

In the previous chapter on Availability we investigated one of the aspects of a CORBA Service's availability, i.e. how to locate the service. In this section we will investigate the second concept of availability and that is keeping the Service available. This area includes an investigation as to why we need to keep the systems up, how we handle the situation when a crash occurs, CORBA Exception Handling at the application level and Load Balancing as a technique for Availability.

10.2 Mainframe Availability

Mainframes can achieve 99.99% or 99.999% uptime in enterprise solutions, resulting in anywhere between 5 and 53 minutes downtime per year **[IBM]**. These figures are very impressive when compared with some of the more modern Operating Systems available today.

A study by **[Turner Brill 2001]** finds that the best possible reliability for mainframe data centres is 99.995%, or about 26 minutes of downtime per year. IBM's new z/OS mainframe systems stand for zero unplanned downtime which is quite a claim.

IBM's Server Group also claims that the mean time between critical failures (MTBCF) or the average time between failures that require a reboot (Initial Program Load) for its S/390 mainframe is 20 to 30 years. If we compare these figures with other decentral platforms such as the PC it is easy to see what we are dealing with in terms of reliable systems.

Mainframes are big, complicated systems that often have clusters of CPUs, terabytes of main memory and many thousands of users, yet they are still very reliable. It is the case that where frequent crashes on a PC or other decentral platforms are accepted as part of the routine hazards of computing, a crash of the mainframe is a massive problem and simply would not be accepted.

In reality, any large banking organisation will have dedicated teams ensuring the smooth running of the mainframe. In addition, there will be redundant hardware, extremely protected operating systems and stable applications, all of which provide for highly available systems.

Our problems really begin when we attempt to introduce the mainframe as a peer in a distributed CORBA computing environment. Typically distributed systems do not have the same guaranteed availability and we are now making host applications available throughout the private company network and beyond. We must look at various solutions and system services that can be applied to try to bring the availability of the enterprise network as a whole to where the mainframe already is.

[Maffeis, Schmidt 1997] define a distributed system as reliable if its behaviour is predictable despite partial failures, asynchrony, and run-time reconfiguration of the system.

In addition, we require reliable applications to be highly available. This means that the application must provide its essential services despite the failure of computing nodes, software objects and communication links.

10.3 Failures

There are various types of failures that can happen in an enterprise system, as by its very nature there are various types of technology used. The most common types of failure include the process crashing, the processing hanging or being in a deadlocked state, the host system crashing, and a network partition due to link or route failure.

As outlined in **[Scallan 2000]** some of the more common problems that occur in distributed systems include:

- Performance bottlenecks
- Network resource limitations
- Network failures
- Race Conditions (not properly synchronised modules)
- Deadlocks (the synchronisation protocol between modules prevents each from completing its task
- Design errors in control flow
- Timeout Failures

Other types of failures that are more difficult to deal with are failures when a process performs incorrect actions or persistent storage faults.

We will see that with all these failures, some are easier to handle than others are. However, with a combination of fault tolerance, load balancing and exception handling we expect to have the enterprise architecture in such a stable fashion that any or all of these failures would not affect the continued availability of the system as a whole.

10.4 Exception Handling

10.4.1 Introduction to Error Handling.

Put simply, Error Handling is the technique applied by programmers to ensure that their applications cover all eventualities. That is they should perform the tasks they have been implemented to perform but they should also be ready for the unexpected.

In fact, not only should they be ready for the unexpected, but also they should have built in frameworks so that they can act decisively to inform all parties involved of the problem and maybe do something to fix it depending on the circumstances.

10.4.2 Early Error Handling

The concept of error handling has been around since the earliest days of computing. With the early mainframe applications it became clear to the developers that they needed to account for the cases when everything did not go according to plan.

The idea was that any errors detected at run time would be relayed by the originator of the message to a recipient. This recipient would also know how to handle the error.

Of course, the ideal scenario is that all errors should be caught at compile time before the application is even run but reality dictates otherwise.

In these earlier languages an error was dealt with by returning a special value or a flag and the recipient could look at this value or flag and work out that something was wrong.

One of the problems with this approach was that often developers would not check **every** method they called to see if something was not amiss and those that did ended up with massive, unreadable code. The discrepancy between applications in terms of error handling techniques was something that needed to be dealt with.

10.4.3 Dealing with Exceptions

The concept of Exceptions was the solution to the problem in that they helped clean up error handling code. The idea was that at the point where the error occurred, the problem would send to a higher authority where someone qualified to make a decision on what to do would take over.

This approach removed the scenarios where at the point of failure the application was not in a position to make a decision on what to do, so it would simply pass the problem on to someone who would know.

So now, instead of checking for a particular error and dealing with it at multiple places in an application, the exception handler will always deal with every exception occurring in the code. There was no longer a requirement to do a check after every method call since the exception will guarantee that someone catches it.

The code now becomes more maintainable as you can now separate it so that it describes what you want to do distinctly from the code that is executed when things to wrong. Reading, writing, and debugging code becomes much clearer with exceptions as opposed to the traditional error handling techniques.

10.4.4 Distributed Exception Handling

Catching Errors is important in any application but it becomes even more so in distributed systems where there may be several different tiers. If something goes wrong in the back end of a system, it is important that the user or the client receives a suitable message letting them know that things did not go exactly as they planned it.

Take any banking transaction for example. If a user is paying a bill online and the system crashes before the user's details can be written to the database, a message must be sent to the user letting them know that their bill payment was not successful. It should also inform this user that if they want to pay it they must go through the procedure again.

In general, remote method calls are much more complex to transmit than local method calls so there are more possibilities for error. This makes it even more important to track these errors, inform the relevant parties, and then decide what should be done in the case of such a failure.

10.4.5 CORBA Exception Handling

The CORBA Fault Tolerance specification defines a standard set of system exceptions. The exceptions are typically raised by the ORB when something major goes wrong with the basic client-server communication. Typically the error occurs during the transmission of remote operation calls. As outlined above, the error must be immediately reported to a client or a server.

CORBA applications used in critical scenarios must be robust. But, with a heterogeneous environment, the use and reuse of commercial off-the-shelf, third party and legacy software modules, and their complex interactions will all be likely to trigger exceptions. **[Pan 2000]**

Thus, the graceful handling of expected and unexpected exceptions is critical for the robustness of CORBA-based systems.

The most common types of system exception range from communication failures due to network problems, to looking for an object that doesn't exist, or looking in the wrong place for an object, or failures due to problems marshalling operation parameters

There is another type of exception known as a user exception. These exceptions are specific to applications and allow an application to define a set of exceptions or errors that may occur in that application and that need to be caught and dealt with accordingly.

10.4.6 CORBA User Exceptions

The OMG defined CORBA specification allows for a raise clause to be defined as part of an IDL operation and this can then return a more detailed error message to the caller depending on what exactly happened. Typically the application will not raise a system exception itself but should be designed so that it can handle both types of exception.

A CORBA client must handle system exceptions raised anywhere on the server side and returned via the ORB, either during a remote invocation or through calls to the ORB. Such a system exception might be raised if the ORB encounters problems with the network

Example of Exception Handling

```
//OMG CORBA IDL

interface Employee {

        exception Reject {
                string Reason;
        };

        boolean IncreaseSalary(in string EmployeeId, in float newSalary) raises (Reject);
        .....

};
```

FIGURE 10.1 : ADDING EXCEPTIONS TO IDL

In the IDL above we have an interface defining an Employee. There is one operation called IncreaseSalary and it takes in the Employees ID and the amount of their newSalary. If for some reason the Employee is not valid or is not due a raise we
can return this information with the User Exception "Reject" that can be thrown is something is not correct.

10.5 Fault Tolerance

10.5.1 Introduction

To make sure that a system is "fault tolerant" the design of that system should integrate several levels including the hardware, the software and the administration.

There are also different levels of fault-tolerant service. The first of these is Recoverability and ensures that components in the system can be re-started after failure and returned to their previous state. Some clients may need to be delayed during this restart.

A second level of fault-tolerance is providing continuous availability where the system remains fully available during component failure. This means the system must have redundant components. Any clients will be moved from faulty components to the live ones or use **multicast** requests to multiple servers.

Performance considerations also need to be looked at. Throughput and response times can be affected by a fault and we need to decide if the system can degrade during a fault or do our performance goals always need to be met.

10.5.2 Realising the failure

In a CORBA system the first sign that something has gone wrong will be when an exception is raised. When a system exception is returned we realise that something has gone wrong with the system. If a user exception is returned we can be sure that a user is at fault.

In the case of a system exception, when the remote call fails this means that a process, a host or a network failure has occurred resulting in the TCP read or write failing. We can easily determine which of these has occurred by testing if it is possible to contact the remote host or the remote server and if the host responds to IP ping calls.

When the failure is completely hardware related however, it can be more difficult for the system exception to return useful information. IIOP failure detection that raises such a system exception is based on TCP connections but when a host crashes or a network partition occurs, TCP cannot distinguish between them and thus it can be difficult to ascertain what went wrong.

In the case of a hanging server we can use timeout mechanisms supplied by various ORB implementations that will result in a client only trying a given number of times before giving up. However, these timeouts have to be carefully used so that a slow system is not confused with a "hung" system.

When a process crashes, the TCP connection will be closed. However in the case of a host crash or network failure these connections are not closed. Remaining processes will realise that the connection is closed, either when they next attempt to make use of this connection, or if they have enabled TCP keepalive.

In reality, detecting closure may not be useful if the clients do not have direct connections to the servers in the case of firewalls or concentrators. Also, clients or servers may have a connection management policy where idle connections are closed to conserve resources and reopened when they are needed. Finally, we must also realise that this method will not easily distinguish between host crashes and network failures.

The OMG issued a RFP in 1998 and produced in 2000 the FT CORBA specification. FT CORBA provides a set of IDL interfaces to an infrastructure that allows the management of replicated, fault-tolerant CORBA objects. **[Marchetti, Virgillito, Mecella, Baldoni 2001]**

The Fault Tolerant CORBA specification adds fault tolerance features to standard CORBA with minimal modification to existing ORBs.

The specification does have limitations and leaves several issues for the vendors to solve. One of the major limitations lies on the server side interoperability. All ORBs within a Fault Tolerance Domain must be from the same vendor. **[Korhonen 2001]**

10.5.3 Recoverable Servers

This approach to fault tolerance involves the server recovering immediately after a crash or reboot and returning to a consistent state without losing the results of requests that were processed before the crash. To do this the server needs to maintain a persistent log of the state of all objects.

This persistent state must change as quickly as possible after an update occurs so that there is little chance of a lost update due to a crash. Typically such approaches will involve the use of a marker to indicate a complete update has been stored, so that if the server crashes while updating the logs, any incomplete updates can be ignored on restart.

The use of a recoverable server means that the system can handle a server process crash without much impact to the system as a whole. After a crash occurs, the server should be able to rebuild its state from its logs. Different Fault-Tolerant approaches dictate when this should be done. Some mandate that server builds its state immediately after restart whilst other mandate that this would be done on demand. The latter approach will reduce the delay that clients might experience while awaiting the server restart.

We are assuming that after re-start that all the applications important state information has been successfully recovered and that any information that was lost is re-creatable.

CORBA clients can often transparently re-launch a server that has crashed between client requests. However, if the crash occurred while a server was processing a client request, an exception will certainly be passed back.

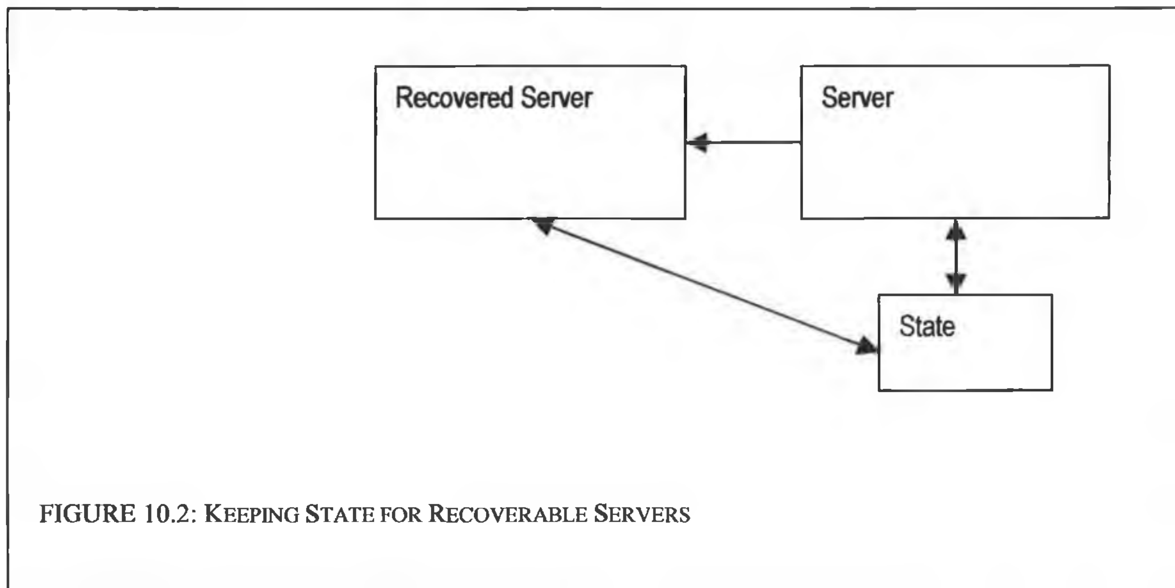


FIGURE 10.2: KEEPING STATE FOR RECOVERABLE SERVERS

10.5.4 Server Monitors

Another method of detecting possible problems with servers is to have a special dedicated standalone server that monitors all the other servers in the system and restarts them in the case of failure.

It can detect shutdown by connecting to any of these servers and noticing when a TCP connection closure occurs.

A Server Monitor must be able to also correct deadlocks perhaps by associating a timeout with each invocation. If the request times out and we are sure that it its not just a slow system, the server monitor can kill and restart the hung server.

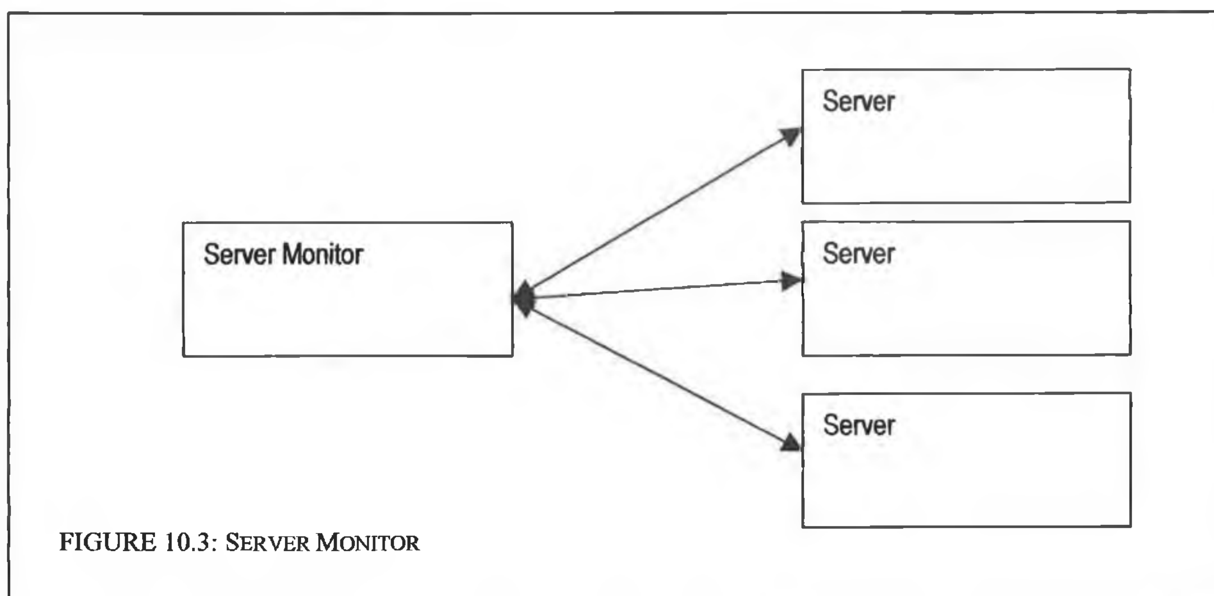


FIGURE 10.3: SERVER MONITOR

10.5.5 Replicating Objects

Both of the solutions outlined so far will provide a suitable Failover mechanism in the case of a process or server crash. However neither will adequately deal with the scenario of the host crashing or network failures. To adequately deal with these scenarios, the only real solution is to have multiple copies of the same service running on physically separate hosts. In the case of failure, clients must then be able to switch to a working replica or else multicast requests to all members of the group and take the first response.

[Marchetti, Mecella, Baldoni 2000] define a distributed application as being fault tolerant if it can be properly executed despite the occurrence of faults. Fault-tolerance can be obtained by software redundancy in that a service can survive a fault if its provided by a set of sever replicas

[Marchetti, Virgillito, Mecella, Baldoni 2001] outline how fault-Tolerance and high availability can be obtained by software redundancy. A service can survive a fault if it is provided by a set of server replicas hosted by distinct servers. If some replicas fail, the others can continue to offer the service. At this end, servers can be replicated according to one of the following replication techniques: Active replication or Passive Replication.

