



A Framework for High Availability and Optimum System Performance in an Application Service Provider (ASP) Environment

M.Sc. Thesis

By

Kenneth Kirrane, B.Sc.

June 2006



Research Supervisor
Sean Duignan

To Sabrina... my soul mate, my inspiration, my friend. And to my family - for all their support and guidance.

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 Computing is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work. It has not been previously used to obtain any other award in this institution, or elsewhere.

Name of Candidate: Kenneth Kirrane

Student ID Number: 10007165

Signature of Candidate:

Date:

TABLE OF CONTENTS

<u>DEDICATION</u>	IV
<u>DECLARATION</u>	V
<u>TABLE OF CONTENTS</u>	VI
<u>ABSTRACT</u>	X
<u>ACKNOWLEDGEMENTS</u>	XI
<u>LIST OF FIGURES</u>	XII
<u>SECTION ONE – INTRODUCTION & RESEARCH FRAMEWORK</u>	
CHAPTER 1: INTRODUCTION & THESIS STRUCTURE	2
1.1 INTRODUCTION	2
1.1.1 APPLICATION SERVICE PROVIDER (ASP)	2
1.1.2 HIGH AVAILABILITY	3
1.1.2.1 FAILOVER	3
1.1.2.2 DISASTER RECOVERY	3
1.1.3 SCALABILITY	4
1.2 RESEARCH CONTRIBUTION	4
1.3 STRUCTURE OF THESIS	5
CHAPTER 2: RESEARCH BACKGROUND & METHODOLOGY	7
2.1 RESEARCH BACKGROUND	7
2.2 LIMITING FACTORS	11
2.3 RESEARCH METHODOLOGY	11
<u>SECTION TWO – LITERATURE & TECHNOLOGY REVIEW</u>	
CHAPTER 3: APPLICATION SERVICE PROVISION	14
3.1 ASP INTRODUCTION	14
3.2 ANALYST ASP PREDICTIONS	18
3.3 ASP ADVANTAGES	19
3.4 ASP GROWTH INHIBITORS	21
3.5 TYPES OF APPLICATION SERVICE PROVIDERS	22
3.6 CHAPTER SUMMARY	24
CHAPTER 4: HIGH AVAILABILITY AND SCALABILITY	26
4.1 HIGH AVAILABILITY	27
4.1.1 HIGH AVAILABILITY PLANNING	33
4.1.2 AVAILABILITY PARADIGMS	41

4.2	CLUSTERING	44
4.3	LOAD BALANCING	50
4.3.1	DNS LOAD BALANCING	51
4.3.2	HARDWARE LOAD BALANCING	52
4.4	SCALABILITY	53
4.4.1	SCALABILITY TYPES	54
4.4.2	SCALABILITY ARCHITECTURES	55
4.4.3	DESIGN PRINCIPLES FOR SCALABLE SYSTEMS	57
4.4.3.1	PRINCIPLE OF INDEPENDENCE	57
4.4.3.2	PRINCIPLE OF BALANCED DESIGN	57
4.4.3.3	PRINCIPLE OF DESIGN FOR SCALABILITY	59
4.5	DISASTER RECOVERY	60
4.5.1	FULL VS INCREMENTAL BACKUPS	61
4.5.2	COMMERCIAL DR SOFTWARE FEATURES	63
4.5.2.1	COMPLETE HARDWARE USAGE	63
4.5.2.1	HOT BACKUPS	64
4.5.2.2	OPEN TAPE FORMAT	64
4.5.2.3	CENTRALISED MANAGEMENT	64
4.5.2.4	QUICK DISASTER RECOVERY	65
4.5.2.5	HARDWARE SUPPORT AND FLEXIBILITY	65
4.5.2.6	MATURE PRODUCTS WITH REFERENCE SITES	65
4.5.2.7	MULTIPLE PLATFORM SUPPORT	66
4.5.3	SEVEN TIERS OF RECOVERABILITY	66
4.5.3.1	TIER 0 – NO OFFSITE DATA	66
4.5.3.2	TIER 1 - PICKUP TRUCK ACCESS METHOD (PTAM)	66
4.5.3.3	TIER 2 – PTAM & HOT SITE	67
4.5.3.4	TIER 3 - ELECTRONIC VAULTING	67
4.5.3.5	TIER 4 - ACTIVE SECONDARY SITE	68
4.5.3.6	TIER 5 - TWO SITE TWO PHASE COMMIT	68
4.5.3.7	TIER 6 - ZERO DATA LOSS	69
4.6	SINGLEVIEW AND BILLING4RENT	70
4.6.1	SINGLEVIEW COMPONENTS	72
4.6.1.1	CONVERGENT BILLING	72
4.6.1.2	CUSTOMER MANAGEMENT	73
4.6.1.3	FINANCIAL ASSURANCE	74
4.6.1.4	LIFECYCLE MANAGEMENT SUITE	74
4.6.1.5	COMMERCE ENGINE	75
4.6.2	SINGLEVIEW AVAILABILITY AND SCALABILITY	75
4.7	CHAPTER SUMMARY	76

SECTION THREE – RESEARCH CONTRIBUTION

CHAPTER 5: PROPOSED FRAMEWORK	79
5.1 BILLING4RENT FUNCTIONAL REQUIREMENTS	80
5.2 BILLING4RENT PROTOTYPE	82
5.3 B4R ADMIN INTERFACE	88
5.4 PROPOSED NETWORK ARCHITECTURE	100
5.4.1 B4R NETWORK DIAGRAM	100
5.4.2 ARCHITECTURE BREAKDOWN	101
5.4.2.1 FIREWALL_1	101
5.4.2.2 LOAD BALANCING SWITCH	102
5.4.2.3 HTTP SERVER	102
5.4.2.4 E-MAIL SERVER	103
5.4.2.5 DMZ	103
5.4.2.6 FIREWALL_2	104
5.4.2.7 APPLICATION SERVER	104
5.4.2.8 DATABASE SERVER	104
5.4.3 INTER-TIER TRAFFIC FLOWS	104
5.4.4 INTER-TIER TRAFFIC FLOW DETAILS	105
5.4.5 VERTICAL SCALING	106
5.4.6 BACKUP INTERNET CONNECTION	107
5.5 CODE OPTIMISATION FOR IMPROVED PERFORMANCE	108
5.5.1 MEMORY USAGE	108
5.5.2 SESSION MANAGEMENT	108
5.5.3 SERVLETS AND JAVA SERVER PAGES (JSPs)	109
5.5.4 LOGGING	109
5.5.5 ENTERPRISE JAVA BEANS	110
5.5.6 DATABASE ACCESS	110
5.5.7 GENERAL CODING CONSIDERATIONS	111
5.6 BILLING4RENT DISASTER RECOVERY	112
5.6.1 BILLING4RENT BACKUPS	112
5.6.1.1 OFFSITE DR LOCATION	113
5.6.2 DISASTER RECOVERY POLICIES	114
5.6.3 DISASTER RECOVERY PREVENTION	116
5.7 CHAPTER SUMMARY	117

SECTION FOUR – RESEARCH EVALUATION

CHAPTER 6: RESEARCH EVALUATION	119
6.1 FULFILMENT OF OBJECTIVES	119
6.1.1 BILLING4RENT ARCHITECTURE	120
6.1.2 MODEL/VIEW/CONTROLLER AND THE B4R PROTOTYPE	122

6.1.3	B4R AND CODE OPTIMISATION	124
6.1.3.1	MEMORY USAGE	124
6.1.3.2	SESSION MANAGEMENT	124
6.1.3.3	LOGGING	125
6.1.3.4	DATABASE ACCESS	126
6.1.3.5	GENERAL CODING CONSIDERATIONS	127
6.1.5	DISASTER RECOVERY	127
6.2	B4R PROTOTYPE AND ACCEPTED BENCHMARKS	130
6.2.1	ORACLE DATABASE	130
6.2.2	JAKARTA TOMCAT WEBSERVER	132
6.2.3	JAVA VIRTUAL MACHINE	132
6.3	OUTSOURCING B4R DEPLOYMENT	133
6.3.1	OUTSOURCING FEATURES & BENEFITS	135
6.3.2	B4R OUTSOURCING	135
6.4	CHAPTER SUMMARY	137
 CHAPTER 7: RESEARCH CONCLUSION		138
7.1	SUMMARY OF RESEARCH CONTRIBUTION	139
7.2	RESEARCH CONCLUSIONS	140
7.2.1	FUTURE RESEARCH POTENTIAL	142
 <u>BIBLIOGRAPHY</u>		144
 <u>LIST OF ACRONYMS USED</u>		152
 <u>APPENDIX ONE – RELATED RESEARCH</u>		155
RESEARCH PAPER PRESENTED AT THE <i>5TH ANNUAL INFORMATION TECHNOLOGY & TELECOMMUNICATIONS (IT&T) CONFERENCE, CORK, IRELAND, OCTOBER 2005, AND PUBLISHED IN THE CONFERENCE PROCEEDINGS</i>		
 <u>APPENDIX TWO – B4R PROTOTYPE SCREENSHOTS</u>		164
 <u>APPENDIX THREE – B4R PROTOTYPE CD-ROM</u>		193

Abstract

The Application Service Provision (ASP) model offers access to centrally managed and hosted applications, via the Internet, on a subscription / rental basis. From its inception in the late 1990s, many have heralded the ASP delivery model as the new wave of IT outsourcing. Initially however, uptake of the ASP paradigm failed to match the hype, due in no small part to the de-valuation of *dot.com* share prices in early 2000, which caused the closure of numerous on-line service providers, and which eroded the confidence of many with regard to renting application access. However uptake of the ASP model is steadily increasing (recent estimates indicate spending on software-as-a-service should reach \$15.2 billion by 2007) and the emergence of hosted applications from key industry players indicate that the delivery of software-as-a-service may become an increasing feature of software product releases. Presented herein, is a framework, including the architecture and system configuration required, to ensure High Availability and Optimum System Performance in an Application Service Provider (ASP) Environment. This thesis contends that High Availability and Scalability are becoming increasingly important components of overall system architectures. For many, application performance and cost-effective scalable platform configurations are emerging as the distinguishing features for software solutions. The function of most products is well understood, and well documented. The ASP innovation of hosting the application and making it available to end-users via the Internet, relies heavily on scalability and availability considerations in order to deliver a service with sufficiently high performance levels. This research project then, investigates the current state of Application Service Provision, and proposes a framework for Availability, Scalability and System Performance that can be used by ASPs going forward. Furthermore, an ASP prototype solution (Billing4Rent) provides the platform on which recommendations arising from this research were implemented and quantified.

Acknowledgements

I am very grateful to many people who have helped me over the course of this M.Sc. I would particularly like to express my gratitude to my supervisor Sean for his input, guidance, motivation and support throughout the duration of this research. I also wish to thank Gabriel Hicks, and the rest of the lecturers and technicians in the Computing & Maths department for their help and encouragement, who have not only assisted in this work but who have made working in GMIT a pleasure. I would also like to acknowledge my family and friends for all their continued support, especially Stephen for his meticulous proof-reading (minus the toothcomb). Last, but certainly not least, I thank Sabrina – for everything.

List of Figures

Figure 3.1 – Essential ASP Components	Page 16
Figure 3.2 – n-tier ASP Architecture	Page 17
Figure 3.3 – ASP types	Page 22
Figure 4.1 – Operate-Repair Cycle	Page 29
Figure 4.2 – Representation of Availability	Page 30
Figure 4.3 – Relative Cost of Availability	Page 32
Figure 4.4 – Sample Internet Application Configuration	Page 34
Figure 4.5 – Introducing Redundancy	Page 35
Figure 4.6 – Webserver Configuration with Increased Redundancy	Page 36
Figure 4.7 – Sample HA Configuration	Page 36
Figure 4.8 – Causes of Downtime	Page 40
Figure 4.9 – Standby Cluster Configuration (Hot Standby)	Page 45
Figure 4.10 – Takeover Cluster Configuration	Page 45
Figure 4.11 – Mutual Takeover Configuration	Page 46
Figure 4.12 – Heartbeat Network	Page 47
Figure 4.13 – Sample Load-Balancing configuration	Page 51
Figure 4.14 – Hardware Load Balancing Configuration	Page 52
Figure 4.15 – Shared-nothing Architecture	Page 55
Figure 4.16 – Shared-disk Architecture	Page 56
Figure 4.17 – Shared-memory Architecture	Page 56
Figure 4.18 – Total Execution Time	Page 58
Figure 4.19 – Incremental Backup Schedule	Page 63
Figure 4.20 – Tier 1 Recovery Solution	Page 67
Figure 4.21 – Tier 2 Recovery Solution	Page 67
Figure 4.22 – Tier 3 Recovery Solution	Page 68
Figure 4.23 – Tier 4 Recovery Solution	Page 68
Figure 4.24 – Tier 5 Recovery Solution	Page 69
Figure 4.25 – Tier 6 Recovery Solution	Page 69
Figure 4.26 – Data Loss & Service Loss	Page 70
Figure 4.27 – Billing4Rent / Singl.eView n-Tier Architecture	Page 71
Figure 4.28 – Singl.eView Components	Page 72
Figure 5.1 – B4R Prototype Home Page	Page 83
Figure 5.2 – B4R Contact Details	Page 83
Figure 5.3 – B4R Client Login	Page 84
Figure 5.4 – B4R Client Home	Page 84
Figure 5.5 – B4R Administrative Interface Hyperlink	Page 88
Figure 5.6 – B4R Administrative Interface	Page 89
Figure 5.7 – B4R Administrative Interface Home Page	Page 89
Figure 5.8 – No Privileges	Page 90
Figure 5.9 – Database Properties	Page 92
Figure 5.10 – Tablespace Properties	Page 93
Figure 5.11 – Table Sizes	Page 93
Figure 5.12 – JAMon Performance Monitor	Page 94
Figure 5.13 – B4R.properties	Page 94
Figure 5.14 – Filter Logger	Page 97
Figure 5.15 – Display Log Records	Page 98
Figure 5.16 – Billing4Rent Architecture	Page 101
Figure 5.17 – Inter-tier Traffic Flows	Page 105
Figure 5.18 – Application Server Vertical Scaling	Page 106

Figure 5.19 – HTTP Server Vertical Scaling	Page 107
Figure 5.20 – Backup Internet Connection	Page 107
Figure 5.21 – B4R Disaster Recovery Solution	Page 114
Figure 6.1 – Billing4Rent Architecture	Page 120
Figure 6.2 – MVC in a B4R Transaction	Page 123
Figure 6.3 – Hidden form fields used in B4R	Page 125
Figure 6.4 – B4R.properties	Page 126
Figure 6.5 – Closing B4R resources	Page 127
Figure 6.6 – Downtime Estimate Formula	Page 128
Figure 6.7 – B4R Component Benchmark Summary	Page 133
Figure 6.6 – DataElectronics Co-location Package	Page 136

SECTION ONE



Introduction and Thesis Structure

Chapter One: Introduction & Thesis Structure

This chapter introduces the dissertation topic by outlining Application Service Provision in general, as well as a brief introduction to High Availability and Scalability. Finally, an overview of each of the chapters of this dissertation is provided.

1.1 Introduction

Regardless of its continual rebranding; from ‘software-as-a-service’, to ‘on-demand’ computing, and to the latest (2005) embodiment as ‘hosted application management’, the Application Service Provision (ASP) model offers multiple users subscription-based access to centrally managed and hosted applications, via the Internet. Due to centralised hosting, ASPs provide access to software on a one-to-many basis. Consequently the cost of ownership and maintenance of the solution is shared by several clients, which introduces enormous economies of scale for end users. Despite the initial hype, user uptake of the ASP model has been slow to materialise. This can be attributed in part to the bursting of the *dot.com* bubble in early in 2000, which caused the closure of many on-line service providers, and which severely dented the confidence of many with regard to renting application access. However uptake of the ASP model is steadily increasing. Indeed, some of the largest players in the software industry have recently introduced ASP offerings including Microsoft (Office Live), Google (Wrightly) and Oracle (Oracle On Demand). Current estimates indicate worldwide spending on software-as-a-service (and associated software license revenue) should reach \$15.2 billion by 2007 - much lower than earlier predictions but substantial nonetheless.

1.1.1 Application Service Provider

An ASP (Application Service Provider) is a third party entity that deploys, hosts and manages access to a packaged application or service, and delivers this software

based service to customers across a wide area network, usually from a central data centre. Applications are typically provided on a subscription or rental basis.

1.1.2 High Availability

High availability refers to a system or component that is continuously operational for a desirably long length of time. A fault-tolerant application's goal is to be available 24 x 7 x 365 to minimize the potential losses that will be incurred if the application suffers downtime. Availability can be measured relative to '100% operational' or a 'never failing' system. An ideal, but difficult to obtain standard of availability for a system, would be 99.999% availability. Downtime of a system usually affects customers and will have a drastic effect on the company's image or profitability.

1.1.2.1 Failover

Failover is a backup operational mode in which the functions of a system component (such as a processor, server, network, or database) are taken over by secondary system components when the primary component becomes unavailable through either failure or scheduled down time. Used to make systems more fault-tolerant, failover is typically an integral part of mission-critical systems that must be constantly available. The procedure involves automatically offloading tasks to a standby system component so that the procedure is as seamless as possible to the end user. Failover can apply to any aspect of a system.

1.1.2.2 Disaster Recovery

Disaster recovery, or Disaster Recovery Plan (DRP) - sometimes referred to as 'business continuity plan' (BCP) - describes how an organization deals with potential disasters. Just as a disaster is an event that makes the continuation of normal functions impossible, a disaster recovery plan consists of the precautions taken so that the effects of a disaster will be minimized, and the organization will be able to either maintain or quickly resume mission-critical functions. Typically, disaster recovery planning involves an analysis of business processes and continuity needs; it may also include a significant focus on disaster prevention.

Disaster recovery is becoming an increasingly important aspect of enterprise computing. As devices, systems, and networks become ever more complex, there are simply more things that can go wrong. As a consequence, recovery plans have also become more complex. Current enterprise systems tend to be too complicated for simple and hands-on disaster recovery approaches, however, and interruption of service or loss of data can have serious financial impact, whether directly or through loss of customer confidence. Disaster recovery planning may be developed within an organization or purchased as a software application or a service. It is not unusual for an enterprise to spend 25% of its Information Technology (IT) budget on disaster recovery.

1.1.3 Scalability

Scalability is the ability of a computer application or product (hardware or software) to continue to function appropriately when it (or its context) is changed in size or volume in order to meet new user needs. In most cases, the rescaling is to a larger size or volume to accommodate larger volumes of users and processes to use a system. Scalable systems should not only function well in the rescaled situation, but also take full advantage of it. For example, an application program would be scalable if it could be moved from a smaller to a larger operating system and take full advantage of the larger operating system in terms of performance (user response time and so forth) and the larger number of users that could be handled.

1.2 Research Contribution

Thesis title: *'A Framework for High Availability and Optimum System Performance in an Application Service Provider (ASP) Environment'*

The aim of this project is to develop a framework, which includes the architecture and system configuration required, to ensure High Availability and Optimum System Performance in an Application Service Provider (ASP) Environment. This thesis investigates the current state of Application Service Provision, and proposes a framework for Availability, Scalability and System Performance that can be used by

ASPs going forward. The ASP solution, **Billing4Rent**, will provide the real world platform on which any recommendations or proposals arising out of the body of research will be implemented and allow the research to be objectively analysed and quantified in a real world environment.

1.3 Structure of Thesis

The remainder of this document is structured as follows-:

Chapter Two – describes the research framework used. This includes a description of the background of the Billing4Rent project (the real world ASP platform on which any recommendations will be implemented), the research methodology, as well as the research objectives.

Section Two – Literature and Technology Review

Chapter Three – reviews the current literature available in the general area of Application Service Provision. This chapter will provide an insight into the origins of ASP; it's evolution over the years; the advantages / disadvantages of the ASP model; analyst predictions for the ASP paradigm, as well as the various types of ASPs currently serving the market.

Chapter Four – reviews the current literature available in the general area of High Availability and Scalability. This chapter provides an introduction to High Availability in general, highlighting the historical reasons for designing Highly Available systems. An outline of the metrics and benchmarks used to evaluate availability as well as desired levels of availability is discussed. The consequences of server downtime, and high availability planning will also be highlighted. In addition, Scalability (and scalability drivers) is discussed. The various types of scalability, as well as the architectures and design principles are also covered. Finally, an outline of clustering, load balancing and Disaster Recovery, and their importance within an ASP environment is examined.

Section Three – Research Contribution

Chapter Five – this chapter details an ASP solution (Billing4Rent) which will provide the real world platform on which any recommendations or proposals arising out of the body of research will be implemented and allow the research to be objectively analysed and quantified. Chapter five also describes the proposed ASP framework for Billing4Rent. The architecture of Availability and Scalability is detailed here.

Section Four – Research Evaluation

Chapter Six – this chapter provides a substantive evaluation of the proposed framework and how it performs in comparison with accepted industry benchmarks. The evaluation considers the real world implementation (Billing4Rent) discussed in the previous chapter.

Chapter Seven – this chapter details the conclusions of this research project. It also identifies avenues of possible future research that arise from this project.

A **Bibliography**, a **List of Acronyms** used in the Thesis, as well as a number of **Appendices** follow Chapter Nine.

Chapter Two: Research Background & Methodology

This chapter details the background to this body of research (*i.e.* the development of a framework for Application Service Providers) in order to achieve Optimal System Performance and High Availability. The proposed framework is then implemented in a real word solution (Billing4Rent) which allows objective analysis of the framework within a live production environment.

2.1 Background

The following abstract and background material is reproduced from the Enterprise Ireland ‘Innovation Partnerships - Company / Third Level College Collaboration for New Product / Process Development’ [1] proposal document. The proposal was subsequently approved, and was given the title ‘Billing4Rent - The ASP Hosted Billing Service’ (Project Code: IP-2004-333).

Abstract

“The Billing4Rent project will analyse, design and implement a highly innovative billing solution for communications providers. The solution will be realised using Application Service Provider (ASP) technology, enabling tier 3 and 4 network operators, content providers, service aggregators and other genres of service providers to access state-of-the-art billing functionality on a subscription/rental basis.

The project partners, ADC, WIT and GMIT, have identified a unique and innovative way of overcoming these barriers. The approach will harness the functionality provided by ADC’s flagship Singl.eView† product, which will

† At the time of the Billing4Rent project proposal (*circa* May 2004), Singl.eView was an ADC product. However on 4th June 2004 - Intec Telecom Systems announced the acquisition of ADC’s ‘Singl.eView’ retail billing software division for \$74.5 million.

be enhanced to provide the core of an ASP billing service providing the flexibility, ease-of-use, architecture and business model required by the intended customer base. The results of the project will provide a full solution to an underserved target market and, in doing so, will provide a major boost to Ireland's competitive positioning in both the billing and the communications industries. The Billing4Rent ASP billing service will enable providers to enter the market by offering an affordable, effective billing solution."

Background

"The tier 1 and 2 billing market is today dominated by five product vendors: Amdocs, Convergys, CSG Kenan, Portal and ADC, with Amdocs and Convergys being in a dominant market position due to their relatively larger revenue earnings. A large number of additional vendors compete for specific market niches such as adjust rating, inter-connect rating, content rating and pre-paid services support.

The ongoing rollout of 3G networks and increasing deregulation of the wired telecommunications market is fostering a very dynamic services and application market place involving a large number of creative and innovative SMEs. Such SMEs require billing solutions to allow them to capture their share of the revenue and to break down existing entry barriers to markets. Hence, the project partners have identified a significant underserved market – provision of outsourced billing solutions for tier 3, 4 and 5 mobile and wired operators, content providers, service aggregators and other genres of service providers. Hence, the project partners have identified a significant underserved market – provision of outsourced billing solutions for tier 3, 4 and 5 mobile and wired operators, content providers, service aggregators and other genres of service providers. The lack of effective, affordable outsourced solutions has prohibited a lot of these entities from entering the market on their own terms and has, in a lot of cases, allowed them to be exploited by the larger network operators who perform billing on their behalf.

The solution proposed here is to create a robust online billing service that could be shared by these entities, hence making it more affordable. Such a solution can

be assembled using ASP technologies founded on the third generation of distributed component technologies – HTTP based transport, XML interchange using SOAP and contract based management using WSDL.

A key differentiating factor of the Billing4Rent ASP service will be that its core (the Musketeer platform) will incorporate core components of ADC's highly successful Singl.eView dynamic transaction management platform. In addition the service will provide customers with the option of utilising innovative dynamic tariffing schema which can vary the charges applied for service usage based on service usage information.