Replicating these services is quite a trivial task when the service is stateless. However it is an entirely different matter when stateful services are being used.

Implementing this solution involves the client being made aware of a failure by means of a system exception. In the case of a host or network failure, this usually involves receiving a COMM_FAILURE system exception on the remote invocation. The client will then switch to

another member of the replicated server group, which implies that the client needs to know about the group members.

The client can have details of the group members either via a lookup service or by just switching to another group member it knows about (a cached instance for example).

The concept of distributing group information among clients can make group updates quite difficult and very inefficient but the concept of a lookup service adds a single point of failure to the system

In this system requests are delivered to just one object but if we are dealing with stateful objects then update requests must be propagated to the group.

10.5.5.1 Primary-Secondary Replication

There is a special type of replication known as Primary-Secondary Replication whereby client requests will be sent to a primary server. This primary will update the secondary and then if the primary fails, the secondary is in a perfect condition to take over as the primary and a new secondary will be started up once this has been completed.

The major benefit of this approach is with the consistency of updates as only the primary is able to make updates there is less chance of difficulties arising. Once again however, a reliable network connection between the primary and secondary is essential and once again we will not be able to distinguish between a host failure and a network failure and such a network partition can create confusion as to who is the primary.

10.5.5.2 Stateful Objects

As mentioned in the previous section, replicating objects is a straightforward way of providing a Failover mechanism when the objects in question are stateless. However, once we introduce objects with state into the equation, things become significantly more difficult.

We now require some way of keeping all the server side objects in synchronisation with each other so that they all have access to the same state information. One way of doing this would be to provide a shared database. This will simplify the consistency of the data but adds a reliability concern as the database is now a single point of failure.

The alternative would be not to use a shared database but to have a scenario where the replicas update each other. This would involve the use of a shared lock service or a transaction mechanism to make sure the updates are consistent. Again, such a transaction or lock service needs to be reliable.

Any new replicated object joining the other objects in the group will need to acquire the current state. This can be retrieved either from the database or from the other replicas. In addition, the clients of these objects will need to be notified that there are new objects in the group (alternatively they can stay hidden until they are discovered).

For this scenario to work we need to assume a reliable network between replicas as they cannot stay consistent if separated by a network partition or if separated from the shared database.

10.5.6 Multicast

Multicast is a system where clients will not just make a request on one object but rather each request will be directed to all members of the group. The client will then use the first response it receives and ignore the others. If the client requests are reliably delivered, then there is no need for state sharing between the objects as they will be kept in lock step by receiving the same client requests. For this approach to work effectively, a guaranteed delivery multicast protocol should be used.

10.5.7 Fault Tolerance Patterns

There are some well-known patterns occurring in the design of high availability systems defined by **[Coplien et al. 1996].**

[Minimise Human Intervention]: Machines don't make mistakes, people do.

[People know best]: Human authority is required to sense the importance of external faults and the actions needed to repair them. System administrators should be able to override automated controls

[Riding Over Transients]: Check the condition really exists before reacting to detected conditions. (Situation might resolve itself)

[SICO First and Always]: The System Integrity Control Program (SICO) is the core component that provides diagnostics and operational control of the system. Trust this component to control the actions of the other system components as well as the initialisation process and normal application functionality

Further general patterns for use in fault Tolerant CORBA systems are defined in **[Natarajan et al. 2000]**

The Leader/Follower pattern provides an efficient concurrency model where multiple threads take turns sharing a set of event sources in order to detect, demultiplex, dispatch, and process service requests that occur on these event sources. This Architectural pattern helps avoid missed polls in the fault detector.

This Active Object design pattern decouples method execution from method invocation to enhance concurrency and to simplify synchronised access to an object that resides in its own thread. This avoids excessive overhead of recovery.

There are a group of optimisation patterns that reduce excessive overhead of service lookup. These are Optimise for the common case, Eliminate gratuitous waste and Store extra information which optimise by storing information and eliminating gratuitous waste.

The Strategy design pattern factors out similarities among algorithmic alternatives and explicitly associates the name of a strategy with its algorithm and state. Using this pattern helps with the problem of tight coupling of data structures.

Abstract Factory is another design pattern used in CORBA Fault Tolerance and provides a single access point that integrates all strategies used to configure the FT-CORBA middleware.

The Component Configurator design pattern employs explicit dynamic linking mechanisms to obtain, install, and/or remove the run-time address bindings of custom Strategy and Abstract Factory objects into the service and installation-time and/or run-time. This overcomes the inability for dynamic configuration.

The Chain of Responsibility Pattern decouples the sender of a request from its receiver, in conjunction with The Perfect Hashing pattern to perform optimal name lookups. The Chain of Responsibility pattern links the receiving objects and passes the request along the chain until an object handles the request.

Initial efforts to enhance CORBA with fault tolerance have taken an integration approach, with the reliability mechanisms incorporated into the ORB itself. With the advent of Object Services in the CORBA standard, other research efforts have taken a service approach, with the provision of a reliable object group service as part of the Object Services. **[Narasimhan et al. 1997]**

A combination of these approaches results in the Interception approach that involves capturing the system calls of the objects hosted by the ORB. The intercepted calls, which were originally directed by the ORB to TCP/IP, are now mapped onto a reliable ordered multicast group communication system. The advantages of this approach are that neither the ORB nor the objects need ever be aware of being "intercepted" and, thus the fault tolerance is not visible to the application objects.

Finally, the internal structure of the ROB requires no modification since the mechanisms that provide reliability are external to the ORB.

CORBA did not provide tools for enhancing the reliability of distributed systems. This had two consequences. Many CORBA systems added replication logic to standard ORBs to cope with object failures and site crashes. Examples of such projects include:

- Eternal **[Moser et al. 1999]**
- OGS **[Felber 1998]** and **[Felber et al. 1996]**
- DOORS **[Chung et al. 1998]**
- Isis and Orbix [*from IONA Technologies*]
- Electra **[Landis Maffeis 1997]**
- AquA **[Cukier et al. 1998]**
- IRL **[Marchetti, Mecella, Virgillito, Baldoni 2000]**

10.6 Load Balancing

10.6.1 What is Load Balancing

The concept of load balancing involves dividing the workload of a banking system across the network so that bottlenecks are avoided as much as possible and throughput is increased. If we successfully distribute the processing and communications activity throughout the system we can be relatively sure that no one process or machine will reach over capacity and bring the entire system to blockage point.

A load balancing solution can be used to improve a systems performance as well as its availability and can be used in a fault tolerant approach. The most common load balancing solution is something we have just looked at - i.e. Replication.

In a CORBA System, a load balancing policy will typically involve having multiple copies of the same service throughout the network so that client requests can be distributed among these services.

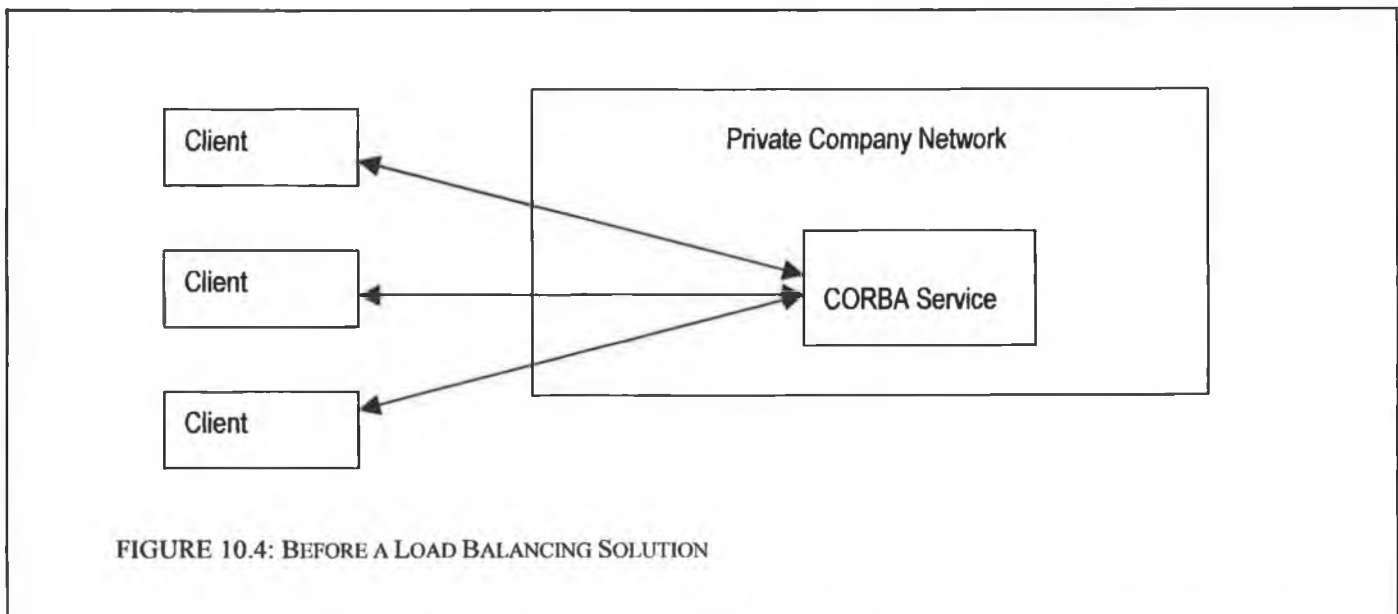


FIGURE 10.4: BEFORE A LOAD BALANCING SOLUTION

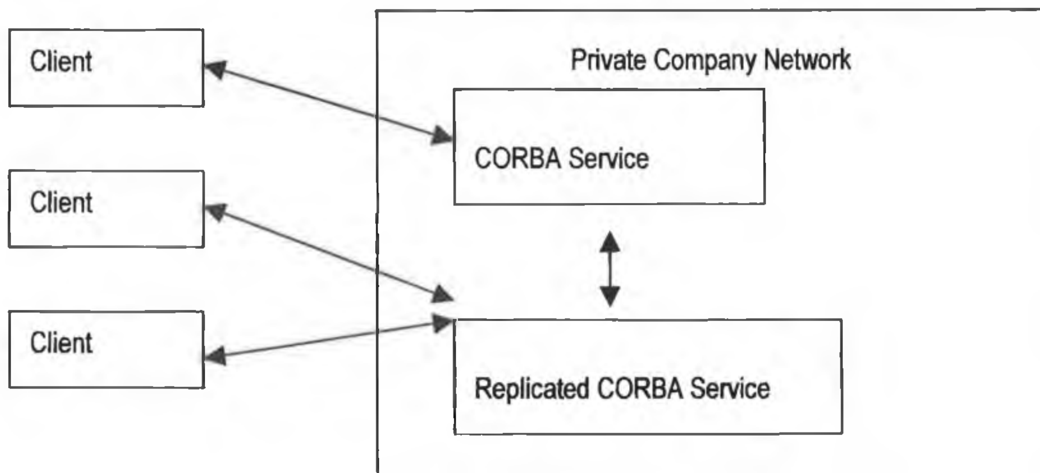


FIGURE 10.5: SIMPLE LOAD BALANCING SOLUTION

10.6.2 Requirements of a Load Balancing Policy

In this context we are considering a load balancing policy for its performance improvements to the system. However in an enterprise application, a load balancing policy will also have additional requirements such as improved scalability, improved reliability, and increased availability.

The load balancing policy is deployed to share or equalise the workload among the available resources in order to optimise performance. There are several popular policies as defined by **[Hoon et al. 2001]**

A **Selection** policy selects a task for transferring. This policy is relatively simple as it selects the task based on First Come First Serve basis.

A **Location** policy selects a location for transferring the selected task. Basically, this policy will need to rely on the information policy for collecting certain information (e.g. load level) from different locations. Usually, this policy will attempt to choose the location with the least load level or below certain threshold. However, random location policy that was mentioned earlier does not need collection of any information.

An **Information** policy determines the level of information needed for task selection or location selection. It decides on when to collect the information and what information to be collected. It also decides whether to collect information from all locations or just part of them, to distribute the information at different locations or to centralise them etc.

The requirements in terms of performance are simply that the load should be divided across the system in a predictable and reliable way that will guarantee the same availability of the CORBA services that an older mainframe application would provide.

[Othman et al. 2001] define key requirements that CORBA load balancing services should be designed to address:

- Support an object-oriented load balancing model
- Client application transparency
- Server application transparency
- Dynamic client operation request patterns
- Maximise scalability and equalise dynamic load distribution
- Increase system dependability
- Support administrative tasks
- Minimal overhead
- Support application-defined load metrics and balancing policies.

10.6.3 Benefits of Load Balancing

Typically a load balancing policy will be implemented for services that have a reasonable amount of clients where there is no obvious pattern to the number of concurrent requests.

In such cases, with increasing numbers of users, early adoption of a load balancing policy will mean that continuing increases in user numbers will not affect the overall performance of the service.

10.6.4 Dangers of implementing a load balancing policy

Load Balancing can be a tricky addition to a bank's infrastructure. There are occasions when a particular policy will not provide the performance improvements required and can even reduce the scalability of the system.

One example of load-balancing failing is where there are a significantly larger number of clients than services. Another case is if certain methods take a long time to complete, then distributing the services across the network might not be beneficial. This is less likely in a mainframe environment where many legacy applications will have been implemented as CICS or IMS transactions and by their very nature will be short running.

10.6.5 Real World Uses

To implement a load balancing solution, techniques such as replication and caching are used and we will see exactly how these are used in the following sections.

In any CORBA system there are Services that can be load balanced and there are CORBA Services such as a Naming Service that can be load-balanced. The idea being that the number of concurrent client requests on a particular service can be kept to a certain minimum by distributing these requests among several instances of this service.

One important detail to note is that any load balancing is hidden from the client. As far as they are concerned they are accessing a particular service and need not be concerned with details such as which instance of that service they are accessing.

There is no single way to implement a load balancing policy. CORBA Services can use network based load balancing at the network layer and the transport layer. This layer use the IP address or the hostname and port to determine where to forward packets.

Modern banking systems may have requirements to support many clients accessing from anywhere in the world. There can be no limit on the number of client requests, nor when the peaks of client interest will occur. The resources of the hardware implementing these services needs protection and can be costly and the purchasing of extra hardware or server cycles is not always cost effective.

Rather, a load balancing solution can result in hardware and software mechanisms determining which server will execute each client request. Such a load balancing solution will distribute any incoming client requests over all the back-end servers so that the system response time as a whole will be favourable.

The BaBar Experiment outlined in **[Becla, Gaponenko 2001]** collected around 20 TB of data during its first 6 months of running. After 18 months, the data size exceeds 30 TB. Even this is expected to be only a fraction of the size of data coming in the future.

In order to keep up with the data, significant effort was put into tuning the database systems. It led to great performance improvements, as well as to inevitable system expansion - 450 simultaneous processing nodes alone used for data reconstruction. It is believed that further growth beyond 600 nodes will happen soon.

In such an environment, many complex operations are executed simultaneously on hundreds of machines, putting a huge load on data servers and increasing network traffic. Introducing two CORBA servers halved start-up time, and dramatically offloaded database servers: data servers as well as lock servers.

10.6.6 Load Balancing Algorithms and Policies

Some of the most common policies for load balancing include

- Random
- Round Robin
- Host Response Time
- Availability of Hosts

Any load balancing policy should be considered at design phase. When deciding on such a policy, other scalability issues should be taken into account. For example, a load balancing policy can be used to overcome resource limits, and limit failure damage as part of a fault tolerance policy.

An enterprise application might consider load balancing as a solution if there are workload variations on the system. If the timing and frequency of requests differs and the type of requests

incoming to a system are subject to variations, it is a good idea to get a feel for the access patterns of the system. Another consideration is if the client request variations are time dependent or are there peaks and periods of inactivity on the system.

If certain parts of the system get more client requests that others, i.e. access to different functional areas has variations, a partitioning of the application logic could yield some immediate performance results. This partitioning could be based on the usage patterns of the system. For example, query requests could be partitioned from read/write requests, which in turn could be partitioned from transactional requests.

The granularity of the components in a system must be also considered when implementing a policy. For example, if the system contains mostly fine-grained components that are well encapsulated, the amount of interaction they have with each other must be taken into consideration. If there is a lot of communication between the components themselves, the performance benefits could become minimal and there may even be a negative performance benefit.

Even with today's high-speed networks, the runtime topology i.e. the geographical concentration of accesses must be looked at carefully. If network speed and network reliability and even bandwidth are not considered, again there may be a net loss. The topology of the deployed application itself must be also looked at.

10.6.7 Implementing Load Balancing using the CORBA Naming Service

A CORBA Naming Service can be used to implement Load Balancing. The OMG defined CORBA Naming Service specification describes a model where a name maps to a user defined object. A Naming Service can be extended so that a name can instead map to a group of objects. When a client selects an object from the Naming Service, the Naming Service will select an object from a group of objects to resolve that request.

10.6.8 Network Based Load Balancing

I.P. routers and domain name servers (DNS) can be used to provide a load balancing policy that supply a pool of host machines. The DNS can decide which IP address to use when a client resolves a hostname depending on the resources available to the system. This can be based on the current load on the systems resources or the response time from the various servers. This load balancing is completely transparent to the client and it may be the case that each time a client resolves a host name, it is a different server that is returned with the DNS resolution. Routers can be used, depending on the load on the system at a particular moment in time to bind a TCP flow to any of the servers available. It can then use that binding for the duration of the flow.