This Innovation Partnership will address a number of research and technical challenges required to leverage the functionality of Singl.eView in an innovative ASP billing solution to target the tier 3/4 operator and communications service provider market. The project partners recognise that an ASP billing solution will need to offer an end-to-end solution that supports a low cost of usage for target customers. The major areas of innovation required to realise this vision include:

- Creation of intuitive user interfaces for configuration of the service;*
- Creation of innovative tariffing schema design tools;*
- Strategies to address ASP performance and security issues.*

The Telecommunications Software & Systems Group (TSSG) at WIT has researched the area of on-line (ASP) billing through a number of European Commission and EI funded projects and has developed a prototype "Rating Bureau Service" (RBS) that addresses some (but not all) of the needs of an ASP billing solution. Both TSSG and GMIT have significant experience in creating ASP type services using web services and similar technologies. ADC will contribute market research, industry experience, software integration, training, partner relationships and core components of Singl.eView to the project. Together the proposal parties will create a partnership to apply their expertise to create an innovate ASP billing solution.

ADC see the opportunity for the results of the project to be fed into a start-up organisation called Billing4Rent. ADC is currently discussing the possibility with an external third party with an interest in commercially exploiting any viable results arising out of the partnership, who would create the company and bring together a team and funding to exploit the results of this project and launch the ASP billing solution. This external third party may also play a critical mentoring and advisory role throughout the lifetime of the project to ensure that he can sell the ASP billing service that will emerge. For their part ADC see Billing4Rent as a potential channel to address a focussed market. The TSSG has an opportunity through this initiative to further develop its online portal billing IPR for exploitation by ADC.

ADC is currently working with industry partners, GMIT and Enterprise Ireland on a High Performance Commercial Computing initiative to address the business challenge of selling, delivering and supporting always on, highly scalable solutions for Operational and Business Support systems targeted at the communications and content industry. This project will support this initiative building on the highly successful ASP model to provide a very large centralised solution that will serve as a proof point for the industry.

In summary, the Billing4Rent solution will target the innovative builders of the new economy, the new smaller communications providers, competing in the deregulated market and the new economy digital media companies building a new market around content. Specifically Billing4Rent will prove to be a critical service for new entrants that will emerge as part of the EU enlargement process. It is important to note that a major problem for these new companies is the ability to bill for the delivery of their products and services. Billing4Rent would thus not only create new value for Ireland but also support the development of existing Enterprise Ireland ICT companies.”

2.2 Limiting Factors

The author acknowledges the important role security plays in achieving high levels of availability within an ASP model. Robust security policies and plans (e.g. data encryption and strong passwords) ensure that unauthorised persons do not gain access to the ASPs systems for the purpose of pilfering / corrupting data, or compromising the availability of the service. It is worth noting that the Billing4Rent project proposal identified two areas of research relevant to the ASP model. One of these areas was security, with the other being:

“Performance: The huge increase in the volume of metering data associated with emerging IP-based services, together with the necessity for real-time processing to accommodate prepaid service provision, places stringent performance requirements on billing components. The adoption of an ASP model introduces particular performance issues relating to the transfer of information to an externally hosted billing service, using web service technologies. A careful analysis of these issues will be carried out to ensure that the Musketeer platform can meet performance targets, especially in prepaid scenarios.”

It is recognised in the project proposal that although the areas of security and performance have several commonalities, each is distinctly different from the other. Accordingly, this thesis focuses on Optimal System Performance and, consequently, security-only concerns and plans are beyond the scope of this dissertation.

2.3 Research Methodology

The principal research objective is to design a framework / architecture, for an ASP solution that is Highly Available, Scalable, Maintainable, and performs optimally within a production environment. The steps involved in achieving this objective are summarised as follows:

- The completion of a comprehensive literature review to fully understand the nature of Application Service Providers in general, and High Availability in particular. The knowledge gained from this phase of the project served as a solid theoretical basis for the design of the deployment architecture for the

Billing4Rent ASP. In particular, acclaimed textbooks, research papers and Internet websites dealing with Application Service Provision, High Availability and Scalability were studied and evaluated to ensure that any decisions arising out of the literature review will be following best practices and industry standards.

- The completion of a comprehensive technology review to evaluate the latest developments and advances in technologies relevant to Application Service Provision. In particular the design, implementation and deployment of Highly Available solutions was studied and evaluated. The knowledge gained from this phase of the project meant that technical design decisions taken at later stages of the project were based on best practices as evidenced in current technology standards.
- A formal proposal for the deployment architecture necessary for the Billing4Rent ASP will be made, reflecting the lessons learned from the literature and technology review phases of the project. The proposal will incorporate the design, definition and deployment of the Billing4Rent ASP, in order to meet the criteria of High Availability, Scalability and Optimal Performance.
- The design and efficient implementation of a deployment architecture based on the above formal proposal. An evaluation of the effectiveness of the proposed Billing4Rent deployment architecture will also be carried out. This step will involve a critical and comprehensive evaluation of the Billing4Rent ASP against accepted criteria for High Availability, Scalability and Optimal System Performance in an ASP environment.

SECTION TWO



Literature & Technology Review

Chapter Three: Application Service Provision

Increasingly, the defining characteristic of Internet era software is that it is delivered as a service, not as a product [2]. This new delivery model is known as *Application Service Provision*. This chapter will provide an overview of Application Service Provision, and will include a review of the origins of ASPs. A comprehensive evaluation of recent research literature and industry analysis reports provides an indication of the state of the ASP sector, as well as the various types of ASP players. The benefits associated with adopting an ASP solution are discussed, along with the drawbacks. The following areas will be addressed:

- An introduction to Application Service Provision in general, outlining the historical reasons for the existence of this application outsourcing option. Industry analyst predictions with respect to the application outsourcing option will also be discussed.
- An outline of the advantages of the ASP model, from the provider, and the end-user perspective. The various types of ASPs - and the market segments they occupy - will be discussed. This section will also highlight the obstacles and inhibitors to ASPs serving a larger segment of the business community.
- The five distinct types of ASP offerings identified in the literature will be examined and a commercial example of each type will be identified.

3.1 ASP Introduction

Application Service Providers (ASPs) are defined by the Information Technology Association of America (ITAA) as: “*Any ‘for profit’ company which provides aggregated information technology resources to subscribers / clients remotely via the Internet or other networked arrangement*” [3]. The term ‘ASP’ is slowly being superseded by terms such as ‘Software-as-a-Service’ (SaaS), and ‘on demand’ [4],

[5], as well as its latest embodiment (2005) by the International Data Corporation (IDC) as 'hosted application management' [6]. However, what does remain unchanged is the underlying principal of renting software services via the Internet. An ASP will typically be responsible for hosting and deploying the application or service, as well as its day-to-day management thereafter. ASP revenues typically fall into two main categories: *start-up fees* and *usage fees* [7]. Start-up fees encompass any integration, training or customisation activities required to get a customer setup for an ASP services. Usage fees are an important aspect of Application Service Provision as users are charged only for the portion of an application they actually use. With an estimated 60% of an application not accessed during typical usage [8], per usage billing could amount to a considerable saving for an organisation.

Heralded by many as the third wave of IT outsourcing, the ASP business model came to prominence during the late 1990's. While the term 'Application Service Provider' may be relatively new, the concept of application hosting dates back to the 1960s. At that time, many customers could not afford expensive mainframes, and sourced business applications through the first wave of outsourcing, called *time-sharing* [9]. Commercial organisations and educational institutions took turns renting mainframe-processing capabilities. All the processing capacity, as well as the memory and disk storage resided on the mainframe. A 'dumb terminal' – the terminals did not do any thinking or processing - relayed messages to and from the mainframe. This allowed users to access applications online on a pay-per-use basis as an alternative to purchasing and maintaining expensive and complex software at each company's physical site.

However there were problems with the mainframe/timesharing approach. This was due for the most part to availability; for example if too many people were connected to the mainframe at the same time, the response was often slow [10]. During the late 1960s and 1970s, the advent of minicomputers drove hardware costs low enough to justify customer ownership and control of assets. The early 1980's saw the advent of client/server computing. The Personal Computer (PC) allowed users to perform much of the processing load at their desks. Given that they were the only one accessing the processing power of their PC, users did not encounter the slow response time synonymous with timesharing or concurrent usage. Over time

however, IT managers became aware that large applications installed on users machine were not being utilised on an ongoing basis. Enterprises spent large amounts of money on licences for software that remained unused for large periods of time. Consequently, towards the end of the 1990's, organisations took the opportunity to outsource their IT application needs to an Application Service Provider, who would charge them on a per-usage basis. The 180-degree turnaround from client/server computing back to centralised computing was complete.

In addition to organisations becoming more enthusiastic about outsourcing IT applications, and the emergence of application hosting companies, the final element required for greater ASP adoption was the acceptance of the Internet Browser as the new application interface. Due to the advent of the World-Wide Web during the 1990s, when the Internet evolved from being almost entirely unknown outside universities and corporate research departments, to becoming an almost-ubiquitous aspect of modern information systems, the usage rate of the Internet has increased exponentially - resulting in users becoming increasingly familiar with performing many tasks and procedures via their Web Browser. Figure 3.1 summarises the essential ASP components (Adapted from [11]):

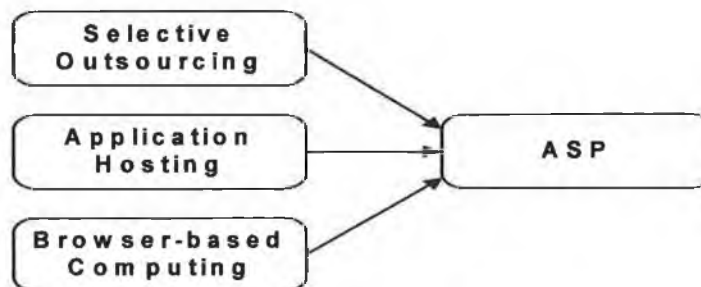


Figure 3.1: Essential ASP Components

A common ASP framework is the *n-tier* architecture. N-tier architecture refers to an application with at least at least 3 'logical' layers: user interface, functional logic (*i.e.* business rules), data storage and data access. N-tier application architecture provides a model for system designers, architects and developers to create flexible and reusable applications. Splitting applications into tiers allows easier modification or additions rather than have to rewrite the entire application, as each tier is developed and maintained as an independent module. This allows the modules to be upgraded

or replaced independently as technology or application requirements change. A particular advantage to the n-tier architecture is scalability should an ASP decide to cater for increased workloads. A three-tier ASP architecture is shown in Figure 3.2 (adapted from [12]).

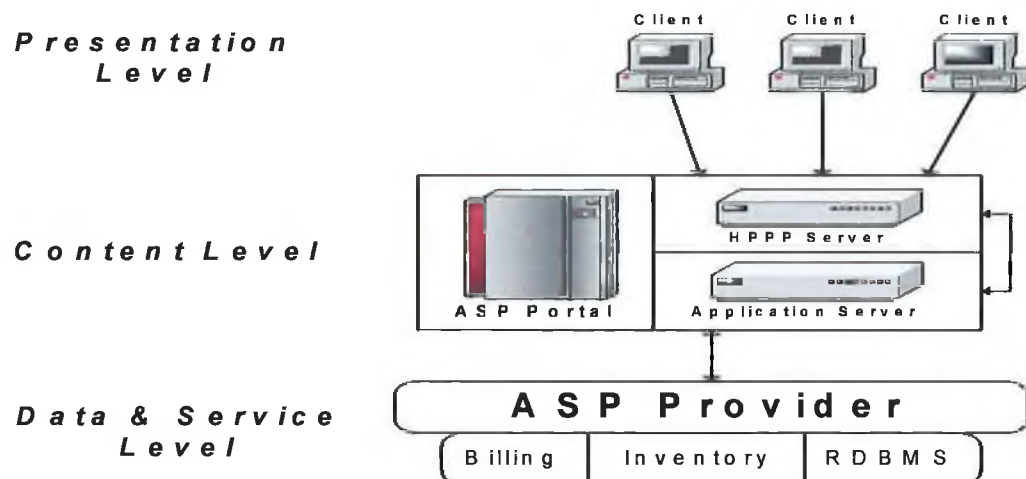


Figure 3.2: n-tier ASP Architecture

The three-tier architecture is made up of the following components:

Presentation: This layer is concerned with displaying data to the user, and accepting data from the user – it represents the ‘look and feel’ of the ASP. Users use a thin client (*i.e.* a Web Browser) to interface with the ASPs web offerings.

Content: This layer is concerned with Business Logic, and is usually powered by an Application Server. The integrity of the data is verified before it is added to the database, or is displayed for the end-user. This tier will drive the Relational Database data queries, updates, and transactions.

Data: The data tier is the layer that manages the persistence of ASP information - data manipulation performed by the ASP will be carried out at this tier. A Relational Database is commonly used at this tier.

3.2 Analyst ASP Predictions

Many ASPs entered the market in the latter years of the 20th century, but struggled to stay afloat due to a number of factors: the large amount of competition, service offerings that were poorly conceived, and the economic downturn after the *dot.com* bubble burst in the early years of the 21st century. As a microcosm of the ASP boom and bust cycle, the *ASP Industry Consortium* began with 25 members in 1999, and boasted more than 300 by the middle of 2000 [13]. However by November 2001, the ASP Industry Consortium met the very fate of many of its members – it was taken over by the Computing Technology Industry Association (CompTIA). Therefore, analyst predictions and expectations with respect to the growth of this new e-business model have not been realised, and in some cases, varied dramatically:

- In 1999, Deloitte Research estimated that ASP revenue would reach \$48.5 billion by 2003 [14].
- By 2001, IDC estimated that ASP revenue would reach \$24 billion by 2005 [15].
- Also in 2001, Gartner estimated that ASP revenue would reach \$25.3 billion by 2004 [16].

However the collapse of *dot.com* shares prices in April 2000 and the subsequent decline in global IT spending resulted in analysts revising down their earlier predictions: In 2002 IDC estimated that ASP revenue would reach \$7.7 billion by 2004 [13]. Also in 2002, Desai *et al* [17] predicted that only 40% of ASPs created before 2001 would survive in their initial form until mid 2002. A follow-up investigation in 2003 showed accuracy of this prediction: of the 424 companies reviewed in the study, 203 had failed, 40 had been acquired and eight had merged. Only 173 companies out of 424 were surviving – just 40.8% [18].

In 2004, actual worldwide spending on software delivered as a service only reached \$4.2 billion, which, although up 40% from 2003 [19], was still a long way short of earlier figures. Although the ASP paradigm is slowly beginning to realise its

potential, as of 2004, software revenue derived from ASP delivery accounted for only 3%-to-5% of all software revenue [81]. Consequently the most recent prediction from the IDC (2005) forecasts that revenue from software-as-a-service would reach \$10.7 billion by 2009 [19].

3.3 ASP Advantages

There are many suggested benefits to the ASP model [10], [11], [17], [20], [21], [22], [23], [24], [25]. ASPs by definition provide application/service access to multiple users/subscribers on a rental or subscription basis. This one-to-many application delivery method affords huge *Economies of Scale* to ASP users. In economic terms, Economies of Scale are achieved when the average cost of producing a product diminishes as each additional product is produced, as the fixed costs are shared over an increasing number of products – *i.e.* the cost per unit made declines with the number of units produced [26]. The Economies of Scale inherent in the ASP business model allows the ASP to operate a secure, reliable data centre at a lower cost per user [20]. Indeed, analyses of ASP cost savings have borne this out. One finding by Miley [27] suggests the cost savings to be in the order of 20% to 50%, while 14% of respondents in one survey by the Information Technology Association of America (ITAA) [28] placed the cost savings at between 51% and 100%.

Due to the global nature of the Internet, companies have a extensive choice as to which ASP offering worldwide they can choose, regardless of national borders or time zones. In a typical ASP service encounter, applications are transmitted to the user machine over a network using a variety of thin client models [20]. *Thin Client Networking* refers to any network in which the lions share of all application processing takes place on a server, instead of a client, with little in the way of local processing power [29]. Typically, the local client interface is a Web Browser. Web sites were traditionally the home of static content, but they now host live applications with increasing frequency. The Web Browser interface has advantages for both the ASP and the end user, as can be evidenced in the removal of the installation procedure. From an ASP perspective, installation of software via a Web Browser eliminates the expense of having to provide installation support, whereas end users

benefit by eliminating the need to employ expensive administration staff to operate complex software installations [11]. From an ASP perspective, the inconvenience and expense of an offsite client installation is eliminated as software can be used immediately. Users do not have to worry about compatibility with their existing legacy systems, or whether their system is powerful enough to run online software, as access to the ASP offerings is via a Web Browser.

The ASP delivery model reduces an ASPs software release and distribution overheads enormously. Because applications are transmitted to the user machine over a network, this eliminates the need to print manuals, press disks, create packaging, manage stock and operate a stock return policy. The network-centric delivery model allows the ASP service provider to produce instant upgrades, bug fixes, and new features to the software without the user having to discover (or be notified), download and install the upgrades. From a support viewpoint, a knock-on effect would be the consistent user base; users will all be using the same version of the software, so the ASP does not have to support different versions and release levels. Additionally, given that the bulk of the processing resides on the server, this reduces software piracy, as unscrupulous users cannot copy and distribute the full version of the software.

ASPs will generally facilitate reduced down time, as service providers will be better equipped to provide 24/7 availability, and at a lower cost. The cost of this expertise is significantly lower, due to the cost being spread among multiple users. Kern, *et al* note that the ASP outsourcing model reduces a users need to retain in-house skilled IT professionals [21]. Because the ASP usually has more expertise in technology forecasting and a larger customer base, it can usually make better technology investment decisions than customer organisations that are not IT specialists [20]. ASP outsourcing also gives the enterprise the flexibility to change direction without losing major IT investments [10], whereas the pricing model of ASPs enables predicable and controllable usage and application costs [21].

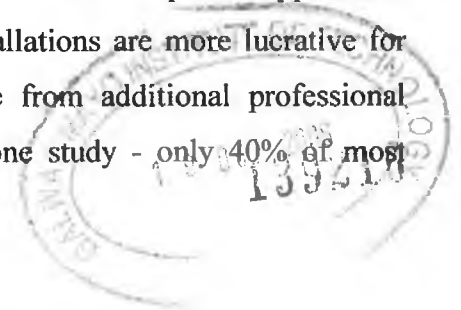
An ASP has the advantage of distributing the cost of surplus capacity across several organisations [22]. Because the ASP can constantly monitor usage, it is able to evaluate the features used most often, identify any features that cause the user

problems, and any features that can be streamlined to increase productivity [11]. The ASP benefits from knowing it has a constant revenue stream – it is not necessarily under pressure to deliver annual releases to generate revenue [11].

3.4 ASP Growth Inhibitors

There are several hypotheses put forward as to the reason for the slow take up of the ASP model. Firstly, one of the main reasons cited was the ASP users loss of control of IT functions and their management [17], [21]. Many organisations are reluctant to become increasingly reliant on external suppliers, where they have little or no input in how their IT operations are managed. In the early days, users were reluctant to adopt the paradigm, mainly due to lack of understanding of the ASP model. As with all emerging technologies, many organisations decided on a ‘wait and see’ policy, whereby a decision would be made on ASP adoption once the embryonic nature of the ASP marketplace had taken a proper shape. Due to the large mortality rate of early ASP players (as noted in section 3.2, in 2001 it was forecast that only 40% of exiting ASPs would still be in business in 2002), there were many companies that were unwilling to outsource critical IT applications to ASPs that may or may not be in business long enough to fulfil their contractual obligations [7]. Also, there was considerable resistance to the ASP paradigm from organisations IT Departments. Many IT Departments perceived ASPs as a threat to their livelihood. Mark Clancy, in his 2001 article “The Insidious Resistance to ASPs” [16] identified the large power displacement felt by many IT Departments as they felt that they were becoming surplus to requirements. For example, if a company adopts an ASP model for a sales application, it may make more sense for the operating budget to be transferred from IT to the sales department. Given that that many IT departments would have been entrusted with responsibility for deciding whether or not to outsource IT applications, this may have been a contributing factor to the slow adoption of ASPs.

Finally, some software vendors are also resistant to the concept of Application Service Providers. Often, full software package installations are more lucrative for vendors [16] and can give rise to further revenue from additional professional services contracts. Given that - as highlighted by one study - only 40% of most



shrink-wrapped software was utilised [8], a software delivery model where the customer paid by usage would seriously affect the revenues of software vendors. Even if a software vendor becomes its own ASP, they may still advise their customers toward purchasing a packaged solution in order to generate higher short-term revenue.

3.5 Types of Application Service Providers

The most common types of ASPs to have engaged in the market in recent years is summarised in Figure 3.3 (Adapted from [4], [7], [17], [18]).

No	Name (Type)	Main Features
1	Horizontal	An Independent Software Vendor (ISV) or start-up firm, which delivers 'business' software such as CRM or payroll, as well as collaborations tools (Groupware)
2	Pure Play	A start-up firm, which enters into partnership with an ISV to deliver software on a remote model over the Internet
3	ASP enablers	Telecommunications Companies with the necessary IT infrastructure (backbone) to deliver software using a remote model
4	Vertical	An ISV, or start-up ASP, focusing upon a specific industry sector
5	Enterprise	A large ISV, or start-up ASP, which aims to deliver enterprise wide or ERP software to the end-user via a remote model, or via a virtual private network

Figure 3.3: ASP types

- Horizontal ASPs are the most popular of ASP offerings, where the ASPs Web enabled applications were relevant to most companies. The main advantage of Horizontal ASPs is their ability to gain market share without having to acquire vertical (*i.e.* industry specific) knowledge of their customers business sector. **Salesforce.com** [30] is one of the worldwide leaders in on-demand Customer Relationship Management (CRM) services, which encompasses all aspects of interaction a company has with its customer, whether it be sales or service related.

- Pure-play ASPs are typically independent start-up companies that do not own any applications or infrastructure. Pure-play ASPs pull together many elements that make up the ASP solution, as they may licence the hosted application from one company, buy network access from another, rent server space in a server farm at a hosting company, as well as application support elsewhere. Pure-play ASPs typically provide minimal service integration activities for their customers, as their focus is primarily on the customer interface. In general Pure Play ASPs offer web-enabled solutions only. An example of a pure-play ASP was **Corio** [31] (Corio was purchased by IBM in March 2005 for \$182m) who offered subscription-based services for several applications, including software from companies such as Oracle, SAP and Siebel Systems among others.
- ASP enablers are large companies that provide telecommunications, hardware, or Internet connectivity, as well as co-location services to ASPs. ASP enablers have the advantage of partnering with existing (enterprise, vertical, horizontal and pure-play) ASPs, as well as those seeking to become ASPs offering a range of web-enabled applications. A leading ASP enabler is the South Africa based **Korbi.net** [32], who bring together a wide variety of applications to achieve economies of scale and make it possible for an organisation to become an ASP by removing the associated cost of technology and infrastructure.
- Vertical ASPs target specific market segments for their services. The primary focus of this type of ASP is to provide vertical industries (*e.g.* the travel agency industry) with all the industry specific tools and applications they need ‘under one roof’. An example of a Vertical ASP would be **LearningStation.com** [33], a leading provider of customised web desktops for schools, used in order to support and advance learning.
- Enterprise ASPs deliver a variety of high-end applications to enterprises, for example to divisions, subsidiaries and business units of very large enterprises. This type of ASP generally offers some degree of customisation availability guarantees. An example of an Enterprise ASPs is **Usi.net** [34], who deliver application outsourcing, remote management, professional services, ISV

enablement, eBusiness development, as well as hosting, information security and risk management services.

In recent years, a new type of ASP has emerged onto the market in the form of Independent Software Vendor (ISV) ASPs. These types of ASP are software vendors that have decided to expand beyond solutions provisioning and offer ASP services directly to the customer [7]. ISV ASPs generally have two sub-categories: the first is where the software vendors have decided against allowing other ASPs to host or provide their applications and is hosting it themselves. The second sub-category is a software vendor who licences their software to an ASP for delivery.