Load balancing at the lower layers such as the network layer or the transport layer is not always optimal depending on the actual volume of client requests. Load balancing at higher layers can base their policy on the content of the requests, for example the pathname information contained in a URL. Such a policy can be used to determine which Web server should receive a particular client request for a certain URL. Using this technique, the most popular URLs can already have extra resources waiting for them.

10.6.9 Operating System Load Balancing

The techniques of clustering, load sharing and process migration can be used by distributed operating systems to provide a load balancing solution. Clustering can result in high availability and high performance but at a lower cost. It combines different machines to improve the power of the system and processing power as a whole. It can distribute processes across the system, again transparently to the processes themselves. Clusters use techniques such as load sharing and process migration.

Process migration is a technique used to balance the load across processors or network nodes. The state of the process is transferred between nodes, but this requires quite a lot of platform infrastructure support to handle the differences between nodes. Programming languages based on Virtual Machines such as Java are going to have limited applicability when using this approach.

10.6.10 Software approaches to implementing a Load Balancing Solution

The Partitioning Pattern and Replication Pattern are two solutions commonly used in today enterprise solutions as load balancing techniques. We have seen how replication works but partitioning is a technique that comes in two flavours - Horizontal and Vertical.

Horizontal partitioning is a technique whereby objects are assigned to a particular partition in a system. Using this technique, each service a system provides will only exist on a particular partition and it is the combination of these objects on their different partitions working a whole that provide the functionality of the system as a whole. This technique is also known as "interface partitioning" and it essentially a way of distributing the load of a particular system across the system.

Vertical Partitioning works like Horizontal Partitioning but, in this case, the partitioning of the system is based on the data it holds. Every server provides the services exported by the system and the particular group of objects will exist on each server. In N-tier CORBA systems there will be N event channels representing the service. This technique is easily applied to data centric applications.

10.7 Conclusion

The purpose of this section is to highlight the various approaches that can be used in an enterprise solution to keep the legacy systems that are now CORBA Services available to users of the system. As was mentioned, these legacy systems were traditionally highly available in their own environment but when we reengineer them to be peers in the distributed computing environment we run the risk of this availability being compromised due to the fault-prone world of network computing.

[Froidevaux et al. 1999] found that full integration of CORBA with their legacy systems allowed Credit Suisse to implement CORBA applications with the same availability as traditional IMS-

TP-based which is typically > 99% in the Credit Suisse's data centre operation. Thus the full integration approach provides a CORBA platform for mission-critical applications.

We have seen possible solutions to handling faults in the system. These include a Fault Tolerance solution, a Load Balancing solution and a comprehensive Exception Handling solution. The combination of these features added to the enterprise can make for availability figures that we would expect from mainframe based applications.

This section completes this phase of the research. We have now investigated in some detail, all of the key areas of concern in an enterprise integration project.

As per the objectives set out at the beginning of the research, our next task is to propose an approach to solving these problems. This is achieved by applying various approaches discovered in the last few sections to an "actual" enterprise integration project.

This next section outlines our approach to enterprise integration that forms the key to this research.

11 Implementation and Recommendations

11.1 Implementation Introduction

For the implementation of this thesis I will detail how real-world mainframe based, legacy applications were integrated within the Credit Suisse CORBA Project of whom I was a team member. In this project we applied many of the concepts, patterns and techniques outlined in previous chapters. Only those that provided real business value to the bank were considered and those affecting performance and security were given the highest priorities. This chapter will detail which of the patterns and techniques were chosen and how the implementation proceeded.

This chapter combines a case-study on how a major bank (Credit Suisse) chose CORBA as their integration solution, and an implementation of the additional the problems and issues that were discussed throughout this thesis. **[Froidevaux et al. 1999]** and **[Murer, Koch 1999]** outline the reasons for the choices Credit Suisse made in each case. The application of patterns and solutions in the areas of Scalability, Performance, Failover and Security by this author, complete the implementation of this thesis.

11.2 Credit Suisse Mainframe Architecture

Like most major banks, Credit Suisse has relied heavily on technology for some twenty years to streamline its operations as well as to compete with the rest of the market to match the business and technical requirements of modern banking.

Some figures for Credit Suisse's Technology Departments **[Hagen 2000]** include:

- 40 million lines of code in the central system
- More than 100 Credit Suisse productive self-made applications
- More than 100 simultaneous projects
- Up to 1 million payments per day
- 25000 work stations networked
- 1000 Servers
- 400 Million Pages of output on paper per year
- 16 Million IMS Transactions per year
- 50000 Databases, 30 TeraByte Disk
- Typically over 99% availability
- Several different electronic banking channels (internet, phone, ATM etc)
- Approximately 1500 employees in the Information Technology Sector.

FIGURE 11.1 CREDIT SUISSE TECHNOLOGY DETAILS

However, in the last decade, the architecture of Credit Suisse's Information Technology has changed. Like many Financial Institutions, they have moved away from the traditional centralised mainframe solution and moved towards client/server and n-tier solutions. Credit Suisse has felt the same pressure for change that all of its competitors in the industry have. These include

pressure for new technologies such as ATMs, 24-hour banking, Online banking, Phone Banking etc.

However, Credit Suisse are not alone in the industry in recognising the mainframe as a key part of their Information Technology Architecture. This platform has served them well over the last decades, being powerful, secure, reliable and very fast. Naturally, there are questions being asked of these new technologies so that none of these benefits will be lost with any change.

Not only was it apparent that the benefits of the mainframe could not be lost, it also became quickly clear that even if a new technology or operating system was available that could guarantee the benefits, it would be quite a difficult task technically and financially to move.

Looking at the old applications in IMS and CICS, it was quickly realised that these would have to be re-written from scratch, as not all the original documentation was available. In addition a look at the costs of such a move indicated that it would cost some 900 million Euro over 5 to 7 years to implement a new system in the Credit Suisse environment. **[Koch Murer 1999]**

11.3 Choosing a suitable Integration Architecture.

It was decided at an architectural level that the Managed Evolution Pattern outlined previously would be most appropriate in the case of Credit Suisse. This leads to a good balance between risk and opportunity. Credit Suisse could decide on a target architecture and work towards it in small steps.

One of the requirements of using Managed Evolution is that the system be partitioned into manageable components or layers separated by clear interfaces. Because the Credit Suisse Information Systems, as a whole, will live longer than any of its technologies, there should be an interface technology that bridges technology and space. This model results in a Service-Based Architecture

I found this to be a clever decision from the technology side. It meant there was no "all-in-one" approach and we could move towards the target architecture in steps. Furthermore, as there were advances in the CORBA implementations by IONA Technologies, these could be included in the next Architectural Release.

The next decision that was made was to use the Standards-Based Solution pattern. This pattern involves using a technology solution that is independent of vendor implementation but adheres to industry standards. Examples of this solution include the EJB Specification from Sun Microsystems, the .NET Specification from Microsoft, or the CORBA Specification from the OMG. Other non Standards-Based Solutions such as Screen-Scraping were not accepted due to lack of a well-defined interface between Services.

I found that using such a Standards-Based Solution made defining the Architecture of our mainframe systems quite easier. We avoided possible pitfalls that could have arisen in trying to define a suitable cross-platform interoperability solution that our predecessors in proprietary solutions had failed in. These pitfalls were discussed in [Chapters 1] and [Chapter 2] and detail how quickly such proprietary solutions become Legacy solutions in themselves.

Another requirement of any solution was that it have a mapping for integration of legacy programming languages such as COBOL and PL/I which are key in Credit Suisse. The full list of requirements for Credit Suisse include:

- Interface definitions must be independent of the programming language and platform.
- A standards based solution allows easy integration of third party products
- Meta-data such as interface definitions should be available within the system
- The service architecture must be available on all relevant platforms
- Interface definitions must be extendable to allow a managed evolution
- Support for several programming languages is required
- Systems management capabilities are important.
- A Naming Service is needed to ensure location transparency

In the end only the CORBA specification from the OMG adhered to the Managed Evolution and Standards-Based Solution patterns and could work with a mainframe integration project. .NET and EJB restricted the architecture to an operating system or programming language and did not provide support for COBOL or PL/1.

The full list of reasons for choosing CORBA were:

- The standards based solution (IIOP in particular) give Credit Suisse the freedom to combine several different middleware products in the same system. Credit Suisse combines Promia's SmalltalkBroker with IONA's Orbix. Other major companies provided full integration products that were not standards based. This would tie Credit Suisse into a specific vendor whereas choosing CORBA meant that Orbix could be changed for another CORBA implementation with relative ease.
- Credit Suisse wanted a technology from the market. Previous experience showed that in-house development of complex middleware is too expensive. It was also important to get a reasonably mature technology. Various Java based mainframe integration solutions were starting to appear on the market but as yet had not been proved in mission-critical situations.
- The strong focus on interface definitions (IDL) in CORBA is an ideal match for the service architecture, where interfaces are the contracts between service users and providers. Techniques such as screen scraping and other non-invasive techniques would not allow this.
- CORBA is an excellent technology integrator: IDL interfaces can be mapped into such diverse technologies as IMS transactions written in PL/1, Java Applets, or Visual Basic.
- Useful additional middleware services like security, naming, trading and transactions are part of the CORBA architecture. Again these are standards based and so do not tie Credit Suisse into a specific implementation.
- There are a number of different CORBA implementations spanning all relevant platforms (programming languages and systems) within Credit Suisse.
- Many CORBA products are now mature enough to be used in an enterprise environment. Necessary features like integration into a systems management framework or logging facilities for accounting and security can be integrated with reasonable effort.

During different phases of the project we considered other non-CORBA solutions to test whether our original goals were still being met. There had been various Web-Services integration solutions available on the marketplace during this time and these would have also allowed a Managed Evolution approach. Such products were also Standards Based. However, I recommended that we did not pursue these further as they did have the maturity of the CORBA industry

implementations. This conclusion was arrived at by an examination of the Security and Failover possibilities (or lack thereof) coming with these solutions.

11.4 Building An Architecture based on Managed Evolution

The Managed evolution pattern allows continuous adaptation of the IT system. To ensure that these changes lead to a long-term improvement of the system, each step should be directed towards target architecture. In addition, design of new software modules and wrapping of existing transactions should follow a coherent style to ensure a smooth operation.

11.4.1 Services-modules instead of components

It became clear that the existing applications in Credit Suisse do not fit into the conventional definition of components. It was decided that the name *Service-Module* was more appropriate than *component.*

[Koch Murer 1999] explain this naming decision:

> *The deficiencies of existing applications with respect to the component definition vary in a wide range. Many of the existing applications are not accessed at their interfaces only, they can not be independently delivered and deployed, and they are tightly integrated with their environment. While we expect new applications to improve the situation significantly with regard to clearly defined interfaces and data en-capsulation, the integration into the Credit Suisse mainframe system will still be very tight.*

Service-Modules are encapsulated and accessed at interfaces only. Each interface offers one or more operations (services)

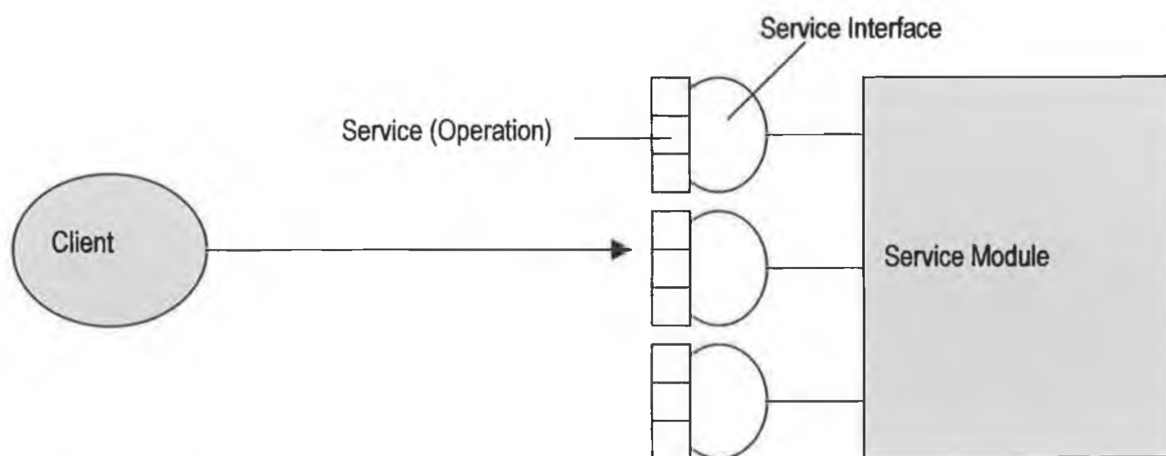


FIGURE 11.2 SERVICE-MODULES

Credit Suisse Service-Modules:

- Contain data and appropriate functionality to manipulate this data. This is achieved by providing well a well-defined interface, and the Service-Module can only be accessed via this interface.
- Are stateless entities, in that they do not maintain a session state, but they can operate on persistent data entities.
- Allow inheritance to be provided only between interfaces.

As part of the implementation of this project we had to map the concept of a Service-Module to a real-world CORBA implementation. This turned out to be particularly easy as Interface Definition Language (IDL) is the key to CORBA communication and our Service-Modules could be easily defined in this way.

11.4.2 Bottom-up approach for the existing system

All of the initial CORBA applications in Credit Suisse were those that were being re-engineered and therefore were designed with a bottom-up approach.

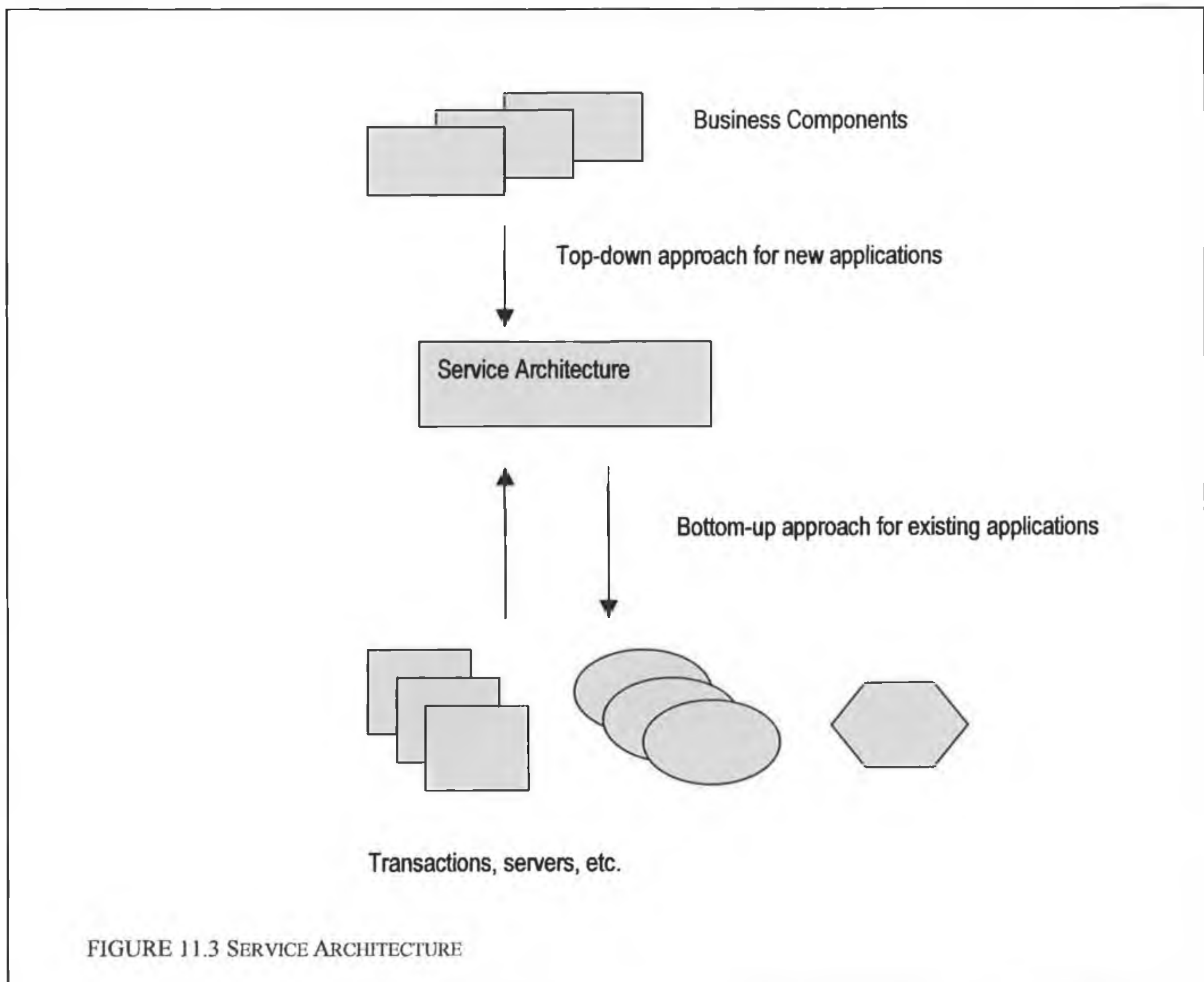


FIGURE 11.3 SERVICE ARCHITECTURE

The design process was started with an investigation of the existing system. The applications were grouped into several domains. Each domain is further refined into service groups such as "account services" and "customer information services" which are service groups within the "Core Banking Domain". Each service group contains one or more Service modules.