3.6 Chapter Summary

This chapter has focused on providing summary detail on Application Service Providers, and the origins of the application outsourcing option. Also analysed were the advantages and disadvantages of the ASP paradigm, as well as the types of ASP players serving the market at the present time. Although the emergence of the Application Service Provision model suffered during the bursting of the dot-com bubble in the Autumn of 2001, there is little doubt that *software-as-a-service* is here to stay. A memo released by Microsoft [35] in November 2005, announced the emergence of a new generation of web-based software, commonly referred to as *Web 2.0* [2]. Microsoft's Chief Technology Officer stated that Web 2.0 is "about 'services' (ranging from today's web-based e-mail to tomorrow's web-based word processor) delivered over the web without the need for users to install complicated software on their own computers". To this end, Microsoft announced plans to launch Windows Live and Office Live in early 2006 [36]. Office Live promises to provide an organisation's business management applications (*i.e.* customer, project, and document management tools), all hosted and maintained by Microsoft. While many users may consider ASPs solely useful within a business context, there are increasing examples of Application Service Provision encroaching into everyday life. Consider for example the huge upsurge in mobile phone usage in the early years of the 21st century. Many network operators offered a value added service whereby the user can send *text messages* to other mobile phone users via the operators website. This

offering has all the hallmarks of an ASP encounter – the primary interface between customer and texting service is a Web Browser; the service is based on a one-to-many model.

While this chapter has focused on the abstract details of ASPs, Chapter Four will focus on High Availability and Scalability. As already discussed, ASPs are typically hosted in large datacenters, where the concepts of high availability and scalability are essential components in delivering a solution that has near 100% uptime and a high Quality of Service (QoS) to its end users.

Chapter Four: High Availability and Scalability

ASPs are typically hosted in large datacenters, where the concepts of High Availability (HA) and Scalability are hugely important. Indeed, HA, Scalability and Reliability are often cited as the major inhibitors to the widespread uptake of the ASP model [7], [10], [17], [37]. This chapter will provide an overview of HA and Scalability. Supported by a comprehensive evaluation of recent research literature and industry analysis reports, this chapter will commence with a review of the origins of HA. It will provide an indication of HA planning and the rationale for introducing HA. The motives behind designing scalable systems will be also explored. The following areas will also be addressed:

- An introduction to High Availability in general, highlighting the historical reasons for designing Highly Available systems. An outline of the metrics used to evaluate availability, as well as desired levels of availability will be discussed.
- The consequences of server downtime, highlighting the drivers of high availability. This section will examine high availability planning, as well as various high availability paradigms.
- Scalability, encompassing the drivers of scalability will be highlighted. The various types of scalability, as well as the architectures and design principles will also be covered.
- An outline of clustering, load balancing and Disaster Recovery, and their importance within an ASP environment will be discussed.
- A brief overview of SingleView, the dynamic transaction management platform from Intec, which will provide the backbone of the Billing4Rent ASP service.

4.1 High Availability

Application hosting servers may need to support tens of thousands of concurrent service sessions with high availability and short response times [11]. In order to truly benefit from the ASP model, ASPs need to ensure their service offerings are not only reliable, but have near 100% availability. High Availability (HA) is a proactive application of specific hardware and software technologies that minimizes unplanned downtime [38]. A highly available system should provide immediate and automatic recovery in the event of system failure and the recovery process should insulate the end-users from the effects of downtime. The birth of concepts - such as mission-critical systems and high-availability - are to be found in military and space exploration systems where computer failures can place missions, individuals and even nations at risk [39]. There are many known threats to system availability, ranging from natural disasters to malicious attacks. Threats can include, but are not limited to:

- Loss of power.
- Denial-of-service attacks
- Loss of keys to encrypted data.
- Physical damage to equipment from accidents, sabotage or terrorism
- Natural disasters, including lightning, flooding, earthquakes or war.
- Magnetic erasure from electromagnetic pulses, electric currents, or magnets.

Malicious attacks on corporate IT systems, such as viruses, worms and Trojan Horses†, are on the rise and are growing by 15% a year, according to recent data released by ICSA Labs [40]. Consequently, over the past few years, system owners and operators have placed increased emphasis on the actual amount of time equipment is capable of performing its intended function, and is accessible to users.

† Viruses and worms are programs, or pieces of code, that are loaded onto a computer without the users knowledge, and run against their wishes. All computer worms and viruses are manmade, and can replicate themselves over a computer network *i.e.* can make a copy of itself over and over again. Viruses and worms usually perform malicious actions, such as using up the computer's resources and possibly shutting the system down. A Trojan horse is a destructive program that masquerades as a benign application. Unlike viruses, Trojan horses do not replicate themselves but they can be just as destructive.

In the 1960s and 1970s, hardware components were the major source of faults and outages. Today, hardware faults are a relatively minor cause of systems outages compared to operations, environment and software faults [41]. Because of these factors, system requirements and Service Level Agreements (SLAs) usually specify availability goals. The SLA usually defines a minimum level of availability a system must offer. High Availability refers to the availability of resources in a computer system, in the wake of component failures in the system [42]. The basic requirement of a Highly Available system is that it should tolerate faults. The system should have the ability to not only detect a fault but also to report the fault, mask the fault so that it is transparent to the user and then continue service while the fault is repaired offline [41].

A fault usually takes the guise of a component failure, which is the failure of some service component to behave as expected – a hard drive failing to spin up when it is power cycled, a software crash, an operator misconfiguring a network switch, *etc.* A component failure can cause a service failure if it prevents an end user from accessing the service or part of the service [43]. In the event of a component failure, a HA system should have the ability to fail over to a redundant node. A redundant node is a component of a computer system that is used as a backup system in the event of a failure to the primary system. Redundant components can include both hardware elements of a system (such as disk drives, peripherals, servers, switches, routers) as well as software elements (such as operating systems, applications and databases [38]). A HA system should support user connections on either the primary node or the redundant node and have well-defined procedures to deal with startup and shutdown of the system, coupled with clear and specific backup, restore, and upgrade policies.

There are various metrics available to measure availability [41], [44], [45]. As highlighted in Figure 4.1, a computer system operates normally for a period of time before it fails. The failed system is then repaired and the system returns to normal operation.

The operate-repair cycle is shown in Figure 4.1 (Adapted from [46]).

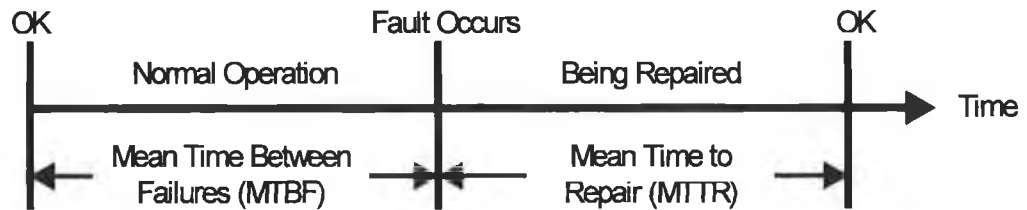


Figure 4.1: Operate-Repair Cycle

High Availability is the proportion of time a system is productive and is usually expressed as a percentage:

Expected reliability is proportional to the *Mean Time Between Failures* (MTBF). Each failure has some *Mean Time To Repair* (MTTR). Availability (Av) can be expressed as a probability that the system will be available:

$$Av = \frac{MTBF}{(MTBF + MTTR)}$$

As the *Mean Time Between Failures* (MTBF) gets larger, Av increases and *Mean Time To Repair* (MTTR) has less impact on Av . As *Mean Time To Repair* (MTTR) approaches zero, Av increases towards 100% [44]. Computers built in the 1950s offered a 12-hour *Mean Time To Failure*. A maintenance staff of a dozen full-time computer engineers could repair the machine in eight hours. This failure-repair cycle provided 60% availability [41].

$$60\% = \frac{12}{(12 + 8)}$$

In a distributed system, some parts may be available while others are not. In these situations, the availability of all of the devices is taken into consideration:

If 90% of the database is available to 90% of the terminals, then the system is $0.9 \times 0.9 = 81\%$ available.

As increasingly critical business applications are placed on computer systems customer tolerance to downtime has decreased dramatically. Some years ago organisations could tolerate hours of downtime caused by computer failures or planned maintenance. Today the majority of enterprises tolerate only seconds or minutes of downtime. A recent study [47] found that 29% of enterprise customers can tolerate only 0-3 seconds of downtime per outage of their critical applications and another 37% of these customers can tolerate only up to 3 minutes of downtime. When these figures are aggregated 66% of respondents can only tolerate up to 3 minutes of downtime per outage. For many enterprises, a desired level of availability is 99.999% (commonly referred to as 'five nines'). This level of reliability, for a service running twenty four hours a day / seven days a week / three hundred and sixty five days a year, equates to just over five minutes of downtime per year, or six seconds downtime per week. Figure 4.2 summarises availability levels and the corresponding average weekly and yearly downtimes (Adapted from [41], [44], [45], [48]).

<i>9s</i>	<i>One 9</i>	<i>Two 9s</i>	<i>Three 9s</i>	<i>Four 9s</i>	<i>Five 9s</i>	<i>Six 9s</i>
Uptime (%)	90%	99%	99.9%	99.99%	99.999%	99.9999%
Downtime per week	16.9h	1.7h	10.1m	1m	6s	605ms
Downtime per year	36.5d	3.7d	8.8d	52.5m	5.3m	31.5s

Figure 4.2: Representation of Availability

When a service is required to run twenty-four hours a day, seven days a week, three hundred and sixty five days a year without any planned and unplanned downtimes, this is commonly referred to as fault-tolerant computing by some commentators [49], and continuous availability by others [50].

- **Fault tolerance** relies on specialised hardware to detect a hardware fault and instantaneously switch to a redundant hardware component, regardless of whether the failed component is a processor, memory board, power supply, I/O subsystem, or storage subsystem. When this cutover appears seamless to the user and offers non-stop service, a high premium is paid in both hardware cost and performance because the redundant components do not perform any processing. This contingency approach to maintaining system availability, whereby a second system - with the same configuration as the main system -

is kept running and ready to take over the processing load instantaneously is known as a *Hot Standby*†.

- **Continuous Availability** implies non-stop operation, with no lapse in service. For most computer systems, this represents an ideal state and is generally used to indicate a high level of availability in which only a small quantity of downtime is allowed. High Availability, however, does not imply continuous availability [42]. In reality, unmanaged computer systems on the Internet typically fail every two weeks and on average take ten hours to recover. This equates to roughly about 90% availability [41].

The difference between fault tolerance and high availability is that a fault-tolerant environment has no service interruption, while a highly available environment has (minimal) service interruption. Many sites are willing to absorb a small amount of downtime with high availability rather than pay the much higher cost of providing fault tolerance. However the fault-tolerant model does not address software failures, by far the most common reason for downtime [49].

Figure 4.3 is an approximation of the cost of implementing a *Fault Tolerant*, or *Continuously Available*, server and illustrates how the cost gets more and more prohibitive the closer the curve gets to 100% availability (Adapted from [49], [51]).

† Standby configurations can be classified as hot, warm or cold. *Hot Standby* occurs when the primary and backup systems run simultaneously. The data is mirrored to the backup server in real time, so that both systems contain identical information. *Warm Standby* occurs when the secondary system runs in the background of the primary system. Data is mirrored to the backup server at regular intervals, resulting in times when both servers do not contain the exact same data. *Cold Standby* occurs when the backup system is only called upon when the primary system fails. The system on cold standby receives scheduled data backups, but less frequently than a warm standby.

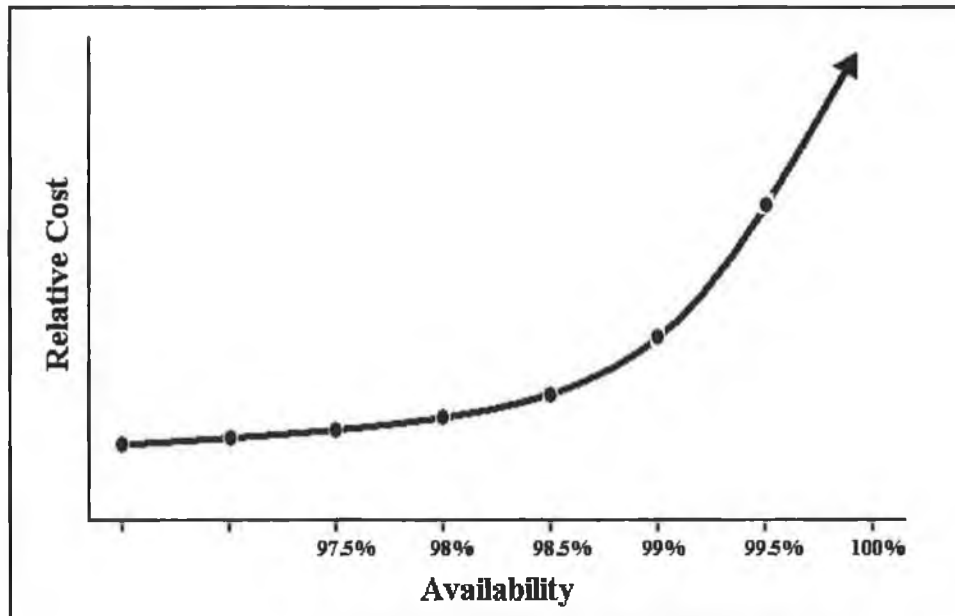


Figure 4.3: Relative Cost of Availability

For the enterprise, there are a number of consequences for server downtime:

- 1. Loss of Business revenue:** When an e-Business Webserver goes down, any business transactions that were in progress will be unable to complete, leading to a loss of revenue for any transactions that would have completed had the server remained operational.
- 2. Loss of Productivity:** While the system is unavailable users cannot work on it, thus impacting their productivity and leading to a waste of resources. A report by IDC [39] highlighted the fact that computer-based design and development tools are becoming standard equipment for a wide variety of engineering tasks. If these tools are not available, then - as noted in the end-user survey - the engineers are not working.
- 3. Loss of Reputation:** A lost Customer never comes back. Many economists acknowledge that it is five to seven times more expensive to find a new customer than to retain an existing one [52]. Negative publicity caused by frequent outages, or downtime at peak business hours, could possibly result in the loss of a customer, which is far worse than the loss of a single purchase. The credibility of the IT department may be diminished within the company should recurring outages occur.

4. **Loss of market opportunity:** Many products have limited life spans, in that they can only be kept on the market for a finite amount of time before they can no longer be sold profitably. For example, in the integrated circuit industry, each day a chip is late to market represents for the company involved a million dollar loss [39]. A reduction in the design and initial manufacturing times can increase the effective sales time of the product. Also, companies that are first to market with products can charge premium prices, at least for a time, and can establish product brands.
5. **Compromised product quality/competitiveness:** In many industries engineers 'design to schedule,' (*i.e.* they are given a fixed schedule for designing a product and they work to design the best product possible in the given amount of time). However, lost resources can spell the difference between a 'workable' design and an optimal design.
6. **Lost data:** Data pertaining to product design is critically important to most organisations. System failures leading to lost or compromised data can be equated to the loss of months or years of product development work and market opportunity.
7. **Regulatory and legal concerns:** Deadlines may be missed due to lack of availability or customer response time may be increased. In some instances, where minimum levels of application availability are guaranteed by SLAs, this may result in penalties and/or fines. It may even lead to legal action from those affected.

4.1.1 High Availability Planning

It is vitally important when implementing a HA system, that proper planning and testing is done at the design and pre-production stage [45], [51], given that failure in operation and the consequent diminished reputation can be very costly to an organisation. Frequently however, IT professionals entrusted with making purchase and policy decisions often get caught in a tug-of-war between (a) the demand to improve service levels, and (b) the pressure to reduce service level costs. High Availability planning should be tackled just like any other business policy. The

organisation involved should document the availability objectives to be met, threats and vulnerabilities to be countered, the risks to estimate, security mechanisms to be used, any real-world constraints and measures of effectiveness, as well as any legacy availability policies. Perhaps the most important consideration when designing a HA system is removing *Single Points of Failure*. A Single Point of Failure (SPOF) is a single component of a system (hardware, firmware, software or otherwise) whose failure will cause some degree of downtime [53]. A SPOF is the weakest link in a system; when that link breaks the entire system fails regardless of the quality or cost of the rest of the system. Most systems have obvious potential SPOFs, such as servers, disks, network devices and cables. For most system designers the most obvious protection against SPOFs is via redundancy. However the cost involved in introducing redundancy may be prohibitive. Consider the Web Application server detailed in Figure 4.4, deployed on a single machine configuration *i.e.* the HTTP Server, Application Server and Database server are all co-located on the one machine, with a firewall† protecting the system from the external network.

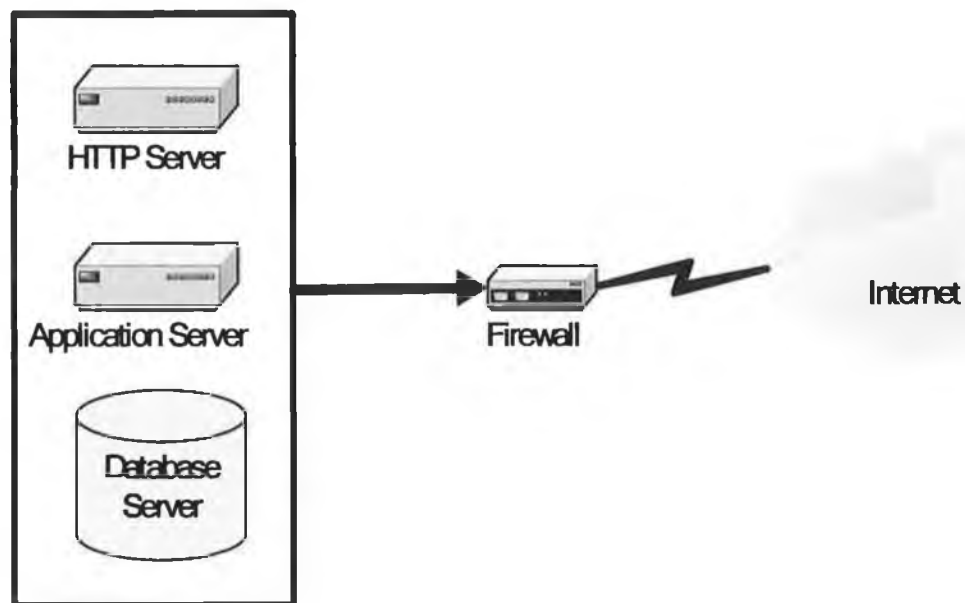


Figure 4.4: Sample Internet Application Configuration

† Firewalls are products designed to prevent unauthorised access to, or from, a private network. Firewalls can be implemented as a dedicated piece of hardware, as installed software, or as a combination of both.

In Figure 4.4, there are two obvious SPOFs. Aside from the fact that the HTTP server, application server, and database server processes will be competing for CPU resources, if a failure occurs on the machine, then the entire system fails. Similarly, the firewall protecting the system from malicious external, and internal, attack is a SPOF. In order to reduce these SPOFs, a second firewall is introduced. Also, the HTTP server, the Application server, and the Database server are all migrated to separate machines. This removes the SPOF inherent in all three being on the one machine, while it allows resource intensive applications like Database servers access a dedicated CPU, rather than compete for CPU cycles. The new configuration is illustrated in Figure 4.5.

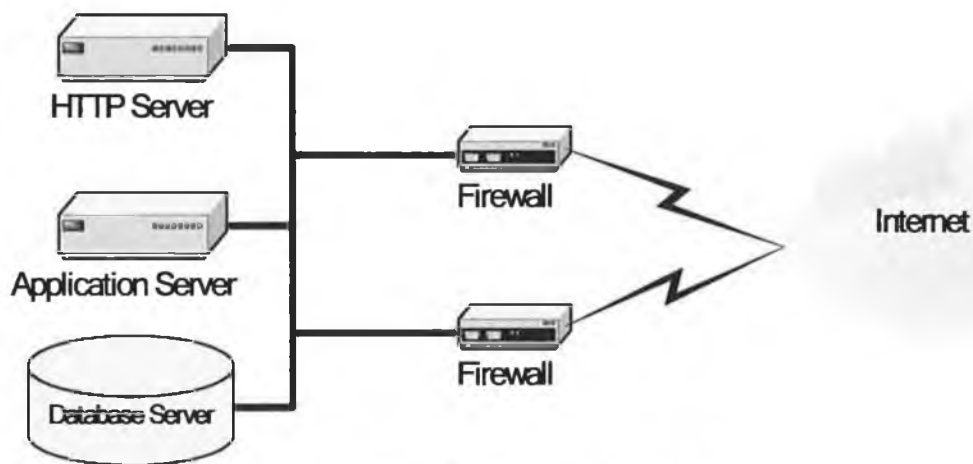


Figure 4.5: Introducing Redundancy

There are still clear SPOFs evident in the configuration in Figure 4.5. While the HTTP server, the Application server, and the Database server have been moved to separate machines in order to reduce the SPOF inherent in having them reside side-by-side in one machine, each of them have now become SPOFs in their own right. For example, the database server *is* on a separate machine. However if it were to be compromised, the overall Web Application would be no longer able to function. A simple remedy would be to replicate the HTTP server, Application server and Database server in order to introduce further redundancy, as shown in Figure 4.6.

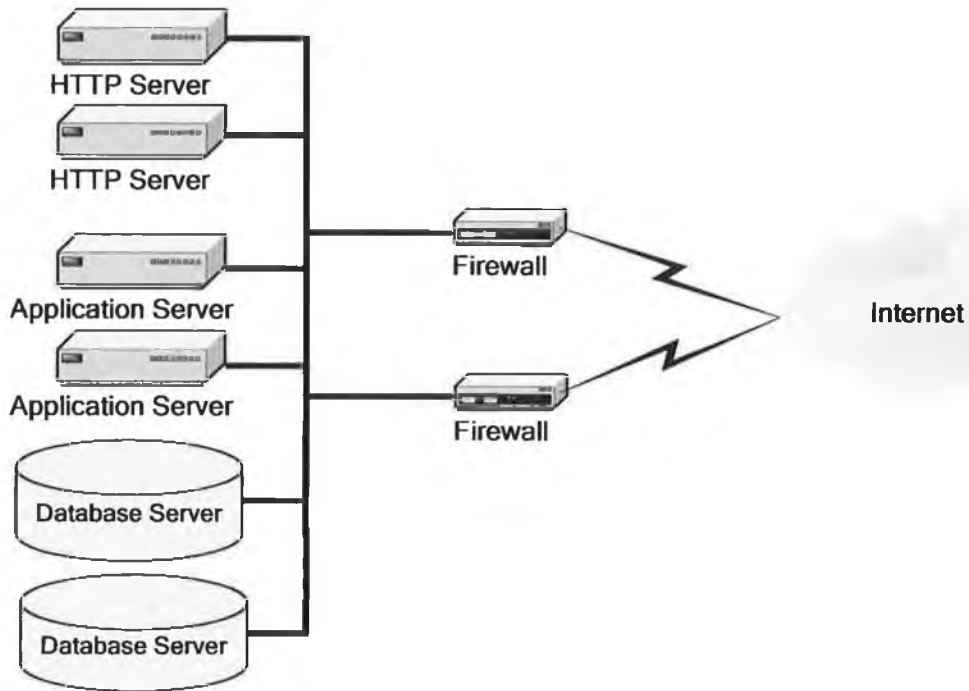


Figure 4.6: Webserver Configuration with Increased Redundancy

This new configuration has eliminated many SPOFs, however one still exists in the form of the underlying network that provides connectivity between the various elements. If for example, the underlying network connection from the HTTP server to the Application server was accidentally severed, the application would again be jeopardised. In order to avoid this, redundant cabling is introduced, as shown in Figure 4.7.

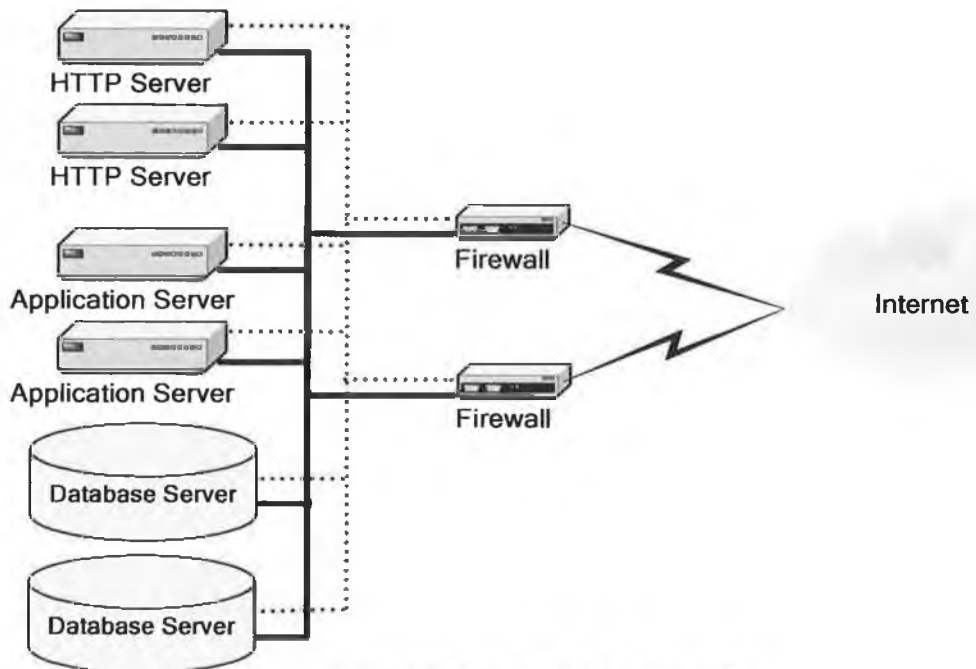


Figure 4.7: Sample HA Configuration

While the above example is by no means an optimal HA configuration, it does illustrate some important points:

1. Firstly, in the examples above, a two-machine Web application server configuration became an eight-machine HA configuration, which incorporated redundant network connectivity between nodes. The consequent increase in hardware expenditure to cover the purchase of the equipment necessary, as well as the added network complexity, can add enormous strain to an organisations operating budget.
2. Secondly, given the increased amount of equipment needed to implement HA policies, it is recommended that hardware and software be standardised within the proposed solution [51]. Industry-standard hardware and software should be favoured over proprietary solutions. The rationale behind using industry-standard solutions is that these solutions all have a public, documented standard. Implementing a solution using various servers, firewalls, software *etc.* from a multitude of vendors - some of them using proprietary solutions - can add confusion, be difficult to document, and may result in the organisation having to spend extra time and resources on maintenance, recovery and troubleshooting.
3. Finally, it may not be possible to eliminate every single SPOF [53], mainly due to either budget constraints, or the fact that while the set-up may be fully redundant - if it were to be housed in one datacenter - then that datacenter itself becomes an SPOF. Similarly, if separate datacenters were used, but each was geographically located within close proximity to each other (*e.g.* on the same power grid), then the power grid serving both datacenters could be regarded as an SPOF.

When planning for HA, it is important that availability targets for systems are well defined in advance [51]. Measurable, yet achievable availability goals should be set, and if possible written into Service Level Agreements. Availability goals in SLAs result in the IT Department having clearly defined responsibility for systems availability. It is also equally important that, prior to implementing a HA policy, a test environment is set-up. Having a test platform will allow an organisation to test

their HA policies before they are incorporated into a production domain. Any or all possible failure scenarios can be simulated to ensure the proposed architecture is resilient, recovers quickly from failure, as well as ensuring that the different hardware and software components work together as expected. Any problems can be found and rectified without affecting end users or live data. Having a test environment also has added benefits when any changes to a live system configuration are mooted. The changes can be tested thoroughly, prior to propagating the update to production systems.

Planning for high availability does not stop after the HA configuration has been implemented. Procedures and codes of practice should be put in place, *e.g.*

- Colour coding and correctly labelling all cables and components in a system. This could save time when correcting a system outage.
- Implementing a management environment capable of informing the IT Administrator, or other authorised persons, of the status of systems in a correct and timely way [51].
- Servers should be stored in a secure, fire-protected room where temperature and humidity are adequately controlled.
- Written procedures should be put in place to ensure that physical access to servers is restricted only to people who are responsible for the server.

No departure from accepted procedures should be permitted without the person responsible for the server being aware of the action, and assuming responsibility for it.

A monitoring tool should be used, and any outputted data from the monitoring tool should be analysed on a regular basis, as trends may become evident that indicate potential problems (*e.g.* degraded software / hardware performance). A remote control / monitoring tool is an invaluable utility allowing a System Administrator to remotely monitor a systems health from their own desk / workstation. It is also recommended [51] that common tasks be automated, as this Proactive Management

allows the Systems Administrator to spend more time on failure prevention rather than reacting to systems failure.

In 2003, Sun Microsystems identified a *Seven Step Approach to Availability* [54], designed to aid HA planning, which enables the delivery of a highly available system while also reducing costs.

Step 1 - Inventory Systems by Business Impact:

Classify systems as task-critical, business-critical, or mission-critical. This allows organisations to identify which of its systems are critical, and require as near to continuous availability as possible (not all systems require 99.999% uptime). One study by Greschler and Mangan (2002) showed that for most shrink-wrapped software, only 40% of the application was utilised [8]. For example, the failure of a task-critical application like a print server may disrupt a few users, however the effect on the company as a whole is negligible. When a mail server goes down, the impact will be higher because it affects employees' ability to do business. While the loss of the e-mail server will affect the business, its effect is not as integral part of how an ISP earns revenue.

Step 2 - Analyse Availability in Tiers:

This step is concerned with understanding which components of the organisations infrastructure support each system. A tiered analysis approach to availability results in identifying various tiers (such as a system layer, data layer, and application layer). Inspection of each tier to determine the systems ability to recover quickly from failure, the systems overall reliability, as well as the ease of maintenance is vitally important. Knowledge of the availability requirements at every tier and for each system helps planning for different levels of availability based on the criticality of the application.

Step 3 - Migrate Availability Costs and Risks Architecturally

The system designed should provide the agreed-upon service levels, at a cost proportional to how critical to the organisation the system is. Simple, yet flexible, architectures allowing fail-over and redundancy can considerably reduce downtime without incurring too many costs. Another factor worth considering is the predictability of the system. For example, a system with predictable need for

maintenance, allows maintenance to be scheduled at a time least disruptive to the business. Thus a predictable system allows an organisation to achieve high service levels without taking on the cost of implementing a mission-critical system.

Step 4 - Reduce the Time to Value:

Time to Value is the amount of time it takes to draw value out of a solution after implementation. One method to reduce costs, as well as time to value, is to choose a solution from a vendor that not only provides the availability infrastructure but also takes responsibility for its performance. This method is preferable to buying separate products and integrating them with existing legacy systems. This takes time, is prone to glitches, adds extra integration costs to an availability solution and it is more difficult to pinpoint the cause of system outage.

Step 5 - Address the Complete Environment:

Typically, costs are only associated with acquisition of products, and implementation of HA systems. In reality, process and people often influence the cost of providing high service levels. Good HA planning also involves the processes and people that support the environment. One Gartner Group Study showed that physical reasons for failure are responsible for only about 20% of all downtime (Figure 4.8, adapted from [51]).

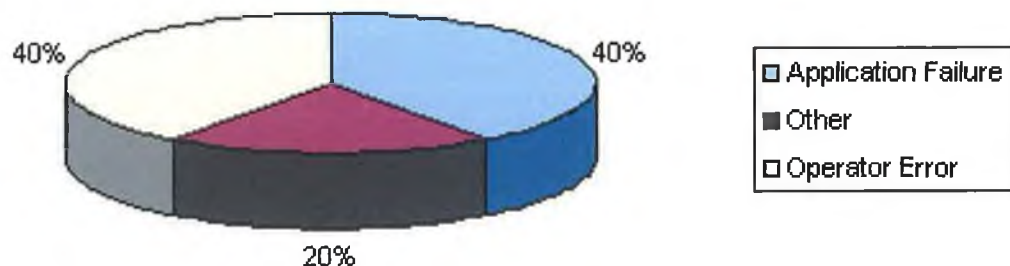


Figure 4.8: Causes of Downtime

Systems failures are more likely to result from operational or human errors than software or hardware glitches.

Step 6 - Design and Implement Comprehensive Recovery Plans

A good availability solution should include a comprehensive recovery plan for each system. Being able to quickly detect failure ensures a speedy recovery when a system outage occurs. A first-rate recovery plan can help reduce the overall cost of downtime to the business

Step 7 - Partner for Experience

Organisations should provide a complete availability solution that meets service level requirements, and also keeps costs to a minimum. While this is an obvious goal, partnering with an company that specialises in providing HA solutions will bring expertise, practical methodologies, real customer experience, and evidence of successful implementations, so an organisation can implement a HA solution with greater ease, lower cost, and above all, confidence.

4.1.2 Availability Paradigms

Availability paradigms have been evolving the over the years as the shift from large centralised systems to distributed computing has gathered pace. Hosmer (1996) identified the *Single Computer*, *Network* and *Cyberspace* paradigms [55], which mirror the move away from large monolithic systems. The single computer paradigm cantered on large mainframe, minicomputer, workstation or personal computers that housed applications crucial to the organisation. The primary threats to such systems were mechanical (*e.g.* loss of air-conditioning in the server room), or human accidents (*e.g.* dropping a disk pack). HA systems had goals of 99.9% availability, whereas low availability systems tolerated 60% availability. Systems designers had to anticipate the maximum loads an application would experience, and plan / design for them accordingly, which invariably meant fail-safe and fault tolerant components, duplicated systems, alternate communications routes, and back-up procedures. The network paradigm offered increased availability due to redundant systems, distributed processing, and distributed databases. The major threats to availability in this model were SPOFs, and external threats such as denial-of-service attacks, viruses, worms, and Trojan horses. The Cyberspace Paradigm emphasises information availability as much as system and network availability. In this model, threats are social (terrorists, hackers, and competitors) -

as well as technical (censorship, electronic junk mail, and system or bandwidth limitations). Availability policies must cover a wider range, and while responsiveness may be highly desired, it may depend on circumstances, as slow response time may be the policy of choice in a variety of situations. In the cyberspace paradigm, availability may cover information content as well as system availability. Parents may be able to restrict their children from seeing certain types of programs or playing certain types of games!

A Whitepaper published by Compaq (1997) echoed Hosmer's findings. Within the *Five Levels of Availability* [47] model, customer requirements regarding HA were identified. The resulting solutions model proposed by Compaq ranged from simple to complex, incorporating single machine systems, multiple machine systems, single sites and multiple sites.

Level 1: In level 1 HA, customers utilise single, standalone servers for their computing services. Redundancy can be incorporated into a single machine set-up, with many server models equipped with redundant cooling fans that ensure continuous cooling of the server even when one fan fails. Similarly, redundant power supplies enable the server to continue to receive power even if one power supply fails. The use of modular architecture design allows users to upgrade their systems gradually and easily over time, while the ability to install software programs and upgrades quickly and easily is also important in increasing system uptime.

Level 2: Level 2 HA builds on the features present in Level 1. In this level, data availability is improved considerably, commonly by implementing a relational database (RDBMS), while also moving to hardware RAID† technology. DBMS are typically used in smaller organisations, where security

† RAID now stands for Redundant Arrays of *Independent* Disks, however during the 1980s, RAID stood for Redundant Arrays of *Inexpensive* Disks. As hard disks became cheaper, the RAID Advisory Board changed 'inexpensive' to 'independent'. While RAID appears to the operating system to be a single logical hard disk, it is a method of storing the same data in various places on multiple hard disks, thereby ensuring redundancy and fault tolerance. Also, by placing data on multiple disks, I/O operations can overlap, improving performance.

of data is not a major concern. An RDBMS however, is designed to process larger amounts of data, and to provide enhanced security. Customers implementing Level 2 HA frequently enhance their services contract to provide faster response time or extended hours of service coverage.

Level 3: While Level 1 and Level 2 HA were concerned with high availability through single server configurations, Level 3 HA is based on multiple machine, or 'cluster', configurations. In Level 3, the focus is on high server and application availability. Clusters are configurations of two or more servers connected together for HA and performance. A simple cluster configuration designed for high availability could comprise of two servers (or 'nodes'), both are operational and actively serving requests from separate application workloads. Both nodes are linked together and communicate with each other. A health manager program constantly monitors the health of each server. If one of the servers experiences component or complete server failure the cluster management software will detect the error or failure. It will then immediately pass ownership of application software, disk, and network resources to the remaining operational node in the cluster, ensuring minimal service interruption for the end-user. Error / failure detection and application failover is fully automated so that no operator intervention is required. Due to the added complexity, HA clusters are more expensive to implement than a single server configuration.

Level 4: Level 4 HA is built upon Level 3 and is usually defined by a move from an availability cluster to a scalability cluster. In a Level 3 HA cluster, each node is active on its own separate application workload. However, in a scalability cluster each node is active on a separate 'instance' (or 'copy') of the same application. Since more than one node is working on the same application, more computing resources can be applied to the same application, thus increasing performance. Consequently, scalability clusters are commonly referred to as 'performance clusters' or 'performance scalability clusters.' In addition to increased performance, scalability clusters also deliver high availability. Given that separate copies of the same

application reside on all nodes in the cluster, a catastrophic failure of one node in the cluster will not cause an application outage.

Level 5: The HA Levels detailed so far incorporate computer systems at a single site, most likely in a single room. As detailed already, single site configurations have limitations, as they become a Single Point Of Failure should a fire, flood, earthquake or other such *force majeure* destroy the building housing the application, or a power grid outage eliminates power to an entire city. Placing servers and storage devices some distance from each other in a separate building can eliminate the SPOF inherent in a single site set-up. Such a configuration is known as a *campus cluster*. Campus clusters ensure high levels of availability, since an outage affecting one building can be recovered by automatically transferring application resources to another node in the cluster, resident in another unaffected building. A campus clusters still has limitations; it cannot recover from a power grid outage, or from a natural disaster, that affects both buildings in the campus. The solution to this limitation is known as a *Geographically Dispersed Cluster (GDC)*. In a GDC, the nodes of a cluster are in a wider distribution, not only building-to-building as in a campus cluster, but also city-to-city, region-to-region, or even country-to-country. A GDC is still a single cluster, whereby a collection of servers is connected by a very high speed interconnect, and they share a single heartbeat. With GDC's, automatic failover in the event of failure occurs not just node-to-node in the same room, or building-to-building in a campus, but city-to-city, region-to-region, or country-to-country, automatically returning service back to the end user.

4.2 Clustering

Increasingly the dominant software platform is evolving from one of shrink-wrapped applications installed on end-user PCs to one of Internet-based application services deployed in large scale, globally distributed clusters [43]. Clustering is the practice of connecting two or more computers together in such a way that their behaviour is that of a single computer, with the obvious benefits of HA, parallel processing, load

balancing and fault tolerance. Clusters can automatically detect and recover from server or application failures, while also eliminating the need for server or application downtime during planned maintenance [56]. Redundant system components provide backup in case of a single component or server failure. Server components - including network adapters, disk adapters, disks and power supplies - are duplicated to eliminate single points of failure. There are two basic types of cluster configurations, *Standby* and *Takeover* [49]. A *Standby Cluster Configuration* is the traditional redundant hardware configuration, whereby one or more standby nodes are set aside, waiting for a primary server in the cluster to fail. This is also known as hot standby (See Figure 4.9).

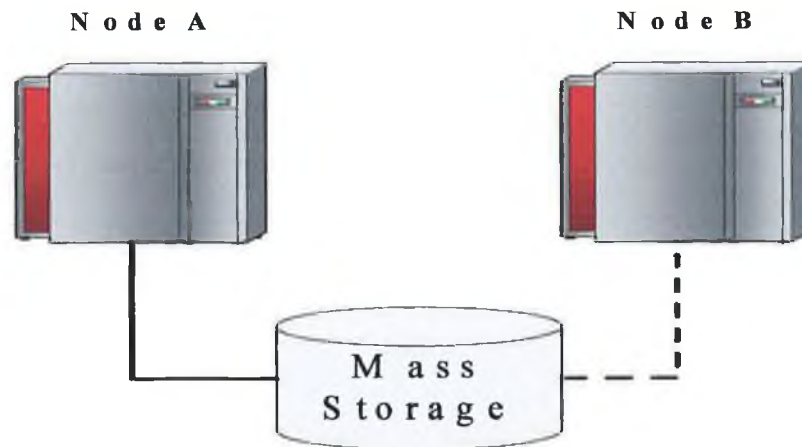


Figure 4.9: Standby Cluster Configuration (Hot Standby)

In the *Takeover Cluster Configuration* all cluster nodes process part of the cluster's workload; there are no nodes set aside as standby nodes. Each node works on a copy of the application, and services the application workload (See Figure 4.10)

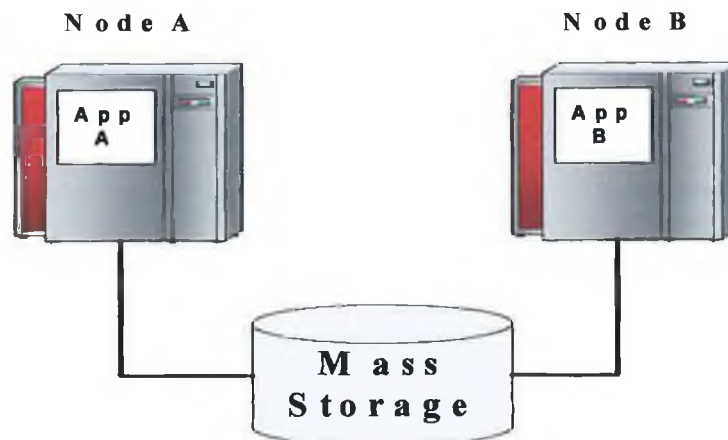


Figure 4.10: Takeover Cluster Configuration

When one node fails, one of the other nodes assumes the workload of the failed node in addition to its existing workload. This is also known as mutual takeover, and is usually considered to be a more cost effective choice since it avoids having a system installed just for hot standby (See Figure 4.11).

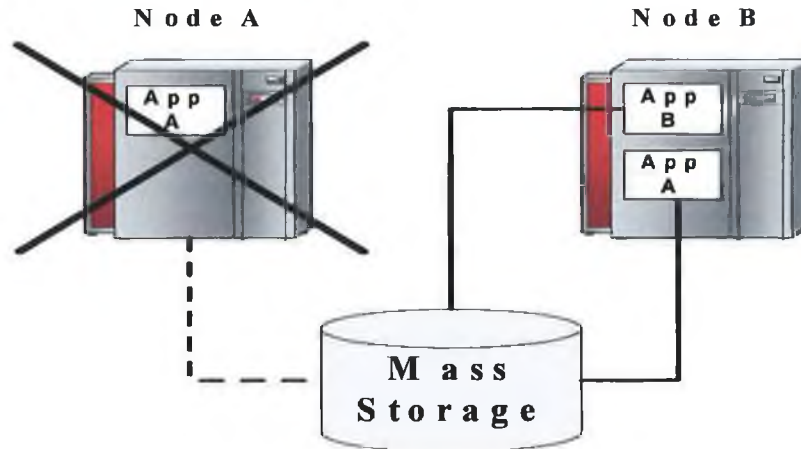


Figure 4.11: Mutual Takeover Configuration

The main benefits of clustering are scalability and high availability. Scalability occurs because the extent of a cluster is not limited to a single server or a single machine [49]. This means that the capacity of the cluster can be supplemented by dynamically adding new servers. If extra hardware or processing power is needed, a new server can be added. Additionally, a cluster uses the redundancy of multiple servers to insulate clients from hardware or software failures, as the same service can be provided on multiple servers in the cluster. If one server fails, another can take over, ensuring no SPOF. From the end users perspective, the availability of the server is constant; the failover from a malfunctioning server to a functioning server appears seamless, so much so that the end user never knows that there was a problem. As previously noted, having redundant cluster members allow routine maintenance of the servers to be scheduled. Indeed, the ability to cleanly migrate services off a cluster member so that routine maintenance can be performed without disrupting service to client systems has been noted as one of the greatest benefits of a high availability cluster [57]. This is because it allows organisations time and opportunity to upgrade software to the latest release or add memory while keeping a site operational. An additional clustering benefit is hardware and operating system independence [58]. Clusters can be run on multiple independent platforms, ensuring

that organisations do not have to rely on specific platform features. Clusters can be comprised of anything from Intel machines running Microsoft Windows NT and its derivatives, to large-scale Unix multiprocessors and IBM OS/390 Mainframes. If an outage occurred on one member of the cluster the other cluster members will conclude that a server has become non-responsive and commence a take over of the services provided by the failed node [57]. This is possible as each cluster member is monitoring the health of the others cluster members, usually over very high speed interconnects. The monitoring process is known as a *heartbeat* network [53] (See Figure 4.12).

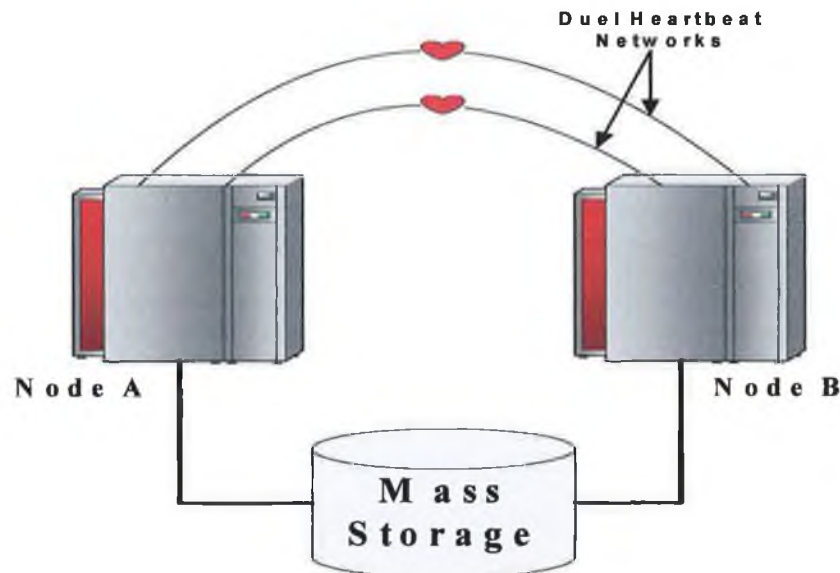


Figure 4.12: Heartbeat Network

Cluster nodes send heartbeat packets to each other, with each packet containing state information about each server, as well as commands from one server directing the other to change states, or execute some function. The primary objective of the heartbeat network is to ensure that cluster nodes can learn of the failure of an individual node when the heartbeats stop. In the eventuality of a stopped heartbeat, the remaining node will take over the services and workload of the failed node and continue service to end-users. Usually heartbeat networks are replicated, with no hardware or equipment in common, or no network paths shared. This greatly reduces the likelihood of the heartbeat stopping for reasons other than a downed server. Failures to networks cards, network cable or hubs, may result in the heartbeat stopping even though all nodes are fully functional. Equally it is important to replicate the heartbeat process itself. In both scenarios, each individual node assumes

it is the only operational node and attempts takeover of services, which could cause data corruption, and a system deadlock or crash. Consider the standby / failover cluster configuration in Figure 4.12, Node A is serving a single database, and Node B is a backup cluster. In the scenario of a heartbeat failure, there are two policy options employed by most cluster products *Pessimistic Assumption* and *Optimistic Assumption* [57].

- 1. Pessimistic Assumption:** Node A is serving the database, but is unaware of Node B's state, so Node A continues to serve the database. Node B is unable to communicate with Node A and, assuming it is down, commences serving the database. This results in both cluster members serving the same database, causing data corruption and a possible system crash.
- 2. Optimistic Assumption:** After a site wide outage, both Node A and Node B boot up at the same time. Because the heartbeat network is not operational, neither node is able to confirm if the other node is alive. In order to avoid data corruption scenario shown in a pessimistic assumption scenario, each node assumes that the other is operational and does not serve the database. This results in a failover cluster configuration with neither node serving the database.

There are situations when a failover from one node to another may be undesirable [53]. One such case is known as a *ping-pong failover*, while the other scenario is known as a *runaway failover*.

Ping-Pong Failover: This occurs when there is a shared disk failure. *NodeA* cannot access the disk, so it shuts down, initiates a failover and reboots. *NodeB* takes over, but also cannot access the disks. Once *NodeA* reboots, *NodeB* fails back to it. This process continues until the System Administrator intervenes.

Runaway Failover: This occurs when *NodeA* fails and *NodeB* takes over. To determine the problem on *NodeB*, the System Administrator initiates a reboot of *NodeA*. Upon rebooting, *NodeB* notices that *NodeA* is operational again, gives up critical resources and fails back over to *NodeA*, before the system administrator has an opportunity to diagnose the problem with *NodeA*.

Both of these scenarios could be avoided if a ‘state locking mechanism’ was used: should *NodeA* failover to *NodeB*, system resources are held by *NodeB* until such time as the System Administrator is satisfied the problem is fixed on *NodeA*, and manually switches back.