For each service an interface definition must be produced based on the features of the current implementation.

Due to limited resources and dependencies between modules as well as maintenance overhead and other problems, a decision must be made as to which part of the systems should be re-engineered and which existing program modules can be used without major changes. Using the Managed Evolution Pattern this can be achieved in small steps so that eventually all core services are re-engineered.

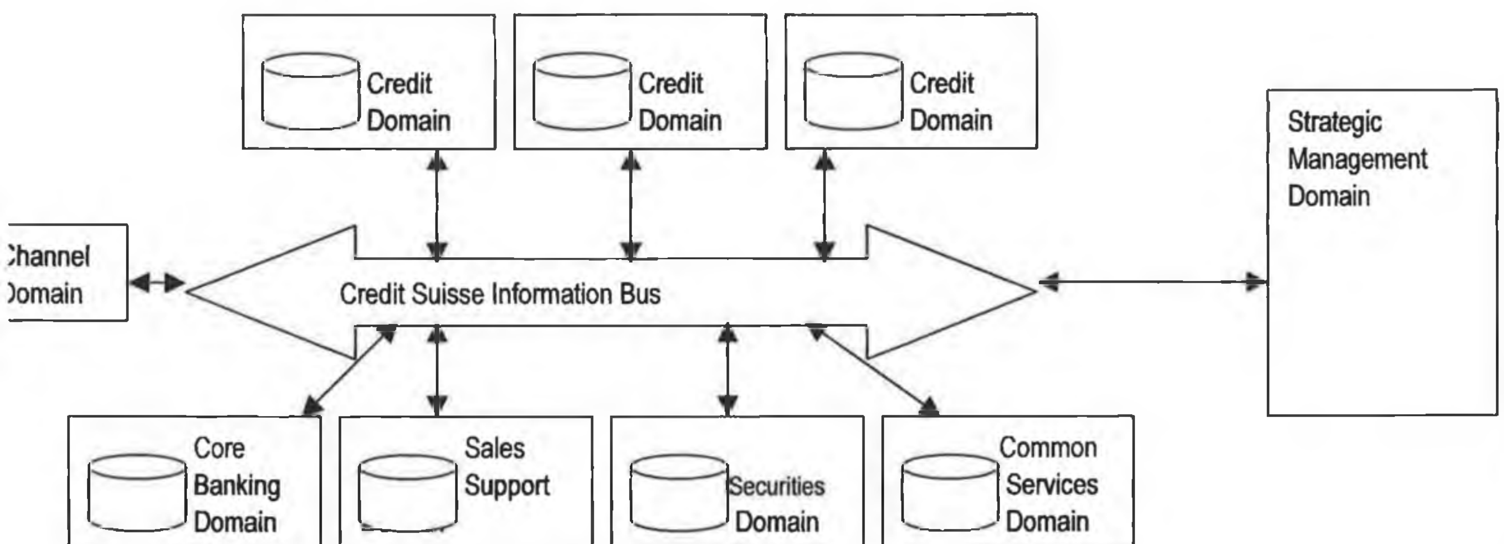


FIGURE 11.4 CREDIT SUISSE SERVICE ARCHITECTURE OVERVIEW

As the system relies on IMS for almost 90% of its applications, the integration of existing IMS transactions provides another motivation for a Service-Based Architecture. Transactions can be offered as services on an interface. Transactions are in most cases stateless services, dealing with objects that "live" in the IMS system. Since the implementation behind an interface is of no interest to a client, we can hide the fact that COBOL (or PL/1) are not Object Oriented languages.

11.5 Performance Concerns

Given that Credit Suisse decided upon on CORBA and distributed computing to integrate their legacy systems, there was always going to be an immediate performance impact due to network latency. As we have seen distributed systems differ fundamentally from monolithic systems:

We have seen in a previously how using a Service-based Architecture leads to having fewer coarse grained objects and having fine-grained objects residing within large grained *Service-Modules*. This approach leads to a system where more information (i.e. a complete customer) is passed with each request, reducing the amount of network overhead.

CORBA's address space model is different from that of most programming languages in that there are no pointers, associations, and relationships between objects are very different. Remote CORBA calls in general show latencies roughly a factor of 1000 higher than local method calls. CORBA itself does not prevent simultaneous access to shared resources so this needs to be implemented by the project. Serialising over a network is much more costly than on a local machine.

However as we have seen there are various patterns and solutions to help reduce these network based overheads. The first of these is to minimise the number of remote operations. A call across the network will always be substantially slower than a local call. With current operating systems even inter process calls on the same physical machine are much slower than calls that execute in the same process. CORBA provides a very convenient sequence abstraction that makes it easy to pass sequences of similar items to method invocations.

In the research on performance enhancement methods, we saw the following techniques:

1. Use sequences whenever several calls to the same operation may occur
2. Communicate structures that contain all/several attributes of an object, instead of asking for each attribute with a separate remote invocation.

In the Credit Suisse Project, I implemented both of these techniques. Looking at the example below, Every time an IDL Operation was defined, rather than simply pass in and out a structure containing *Customer_Details,* a sequence of structures was used. This meant that as this operation was typically called many times from the same client, only one network call was required. Typical IDL design would return an entire customer object for each request resulting in at least n+1 remote invocations for n customers.

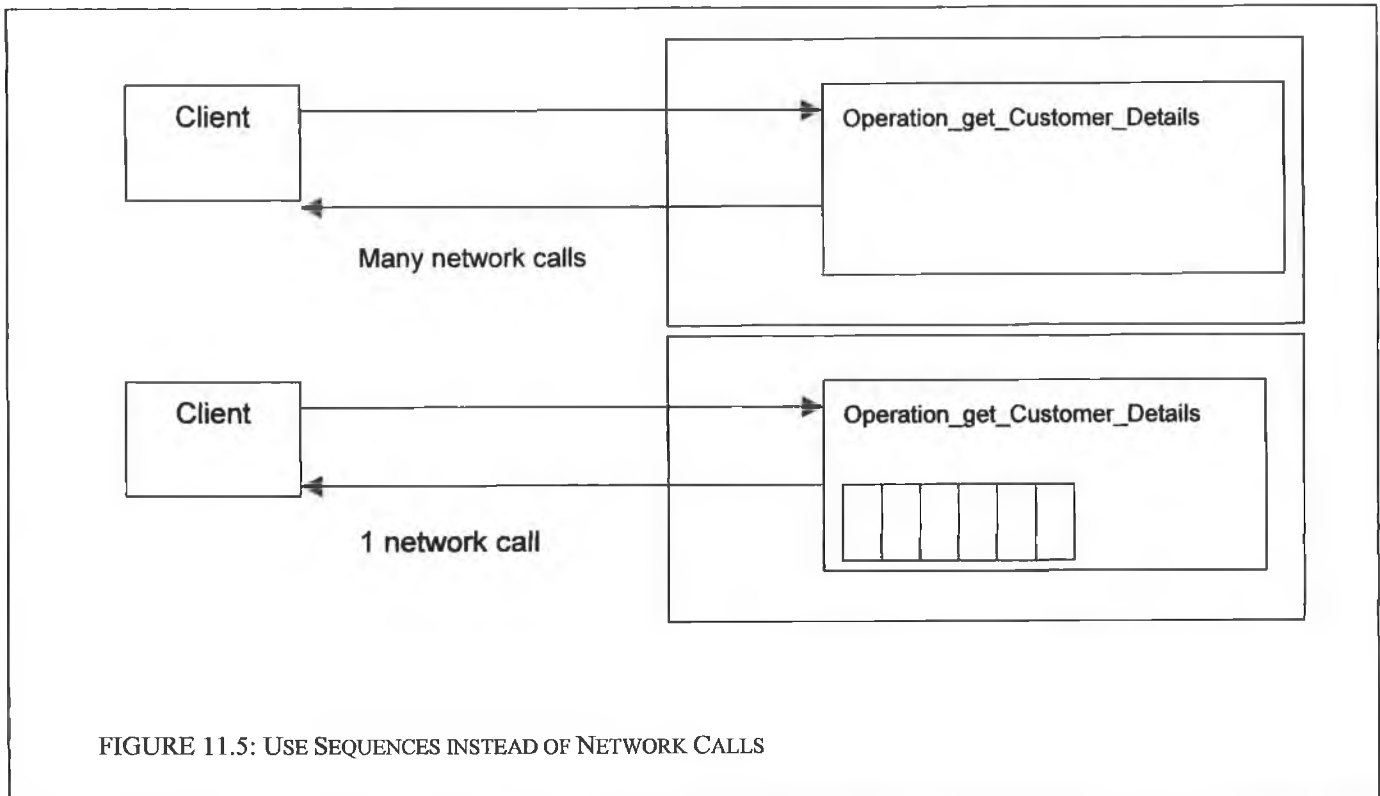


FIGURE 11.5: USE SEQUENCES INSTEAD OF NETWORK CALLS

Implementation of the second technique is easy to see. Instead of having operations such as

String getCustomerName();
Float getCustomerSalary();
String getPhoneNumber();

We can fill up a structure called CustomerInfo{...} and simply have one operation

CustomerInfo getCustomerInfo ();

This results in just one network call instead of the previous situation where there were many network calls. Of course, it does result in possibly more information than is required being passed back from the operation but unless this extra information was a massive amount of data, there will still be performance gains.

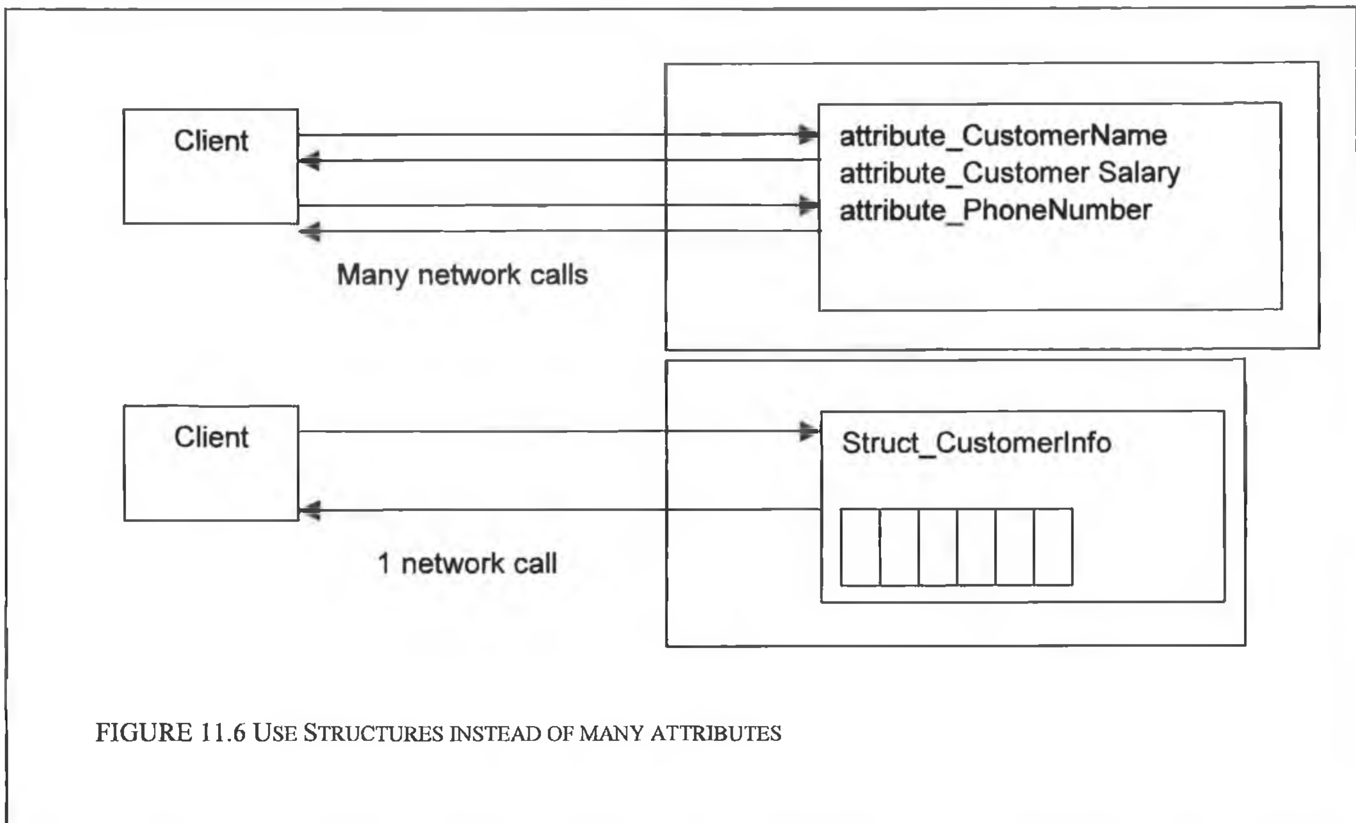


FIGURE 11.6 USE STRUCTURES INSTEAD OF MANY ATTRIBUTES

Both of the approaches outlined here would not be obvious choices for "pure" IDL as the interface definition does not detail exactly what the operation is trying to achieve. However in recent times, designing IDL with performance in mind has become an acceptable option and many enterprise applications seen in various banks today will have such solutions incorporated early on.

Some of the other patterns and solutions that were outlined previously were also considered for use in the Credit Suisse CORBA-Mainframe integration. One well-known solution is to optimise the type of data passed, as some types are more complex to marshal than others are. However, as the projects that were interested in CORBA used a bottom-up design approach, there is less of an opportunity to dictate which datatype should be used as it is required to use a datatype that could contain the legacy data.

As a result, Credit Suisse typically did not enforce the IDL datatypes used by each project this but produced its own IDL design guidelines. These guidelines outlined the different marshalling costs so that those projects concerned with the performance impact of CORBA could make their own decisions.

Optimising the amount of data passed is another way of minimising performance overhead. However, there is a trade-off here because on one hand we would want to pass as much information in each request as possible so this pattern was somewhat ignored.

Many of the other patterns designed to optimise CORBA performance do not apply in the case of legacy systems on the mainframe. Using a Service-based architecture will result in some industry wide patterns becoming redundant (examples include Fine Grained Framework and Flyweight) that show solutions for having many fine-grained objects.

Because IMS/CICS Transactions are typically short running, other industry patterns such as Distributed Call-back are also not required in this architecture. There are certain exceptions to this rule such as conversational IMS Transactions and long-running CICS Transactions but these were not considered in this project.

Others such as the Replication Pattern can improve performance but are more suited to a discussion on the Scalability of the system.

One of my roles within the Credit Suisse CORBA project was to continuously test and adapt performance enhancement techniques. This involved applying patterns such as those above as well as industry-wide idioms and technology updates. The more successful of these included

- Modifying Stack and Heap size usage by Language Environment on OS/390
- Finding faster images in the system to run the CORBA Infrastructure under
- Applying different Multithreading policies to discover the most appropriate
- Test new Java Runtime Environments for possible performance enhancements
- Testing the different Java and PLI types to discover difference in marhsalling costs
- Fine tuning the IMS (Message Processing Regions) MPR and CICS Regions with techniques such as Pre-loading of Transactions, Language Environment Tuning and MVS priorities.
- Testing new versions of IONA Technologies' CORBA implementation software to discover advances in marshalling techniques and lower CPU usage.

11.6 Security Considerations

Credit Suisse, like any major bank included maintaining the security of its data as among the top priorities when considering CORBA as an option for integrating its legacy applications.

The bank required clients (both internal and external clients) to have access to information (for example customer account details) that resided on the mainframe in the private company network. In this first architecture there is no security provided so this had to be built up.

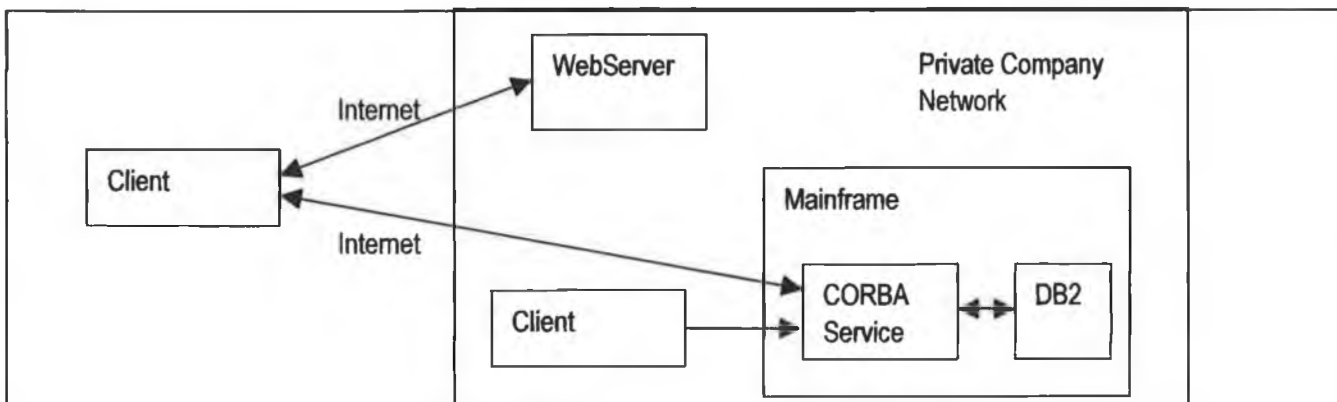


FIGURE 11.7: INITIAL BUSINESS REQUIREMENTS

CORBA supports the use of SSL (Secure Sockets Layer) and so each CORBA request was required to use SSL on top of CORBA IIOP. A Credit Suisse Public Key Infrastructure was created resulting in the bank having its own Certificate Authority (CA). This meant that each user of the system would receive their own certificate and that every request entering the CORBA Infrastructure was encrypted and secure and could also be verified and logged.

Secondly, a RACF/ACF2 lookup takes place for every user trying to access an IMS/CICS transaction via the CORBA Adapters to ensure that the user have the correct permissions to complete this task.

Both of these security measures ensured that users inside the private company network could be verified and meet the requirements of a security service. However, provisions also had to be made for users outside the private company network. The approach taken by the team in Credit Suisse was to add a security system in steps as per the Managed Evolution approach.

The very first step was to secure the perimeter of the private company network.

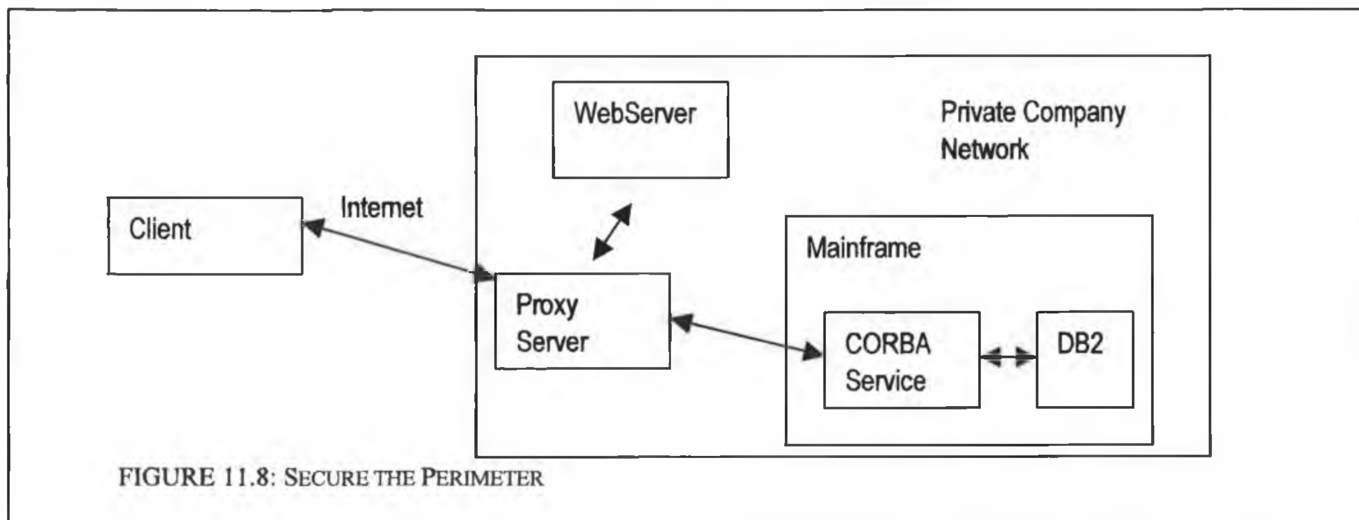


FIGURE 11.8: SECURE THE PERIMETER

Adding a Firewall Proxy Server ensures that all requests entering the private network must pass through this proxy server. As outlined in the research on CORBA Security, using a Firewall Proxy Server ensures that each and every request entering the private company network is monitored and logged, drastically reducing the chances of security holes in the perimeter.

The next phase was to enhance the security of the private company network by adding a demilitarised zone plus an additional firewall.

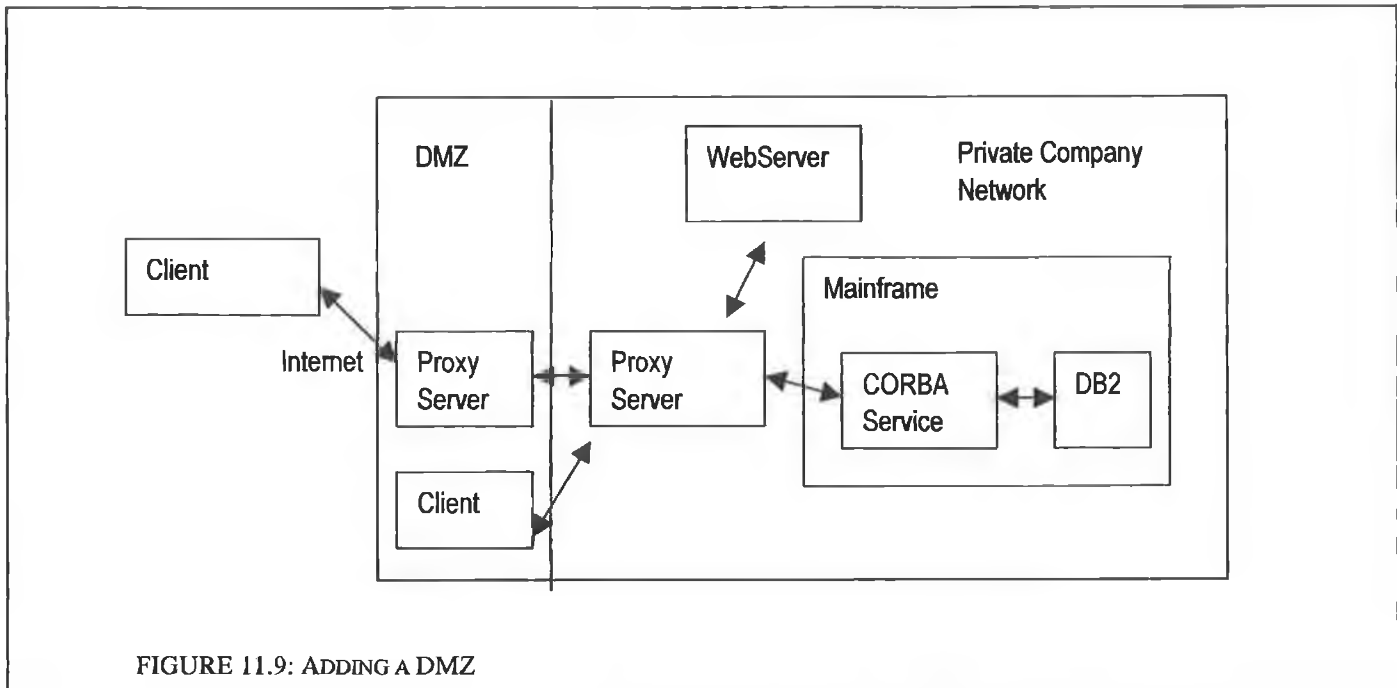


FIGURE 11.9: ADDING A DMZ

With the addition of a (DMZ) demilitarised zone and an additional Firewall Proxy Server, the internals of the private company network cannot be accessed directly from external clients. There is now a buffer zone between the outside world and the private network.

Among other things, this helps ensure that outside attackers trying to create a denial of service can be thwarted as they will only reach the proxy server and not the CORBA Infrastructure or other servers running inside the private company network.

Adding a CORBA Security Service can be achieved by having a Master Security Server plus adding a Security Runtime to each Application Server.

This Security infrastructure with the Credit Suisse PKI also added to the system provides sufficient security to ensure the integrity of the critical data the bank contains in its private database. There is no possibility of a direct access from outside the private company network. With CORBA Security and IIOP/SSL in place, the requests are also secure. Finally with the PKI in place the rest of the requirements of a security model can be met.

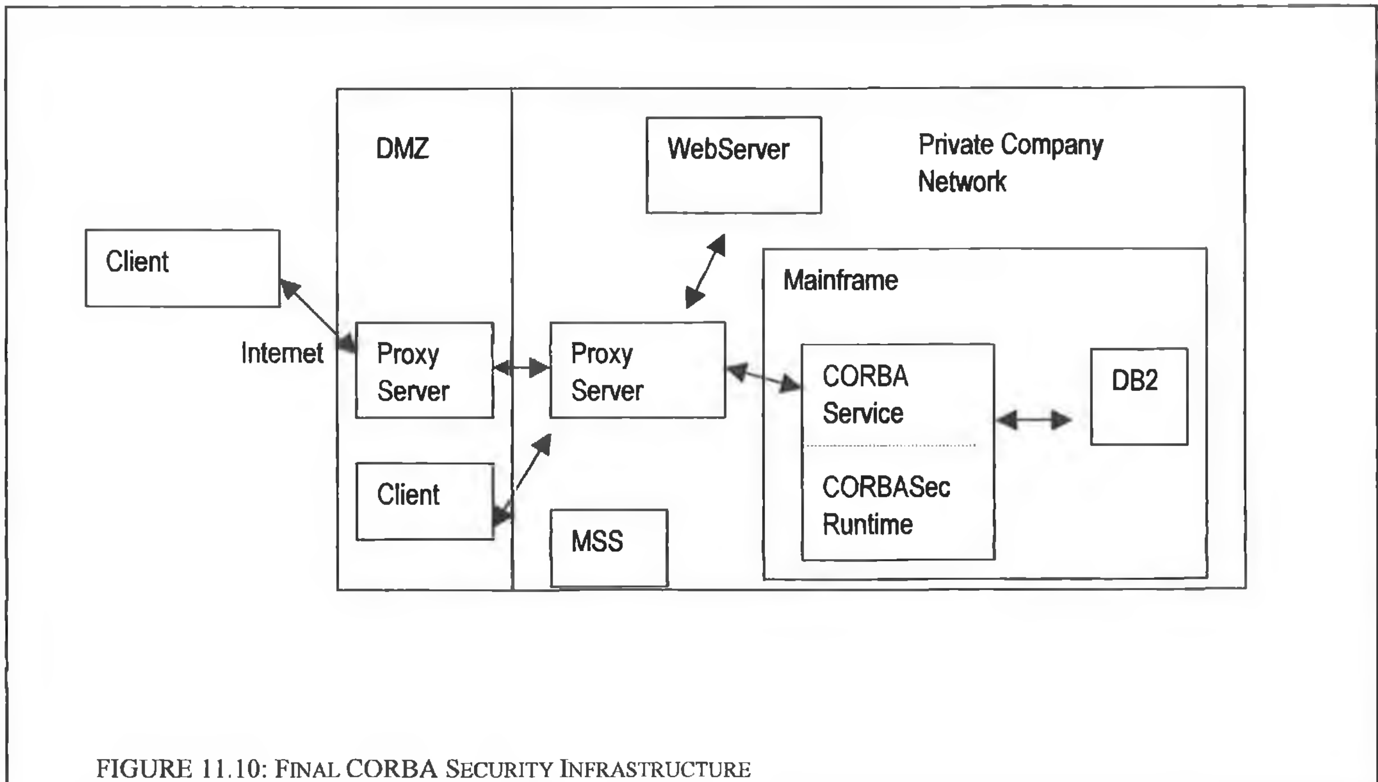


FIGURE 11.10: FINAL CORBA SECURITY INFRASTRUCTURE

One of the problems I found with the approach outlined above is that it added a considerable performance overhead to round-trip-times. This was particularly increased, as logging of all requests was required at each stage of the client to server communication. As part of my performance as described in the previous section, I proposed various solutions that reduced a lot of this additional overhead. These included:

- Optimising the CORBASec Runtime code for efficiency.
- Rather than clients accessing the CORBA Services, we introduced EJB applications servers that acted as the CORBA clients. As these were continuously connected to the CORBA Services, there was a reduction in Connection Management costs.

11.7 Scalability

Scalability was another area of concern for Credit Suisse. All of the areas for improving CORBA scalability outlined in this research were considered.

Multithreading was possible on the client side when using C++ and Java and also with the Orbix IMS Adapter on the mainframe. It was not possible for the PL/I and COBOL developers to use multithreading in their code but IMS and CICS are highly scalable transaction monitors so this was less of a concern.

In terms of connection management, the Orbix CORBA implementation that was used has built-in possibilities to use the client disconnects and the server disconnects patterns. Essentially this means that a client will disconnect when finished communicating with the IMS/CICS Adapters and the Orbix Adapters will close connections to the clients when they are finished. IMS and CICS Transactions are typically short running and do not keep connections open so this was not a problem in this case.

An in-house implementation of the concentrator pattern was developed and ensured that the number of simultaneous connections to the Naming Service does not exceed the limits.

Credit Suisse performed some scalability tests where there was one CORBA IMS/CICS Adapter scaling up to 1500 concurrent clients. There could be 6500 clients distributed over 10 Adapters. One Unix System Services image on OS/390 could run 20 adapters so it could handle 20000 concurrent clients. This amount of concurrent clients was deemed sufficient to provide for the users of the bank applications and so the required scalability was reached.
Credit Suisse also did scalability tests outlining the number of requests per second that can be processed. These tests used a sample application on one IMS-TM system with 10 Message Processing Regions in IMS. This processed 70 CORBA requests per second.

This could be scaled to the limits of IMS meaning several hundred requests per second or several thousands in a larger IMS installation or with newer, more powerful hardware in the future.

Again, these figures underlined the continued scalability of Credit Suisse's legacy systems when made peers in a CORBA based distributed network. One of my roles in this phase of the project was to determine the optimum usage of IONA IMS and CICS Adapters for maximum scalability. The following techniques were used.

- Testing Adapter connection management capabilities to determine after how many concurrent client requests per adapter did scalability and performance degrade
- Determining the ideal number of threads per Adapter for optimum scalability and performance.

By the end of the implementation of this phase of the project, we found that the results outlined above most accurately reflected the ideal scalability capability of the CORBA Infrastructure.

11.8 Availability

As per the discussion on Availability, the major question posed to Credit Suisse was "how could client reach a service". Also, as proposed in this research, probably the easiest way is to use the Naming Service Pattern.

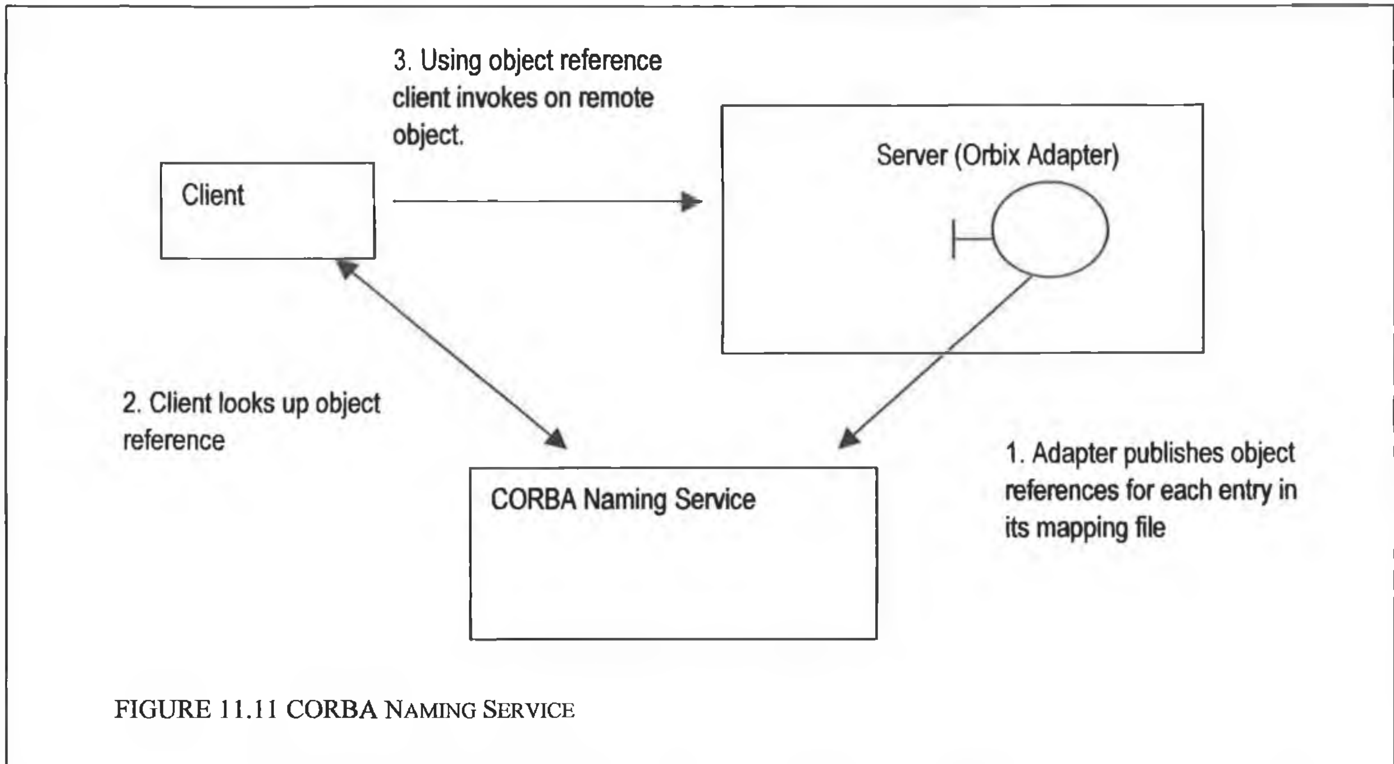


FIGURE 11.11 CORBA NAMING SERVICE

Credit Suisse used the CORBA Orbix Naming Service that maps readable names to the IOR for each Service. For example, a client could bind to the Naming Service and ask for *Services.Customer.Customer_1_0* and would be returned the IOR for this Service.