An important consideration when implementing a failover cluster is to use compatible systems, rather than, for example, having a Windows server failing over to a Unix server. Even though it may be cost efficient to combine two systems already within an organisation possession rather than purchasing new equipment, according to Marcus & Stern [53] a number of important issues need to be resolved in order for a successful failover to occur:

- A compatible Failover Management System (FMS) must be in place on both systems and this management system must be able to communicate successfully with each other. However there are very few examples of FMS that run on different systems
- The applications would need to function identically on both systems, despite architectural differences. There is the potential for incompatibility with the filesystems (word size, big-endian versus little-endian† *etc*).
- If the Network Identification Cards (NIC) are of incompatible types it introduces added complexity to the heartbeat networks. Consequently, it may be necessary to use a network bridge, which introduces another potential Single Point of Failure (SPOF).

† The terms *big-endian* and *little-endian* are derived from Jonathan Swifts “Gulliver’s Travels”. In computing terms, big-endian and little-endian refer to which bytes are most significant in multi-byte data types, and describe the order in which a sequence of bytes is stored in a computer’s memory. In a big-endian system, the most significant value in the sequence is stored at the lowest storage address (*i.e.* first). In a little-endian system, the least significant value in the sequence is stored first. For example, consider the word UNIX stored in two 2-byte words (UN + IX). In a big-endian system, it would be stored as *UNIX*. In a little-endian system, it would be stored as *NUXI*.

- Complications may arise when one of the systems vendors needs to be contacted regarding a support issue. While the support staff for either system will have experience using combinations of its own servers, it is unlikely that they will have experience of any combination of its own server working in tandem with a rival server.
- The System Administrator needs to be accomplished in the management of both of the server's operating systems, hardware environments and scripting languages.
- Any shared disks between the two systems must also be compatible with both servers.

4.3 Load Balancing

When a system is highly available it will have the ability to failover during a component failure to redundant nodes. Redundant nodes increase reliability by allowing system users to predictably and quickly access computing resources. When implementing a highly available system, a method whereby network / user traffic is directed to all nodes evenly so that no single device is overwhelmed and then redirecting traffic away from a downed node in the event of a failure, is essential [53].

The most popular method of achieving this is *load balancing*. Load balancing is especially important for networks where it is difficult to predict the number of requests that will be issued to a server. Busy Web sites typically employ two or more Web servers in a load-balancing scheme. If one server starts to receive extra requests and its response time becomes slower, requests are forwarded to another server with more capacity (see Figure 4.13)

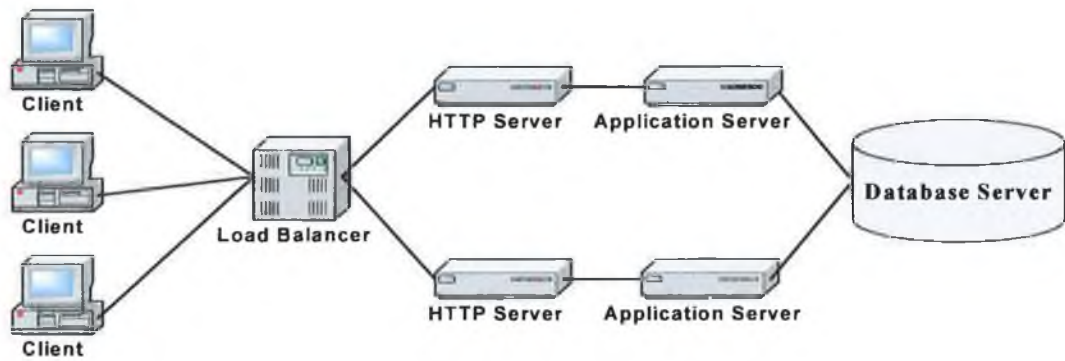


Figure 4.13: Sample Load-Balancing configuration

There are two basic high-level approaches to Load Balancing, a DNS approach and a hardware-based approach.

4.3.1 DNS Load Balancing

DNS (Domain Name System) is an Internet service that translates domain names into IP addresses [59]. It is used primarily because, as domain names are alphabetic, they are easier for users to remember than IP addresses. Each website has its own hostname, and its own interface to the Internet. A DNS hostname can be mapped to a list of multiple IP addresses. When a client connects to a Webserver, the DNS server will cycle sequentially through the list of IP addresses on each lookup of the hostname, so that each time a client resolves the URL it will get the next address in the cycle [58]. This is known as *round-robin DNS*. When a client gets an IP address via round-robin DNS it will use the IP address until the DNS lifetime has expired. A site using round-robin DNS will typically shrink the DNS lifetime to several minutes, which forces the clients to rebind the hostname to an IP address more regularly, and reduces the average window of an outage [53]. Round Robin DNS is a low cost method of load balancing, and is a relatively reliable method of masking failures in a system.

However Round-robin DNS does not manage failovers; it just makes the failover transparent to the user by supplying an alternate server IP address when the client connects to a hostname. It does however have some drawbacks. A reduced DNS lifetime increases the workload of DNS servers and it assumes that clients do not make persistent mappings of IP addresses. Another disadvantage of round-robin

DNS lies in the fact that maintaining state on the server is difficult with this form of load balancing. An extreme case, but a drawback nonetheless is a situation where a four-server configuration is served by round-robin DNS. If every fourth connection is for a large file - while the other three are for much smaller files - one of the servers could be overworked while the remaining three are underworked.

4.3.2 Hardware Load Balancing

Also known as *network redirection*, hardware load balancing avoids the drawbacks of the round-robin DNS approach by operating at the IP level rather than the hostname level. In the hardware load balancing approach, a client connects to the load balancer, which routes the connection to one of the servers behind it. Load-balancing hardware can track the 'health' of each server and avoid sending requests to downed servers; it can also incorporate load information in its load-balancing decisions. Hardware load balancers can store state information for each client so that when a server goes down client information - such as login credentials, shopping cart information *etc* - can be preserved and transferred to a functioning server. Normally a hardware load balancer is replicated for redundancy, and located behind a firewall. Figure 4.14 illustrates an example of a hardware load balancer.

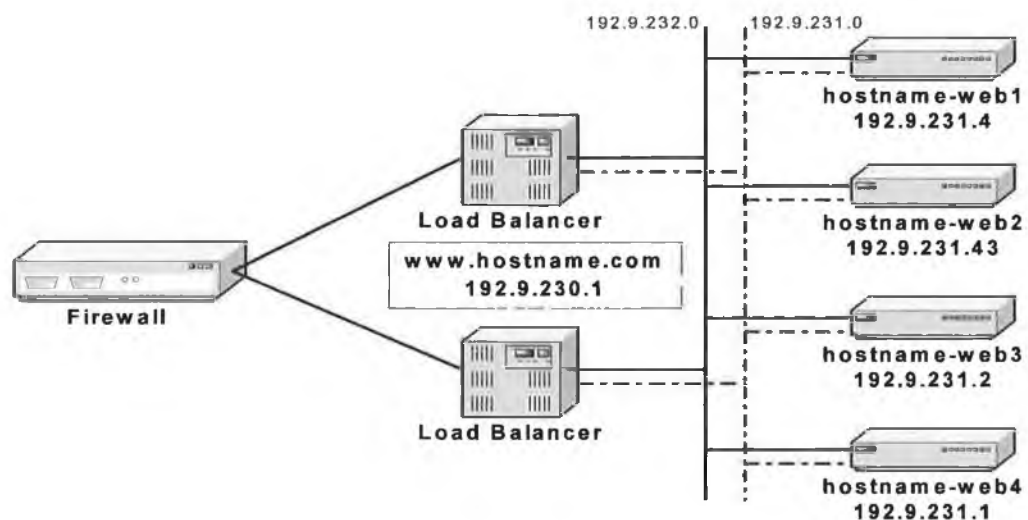


Figure 4.14: Hardware Load Balancing Configuration

Because a hardware load balancer does not need round-robin DNS support it only uses one public IP address. Clients resolve the hostname to this address and are then connected to the appropriate back-end server based on load, time, equal distribution *etc.*

The advantages of load-balancing hardware:

- Wider choice of load-balancing algorithms.
- Health monitoring allows load balancer to skip downed servers, resulting in increased response times.

The main drawback to hardware load balancers is that they are significantly more expensive than the round-robin DNS approach.

4.4 Scalability

Scalability is a measure of the ability of hardware or software systems to adapt to changing demands. For example, a scalable network system would be one that can start with just a few nodes but can easily expand to thousands of nodes. Scalability means not just the ability to operate but to operate efficiently and with adequate Quality of Service (QoS), over the given range of configurations [60]. In today's business environment, organisations must be able to dynamically increase capacity to meet changing demand. The scaled-up system should exhibit increased computing power proportional to the increase in resources. However, it is worth noting that a scalable system should also be able to scale downwards, or reduce its resources to reduce costs. Additionally, an *unscalable* system is defined as a system where the additional cost of coping with increases in traffic or size is excessive or that it cannot cope at this increased level at all [61]. Cost is not just limited to monetary terms but can also encompass response time, processing overhead, space or memory. An unscalable system adds to labour costs or negatively impacts the QoS.

4.4.1 Scalability Types

Bondi (2000) identifies four types of scalability: *Load Scalability*, *Space Scalability*, *Space-Time Scalability*, and *Structural Scalability* [61]. A system or system component may have more than one of these attributes, and two or more types of scalability may mutually interact.

1. **Load scalability.** A system is load scalable if it exhibits the ability to function without unnecessary delay and without unproductive resource consumption at light, moderate, or heavy loads, while at the same time properly utilising available resources.
2. **Space scalability.** A space scalable system is one where the system or applications memory requirements do not grow to unbearable levels as the number of items it supports increases. Various programming techniques can be used to achieve space scalability, including compression. However, because compression takes time, it is possible that space scalability may only be achieved at the expense of load scalability.
3. **Space-time scalability.** A system is space-time scalable if it continues to function competently as the number of objects it encompasses increases by orders of magnitude. It is space-time scalable if its internal data structures and algorithms continue to operate to smooth and speedy operation regardless of its size. For example, a search engine that is based on a linear search is not space-time scalable, while one based on an indexed or sorted data structure such as a hash table or balanced tree is.
4. **Structural scalability.** A structurally scalable system is one where its implementation or standards do not impede its growth, or will not do so within a chosen time frame. This is a relative term, because scalability depends on the number of objects of interest now relative to the number of objects later. Any system with a finite address space has limits on its scalability given that the limits are inherent in the addressing scheme. For example, a packet header field typically contains a fixed number of bits. If the

field is an address field, the number of addressable nodes is limited. Load scalability may be improved by modifying scheduling rules, avoiding self-expansion, or exploiting parallelism.

4.4.2 Scalability Architectures

Hwang (1998) identified three scalable architectures, which have much in common but have increased levels of resource sharing [46]. The first of these is the *shared-nothing* architecture, which consists of a number of nodes connected by an interconnection network. In this configuration, the data on one server is replicated, via a network, to the other server / servers. Each node can contain more than one processor. The shared-nothing architecture is detailed in Figure 4.15.

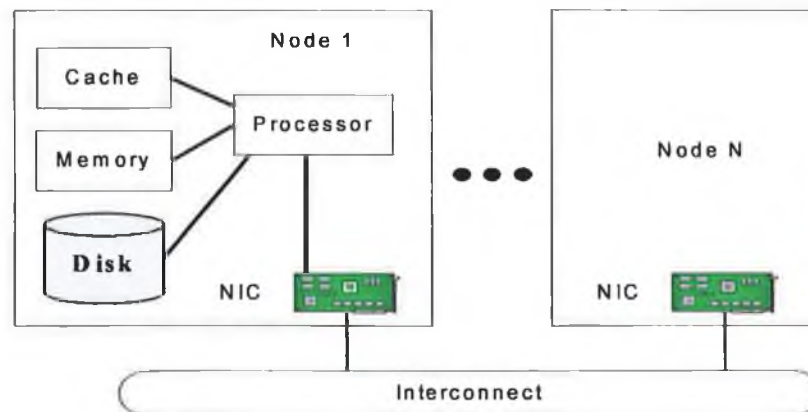


Figure 4.15: Shared-nothing Architecture

The shared-nothing architecture is optimal for remote failover. For local failover, when the systems in question are in close enough proximity to be connected by Small Computer System Interface (SCSI) and / or Fibre Cables, the *shared-disk* architecture is a superior and more reliable architecture [53]. The shared-disk architecture has one main difference from the shared-nothing architecture. In this approach, the disk module where the critical application data resides is moved out of the individual nodes and is shared among all nodes. The unshared, local disk contains the operating system, as well as any other files required by the Node to initiate and maintain the failover process (see Figure 4.16).

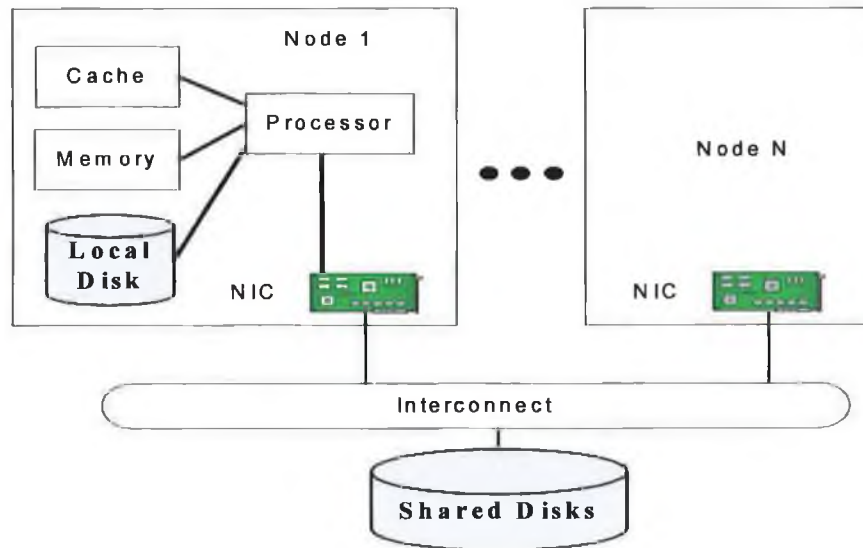


Figure 4.16: Shared-disk Architecture

In the *shared-memory* architecture main memory is also shared among the various nodes (see Figure 4.17). As with the shared-disk architecture, each node also has access to its own local, non-shared, private memory. The disadvantage to the shared-memory architecture is when one processor caches memory that it needs fast access to. Whenever one cache is updated with information that may be used by the other processors, the change must be replicated to the other systems or else all processors will be working with different data.

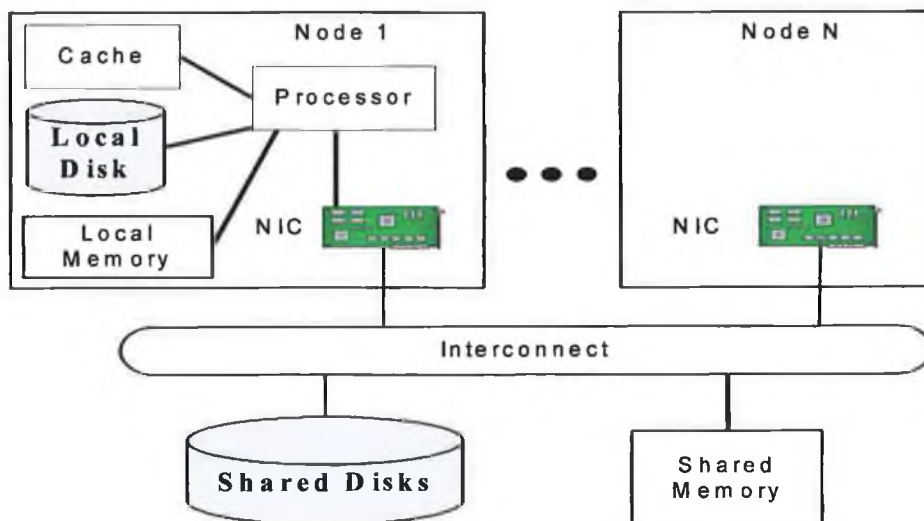


Figure 4.17: Shared-memory Architecture

4.4.3 Design Principles for Scalable Systems

Introducing scalability to an existing system is considerably more difficult than designing a scalable system from scratch. To this end, Hwang (1998) documented three design principles, that when used, will considerably reduce the complexity inherent in building scalable systems [46]. The three principles are:

- 1) Principle of *Independence*
- 2) Principle of *Balanced Design*
- 3) Principle of *Design for Scalability*

4.4.3.1 Principle of Independence

The underlying premise of this principle is that when designing a system, the individual components of the system should be independent of each other. If complete independence is not achievable, then the dependence that some components have with other components should be as small and as clear as possible. The main advantage to having independent components is that it greatly increases the possibility of *Independent Scaling*. Independent scaling (also known as *Incremental Scaling*) is achieved by scaling a system along one dimension by improving one component, independent of the other components. There is no requirement to simultaneously, or subsequently upgrade any of the other components. An example of independent scaling would be in the case of an extra node being added to a system where the existing nodes are running on the Windows 2000 Server operating system. Even though the new node may be running on Windows 2003 server there would be no need to upgrade the operating system of the existing nodes. There is however one major downside to independent scaling: an efficient system cannot be built just by scaling up one, or a few, components alone. An example, consider scaling a system by adding a faster processor. If the existing communications systems or memory is too slow for the newer processor, then the system is unbalanced.

4.4.3.2 Principle of Balanced Design

As if to acknowledge the stated downside to independent scaling, the second design principle is focused on designing a system that minimises any

performance bottlenecks. A bottleneck can occur when a relatively slow component reduces the performance of the overall system, even though the remaining components are fast. Equally, *Single Points of Failure* (as discussed in section 4.1.1) should be eliminated in order to design a balanced system.

One method of ensuring a balanced design is by following Amdahl Law [62]. For example, Figure 4.18 shows an application program which is divided into two types of computational structure: part *X* and part *Y*. Combined, the two components take $X\% + Y\%$ of the total execution time



Figure 4.18: Total Execution Time

If part *X* is improved to run n times faster, the speedup S is defined.

$$s = \frac{\text{Original Time}}{\text{Improved Time}} = \frac{1}{(X/n)\% + Y\%} \rightarrow \frac{1}{Y}$$

For example: consider a system which has an execution time of *X* equal to 60% and *Y* equal to 40%. If *X* were improved to run 3 times faster, the speed up of the system would be 0.0167%.

$$0.167\% = \frac{\text{Original Time}}{\text{Improved Time}} = \frac{1}{(60/n)\% + 40\%} \rightarrow \frac{1}{60}$$

The implications of the above computations show that:

- The methods of component *X* will execute more often, and is a good candidate for optimisation (*i.e.* speed up the common areas).
- The optimum speedup of the application has an upper boundary of $1/Y$.
- The slow component *Y* is the bottleneck. *Y* should be made as small as possible.

- As illustrated above, in order to achieve a relatively insignificant improvement of 0.0167%, it was necessary to improve X to run 3 times faster. There may be a trade-off between the resources required to improve an application, and the improved execution time.

4.4.3.3 Principle of Design for Scalability

When designing a system, scalability should be the main objective from the start of the design process, rather than as an afterthought at the end. The design should have provisions so that the system can scale to achieve greater performance, or scale downwards to allow greater cost-effectiveness. There are two approaches to designing for scalability:

Overdesign: As the name implies, upon design the system should not only satisfy the minimum requirements it was designed for, but also include additional components that allow for scaling in the future. While the extra components may seem superfluous at the design stage, they should allow for a smooth migration into future scaled up systems. For example, if an organisation needs an 8-CPU (Central Processing Unit) server, it is recommended that a 16-CPU server be purchased, and only 8 CPUs be installed into the server [53]. If an 8-CPU server is purchased and fully utilised, when the organisation needs to scale up to 16 CPUs, it may have to buy an additional server. It is important to note that overdesign may be more costly upfront, but should result in greater long-term savings.

Backward Compatibility: In some ways, backward compatibility would seem to be the opposite of overdesign. In this approach, the requirements of a scaled down system are taken into consideration. An example would be a system with a new, faster processor. The new processor should still be able to execute code and run applications designed for older, slower processors. It is important to note that when designing for backward compatibility, not all components of the old system need to be kept – obsolete components should be discarded.

4.5 Disaster Recovery

Disaster Recovery (DR), and *Disaster Recovery Planning* (DRP), consists of a set of activities aimed at reducing the likelihood - as well as limiting the impact - of disaster events on critical business processes [63]. The ability to quickly recover client data after a disaster is becoming an increasingly important component in delivering high levels of availability in an ASP environment. A disaster is an event that causes an interruption of mission critical information services to a firm. Normally, in a disaster situation, users are aware that an outage has occurred. The duration of the outage is mainly dependent on the recovery solution, which can be measured by two different components [50]:

- **Data Loss:** This represents the loss of data an organisation has *i.e.* how much work must be re-executed once the system is recovered.
- **Service Loss:** This represents the loss of computing experienced from the moment of disaster up to the moment when a system has been recovered.

However, it is worth noting that, what constitutes a disaster for company A, may not necessarily be a disaster for company B [64]. For example, a hard disk failure on a PC might be a disaster for a small firm if that PC managed the firms' accounts, but might not be a disaster to a much larger organisation. Indeed, a Disaster Recovery plan suitable for a large organisation may not be achievable for many Small and Medium-sized Enterprises (SMEs). Many SMEs simply do not have the resources for Disaster Recovery – from the hardware / software required to implement a Disaster Recovery plan, to the properly trained IT personnel to execute the plan in the event of a service outage.

Regardless of size, when implemented correctly, a proper DR plan allows for a quick restoration of an organisations' IT services, by making a backup of servers and files critical to the organisation and quickly restoring those files in the event of a disaster. A good example of an extremely efficient disaster recovery plan was seen in the wake of the September 11th 2001 terrorist attacks on the cities of New York and Washington. The New York Board of Trade (NYBT), whose office was located

adjacent to tower 2 of the World Trade Center in Lower Manhattan, New York, suffered complete buildings and systems destruction. However, due to a comprehensive DR plan, by 8pm that evening the NYBT was ready to resume trading [53]. While the NYBT was a commendable example of a good Disaster Recovery plan, it was estimated that prior to September 11th, only one in five Companies in the New York area had a disaster recovery plan [65].

While a DR plan is essential, there are two main factors that hamper the disaster recovery effort [66]. Firstly the daily growth of business information, results in more and more data to be backed up. Due to the increasing need for services to be operational twenty-four hours a day / seven days a week / three hundred and sixty five days a year, the optimal time available to backup data has shrunk dramatically. Secondly, customers expect services to resume rapidly after a business disruption - regardless of the circumstances. From a financial and resource standpoint, larger organisations are better equipped to provide speedy recovery than small or mid-size organisations. There are two main factors that support this assumption. Firstly, many small and mid-size organisations have little or no dedicated IT personnel to enable them to respond quickly to business interruptions. Secondly, it is common for smaller organisations to house all of their business data on one server, with the result that if that server goes down, all business operations cease until the server is fully restored.

4.5.1 Full Vs Incremental Backups

Marcus & Stern [53] contend that backups are the heart of any design of critical systems. Handled properly, they represent the last line of defence against just about any catastrophe. Backing up files means copying files to a second medium, which can be a disk or a tape, as a precaution in case the first medium fails. Because even the most reliable computer will break down at some stage, it is vitally important that files are backed up regularly. Ideally, at least two copies of the backups are made. One copy should be stored close to the servers that are backed up. In the case of a disaster like a hard disk crash, the backups are on hand for quick recovery. A copy of the backup should be stored off-site, in a different location to the servers. In the event of a critical failure where the servers and backups are

destroyed, having a copy of the backup in a different geographic location means that not all data is lost. Many companies may take daily backups but might just make copies for off-site locations only once or twice a week or less frequently. It is important that the copies of backups are up to date, unless the organisation can cope with losing a week, or months, worth of business data. Some especially paranoid organisations make a second copy of the backups and store them in different countries or continents, to guard against a catastrophic countrywide disaster but the cost of implementing this may be prohibitive to most organisations.

It is recommended that every file, on every system, should get backed up, regardless of how trivial the file may seem. The general rule of thumb is that if a file could ever be needed in the future, or if it would take time for a user to recreate data in that file, then the file should be backed up. Special care should be taken to ensure that hidden files, and System registries, are also backed up. However it is not enough to just backup an organisations server. It is also essential that a backup include desktops and especially laptops. It is estimated that up to 60% of all critical data is stored on laptops [53]. Given that laptops are portable, and as such are susceptible to breakage, being dropped, theft *etc*, it is especially important that their contents are backed up.