In addition, Naming helper-classes were provided by our team, for developers to simplify even further the process of locating the object required. These Helper classes were provided for all programming languages applicable (Java, C++, PL/I, COBOL, Smalltalk) and just provided the **bind() and resolve()** functions that come with the Naming Service API. This took all the complexity away from client side object location.

As per the last section on scalability, I performed some tests on the CORBA Naming Service to see how many concurrent clients could be accepted before performance and scalability degradation were seen. In this case, we saw no major problems right up to the TCP/IP limit of 1024 simultaneous clients. After this point, no more connections could be opened an exceptions were thrown back to the client. However, because client interaction with the Naming Service is short-lived, this limit was never actually reached in practice.

11.9 Failover

Partial failure can be a problem in that a CORBA call can fail with a status that does not allow you to determine what really happened (for example when a client sees the COMPLETION_MAYBE exception status). Partial failure requires you to take provisions in your system design, or to provide very sophisticated recovery mechanisms. In some cases even manual operator intervention may be required.

Credit Suisse used the following design principles:

- Make update operations in servers repeatable so that an update can be repeated as often as needed without comprising data integrity
- Given the current mechanisms, use distributed transactions only within a tightly coupled environment with reliable connections between the resource managers. This results in a minimum chance of message loss, and controlled scope of possible repair
- Build application control loops wherever data integrity is crucial to the business

Credit Suisse also made use of the Replication pattern to ensure that the Naming Service was not a single point of failure. Having a replicated Naming Service helped ensure that clients could always get the Object References of the Services they required.

In reality there was a Master/Slave solution where the Main (Master) Naming Service had a backup called the Slave who would become available only when the Master went down or became unavailable. If the Slave also went down there would be no Naming Service so the System Administrators would have to work on getting the Master up again as soon as possible.

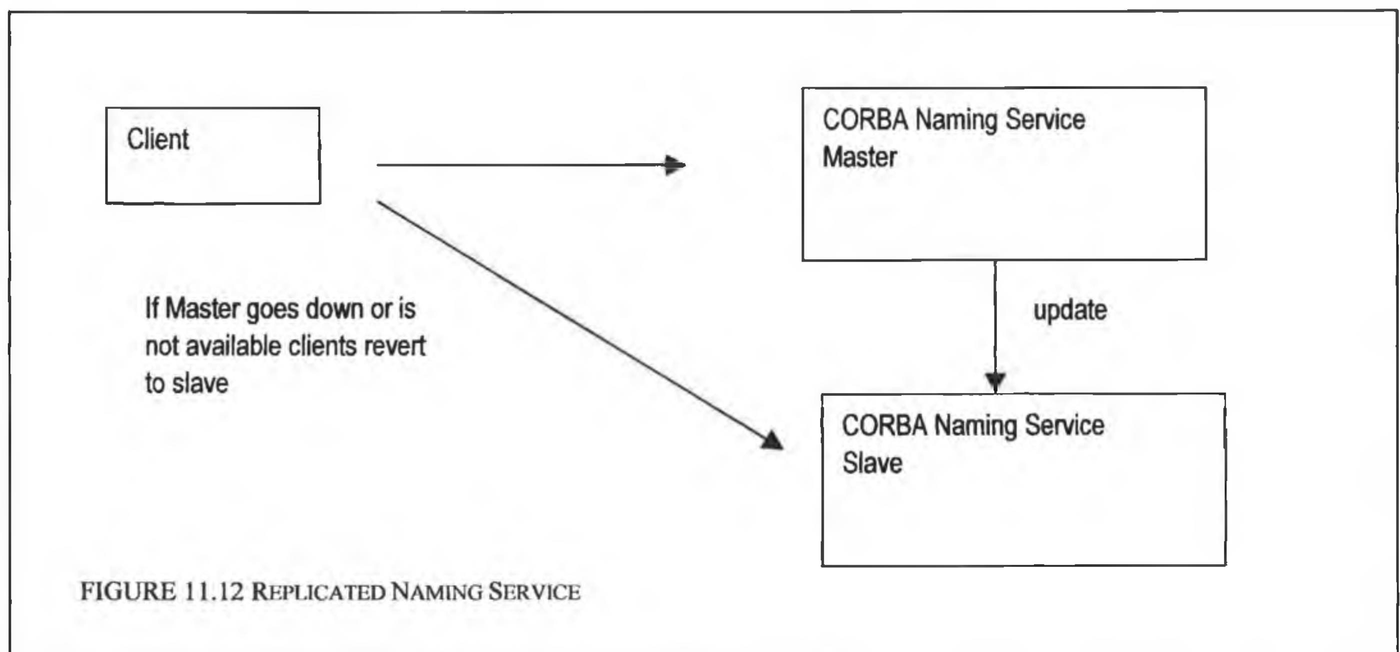


FIGURE 11.12 REPLICATED NAMING SERVICE

The Load-Balancing pattern also ensured both enhanced performance and reliability for the CORBA infrastructure. Each Service was load balanced over three of four CORBA Adapters in a round robin manner so that in the event of an increased load, no one adapter could become overloaded.

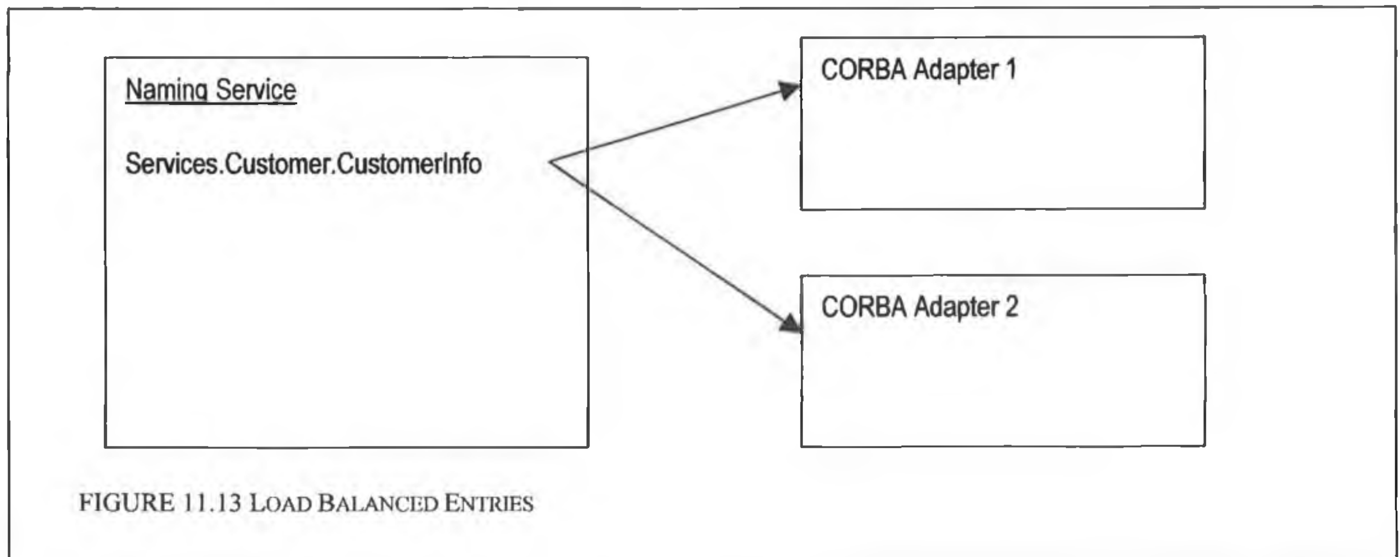


FIGURE 11.13 LOAD BALANCED ENTRIES

In this example, each entry in the Naming Service actually has two different IORs that point to different CORBA IMS/CICS Adapters. Each client request asking for this Service will get an alternate IOR so that the load for the Customer Service is spread evenly between both adapters. In reality this can be extended over many adapters if the load increases.

11.10 Other Idioms and Useful Solutions

There were various other solutions used by Credit Suisse that are not patterns in the correct sense but rather idioms and rules of thumb that made the CORBA-Mainframe Integration easier to maintain.

- Do not change an interface in production. The IDL interface is the contract between client and server and if this changes on either side there can be serious consequences.
- Introduce a versioning convention for IDL interfaces. There should be a major version number and a minor version number. If an operation or attribute is changed or deleted a new major version must be created. If the interface is extended a new minor version can be created.
- At any time up to three versions of an IDL interface must be supported. It can be quite a task for each client to upgrade if a Service is changed so there must be an overlap to allow this to happen over time.
- An IDL must never contain parameters than refer to the location of an object or service. This should always be kept transparent.
- Several conventions about the names of interfaces, modules, operations and types ensure a standardised look and feel and help to avoid misinterpretation.

11.11 Credit Suisse CORBA Infrastructure

The following diagram shows the sequence of events for each CORBA Client request to a CORBA Service implemented on the mainframe (where the Service is implemented using PL/1 in IMS)

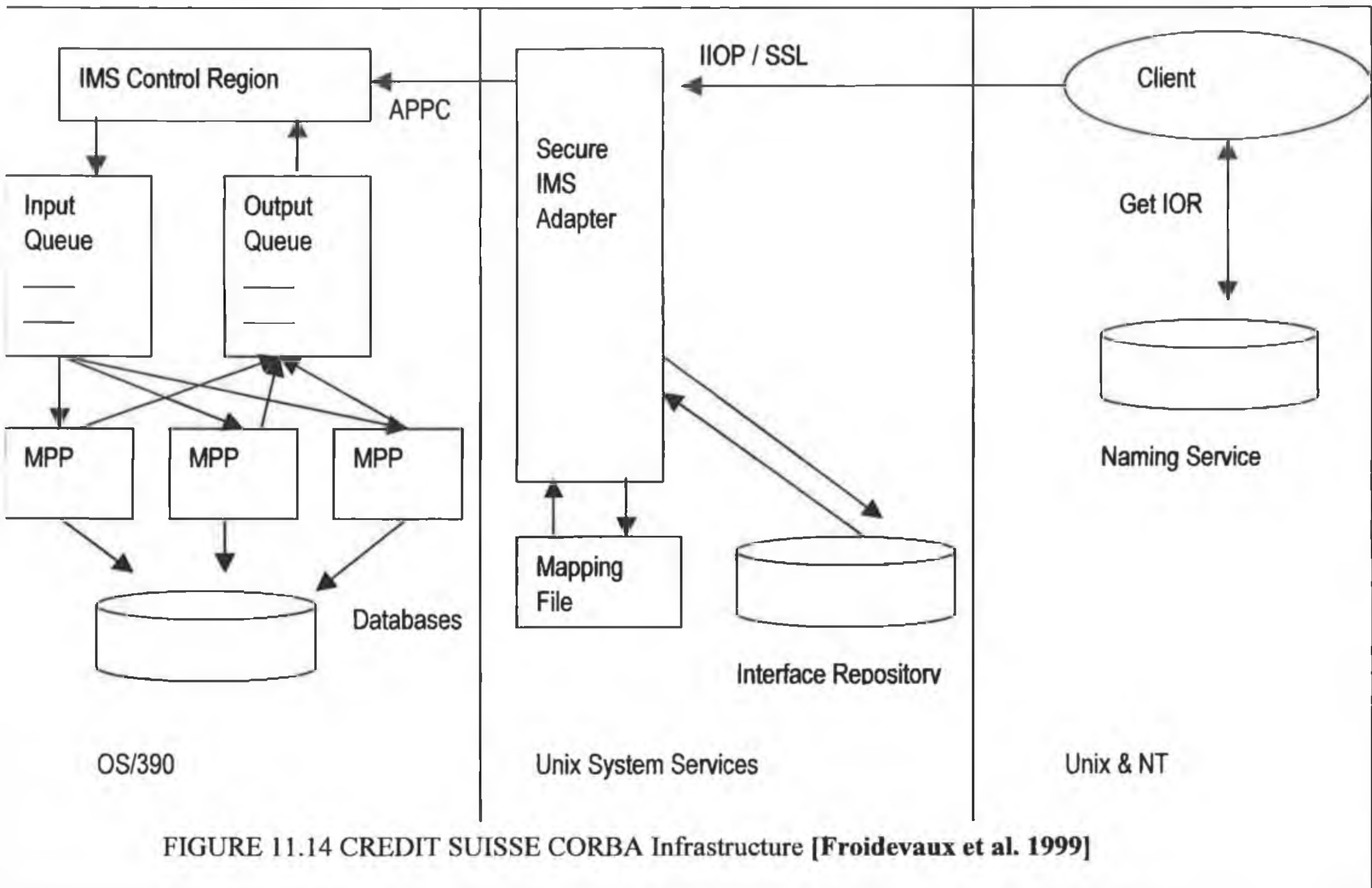


FIGURE 11.14 CREDIT SUISSE CORBA Infrastructure **[Froidevaux et al. 1999]**

A distributed client issues a request to the IMS Adapter. The IMS adapter is implemented using DSI as a dynamic server that reads the interface information at start-up from the interface repository. The request is assigned to the DSI object implementing the requested operation. The mapping of Message Processing Programs (MPPs) to interfaces and operations is provided in a mapping file. This mapping file belongs to the Orbix IMS/CICS Adapter.

Inside IMS, the operation gets a free LU6.2 (APPC) session from the session pool and sends the request to the Control Region. The Control Region receives the request, stores it in a request input queue from which it is read by an MPR (Message Processing Region). The Object Adapter, which is the entry point of the MPP (Message Processing Program), reads and demarshals the GIOP request and builds up a PL/I-compliant data structure which reflects the input parameters of the operation.

At this point, the application routine is called, which processes the request. The application routine reads and writes to the data structures by calling functions from the Orbix POD Library.

After the application program has terminated, the output parameters are marshalled and returned as GIOP response to the CR and to the Orbix IMS Adapter which translates the response to an IIOP response and returns it to the client

11.12 Conclusion

Credit Suisse faced many of the challenges that are faced by every company using older technology in the financial industry. However, they were a little more radical than others in their choice of integration strategies were. Rather than opting for a quick fix solution, they chose a path that is at first more difficult but which is expected to pay off in the longer term.

Choosing CORBA resulted in having a standards-based solution in place that can outlive one vendors' product set. In addition, there are many industry wide patterns and solutions that Credit Suisse could learn from other companies in the same situation and apply them in this case.

This section outlined an approach to integrating an existing legacy system using CORBA and a Service-Based Architecture. In addition, it tackled those topics outlined earlier in the research as being possible areas of trouble in an enterprise situation.

In the original aims and objectives of this research, finding and overcoming possible problems with integration projects involving mainframe based legacy systems and their modern Object Oriented counterparts was outlined as the number one objective. The last few sections located these problem areas and this section outlines an approach to overcome these problems.

Some Advantages of using this approach include

- The Standards Based solution results in industry-wide consensus on how best to perform certain roles.
- A Managed Evolution allows a migration in smaller easier steps
- Using A Service Architecture results in a component-based Architecture so that individual Services can migrate to newer technologies at their own pace
- Various performance patterns ensure less degradation of mainframe application performance when it becomes a peer in a distributed network.
- The various layers in the Security Model allow the mainframe to be opened up to the wider world with less risk of attack or compromise of its data
- The use of various CORBAServices help overcome Availability and Failover issues
- Making full use of Object Oriented Technologies results in better code, enhanced scalability, and helps ensure that a similar integration headache does not arise in 10 years time.

Essentially, we have realised the objectives and aims outlined at the beginning of the research in that the approach outlined above allows mainframe applications to gradually move towards newer technologies whilst avoiding certain pitfalls and dangers.

Any further problems encountered by using the approaches outlined in this section will be looked at in the section on further research possibilities.

12 Future

12.1 The Future of Mainframes

Predictions over the last ten years and certainly before the turn of the millennium indicated that mainframes were finished as operating systems. People were of the opinion that this was a technology that was too old and too difficult to use and maintain to have any real future. However, while these predictions are still with us, large enterprises and especially the financial sector are still buying mainframes. And today this operating system seems more important to the financial sector than ever before.

As we saw from the first sections of this research, e-business is placing new demands on a banks IT infrastructure, in terms of demanding highly scalable, centralised solutions. The strange thing in many peoples eyes is that the S/390 platform and the newer z/Architecture are attracting renewed attention from IT managers who are looking for a technology that is both proven and built on the latest technology.

The next phase of mainframe computing comes with the z/OS platform and the z/Series processors. This is essentially the next release of the OS/390 mainframe except it now has 64-bit addressing support and advanced workload management. These systems have the ability to host thousands of Linux images under VM and for many larger companies this is a platform they want to be tied into over the next decade.