A full backup is a backup where every bit of data on an organisations system is copied to backup media. However a full backup of a system is time intensive, and copies all data regardless of whether it has changed since the last backup. An alternative to daily full backups is an *incremental* backup, which only backs up the data that have been modified since the previous backup. There are two approaches to incremental backups:

Cumulative Incremental: In this backup, all data that has changed since the last full backup is backed up.

Differential Incremental: In this backup, all data that has changed since the last differential, or full, backup is backed up.

Differential backups are faster than cumulative backups as they backup less data. A typical incremental backup schedule is detailed in Figure 4.19 (Adapted from [67]).

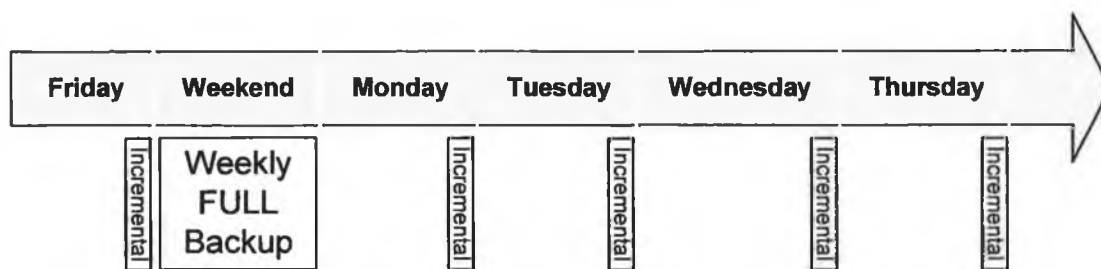


Figure 4.19: Incremental Backup Schedule

While daily full backups are more time intensive, their main advantage is evident in the aftermath of a disaster. With a full backup only one set of backup media are generated. In the event of a disaster, only one full restore operation needs to be carried out in order to completely re-establish an organisations data system. However, with incremental backups, there is more than one set of backup media. When restoring incremental backups, the last full backup needs to be applied first, followed by each additional incremental backup in sequence. For example, consider an organisation that perform a full backup on the 1st of every month, and then perform an incremental backup every other day of the month thereafter. If their system were to suffer a disaster on the 30th of the month, in order to complete the restore process they would need to apply 1 full backup, and 29 incremental backups. This is a time consuming exercise, and can be prone to mistakes.

4.5.2 Commercial DR Software Features

There are many commercial software products available that aid backup and restore operations. Regardless of the supplier, the following are desirable features in a commercial DR product.

4.5.2.1 Complete Hardware Usage

As discussed in section 4.5, due to the frequent continuous operation of some applications, the available window for data backups can be quite small. Therefore, an important feature of a commercial DR product is the optimal usage of backup equipment. For example, if backup media had a maximum write speed

of 1.5 Megabytes per second, but the DR product in use could only write to the media at a speed of 1 Megabyte per second, then this is an inefficient use of hardware. Good backup software should drive the hardware to its maximum capacity.

4.5.2.2 Hot Backups

In an ideal world, an organisation should be able to backup files and databases without taking them offline, and without a performance degradation to the user. This is known as a 'Hot Backup'. However, in a production system, the demand for continuous operation means that the window of time available for backups has diminished considerably [63]; backups may be difficult if users are accessing files while a backup is taking place. Many products advocate temporarily taking a system offline prior to a backup, in order to get the data into a consistent state, making a backup, and then bringing the system back online. While the system is down for backup, users will not have full access to the data. In most cases, read access will still be granted, but write access will be revoked for the duration of the backup. Ideally the time in which the data is in read only mode should be minimised as much as possible to avoid disruption to users. Another solution to backing up files in use is to only back up the files *not* in use, and *log* any open files. When the file, database table *etc* next becomes stable, it is then backed up.

4.5.2.3 Open Tape Format

In a disaster situation, the system administrator, or person charged with carrying out the systems recovery operation should be able to read and restore tapes without needing specialised or proprietary hardware or software. If however the software required for restoration needs a licence key for operation, it is vitally important that this information is stored with the backup files and is replicated in any off site location. In the event of the total destruction of an organisations building, existing licence keys may also be lost. Any delays in obtaining software or licence keys will increase the systems Mean Time to Repair (MTTR) [53].

4.5.2.4 Centralised Management

It should be easy to administer an entire backup environment from a single console, rather than from dispersed consoles throughout the organisation. Many

products also offer a Web enabled management console, which allows remote, centralised management of storage servers from any location worldwide. This helps reduce expenses and operating errors and simplifies administration. Centralised management also allows for the generation of reports that are considerably more accurate than reports collated piecemeal from scattered locations.

4.5.2.5 Quick Disaster Recovery

Some DR products require rebuilding their media databases or catalogs before a post-disaster recovery can begin. In this scenario, should the catalog rebuilding also require pre-reading every tape in the library, it can add hours, if not days, to the recovery process. Therefore it is vitally important that an organisation knows and understands the entire process involved with system recovery, not just the process involved in making backups.

4.5.2.6 Hardware Support and Flexibility

Implementing a site-wide backup and restore environment can be very cost intensive, and the DR budget may not stretch to brand new tape hardware, or even in some cases to brand new tapes. Therefore, it is important that a commercial DR product has backward compatibility with existing backup hardware.

4.5.2.7 Mature Products with Reference Sites

It can be a good idea to purchase products that are well established, with a proven track record of successful implementation, rather than new or start-up solutions. Given that established products tend to come from larger companies, these companies may provide telephone support, on site support, informative websites, user groups, user conferences *etc.* Because of these established support structures, it may be easier to get assistance from many diverse viewpoints. Equally, mature products have generally been tested more rigorously, and improved upon, over time.

4.5.2.8 Multiple Platform Support

When purchasing DR products, it should be possible to purchase one solution that can back up and restore all major platforms, such as Windows, NetWare, UNIX (including Solaris, HP-UX, AIX, Tru64), Linux (including Red Hat, SuSE, Turbo Linux), Max OS X, VMS *etc.* An organisation should not need to purchase one backup solution for PCs running Windows, and another for Solaris boxes, and a third for your Novell servers *etc.*

4.5.3 Seven Tiers of Recoverability

The Seven Tiers of Recoverability is a guideline to Disaster Recovery, whereby seven tiers of recoverability were ranked based on the recovery method used and recovery time taken after a disaster [50]. The IBM association *SHARE* is an independent, volunteer-run association, providing IBM customers with user-driven education and resources to make enterprise-computing specialists more effective professionals [68]. *SHARE* has been in existence since 1955, shortly after IBM released its first computer, when a group of interested IT professionals decided to band together to 'share' ideas about how best to install and implement IBM's new release. At the 1992 *SHARE* conference, the *Automated Remote Site Recovery Task Force* presented seven tiers of recoverability:

4.5.3.1 Tier 0 – No Offsite Data

This tier provides the lowest level of Disaster Recovery preparation. Very little planning is made for saving or replicating information, gathering DR requirements, the establishment of a backup hardware platform, or development of a 'Plan B'.

Typical Recovery Time: The length of time from disaster to recovery at this tier can be unpredictable. In some cases, recovery may be impossible.

4.5.3.2 Tier 1 - Pickup Truck Access Method (PTAM)

At Tier 1 (Figure 2.20 adapted from [50]) an organisation has developed a (limited) contingency plan, has backed up required information and stored this information at an off-site location.

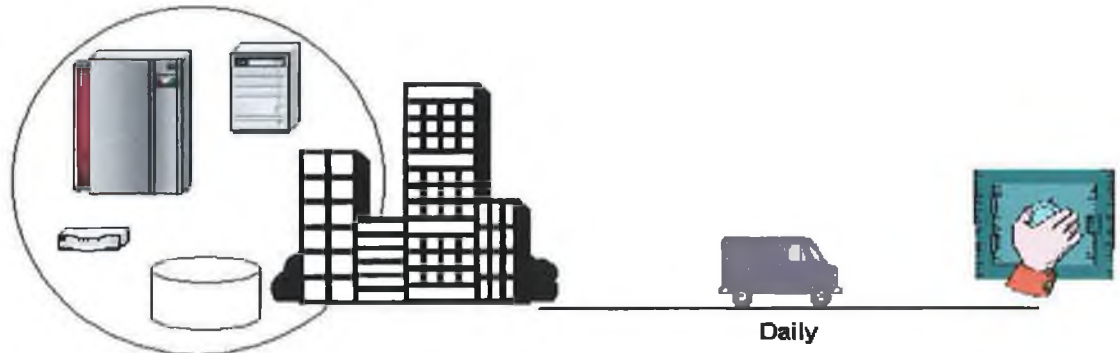


Figure 4.20: Tier 1 Recovery Solution

The organisation had begun to determine recovery requirements and may, at this stage, have established a backup site.

Typical Recovery Time: The length of time from disaster to recovery at this tier is usually more than a week.

4.5.3.3 Tier 2 – PTAM & Hot Site

Tier 2 is made up of the requirements of Tier 1 as well as a backup platform with sufficient hardware and network support for the organisations critical business applications. See Figure 4.21 adapted from [50].

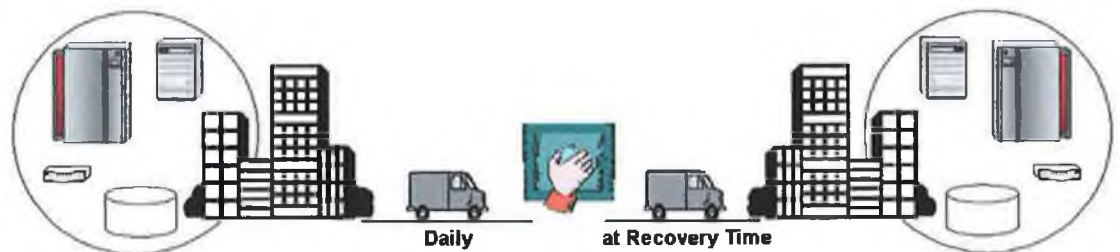


Figure 4.21: Tier 2 Recovery Solution

Typical Recovery Time: The length of time from disaster to recovery at this tier is usually more than one day.

4.5.3.4 Tier 3 - Electronic Vaulting

Tier 3 is made up of the requirements of Tier 2, as well as support for electronic archiving of some of the organisations critical information. The receiving hardware must be physically separate from the primary site and the data stored for recovery after a disaster. Figure 4.22 adapted from [50].

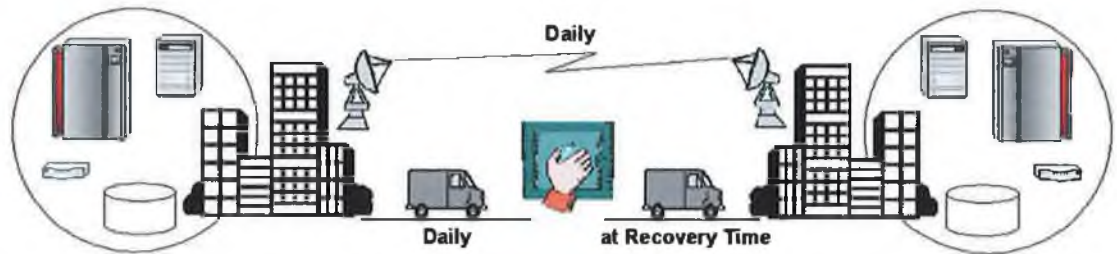


Figure 4.22: Tier 3 Recovery Solution

Typical Recovery Time: The length of time from disaster to recovery at this tier is usually one day.

4.5.3.5 Tier 4 - Active Secondary Site

A CPU at the recovery site, as well as bi-directional recovery makes up tier 4, which is a follow-on from the requirements of Tier 3, as well as the introduction of active management of the recovery data. The receiving hardware must be physically separated from the primary platform. Figure 4.23 adapted from [50].

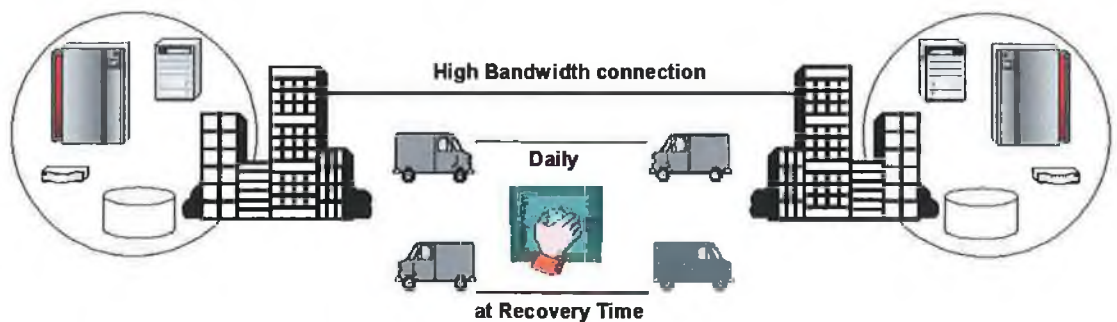


Figure 4.23: Tier 4 Recovery Solution

Typical Recovery Time: The length of time from disaster to recovery at this tier is usually up to one day

4.5.3.6 Tier 5 - Two Site Two Phase Commit

Tier 5 is made up of the requirements of Tier 4, as well as allowing database updates to be applied to both the local and remote copies of the databases with a single commit. A commit is not completed until both the primary and secondary locations are updated. Tier 5 requires hardware on the secondary platform with the ability to automatically accept the workload of the primary site during an outage. Figure 4.24 adapted from [50].

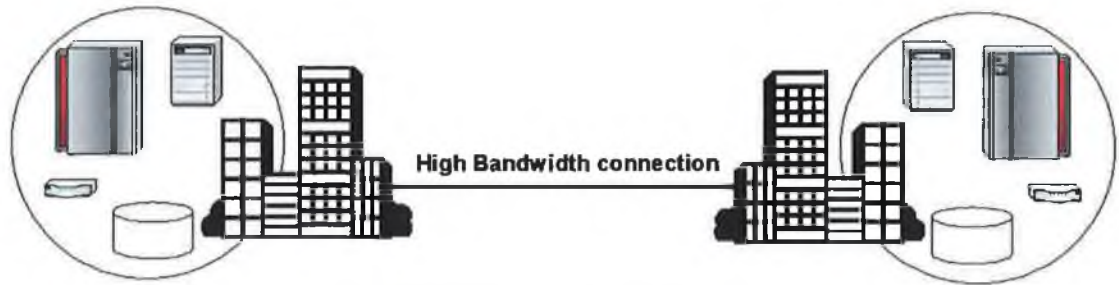


Figure 4.24: Tier 5 Recovery Solution

Typical Recovery Time: The length of time from disaster to recovery at this tier is usually less than 12 hours.

4.5.3.7 Tier 6 - Zero Data Loss

At Tier 6, organisations exhibit zero data loss in the aftermath of a disaster, due to the immediate and automatic transfer to the secondary site. Figure 4.25 adapted from [50].

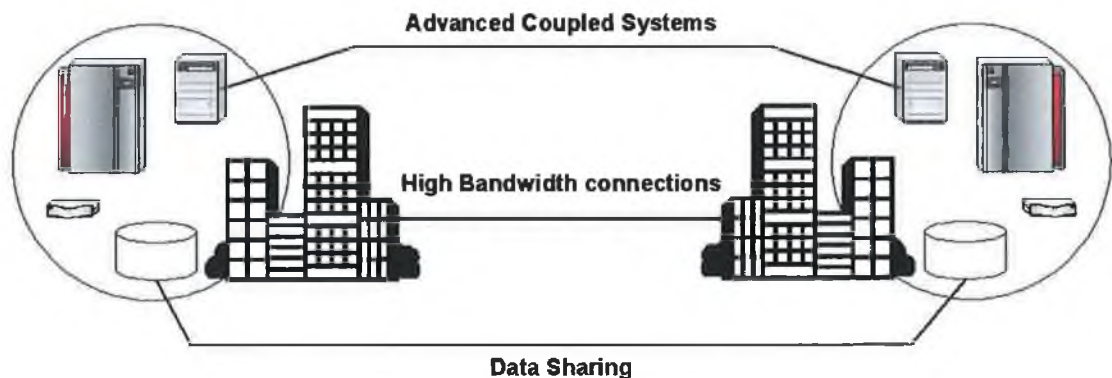


Figure 4.25: Tier 6 Recovery Solution

Typical Recovery Time: The length of time from disaster to recovery at this tier is usually a few minutes.

As with various availability paradigms, the ability of an organisation to achieve Tier 6 recovery is directly proportional to the amount of money the organisation is prepared to invest in Disaster Recovery. Figure 4.26 (Adapted from [50]) illustrates the data loss and service loss aspects of each tier of the recovery solution.

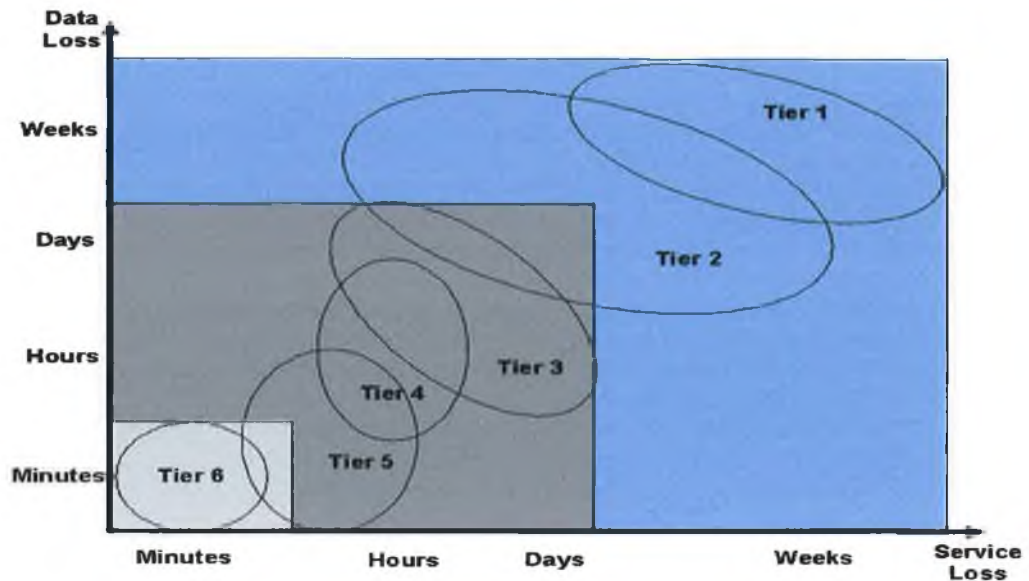


Figure 4.26: Data Loss & Service Loss

4.6 Singl.eView and Billing4Rent

The Billing4Rent (B4R) project provides the real world platform on which recommendations or proposals arising out of this body of research will be implemented. The B4R project will be built upon the Singl.eView [69] billing engine from Intec. Singl.eView is a large-scale enterprise solution, for Tier1 and Tier2 service operators – as an example of Singl.eView’s scale, Intec recently (2005) won a US\$15 million (€11.9 million) contract with a leading African operator to provide a billing system to support over 7.7 million pre and post-paid subscribers [70]. Accordingly, the following summary details are necessary to provide an overview of Singl.eView - its relevance will be shown in the *Thesis Contribution* section of this dissertation.

Singl.eView is a scalable, highly available billing and rating solution that allows service providers to design, deliver, and bill the products and services their customers subscribe to. Singl.eView provides modules for rating, discounting, and bill production. Rating is the process in which events (*i.e.* telephone calls, internet usage *etc*) are converted into rated events, using tariffs that have been defined for a particular service or customer. Billing is the process whereby rated events are processed to generate billing information. Generally the result is a set of invoices, which can either be sent electronically to the customer or printed for postal delivery.

Singl.eView's 'out-of-the-box' configuration is an enterprise wide solution designed to manage the billing and rating of large-scale global Tier1 and Tier2 service providers such as Deutsche Telekom, Virgin Mobile *etc.* However, the costs associated with implementing Singl.eView are prohibitive for many smaller Tier3 and Tier4 service providers. Consequently, the Billing4Rent project proposes to modify Singl.eView to deliver a hosted billing solution, which can be accessed on a subscription / rental basis (see Figure 4.27).

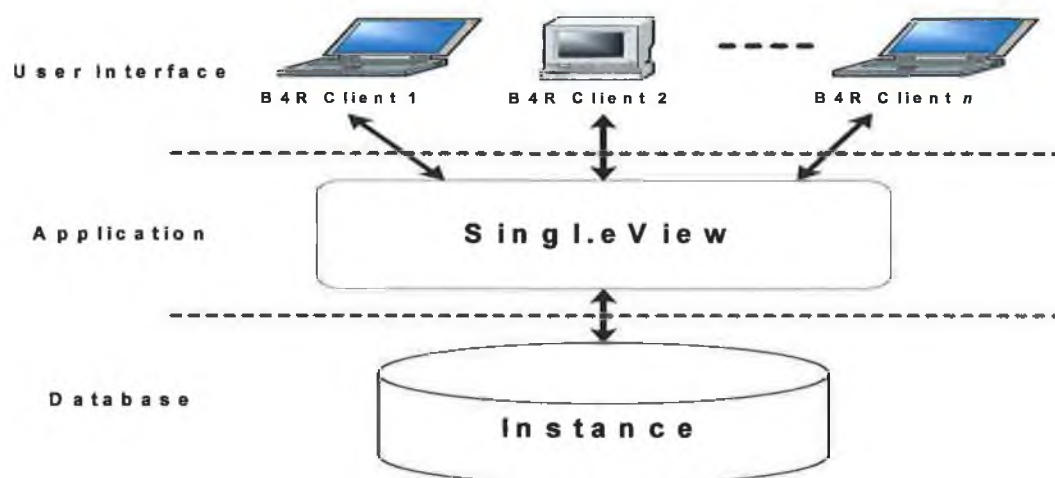


Figure 4.27: Billing4Rent / Singl.eView n-Tier Architecture

In the past, Singl.eView has been targeted at Tier1 and Tier2 operators. Billing4Rent will allow Intec to focus on lower tier operators and hence move into a new market segment. As more and more networks are moving towards consolidated IP based technology [36] (witness the recent emergence of VOIP phone calls over broadband Internet, or streaming video on mobile handsets), as well as the constant change and evolution of mobile phone pre and post payment packages, a secure cost-effective converged billing solution is essential for service providers of any size. As a measure of the enormous potential of the Billing4Rent solution, the 'Innovation Partnership' proposal document [1] conservatively estimated a monthly, recurring revenue stream of € 2,500,000. This figure is broken-down as follows:

Number of Clients subscribing to Billing4Rent:	500
Number of Customers per Client:	5000
Monthly rental fee per Customer:	€1
Potential monthly revenue for Billing4Rent:	€2,500,000

4.6.1 Singl.eView Components

Figure 4.28 illustrates the components of Singl.eView (adapted from [71]).