From a purely technical perspective the z/Architecture brings lots of new functionality and new potential and lots of independent software vendors are trying to bring their technology to mission critical applications on the host. Any CORBA or J2EE conference these days will have at least some mainframe integration theme and with the advent of the very newest technologies such as Web Services, SOAP, UDDI and WSDL; there is already talk of how we integrate the mainframe with this technology.

From an e-commerce perspective, it seems likely that the massive demands of such network centric applications will place increasing demands on the enterprise-servers of tomorrow. It seems that the mainframe and especially the new z/architecture are particularly well placed to meet these requirements.

12.2 The Future of Legacy Applications

Many of the enterprises that use CORBA to re-engineer or rewrite their legacy systems have now made these available to the rest of their distributed computing environment. However, the core functionality behind these applications has not changed, and whether in this form or written in another programming language on another system, will not change in the foreseeable future.

For sure there will be additional requirements by banks over the next decade and there will be applications built to meet these requirements. This time however, there will be one eye looking into the future to ensure that today's applications in as far as is possible don't become obsolete,

12.3 The Future of Distributed Computing

For almost 10 years, CORBA has been the standard of choice for large organisations wishing to integrate heterogeneous applications on distributed systems using different programming languages. For many, CORBA was a little over-complicated and CORBA-based systems almost always ran over budget and into more complications that they envisaged. The alternatives in today's market include J2EE, MQSeries, and Microsoft's .NET

J2EE (Java 2 Enterprise Edition) is the very latest in Java technology and includes in its specification such components as Enterprise Java Beans and J2EE Connectors. Both of these approaches have a distributed computing element to them and the J2EE Connectors specification is looking at integrating legacy applications in COBOL and PL/I with Java Application Servers and Java Clients. The Connector technology is not mature yet and Java as a mainframe language is only now gaining respectability.

MQ Series is IBM's solution for asynchronous messaging on the host. MQSeries can easily be integrated with CICS and IMS and also with Java Clients. It has a guaranteed one-time delivery of each message and is probably the most popular solution of asynchronous messaging. It is not really a competitor of CORBA as it fills a different market space.

.NET is Microsoft's end-to-end product offering. However, as yet there is no mainframe presence for this product suite.

One of the problems with standards based solutions is that if one of the major players takes it over and puts its own thumbprint on the standard - the other players in the market will look at alternative solutions and the original standard starts looking more like a proprietary solution.

The future of distributed computing looks like there will be an offering from the major players in this market including IBM, Sun and Microsoft for the foreseeable future. The new WebServices technology looks like a way of integrating this Websphere, SunOne and .NET technologies.

However, for mainframe integration solutions it seems that the only offerings that will be viable in the future will be IBM's various integration strategies and CORBA mainframe implementations such as the ASP OS/390 product suite from IONA Technologies.

13 Conclusion and Further Work

13.1 General Work for the future

In the last section we considered the future of mainframes, their legacy applications, and the distributed systems of which they can now be peers. We must now look at further work and research that could be continued in this area.

The most obvious way to approach this is to look at the areas where the solution we outlined in this research did not meet the requirements or was lacking in some respects

- Some think that CORBA application programming and design is too difficult. This can be especially true for PL/I and COBOL programmers who need to quickly get and understanding of Object Oriented concepts
- The success of the CORBA Standard is dependent upon industry implementations. On the mainframe platform, only IONA Technologies have both PL/I and COBOL implementations. Having just one company providing this implementation takes away somewhat from the concept of a Standards Based solution.
- Having many extra layers to ensure Security and Failover is of the highest degree in an enterprise solution can effect performance adversely
- Having a solution such as CORBA in place with IDL definitions for a Service does by no means guarantee that the Service will be well written and without bugs. This age-old problem is something that continues to exist.
- Other problems are still occurring on a daily basis for which no patterns have yet been implemented as a solution.

Despite the problems that can still exist using a CORBA based Service Architecture for enterprise integration projects, it is still a preferred solution to re-writing all mainframe applications from scratch. It is certainly less risky and less expensive.

Some of the areas in which further research could enhance even further the approach outlined in this thesis include

- Developing new patterns and general solutions to well known industry problems
- Enabling CORBA solutions to easily communicate with EJB, MQ Series or .NET solutions (as the new Web-Services will eventually allow).
- As newer technology and business requirements appear, these should be incorporated into existing standards in a uniform and timely manner.
- The advent of CORBAFacilities specialising in the Financial Sector could result in more off-the-shelf components and Services that could be added to an enterprise integration solution without much overhead

Even though CORBA is now considered one of the more mature distributed computing approaches in the market, there is further work that can be done to ensure that the Financial Institutions using CORBA today to integrate their legacy systems do not need themselves to be integrated in 10 years time.

13.2 Specific Research Possibilities

The previous section details the "general" problems that exist in the Financial Industry during the migration of legacy applications. However, there have also arisen some more specific research possibilities. These have come about as a result of the implementation's we did in Credit Suisse and include:

- Refine and investigate clearer and more applicable methods of applying Object-Oriented standards such as the CORBA specification to non-Object Oriented languages such as PL/I and COBOL. We found many problems applying such concepts in this project.
- Define further standards-based CORBA Services to avoid the reliance on in house tools for Systems Management.
- Look at other solutions in the industry that would be more lightweight and less complex for those simpler Services in the bank.
- Even after applying the various performance enhancement techniques, we still found that using CORBA can add an overhead not present other solutions. Further investigations in this area are required.
- Despite using a standards-based solution, there are still many difficulties in getting ORB implementations from different vendors to interoperate. Some further work investigating ORB Interoperability could alleviate this problem.
- The area in which our team experienced most of its troubles during this project was on the Fault-Tolerance side. Actually having a real-world solution that implements actual Master/Slave concepts without problems was something that eluded us for quite some time. Some further investigations into this area would provide real benefit to the project outlined here and other similar projects.

It is the opinion of this author that CORBA is still the best solution on the marketplace for those enterprise Financial Institutions wishing to integrate their mainframe systems. However, any such project will still have a high cost both in money and time terms. On the plus side, those Institutions that go down this route will be better placed for future integration solutions and technologies that will come as they are more likely to be able to integrate with today's modern technologies.

Appendix A: Web Resources

Computer Associates ACF2	http://www3.ca.com/Solutions/Product.asp?ID=147
IBM Linux	http://www-1.ibm.com/servers/eserver/zseries/os/linux/
IBM MQSeries Product Suite	http://www-3.ibm.com/software/ts/mqseries/
IBM MVS	http://www-1.ibm.com/servers/s390/os390/
IBM Resource Access Control Facility	http://www-1.ibm.com/servers/eserver/zseries/zos/racf/racfhp.html
IBM Transaction Processing Facility	http://www-4.ibm.com/software/ts/tpf/index.html
IBM Unix System Services	http://www-1.ibm.com/servers/eserver/zseries/zos/unix/
IONA Technologies Products	http://www.iona.com
J2EE Connector Architecture	http://java.sun.com/j2ee/connector/
Java Remote Method Invocation	http://java.sun.com/products/jdk/rmi/
Microsoft DCOM	http://www.microsoft.com/com/tech/DCOM.asp
OMG CORBA Specifications	http://www.omg.org/technology/documents/spec_catalog.htm
SSL v3.0 Specification	http://wp.netscape.com/eng/ssl3/
W3C XML Specification	http://www.w3.org/XML/
Xtradyne Domain Boundary Controller	http://www.xtradyne.de/products/boundary.htm

Bibliography

[Alexander 1977]
Christopher Alexander. *A Pattern Language.* Oxford University Press, 1977.

[Alexander 1979]
Christopher Alexander. *The Timeless Way of Building.* Oxford University Press, 1979

[Alireza, et al. 2000]
A. Alireza, U. Lang, M. Padelis, R. Schreiner, M. Schumacher. *The Challenges of CORBA Security.* Proceedings of the Workshop Sicherheit in Mediendaten. Gesellschaft für Informatik (GI) 2001.

[Ballintijn, et al. 2000]
G. Ballintijn, M. van Steen, A.S. Tanenbaum. *Scalable Naming in Global Middleware.*
Proceeding of the 13th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-2000), Las Vegas, August 8-10, 2000, ICSA 2000.

[Beck Cunningham 1987]
Kent Beck, Ward Cunningham. *Using Pattern Languages for Object-Oriented Programs.*
OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming. ACM Press 1987.

[Becla, Gaponenko 2001]
J. Becla, I. Gaponenko. *Optimising Parallel Access to the BaBar Database System Using CORBA Servers.* CHEP'01 (Computing in High Energy and Nuclear Physics), Beijing, China, 2001. CHEP 2001.

[Bennett 1995]
K.H.Bennett. *Legacy Systems: Coping With Success.* IEEE Software, January 1995, Vol. 12,No.1.

[Bennett, Kannenberg 1996]
Cedric Bennett and Margo Kannenberg. *Student Transactions via the Web.*
Presented at the 1996 CAUSE annual conference "Broadening Our Horizons : Information, Services, Technology". CAUSE 1996.

[Beznosov, Deng, Blakely 1999]
Konstantin Beznosov, Yi Deng, Bob Blakely. *A Resource Access Decision Service for CORBA-based Distributed Systems.* 15th Annual Computer Security Applications Conference December 1999, Phoenix, Arizona, IEEE CS 1999.

[Brodie Stonebraker 1995]
Michael L.Brodie, Michael Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach.* Morgan Kaufmann Series in Data Management Systems, 1995.

[Carzaniga et al. 1999]
Antonio Carzaniga, David S. Rosenblum, Alexander L.Wolf. *Challenges for Distributed Event Services : Scalability vs. Expressiveness.* From the Proceedings of the ICSE '99 Workshop on Engineering Distributed Objects (EDO '99), Los Angeles, CA, May 17-18, 1999, IEEE CS 1999.

[Chan 1998]
Charles Quoc Cuong Chan. *Tolerating Latency in Software Distributed Shared Memory Systems through non-binding prefetching.* Thesis, Degree of Master of Science, Graduate Department of Computer Science, University of Toronto 1998.

[Chang 2000]
Chi-Chao Chang. *Safe and Efficient Cluster communication in Java using explicit memory management.* Degree of Doctor of Philosophy Thesis, Graduate School, Cornell University January 2000.

[Chen et al. 2000]
Li Chen, Rossi G. Marinina. *Banking Merger 2000.* Masters Thesis, Industrial and Financial Economics, Gothenburg University 2000.

[Chung et al. 1998]
P. Chung, Y Huang, S. Yaknik, D. Liang and J. Shih. *DOORS : Providing fault tolerance to CORBA objects.* Poster session at IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98) The Lake District, England, Springer 1998.

[Claesson 2001]
Per Claesson. *Integrating CORBA functionality within an existing COM architecture.* Master thesis in Computing Science, Chalmers Tekniska Hogskola, Gothenburg 2001.

[Clerc 1999]
Vincent Clerc. *UNICIBLE Presentation, IONA World Conference 1999.* Available from http://www.iona.com.

[Coplien et al 1996]
James Coplien, Michael Adams, Robert Gamoke, Robert Gammer, Fred Keeve, Keith Nicodemus. *Fault-Tolerant Telecommunications System Patterns.* In Vlissides, J.M., J.O. Coplien, and N.L. Kerth. (eds.) *Pattern Languages of Program Design 2.* Addison-Wesley, Reading, Mass. 1996.

[Coyle 2000]
Frank P. Coyle. *Legacy Integration - Changing Perspectives.* IEEE Software Magazine March/April 2000.

[Cukier et al. 1998]
M. Cukier, J. Ren, C. Sabnis et al. *AquA : An Adaptive Architecture that Provides, Dependable Distributed Objects.* Proceedings of the 17th Symposium on Reliable Distributed Systems (SRDS-17) West Lafayette, IN 1998, IEEE CS 1998.

[Curtis 1997]
David Curtis. *Java, RMI and CORBA.* A white paper prepared by David Curtis, Director of Platform Technology Object Management Group.
Available from http://www.omg.org/library/wpjava.html.

[Davis Gamble 2001]
L.Davis and R.Gamble. *Conflict Patterns : Towards Identifying Suitable Middleware.* Int'l Conference on Information Reuse and Integration, Las Vegas, NV, 2001. IEEE CS Publishing 2001.

[DSRG 1999]
Distributed Systems Research Group, Charles University, Prague
CORBA Comparison Project. Technical Report, 1999.
Available from http://nenya.ms.mff.cuni.cz/projects/corbac/Report_0899.pdf

[Erlikh, Goldbaum 2001]
Len Erlikh and Lisa Goldbaum. *EAI's Missing Link: Legacy Integration.* EAI Journal, April 2001.

[Ezhilchelvan et al. 2001]
Paul Ezhilchelvan, Mohammad-Reza R.Khayyambashi, Doug Palmer, Graham Morgan.
Measuring the Cost of Scalability and Reliability for Internet-based, server-centred applications.
Sixth 35 International Workshop on Object-oriented Real-time Dependable Systems (WORDS01), Rome, Jan. 2001, IEEE CS Publishing 2001.

[Falkner 2000]
Katrina Elizabeth Falkner. *The provision of relocation transparency through a formalised naming system in a distributed mobile object system.* Doctor of Philosophy Thesis, Department of Computer Science, University of Adelaide. 2000.

[Felber 1998]
P. Felber. *The CORBA Object Group Service : A Service Approach to Object Groups in CORBA.* Ph.D. thesis, Ecole Polytechnique Federale de Lausanne, 1998.

[Felber et al. 1996]
P. Felber, B. Garbinato, R. Guerraoui. *The Design of a CORBA Group Communication Service.* Proceedings of the 15th Symposium on Reliable Distributed Systems, Canada 1996, IEEE Publishing 1996.

[Ferguson 2002]
Roger W. Ferguson JR.
Speech to the Federal Reserve Board by Vice Chairman Roger W. Ferguson JR.
March 2002. Available from http://www.federalreserve.gov/boarddocs/speeches/2002/default.htm

[Foote Yoder 1997]
Brian Foote and Joseph Yoder. *Big Ball of Mud.*
Fourth Conference on Patterns Languages of Programs (PLoP '97/EuroPLoP '97)
Monticello, Illinois, September 1997, Addison-Wesley 1997.

[Froidevaux et al. 1999]
Werner Froidevaux, Stephan Murer, Martin Prater. *The Mainframe as a High-Available, Highly Scalable CORBA Platform.* Published at the International Workshop on Reliable Middleware Systems, October 19, 1999 In Conjunction with the 18th IEEE International Symposium on Reliable Distributed Systems, IEEE CS Publishing 1999.

[Frölich, Gal, Franz 2002]
Fröhlich, Gal, Franz. *On Reconciling Objects, Components, and Efficiency in Programming Languages.* Technical Report No. 02-12 Department of Information and Computer Science University of California, Irvine, USA March 2002.

[Gamma 1995]
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns : Elements of Reusable Object Oriented Software.* Addison-Wesley 1995.

[Glynn 1996]
Davies Glynn. *A history of money from ancient times to the present day.* University of Wales Press 1996.

[Gokhale, Schmidt 1998]
Aniruddha S. Gokhale and Douglas C. Schmidt. *Measuring and Optimising CORBA Latency and Scalability Over High-speed Networks.* Appeared in a special issue of IEE Transaction on Computers, Vol. 47, No 4. April 1998.

[Gokhale, Schmidt 1997]
Aniruddha S. Gokhale and Douglas C.Schmidt. *Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks.* Appeared in the Proceedings of ICDCS'97 (May 97 Baltimore Maryland), IEEE CS Publishing 1997.

[Goldberg, Robson 1989]
Goldberg and Robson. *Smalltalk-80 : The Language and its Implementation.* Addison-Wesley, Reading, MA, USA.

[Grahn, Holgersson 2002]
Hakan Grahn and Marcus Holgersson. *An approach for performance measurements in distributed CORBA applications.* 20th IASTED International Conference on Applied Informatics, symposium on Parallel and Distributed Computing and Networks, pages 326-337, Innsbruck, Austria, ACTA Press 2002.

[Gueheneuc Juissen 2001]
Yann-Gael Gueheneuc and Narendra Juissen. *Using explanations for design patterns identification.* IJCAI 2001 Workshop on Modelling and Solving problems with constraints, Morgan-Kaufmann 2001.

[Hagen 2000]
Claus Hagen. *Credit Suisse IT Architecture.* Presentation at ETH Zürich
Available at : http://www.inf.ethz.ch/personal/iks/Other/PDDBS/pdf/Hagen-PDDBS.pdf

[Harding 2001]
Elizabeth U. Harding. *Programmer Shortage Threatens Mainframe Future.* Software Magazine April 2001.

[Henning 1999]
Michi Henning. *Binding, Migration, and Scalability in CORBA.*
Communications of the ACM Journal, volume 41, number 10, Oct 1998.

[Henning, Vinoski 1999]
Michi Henning, Steve Vinoski. *Advanced CORBA Programming with C++.* Addison Wesley, February 1999.

[Hermansson, Akerlund 1997]
Therese Hermansson, Malin Akerlund. *EJB - A Deployment Evaluation.* Masters Thesis, University of Umea, Sweden, 1997.

[Hoon et al. 2001]
Gan Keng Hoon, Chan Huah Yong and Fazilah Haron. *Load Balancing for Web-based Grid Application.* Asia Pacific Advanced Network Consortium (APAN) Meeting 2001, Asia-Pacific Advanced Network 2001.

[Horswill 2000]
John Horswill and the member of the CICS Development Team at IBM Hursley. *Designing and Programming CICS Applications.* O'Reilly 2000.

[Jacobsen et al. 1997]
E. Jacobsen, B Kristensen, P Nowack. *Patterns in the Analysis, Design and Implementations of Frameworks.* In Proceedings of the Twenty-First Annual International Computer Software and Application Conference, (COMPSAC'97), Washington D.C., USA, 1997, IEEE CS Publishing 1997.

[Jiang 1998]
Qingli Jiangb. *Integration of Real-time Object-Oriented Database and Real-time CORBA into Legacy Software.* Thesis, Degree of Master of Science, Computer Science, University of Rhode Island. 1998.

[Johnson 1989]
Robert H. Johnson *MVS Concepts and Facilities*
Intertext Publications/McGraw-Hill Book Company 1989.

[Jordan 1996]
Jerry L. Jordan. *The Functions and Future of Retail Banking.*
Economic Commentary by Jerry L. Jordan, President and Chief Executive Officer, Federal Reserve Bank Cleveland, 1996. Available from http://www.clev.frb.org/ccca/jj100196.htm.

[Juric et al. 1999]
Matjaz B Juric, Ivan Rozman, Marjan Hericko, Tomaz Domajnko. *Integrating Legacy Systems in Distributed Object Architecture.* Proceedings of International Conference on Enterprise Information Systems, March 1999 Portugal, Kluwer Academic Publishers, 1999.

[Kahkipuro 1999]
Pekka Kahkipuro. *Performance Modelling Framework for CORBA Based Distributed Systems.* From Proceeding of UML 1999, Berlin, 1999, Springer-Verlag 1999.

[Keshav, Gamble 1998]
Keshave, Gamble. *Towards a Taxonomy of Architecture Integration Strategies.*
3rd International Software Architecture Workshop, November 1998, ACM 1998.

[Khandker et al. 1995]
A.M. Khandker, P. Honeymam, and T.J. Teorey. *Performance of DCE RPC.* 2nd International Workshop on Services in Distributed and Networked Environments June 05 - 08, 1995 Whistler, British Columbia, IEEE CS Publishing 1995.

[Kim Bieman 2000]
Hyeon Soo Kim, James M. Bieman. *Migrating Legacy Software Systems to CORBA based Distributed Environments through an Automatic Wrapper Generation Technique.* Proc. Joint meeting of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000) and the 6th International Conference on Information Systems Analysis and Synthesis (ISAS'2000), IIIS 2000.

[Koch, Murer 1999]
Thomas Koch, Stephan Murer. *Service Architecture Integrates Mainframes in a CORBA Environment.* Published at the third IEEE conf. On "Enterprised Distributed Object Computing" Sep. 27-30, 1999, IEEE CS Publishing 1999.

[Kolodziej 1987]
J.Kolodziej. *COBOL Shapes Up.* Computerworld Magazine, Vol 21, (Jan 7, 1987), p. 13-14.

[Korhonen 2001]
Jouni Korhonen. *Fault Tolerant CORBA.* Research Seminar on Real Time and High Availability University of Helsinki, Department of Computer Science November 2001
Available from http://www.cs.helsinki.fi/u/kraatika/Courses/sem01a/korhonen.pdf

[Kudrass et al. 1996]
Thomas Kudrass, Marco Lehmbach, Alejandro Buchmann. *Tool-Based Re-Engineering of a Legacy MIS: An Experience Report.* Proceedings of the 8th Intl. Conference, CAiSE '96, Heraklion, Crete (Greece), May 1996, Springer-Verlag LNCS 1080 1996.

[Kugel 2001]
Herb Kugel. *History of Computing : The IBM 650.*
Dr. Dobb's Computer Magazine Available at http://www.ddj.com/.

[LaLiberte Braverman 1999]
Daniel LaLiberte, Alan Braverman.
A Protocol for scalable group and public annotations. Proceedings of the 3rd International WWW conference Volume 27, Number 6, ACM Press 1999.

[Landis Maffeis 1997]
S. Landis, S. Maffeis. *Building Reliable Distributed Systems with CORBA.* Theory and Practice of Object Systems Journal Vol. 3, no 1. 1997.

[Lang 1997]
Ulrich Lang. *CORBA Security : Security Aspects of the Common Object Request Broker Architecture.* MSc in Information Security, Royal Holloway, University of London. 1996/1997.

[Lauder, Kent 2000]
Anthony Lauder and Stuart Kent. *Legacy Systems Anti-Patterns and a Pattern-Oriented Migration Response.* In: Henderson P, Systems Engineering for Business Process Change, Springer Verlag, 2000.

[Lawrence 1996]
Andrew Lawrence. *IBM System User International Survey.* Computer Business Review, p. 1-4. March 1996.

[Lea 1993]
Doug Lea *Christopher Alexander: An Introduction for Object-Oriented Designers*
An article available from http://gee.cs.oswego.edu/dl/ca/ca/ca.html.

[Levine, Gill, Schmidt 2000]
David L.Levine, Christopher D. Gill, and Douglas C. Schmidt. *A Complementary Pattern for Controlling Object Creation and Destruction.* C++ Report, SIGS, Vol. 12, No. 1, January, 2000.

[Levine Schmidt 2000]
Dr. David Levine, Douglas C. Schmidt.
Introduction to Patterns and Frameworks. Tutorial from the Department of Computer Science, Washington University, St. Louis Available from
http://www.cs.wustl.edu/~schmidt/PDF/patterns-intro4.pdf.

[Little 1999]
M.C.Little, S.K. Shrivastava
Implementing high availability CORBA applications with Java.
Appeared in the proceedings of the 1999 IEEE Workshop on Internet Applications, July 26 - 27, 1999. San Jose, California, IEEE CS Publishing 1999.

[Luomala 2000]
Vea Luomala. *CORBA Based Object Transaction Monitors.* Thesis, Master of Science in Engineering, Department of Information Technology, Helsinki University of Technology 2000.

[Maffeis, Schmidt 1997]
Silvano Maffeis, Douglas C. Schmidt. *Constructing Reliable Distributed Communication Systems with CORBA.* Appeared in the feature topic issue on Distributed Object Computing in the IEEE Communications Magazine, Vol. 14, No.2, February 1997.

[Marchetti, Mecella, Baldoni 2000]
Carlo Marchetti, Massimo Mecella, Roberto Baldoni. *Architectural Issues on Fault Tolerant CORBA* Proceedings of the SSGRR 2000 Computer & Business Conference, L'Aquila, Italy, 2000, Scuola Superiore 2000.

[Marchetti, Mecella, Virgillito, Baldoni 2000]
C. Marchetti, M. Mecella, A. Virgillito, R. Baldoni. *An Interoperable Replication Logic for CORBA Systems.* Proceeding of the 2nd International Symposium on Distributed Objects and Applications (DOA 2000) Antwerp, Belgium 2000, Springer-Verlag 2000.

[Marchetti, Virgillito, Mecella, Baldoni 2001]
Carlo Marchetti, Antonio Virgillito, Massimo Mecella, Roberto Baldoni. *Integrating Autonomous Enterprise Systems through Dependable CORBA Objects.* Published at the Proceedings of the 5th International Symposium on Autonomous Decentralised Systems (ISADS 2001), Richardson, Texas 2001, IEEE CS Publishing 2001.

[McCauley 1999]
Chris McCauley. *Under the covers and ILM, A CORBA System in Action.* A presentation by Chris McCauley, Senior Architect, Irish Life Investment Managers, IONA World Europe Conference 1999. Available from http://www.iona.com/.

[McDonough 1999]
Mr William J. Mc Donough. *Changing nature of banking, risk and capital regulation.*
29th Annual Banking Symposium, Bank and Financial Analysts Association New York City 1999, Federal Reserve System 1999.

[Modi 2000]
Tarak Modi. *Using Space-Based Programming for Loosely Coupled Distributed Systems.* Java Developer's Journal, October 2000.

[Morris, Isaksson 2002]
Rob Morris and Pete Isaksson. *Legacy within the Enterprise: Imagine the Possibilities.*
EAI Journal, March 2002.

[Moser et al. 1999]
L.E. Moser, P.M. Melliar-Smith, P. Narasimhna, L.A. Tewksbury, V.Kalogeraki. *The Eternal System : An Architecture for Enterprise Applications.* Proceedings of the 3rd International Enterprise Distributed Object Computing Conference 1999, Mannheim Germany, IEEE CS Publishing 1999.

[Mowbray, Malveau 1997]
Mowbray, Malveau. *CORBA Design Patterns.* Wiley& Sons 1997.

[Mungee, Surendran, Schmidt 1999]
Sumedh Mungee, Nagarajan Surendran, Douglas C. Schmidt. *The Design and Performance of a CORBA Audio/Video Streaming Service.* Appeared in the HICSS-32 International Conference on System Sciences, minitrack on Multimedia DBMS and the WWW, Hawaii, January 1999, IEEE CS 1999.

[Murer 1999]
Stefan Murer. *Why Does Credit Suisse invest in Orbix on MVS.* IONA World Europe Conference 1999 Presentation. Available from http://www.iona.com/.

[Narasimhan et al. 1997]
P. Narasimhan, L.E. Moser, P.M. Melliar-Smith. *The Interception Approach to Reliable Distributed CORBA Objects.* Published at the Proceedings of the Third USENIX Conference on Object-Oriented Technologies and Systems Portland, Oregon, June 1997, USENIX Association 1997.

[Natarajan et al. 2000]
Balachandran Natarajan, Aniruddha Gokhale, Shalini Yajnik, Douglas C.Schmidt. *Applying Patterns to Improve the Performance of Fault Tolerant CORBA.* Submitted to the 7th International Conference on High Performance Computing, Bangalore India, Dec 2000, ACM/IEEE 2000.

[O'Ryan et al. 1999]
Carlos O'Ryan, David L.Levine, Douglas C.Schmidt, J. Russell Noseworthy. *Applying a Scalable CORBA Events Service to Large-scale Distributed Interactive Simulations.* Proc. IEEE 5th Workshop on Object-Oriented Real-Time Dependable Systems, Los Alamitos, Calif., Nov. 1999, IEEE CS Press 1999.

[ORBOS 1998]
Proposal to the ORBOS Platform Task Force for Benchmarking CORBA Scalability.
Available from http://www.omg.org/docs/bench/98-10-01.doc.

[Othman et al. 2001]
Ossama Othman, Carlos O'Ryan, Douglas C.Schmidt. *The Design of an Adaptive CORBA Load Balancing Service.* IEEE Distributed Systems Online 2(4): (2001)

[Pan 2000]
Jiantao Pan. *Robustness Testing and Hardening of CORBA ORB Implementations.*
MS Thesis, Electrical and Computer Engineering Department, Carnegie Mellon University Pittsburgh, Pennsylvania, USA, 2000.

[Parikh, Girish 1987]
Parikh, Girish. *Making the Immortal Language Work.* Business Software Review, Vol 6, Iss 4, (April 1987), p. 33-36.

[Pyarali et al. 2000]
Ifan Pyarali, Carlos O'Ryan, Douglas C. Schmidt. *Patterns for Efficient, Predictable, Scalable, and Flexible Dispatching Components.* 7th Pattern Languages of Programs Conference (PLoP '00) in Allerton Park, Illinois, August 2000. Addison-Wesley 2000.

[Quinot et al. 2001]
Thomas Quinot, Fabrice Kordon, Laurent Pautet. *Architecture for a reusable object-oriented polymorphic middleware.* In Proceedings of PDPTA'2001, Las Vegas, Nevada, Etats-Unis, June 2001, CSREA Press 2001.

[Rackl 2000]
Günther Rackl. *Monitoring and Managing Heterogeneous Middleware.* PhD Thesis, Technical University Münich 2000.

[Reddy 2002]
Ram Reddy. *The Future of Enterprise Applications, Pieces of a whole.* Intelligent Enterprise Magazine (March 2002).

[Sang et al. 1999]
Janche Sang, Chan Kim, Isaac Lopez. *Developing CORBA-Based Distributed Scientific Applications from Legacy Fortran Programs.* Information and Software Technology, Vol. 44, Issue 3, 175--184, 2002.

[Scallan 2000]
Todd Scallan. *Monitoring and Diagnostics of CORBA Systems : Demystifying the CORBA communication bus to enable 'distributed debugging'* Java Developers Journal June 2000.

[Schmidt 1999]
Doug Schmidt. *Strategised Locking, Thread-safe Interface, and Scoped Locking : Patterns and Idioms for Simplifying Multithreaded C++ Components.* C++ Report, Volume 11, September 1999.

[Schmidt Stephenson 1995]
Douglas C. Schmidt, Paul Stephenson. *Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms* ECOOP '95 conference, August 1995, Springer-Verlag 1995.

[Schmidt et al. 1997]
Douglas C.Schmidt, Aniruddha Gokhale, Timothy H. Harrison, and Guru Parulkar. *A High-performance Endsystem Architecture for Real-time CORBA.* Appeared in the feature topic issue on Distributed Object Computing in the IEEE Communications Magazine, Vol. 14., No 2, February 1997.

[Schultz 2001]
Andreas Schultz. *Multi threading in a CORBA ORB.* Diplomarberit, "Otto-von-Güricke" University, Magdeburg 2001.

[Shen et al. 2000]
E-Kai Shen, Shikharesh Majumdar, Istabrak Abdul-Fatah. *High Performance Adaptive Middleware for CORBA-Based Systems.* Nineteenth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2000), Portland, Oregon, 16-19 July 2000, Springer-Verlag 2000.

[Silva et al. 2000]
Roberto Silva Filho, Jacques Wainer, Edmundo R.M. Madeira. *CORBA Based Architectures for Large Scale Workflow.* Masters thesis Institute of Computing - UNICAMP - State University of Campinas, Brazil, 2000.

[Slama et al. 1999]
Slama, Garbis, Russell. *Enterprise CORBA,* Prentice-Hall 1999.

[Smith, Williams 1998]
Connie U. Smith, Lloyd G. Williams. *Performance Engineering Models of CORBA-based Distributed-Object Systems.* Int. CMG Conference 1998, Computer Measurement Group 1998.

[Stroulia et al. 2000]
E. Stroulia, M.El-Ramly. P.Iglinski, P. Sorenson. *User Interface Reverse Engineering in support of Migration to the Web.* Automated Software Engineering 10(3): 271-301; Jul 2003.

[Stroustrup 1991]
B. Stroustrup and M. Ellis. *The Annotated C++ Reference Manual.* Addison-Wesley 1991.

[Stroustrup 1992]
B. Stroustrup. *The C++ Programming Language.* Addison-Wesley 1992.

[Summers 2000]
Bruce J. Summers. *Remarks of Bruce J.Summers : Director Federal Reserve Information Technology, Bank of Japan.* Conference on the Development of Information Technology and Central Banking, Tokyo Japan, October 2000.

[Szymaszek et al. 1998]
Jakub Szymasek, Andrzej Uszok, Krzysztof Zielinksi. *Building a Scalable and Efficient Component Oriented System using CORBA - Active Badge System Case Study.* Distributed Systems Engineering Journal 5(4): 203-213 (1998)

[TN3270 2001]
Internet Engineering Task Force : Telnet TN3270 Enhancements Special Working Group
Available from http://www.ietf.org/html.charters/tn3270e-charter.html

[Turner Brill 2001]
W. Pitt Turner and Kenneth G. Brill. *Industry Standard Tier Classifications Define Site Infrastructure Performance.* Uptime Institute. 2001. Available from http://www.uptime.com

[Vinoski 1998]
Steve Vinoski. *New Features for CORBA 3.0.* Communications of the ACM Vol. 41, No. 10 October 1998.

[Vinoski 2000]
Steve Vinoski. *Scalability issues in CORBA-Based Systems.* Tutorial presented at 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services. October 17-20, 2000 Enschede, The Netherlands, Springer Verlag 2000.

[Völter 2000]
Markus Völter. *Server-Side Components - A Pattern Language.* EuroPLoP 2000 Conference (Fifth European Conference on Pattern Languages of Programs), 5 - 9 July 2000, Irsee, Germany, Addison-Wesley 2000.

[Wang, Schmidt, Levine 2000]
Nanbor Wang, Douglas C. Schmidt, David Levine. *Optimising the CORBA Component Model for High-Performance and Real-time Applications.* In `Work-in-Progress' session at the Middleware 2000 Conference, ACM April 2000.

[Weik 1961]
Martin H. Weik. *The ENIAC Story.* The Journal of the American Ordnance Association (Jan/Feb 1961).

[Wolff, Schmid, Völter 2001]
Eberhard Wolff, Alexander Schmid, Markus Völter. *Building EJB Applications - A Collection of Patterns.* Presented at the PloP 2001 Conference (8th Conference on Pattern Languages of Programs), September 11-15, 2001, Allerton Park, Monticello, Illinois, USA, Addison-Wesley 2001.

[Yoder, Baraclow 1997]
Joesph Yoder, Jeffrey Baraclow. *Architectural Patterns for Enabling Application Security.* Presented at the PloP Workshop (Pattern Languages of Programs) 1997. September 3-5, 1997, Allerton Park, Monticello, Illinois, USA, Addison-Wesley 1997.