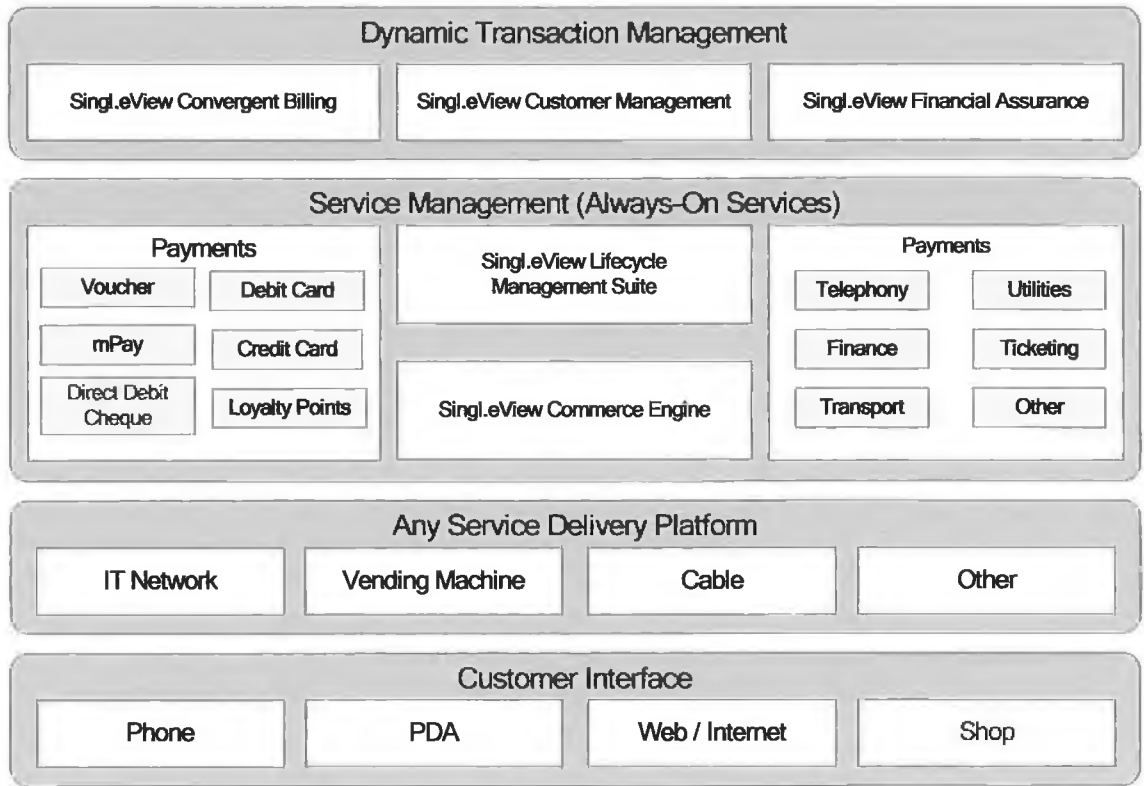


Figure 4.28: Singl.eView Components

The following points illustrate important Singl.eView components:

4.6.1.1 Convergent Billing

The rating and transaction engine within the Convergent Billing module is the most important module in Singl.eView. It manages transactions (*i.e.* phone calls, broadband usage) in real-time using business rules specified by the service providers. Convergent Billing features:

- Only one solution is necessary, regardless of product, service, delivery network, customer type, or payment method. For example, this allows a provider offering phone and Internet service to utilise a single billing solution.
- The flexibility to process and rate transactions in real-time, or as a batch in times of low-usage.

- Real-time processing that not only protects transactions from errors and fraud but also allows accurate up-to-the-minute information on revenue and customer accounts.
- SingleView is expression-driven, which means service providers can easily configure the solution to grow and evolve with changing business needs.

4.6.1.2 Customer Management

Customer Management is an application allowing clients to store all customer information in one place. It allows clients to create and manage all customer information, from Prospect Management (initial customer contact with client company) and Marketing Campaigns, through to Treatment and Collections management. Customer Management features include:

- **Campaign Management:** facilitates tailoring Marketing Campaigns to target specific consumer groups, potentially increasing uptake and reducing the cost of sales.
- **Sales Support:** during interactions with customer support staff, real-time guidance (in the form of screen-pops) can be given to agents on opportunities for selling new products, or premium versions of existing products.
- **Contact Management:** a comprehensive contact history is stored regardless of the customer interaction (phone / email / web), which ensures that follow-ups are targeted, for reduced time-to-sale and increased customer confidence.
- **Payment Assurance:** provides complete functionality for Payments, Collections, Adjustments, invoice disputes and Promises-to-Pay.
- **Analysis And Reporting:** provides built-in or ad-hoc reporting and analysis to enable complete management of customers, and the clients organisation overall.

4.6.1.3 Financial Assurance

Financial Assurance provides clients with a complete view of customer and supplier transactions, enabling them to better manage their business, as clients will have complete and up-to-the-minute data. Financial Assurance is a reporting and error management application consisting of three main features:

- A reporting framework to create and modify data collections and database views.
- A process in which business users can report on underlying data enabling service providers to minimise lost revenue and maximise return on investment by monitoring and controlling revenue streams in real-time.
- A single user interface to prioritise, test and bulk-reprocess event errors.

4.6.1.4 Lifecycle Management Suite

The suite provides an XML based catalogue, tools, and wizard interfaces to populate and manage product information.

- **SingleView Catalogue:** The catalogue is a repository for all SingleView product data. SingleView Lifecycle Management Suite allows any suppliers (a client has) to publish to the product catalogue, enabling the supplier to validate, modify, and deploy their products as integrated offerings with the clients' own products.
- **SingleView Workbench:** The workbench is the client front-end, which utilises wizards to set up common products. The wizard is used to capture variable information. Using a '*Build Once, Deploy Many*' model means that intricate business processes are automated for future use.
- **SingleView Configuration Tools:** These tools manage the catalogue data, including the processes for it to be transferred from one environment to another. It also provides the tools to create the wizards that are used in the workbench.

4.6.1.5 Commerce Engine

Singl.eView's Commerce Engine allows service providers to offer the same level of service regardless of product type: prepay or post-pay mobile, dial-up or broadband Internet. Singl.eView Commerce Engine features:

- **Authentication And Authorisation:** Real-time authentication of users; their account balances and eligible services is determined up-front during service encounters.
- **Real-Time Rating And Discounting:** Singl.eView Commerce Engine works in conjunction with the Rating and Transaction Engine to calculate charges, and ensure the customer has the necessary funds. Fixed charges can be applied to discrete functions (*e.g.* downloading games), while incremental charging can be used for time or other variable dependent transactions.
- **Balance Management:** Customer balances are checked throughout a transaction to ensure there is available credit. If the customer has multiple accounts, they can select which account is to be used.
- **Payments And Settlements:** Singl.eView Commerce Engine provides a variety of payment methods (banks, credit cards, post-paid and prepaid accounts).

4.6.2 Singl.eView Availability and Scalability

Singl.eView supports scalability by allowing application server processes to be split across multiple servers. Processes such as billing, reporting, invoice generation *etc* can be run in parallel, resulting in increased processing performance and efficiency.

In terms of High Availability, Singl.eView supports clustering, component redundancy and data replication. Redundant components and data replication allow upgrades and changes to be made without loss of service. Similarly, replication

allows the failover from a master server to a backup server during a service outage, with a transparent continuity of service to the end users. Finally, SingleView also supports Oracle RAC (Real Application Clusters). Oracle RAC allows multiple computers to run the Oracle RDBMS software simultaneously while accessing a single database [72].

4.7 Chapter Summary

While chapter three presented an overview of Application Service Provision in general, the aim of this chapter was to present a picture of High Availability, Scalability and Reliability, and their importance not just in an ASP environment but also for any application environment. In a survey conducted by the ITAA of key user expectations with respect to Application Service Provision, over 80% of respondents cited guarantees on network reliability as a very important feature of Service Level Agreements (SLAs) between ASP and clients [15]. Consequently, this chapter documented the origins of High Availability and gave an overview of the metrics used to evaluate the availability of an application to its end users. It was shown that many organisations have a desired level of 99.999% application availability. This level of availability (commonly referred to as 'five nines') equates just 31.5 seconds downtime in a year, or only 605 milliseconds a week!

Also examined in this chapter was clustering. Clustering is the process in which two or more machines are connected together in such a way as to act like a single computer. Clusters can automatically detect and recover from server or application failures, allowing routine planned maintenance without the need for server or application downtime.

This chapter examined Scalability - the ability of a system to maintain or increase performance under an increased load when resources are added. The drivers of scalability were highlighted, as well as the four types of scalability (*i.e.* load, space, space-time and structural). Common scalability architectures and design principles were also covered.

An important area of High Availability - Disaster Recovery - was also addressed in this chapter. The merit of full versus incremental backups was discussed, as was the (desirable) features of commercial Disaster Recovery Products. Also illustrated was the *Seven Tiers of Recoverability* (a tiered approach to Disaster Recovery) advocated by the IBM association SHARE, where each tier was ranked based on the recovery method used, as well as the time taken for recovery. Finally, this chapter provided an overview of Singl.eView (the transaction management platform from Intec), which will provide the backbone of the Billing4Rent ASP service. Singl.eView is a billing and rating solution that allows service providers to design, deliver, and bill the products and services their customers subscribe to. The various components that make up Singl.eView were examined, as well as Singl.eView's ability to support scalability (by allowing application server processes to be split across multiple servers) and high availability (by supporting clustering, component redundancy and data replication).

SECTION THREE



Research Contribution

Chapter Five: Proposed Framework

This chapter describes the proposed ASP framework for Billing4Rent. The Billing4Rent (B4R) project provides the platform on which recommendations or proposals arising out of this body of research will be implemented, and allows these proposals to be objectively analysed and quantified. Consequently, an online B4R Billing ‘prototype’ was developed in order to incorporate all these recommendations and allows these proposals to be analysed and quantified in an objective manner. This chapter will address the following areas:

- The functional requirements for developing the Billing4Rent project will be outlined, as well as the technical objectives and challenges.
- An overview of the B4R prototype. The prototype consists of a B4R client website, where clients can add, remove, update customers, products / services, and invoices.
- The design and development of Availability and Scalability components for the B4R ASP will be detailed and objectively examined. An ‘Administrative Interface’ was created in order to administer Billing4Rent and to incorporate these availability and scalability components. The Administrative Interface allows authorised B4R personnel to create and manage user accounts for clients. Potential clients must contact Billing4Rent, and their eligibility for subscribing to B4R services is evaluated against any criteria for joining B4R deemed necessary. From the Administrative Interface, authorised personnel can create users, view statistics relating to B4R storage media (*i.e.* Database or Filesystem) usage, as well as view audit and error logs and launch a Monitoring program which collects statistics about Billing4Rent such as page hits, page completion times etc
- Recommendations for Availability and Scalability of the B4R ASP will be documented: a Proposed Network Architecture, Code Optimisation for Improved Performance and Disaster Recovery Guidelines.

5.1 B4R Functional Requirements

The following functional requirements are adapted from the Billing4Rent ‘Innovation Partnership’ proposal document [1]. The goal of the Billing4Rent project is to build an ASP hosted billing service platform – to be known as the *Musketeer Platform* – based on *SingleView* components, as well as web services technologies. The W3C [73] define Web Services as follows

“Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and / or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.”

Billing4Rent will be designed to allow functionality to be added during several iterations and will allow service providers to:

- Automatically generate and test tariffing schemas for their products.
- Deploy and manage tariffing schemas via a web-based User-Interface (UI).
- Upload customer usage records to Billing4Rent via secure web services interfaces.
- Perform rating and discounting, electronic billing and Payments.
- Monitor and generate reports on usage and earnings.

Successful development and deployment of Billing4Rent will require a number of technical challenges to be addressed. The main challenges, as outlined by the ‘Innovation Partnership’ are:

- **Usability:** One of the reasons cited for the adoption of the ASP business model (outlined in Section 3.3) is the elimination of the need for Small / Mid-Sized service operators to employ IT staff with specific expertise in the installation, configuration and maintenance of complex billing products. Because of this, actual end-users of the Billing4Rent service may not have experience using billing software. Therefore, in order to make Billing4Rent a

success, it is important that Billing4Rent includes intuitive, yet flexible user interfaces. Accordingly, Billing4Rent proposes to design stateful web-based user-interfaces for the creation / configuration of tariffing schema, invoice layouts, report contents / presentation and other configurable elements.

- **Error Handling:** As with any service where financial data is generated and utilised, it is vital the data from which invoices are generated is correct. The Billing4Rent ASP services must include robust error handling to detect and rectify a wide range of faults and error conditions. In addition, the user interfaces must be carefully designed to ensure that incorrect or inappropriate configurations are not applied to the service.
- **Dynamic Tariffing Schema:** A desired feature of a billing service is the ability to implement dynamic tariffing schema which can be automatically adjusted in reaction to service usage.
- **Security:** Customer usage data is an important source of billing information for a service provider; therefore, the transfer of such data securely to an external ASP will raise significant security-related concerns (*e.g.* How will the transfer of data from the client to Billing4Rent across the Internet be secured? How will the integrity of customer data stored by Billing4Rent be ensured?)
- **Availability & Performance:** Availability and Performance of the Billing4Rent ASP service will be a key issue for potential clients. Due to the large volumes of data to be processed - and the potential revenues involved - there must be near continuous uptime of the Billing4Rent ASP while the service should also operate correctly and within an acceptable response time.

5.2 Billing4Rent Prototype

The Billing4Rent development team comprised of five members, with two located in Galway (at the Galway-Mayo Institute of Technology (GMIT)), and three located in Waterford (at the Telecommunications Software & Systems Group (TSSG)). The GMIT team concentrated their efforts on the research and development of two areas identified by the project proposal as being relevant to the ASP model: *Security* and *Performance*. In parallel to these efforts, the TSSG team had responsibility for developing the Billing4Rent User Interface (UI), as well as customising SingleView to support multiple clients and their customers. Due to the logical separation of both teams, the efforts of the GMIT team was analogous to a ‘black box’ (*i.e.* the GMIT team worked in isolation, and developed components to *plug into* the commercial B4R release). How this work was completed was of little consequence to the second B4R team. The GMIT team developed a web-based B4R prototype in order to incorporate components and recommendations relating to both Security and Performance. The following sections details this prototype, and the related Performance, Availability and Scalability components and recommendations.

The ASP web-based billing prototype was developed using Java, Java Servlets and Java Server Pages (JSP) [74]. The prototype has an Oracle 9i RDBMS [75] backend to store client and administrative data and runs in a Jakarta Tomcat [76] web-container. The prototype was developed using the Model / View / Controller (MVC) design pattern†. MVC consists of three components:

- **Model:** Holds all data, state and application logic.
- **View:** Provides a presentation of the model.
- **Controller:** Defines the way the UI reacts to user inputs.

Billing4Rent encompasses all three of the above components. However these components are decoupled to increase flexibility and reuse [77] – for example, the B4R prototype allows the Oracle database to be interchangeable with another data

† A design pattern relates to the use of a solution (or a partial solution) that solves a design problem that keeps occurring across projects. A design patterns purpose is to codify existing design knowledge so that developers do not constantly ‘re-invent the wheel’.

storage medium *etc.* Figure 5.1 shows the Billing4Rent prototype home page.

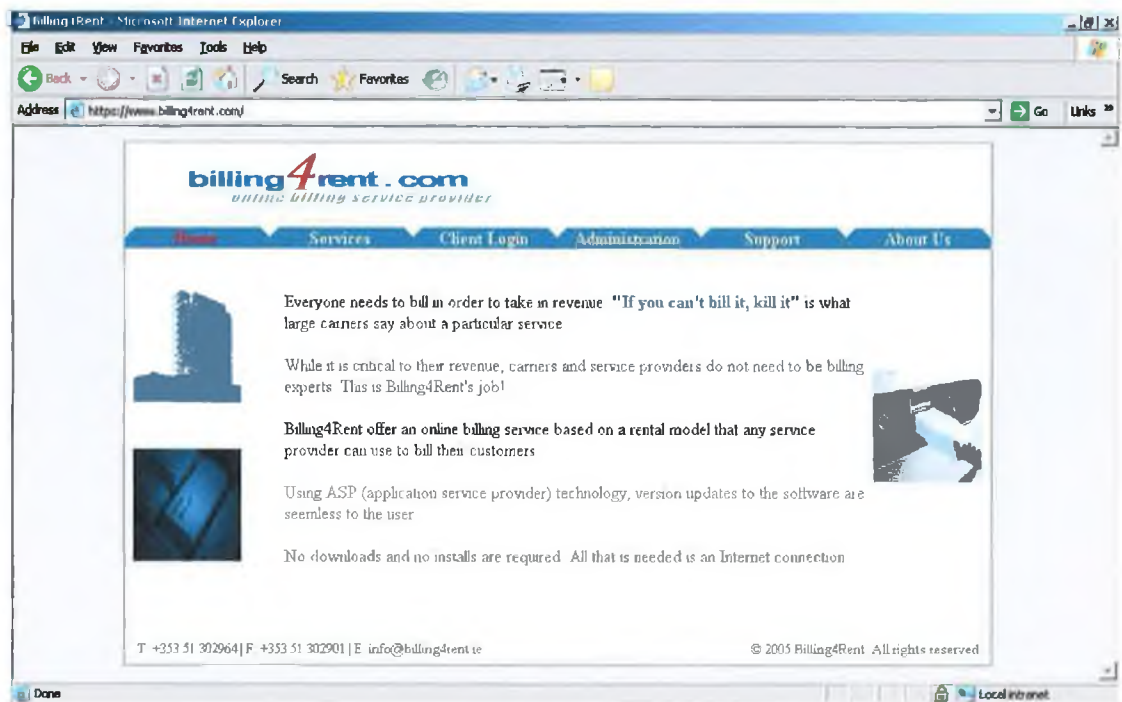


Figure 5.1: B4R Prototype Home Page

The Billing4Rent client site provides details about Billing4Rent (including a biography of the organisation, latest news, staff vacancies *etc.*). The site also provides a client login page, which clients can use to access their data, as well as a link to the Billing4Rent 'admin' interface. In order to subscribe to B4R, the prototype provides a 'contact us' page (Figure 5.2), which lists the various methods available to contact Billing4Rent.



Figure 5.2: B4R Contact Details

This allows an authorised officer of Billing4Rent to appraise potential new customers to ensure they meet any criteria B4R deem necessary for joining. Only at this point are clients given a username and password. Clients can then gain access to Billing4Rent via the 'Client Login' page (Figure 5.3).

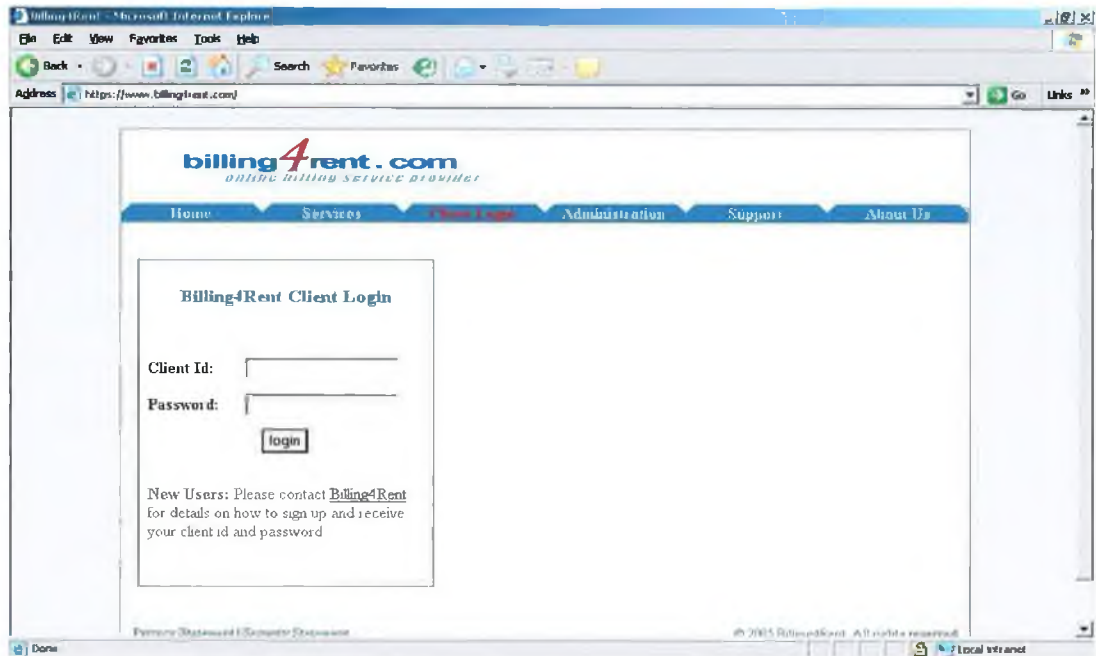


Figure 5.3: B4R Client Login

After logging in, clients are presented with the client home page (Figure 5.4)...

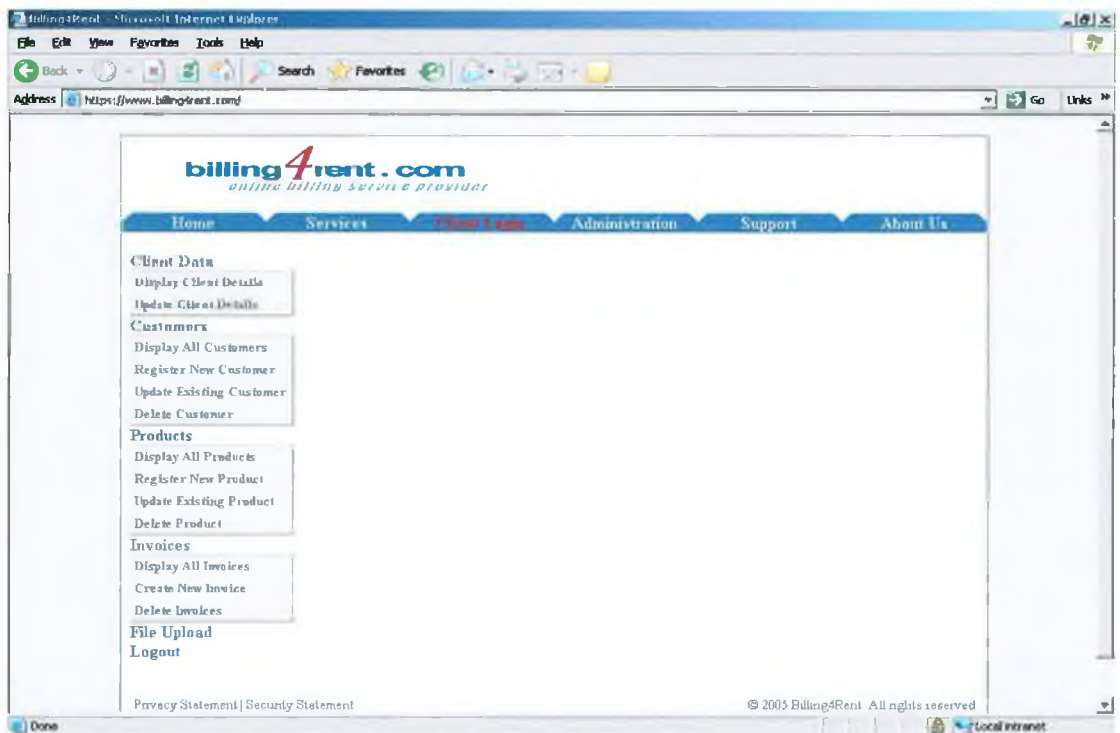


Figure 5.4: B4R Client Home

The home page provides links from which clients can perform various actions (see Appendix 4 for a selection of B4R prototype screenshots):

- 1) **Display Client Details:** This option displays all details for the currently logged in client.
- 2) **Update Client Details:** By selecting this option, the client can change some, or all, of their recorded details including their address, phone number, e-mail address, password *etc.* The only field that cannot be changed is the *Client ID*, which is unique to each client.
- 3) **Display All Customers:** A client can display a list of all of the customers who subscribe to their products or services. By clicking on an individual customer, the client can retrieve more detailed information for that customer, including address, phone number, e-mail address *etc.*
- 4) **Register New Customers:** This screen allows the client to add new customers and record appropriate contact details for that customer.
- 5) **Update Existing Customers:** This option allows a client to update details relating to one, or more, of their customers. By selecting the appropriate customer from the dropdown list, details (including changes of address or phone number) can be recorded.
- 6) **Delete Customers:** By selecting this option, a client can delete all details for any customer(s) who no longer subscribes to the clients products / services. The client is then asked to confirm his / her intention to delete a customer prior to the customers details being completely removed from Billing4Rent systems.
- 7) **Display All Products:** This shows a summary of all products or services the client currently provides. This screen shows the product ID, name, unit price, and the product status. B4R allows products to be enabled or disabled, where enabled products are products or services the client is currently providing, and disabled products are products which the client is not providing at the present time, but may do so again in the future (*i.e.*

a seasonal product). By clicking on the product name, further details relating to that product are shown: a brief description of the product, tax applicable on the product, the invoice string (the description of the product as it appears on an invoice), and the General Ledger (GL) code (code corresponding to the General Ledger account type involved *i.e.* Assets, Liabilities, Revenue *etc.*).

- 8) **Register New Product:** By selecting this option, a client can create a new product or service they can provide to their customers. The following details are recorded: Product ID, name, a brief description of the product, the invoice string (the description of the product as it appears on an invoice), unit price, tax applicable on the product, the General Ledger (GL) code and the product status (*i.e.* disabled or enabled).
- 9) **Update Existing Products:** This option allows a client to update details relating to one, or more, of their products. The client initially selects the product to be updated from a dropdown list and is then presented with a page where all details relating to a product can be changed: name, description, invoice string, unit cost, tax, GL code and profile. The only field exempt from being updated is the *Product ID*, which is the unique identifier for that product.
- 10) **Delete Products:** By selecting this option, a client can delete all details for any product(s), which will no longer be provided by the client. The client is then asked to confirm his / her intention to delete a product prior to the product details being completely removed from Billing4Rent systems.
- 11) **Display All invoices:** This option allows a client to display details of previously generated invoices for any of the client's customers. The client initially selects the appropriate customer from a dropdown list and all invoices for that customer stored on Billing4Rents systems are displayed in summary form. Fields shown are customer ID, invoice ID, invoice date, invoice total and the Purchase Order (PO) number. By

clicking on the invoice ID, full invoice details are displayed onscreen. The invoice lists the name and address of the client, as well as the name and address of the customer the invoice relates to. As on the summary screen, the invoice ID, invoice date, and the PO number are shown. The bottom section of the invoice details the products / services being billed for. For each product, the product ID, name, description, quantity used and product cost is listed. All products are subtotalled, and the aggregate tax for all products is shown. The final invoice total is shown at the bottom.

- 12) Create New invoice:** By selecting this option, a client can create a new invoice for a particular customer. The client initially selects the appropriate customer from a dropdown list. The client is then presented with a form with a unique invoice ID, the customer ID, as well as a field for entering the PO number relating to the invoice. A dropdown list displays all the products the customer is subscribed to. Using the dropdown list and the quantity used field, the client can add products to the invoice, as well as remove products if a mistake is made. No products can be added twice to a single invoice. All products are subtotalled and the aggregate tax for all products is shown as products are added or removed. The final invoice total is shown at the bottom.
- 13) Delete invoice:** By selecting this option, a client can delete some, or all, invoices for a customer. The client selects the appropriate customer from a dropdown list before all invoices for that customer is displayed. The client is then asked to confirm his / her intention to delete a invoice prior to the invoice details being completely removed from Billing4Rent systems.
- 14) File Upload:** Choosing this option allows the client to upload a logo image file to Billing4Rent. This logo can then be used to 'brand' the clients organisation or services and can then be utilised for web page customisation and on generated invoices.

- 15) **Logout:** When a B4R client selects the logout button their current session is invalidated.

5.3 B4R Administrative Interface

In order to perform administrative tasks and to incorporate and test availability and scalability components, an 'Administrative Interface' was developed. The Administrative Interface is a separate website, also developed using Java, Java Servlets and JSP (and implemented using the MVC design pattern). The Administrative Interface allows authorised B4R personnel to create and manage user accounts for clients. A potential client wishing to subscribe to Billing4Rent must first contact B4R in order for their eligibility for joining to be evaluated. Additionally, authorised B4R personnel can also create users, view statistics relating to B4R storage media (*i.e.* Database or Filesystem) usage, as well as view audit and error logs. Finally, from the Administrative Interface, users can launch a Monitoring program which collects statistics about B4R such as page hits, page completion times etc. At present, the Administrative Interface is launched (in a new browser window) via a hyperlink on the B4R prototype client site (Figure 5.5)

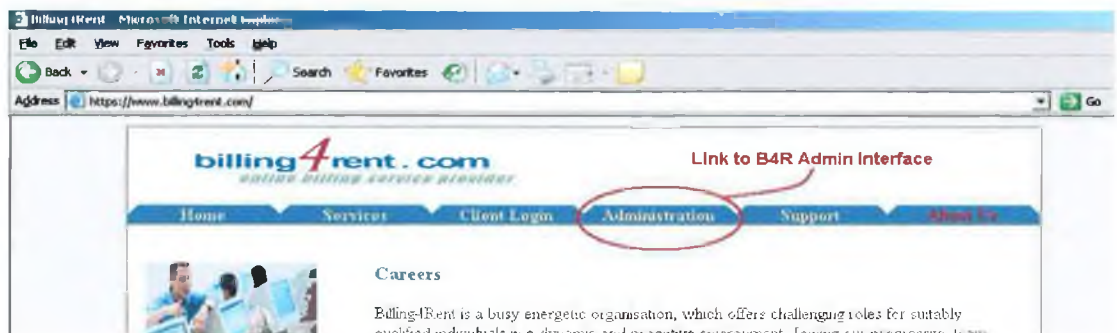


Figure 5.5: B4R Administrative Interface Hyperlink

When launched, the Administrative Interface displays a login dialog (Figure 5.6).



Figure 5.6: B4R Administrative Interface Login

When users are successfully logged in, the home page is displayed (Figure 5.7)

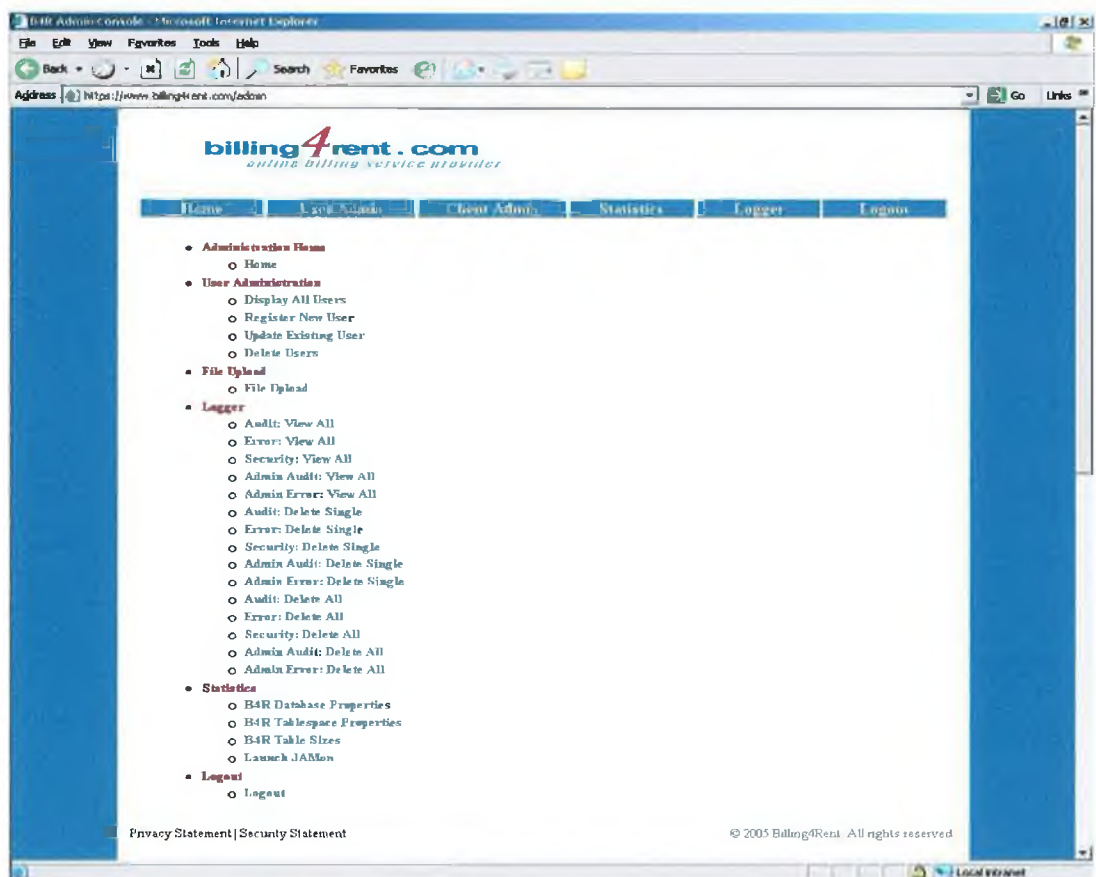


Figure 5.7: B4R Administrative Interface Home Page

The home page provides links from which clients can perform various actions (see Appendix 5 for all B4R Administrative Interface screenshots):

1) **Users Menu:**

- **Display All Users** – When this option is selected, the user can view a list of all B4R Users with their full name, username, role, credential and profile. Profile indicates whether the user is currently active or inactive. There are currently two profiles in B4R, ‘admin’ and ‘user’. ‘Admin’ profile allows full access to the Administrative Interface. ‘User’ profile allows the viewing of Administrative Interface pages, but does not allow the user to make any updates or alterations (Figure 5.8).

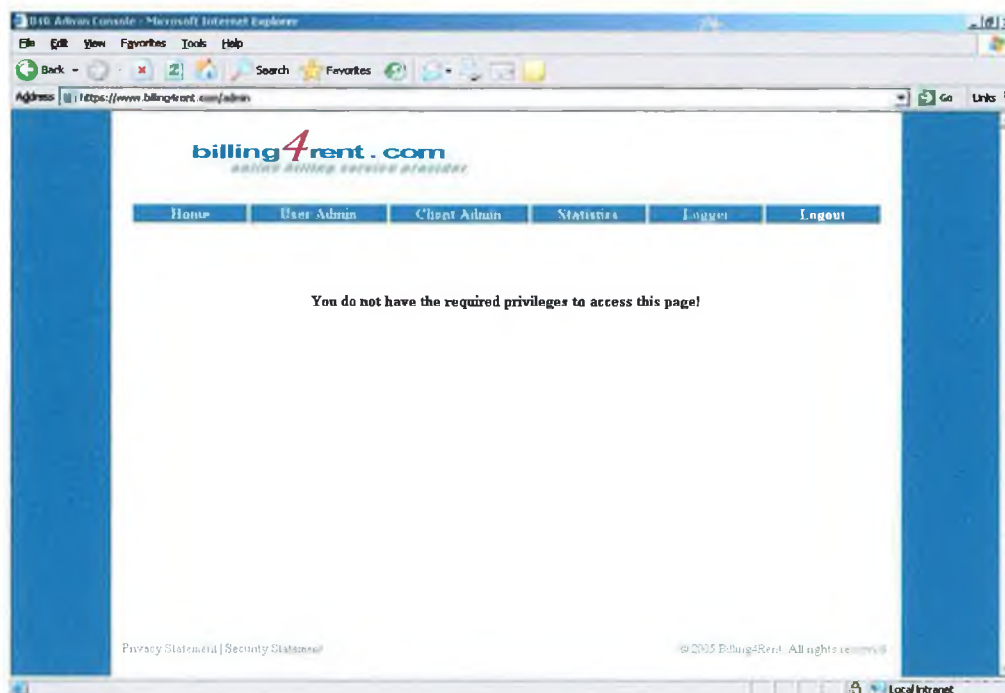


Figure 5.8: No Privileges

- **Register New User** – This screen allows a B4R user with an ‘admin’ role to create a new user and record appropriate details for that user including name, username, password, role, credential and profile.
- **Update Existing User** – By selecting this option, an ‘admin’ user can change some, or all, of any users recorded details. The user is initially presented with a dropdown box containing the usernames of all users on the B4R system. When a user is selected, details for that user including

their name, password, role, credential and profile can be amended. The only field that cannot be changed is the *Username*, which is unique to each user.

- **Delete Users** – By selecting this option, ‘admin’ users can delete all details for any user(s). A confirmation dialog is shown prior to the users details being completely removed from Billing4Rent systems.

2) **Client Menu:**

- **Display All Clients** – This page displays a summary of all clients currently subscribed to Billing4Rent. This screen shows the name, username, company details, role, credential and profile. B4R allows clients to be enabled or disabled, where enabled clients are currently receiving B4R services and disabled clients are clients whose accounts are currently disabled. By clicking on the client name, further details relating to that client are shown: company name, address, postcode, email address and contact phone number.
- **Register New Client** – A potential new B4R client initially contacts Billing4Rent and that client is then evaluated as to their suitability. If a client is deemed suitable, their details are recorded on this page where the following are recorded: name, username, company details, role, credential and profile.
- **Update Existing Client** – This option allows a user to update details pertaining to one, or more, B4R clients. By selecting the appropriate client from the dropdown list, details for that client can be amended.
- **Delete Clients** – By selecting this option, details for any client(s) who no longer subscribes Billing4Rent can be selected for deletion. The user is then asked to confirm his / her intention to delete prior to the clients details being completely removed from Billing4Rent systems.

3) Storage Statistics and Performance Monitoring Menu:

- **B4R Database Properties** – This screen displays statistics relating to the database used for storing Billing4Rent data. As SingleView runs on an Oracle database, Oracle was also selected as the database for the GMT B4R online billing Prototype and Administrative Interface. The properties shown for the database are: instance name, database version, the date and time the database was started as well as the current status of the B4R database (Figure 5.9).

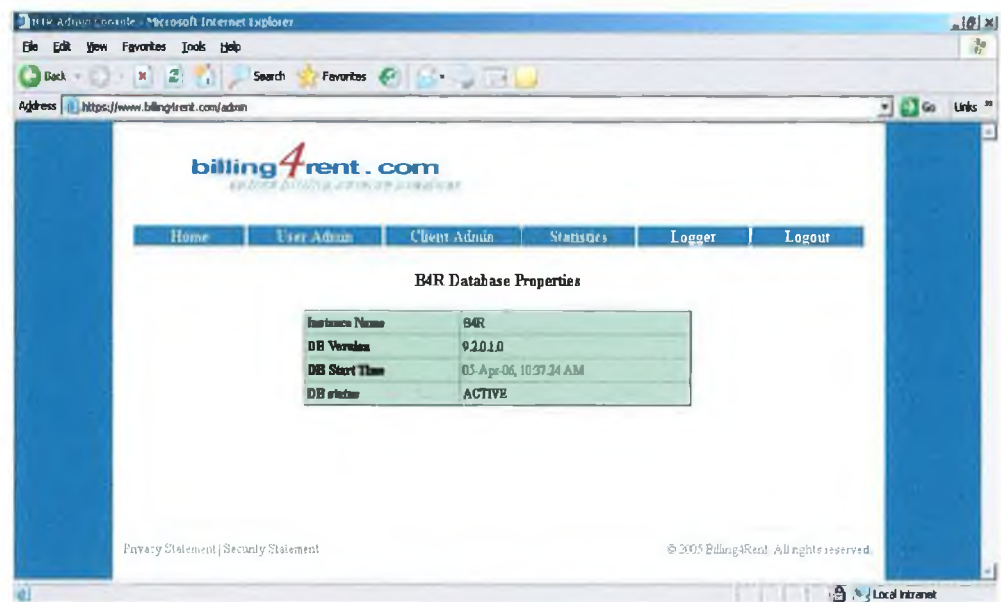


Figure 5.9: Database Properties

- **B4R Tablespace Properties** – A database is divided into Logical Storage Units called tablespaces. This screen displays details relating to the B4R Tablespace used for storing Billing4Rent data. The tablespace properties shown are: name, size, size used, size remaining, and the percentage tablespace used (Figure 5.10).

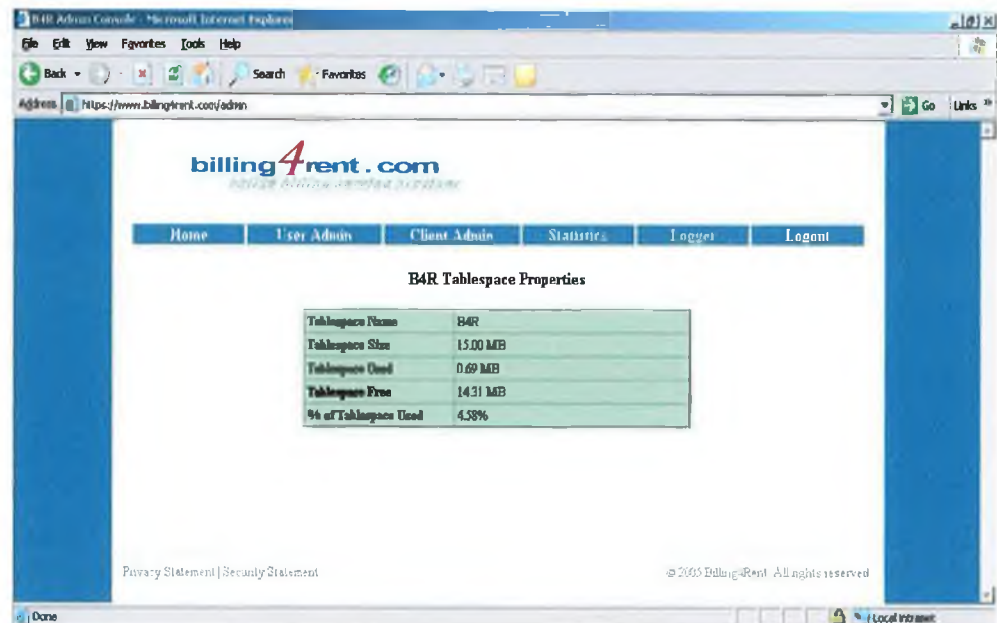


Figure 5.10: Tablespace Properties

- **B4R Table Sizes** – This page displays a list of all eleven tables used for storing B4R data, both client specific and admin. The number of rows in each table is shown (Figure 5.11).

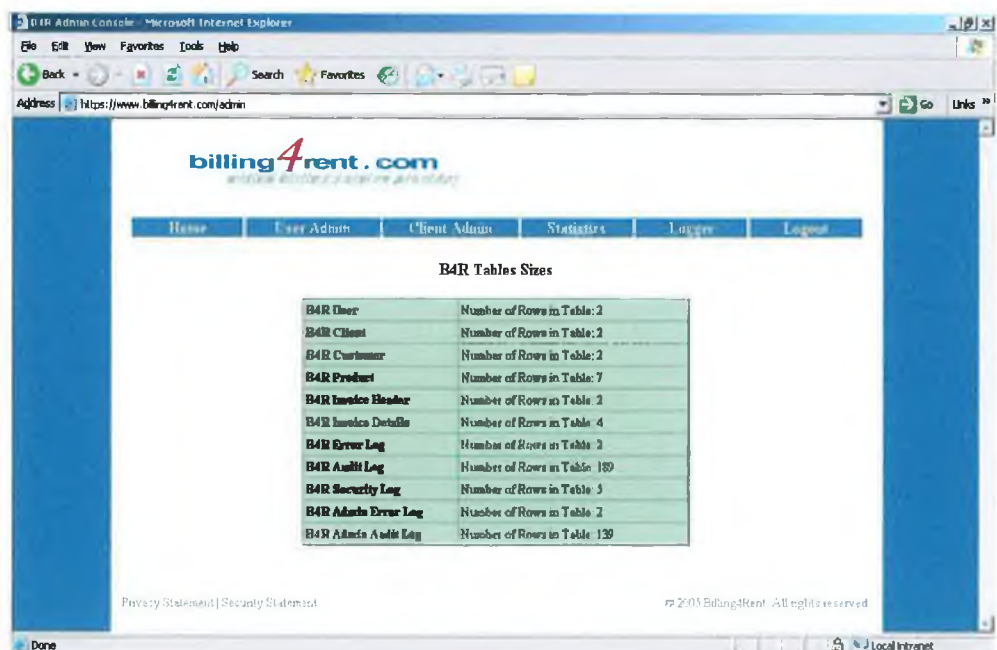


Figure 5.11: Table Sizes

- **Launch JAMon** – The Java Application Monitor (JAMon) [78] is a free Java Application Program Interface (API), which allows the monitoring of production applications (Figure 5.12).

URL	Hits	Total Time	Avg Time	Min Time	Max Time	Concurrent	First Access	Last Access	Errors	Status	Last Update
JSP . b4r/home.jsp	1	10	10	0	10	10	0	1	0	0	0
JSP . client/CustomersDetails.jsp	1	10	10	0	10	10	0	1	0	0	0
JSP . client/DisplayClient.jsp	1	0	0	0	0	0	0	1	0	0	0
JSP . client/DisplayCustomers.jsp	2	25	50	21	10	40	0	1	0	0	0
JSP . client/Home.jsp	1	0	0	0	0	0	0	1	0	0	0
JSP . client/login.jsp	1	10	10	0	10	10	0	1	0	0	0
JSP . client/MyClient.jsp	1	30	30	0	30	30	0	1	0	0	0
JSP . client/UpdateClient.jsp	1	10	10	0	10	10	0	1	0	0	0
SQL . SELECT * from BARCUSTOMER where CLIENTID	2	15	30	21	0	30	0	1	0	0	0
SQL . SELECT * from BARCUSTOMER where CUSTOMERID	1	20	20	0	20	20	0	1	0	0	0
SQL . SELECT * from BARCLIENT where CLIENTID	2	0	0	0	0	0	0	1	0	0	0
SQL . Update BARClient	1	40	40	0	40	40	0	1	0	0	0
Servlet . Client CheckId.jsp	1	0	0	0	0	0	0	1	0	0	0
Servlet . Client CheckPrivilegedAction	13	3	40	4	0	10	0	1	0	0	0
Servlet . client/CustomersDetails	1	300	300	0	300	300	0	1	0	0	0
Servlet . client/DisplayClient	1	341	341	0	341	341	0	1	0	0	0

Figure 5.12: JAMon Performance Monitor

JAMon can be used to identify application performance bottlenecks, user / application interactions, track application scalability *etc.* JAMon gathers statistics such as hits, execution times (total, average, minimum, maximum and standard deviation), as well as concurrency information (*i.e.* simultaneous application requests). Performance monitoring, as well as Logging on the Billing4Rent Prototype is entirely configurable. Performance monitoring and logging can be turned on / off by setting a field in the *B4R.properties* file (Figure 5.13).

```

db.url=jdbc:oracle:thin:@localhost:1521:B4R
db.user=SYSTEM
db.pass=research
b4r.debug=true
b4r.monitorServlet=true
b4r.monitorSQL=true

```

Figure 5.13: B4R.properties

The properties file is read by the B4R application and, depending on the setting - logging and monitoring of B4R is performed. For example, in each servlet, the *B4R.properties* file is read and the value for **monitorServlet** (either *true* or *false*) is stored in a *Boolean* variable. At the start of each servlet execution, the following code checks the value of that variable and monitors the specified code if monitoring is set to *true*:

```
import com.jamonapi.*;

public class ClientCustomerDetails extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        final String MODULE = "ClientCustomerDetails";
        boolean monitorServlet = false;
        Monitor mon = null;

        try {
            // Check if Monitoring is enabled
            if (session.getAttribute("monitorServlets") != null) {

                // Retrieve monitorServlet value from session obj
                String strMonServ = (String) session.getAttribute("monitorServlets");

                // Convert monitorServlet string to boolean
                monitorServlet = Boolean.valueOf(strMonServ).booleanValue();
            }

            // If monitoring Servlets is enabled, start the servlet monitor for this servlet
            if (monitorServlet)
                mon = MonitorFactory.start("Servlet - clientCustomerDetails");
                //
                //      CODE BEING TIMED
                //
        }
        catch (Exception e) {
            //
            //      EXCEPTION HANDLING
            //
        }
        finally {
            if (monitorServlet)
                mon.stop();
        }
    }
}
```

JAMon gathers statistics for any code that comes between the *start()* and the corresponding *stop()* method. The code `MonitorFactory.start("Servlet - clientCustomerDetails")` creates a monitor, with the label *Servlet - clientCustomerDetails* and begins gathering monitoring statistics.

Summary statistics are gathered for all monitors, which are passed identical labels.

JAMon can also be configured to gather statistics without altering existing B4R code. For example, JAMon can gather statistics for all JSP pages in Billing4Rent, including the number of hits a page receives, the page execution times *etc.* In order to take advantage of this feature, *JAMon.jar* should be placed in the Webserver's classpath, with the following code inserted into the B4R *web.xml*:

```
<web-app>
  <display-name>Billing4Rent Performance Monitor</display-name>
  <filter>
    <filter-name> JSP Filter</filter-name>
    <filter-class>com.jamonapi.JAMonFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>JAMonFilter</filter-name>
    <url-pattern>/*.jsp</url-pattern>
  </filter-mapping>
  .
</web-app>
```

- 4) **Logger Menu:** extensive debugging / logging of Billing4Rent can also be configured via the *B4R.properties* file (see Figure 5.13, page 104). The *B4R.properties* file is read and the value for **debug** (either *true* or *false*) is stored in a *Boolean* variable. When an action is performed, and debugging / logging is enabled, the action is logged.

```
if (debug){
    String auditmsg = "SQL insert complete";
    log.addToLogger(B4RauditUser, "5", MODULE, auditmsg, B4RauditDatabase);
}
```

where:

- B4RauditUser is the currently logged in client
- 5 corresponds to the logging level (*i.e.* 1 is a critical issue, 5 is informational)
- MODULE is the module / class the log entry originates from
- auditmsg is the description of the logger entry
- B4RauditDatabase is the database the log entry is stored in

At present in the B4R prototype a scheduled daily DBMS job deletes all entries from the B4R database that are greater than two weeks old.

```
CREATE OR REPLACE PROCEDURE SYSTEM.B4R_HOUSEKEEPING AS
BEGIN
    Delete from B4RauditLog where Date_Logged < SYSDATE-7;
    Delete from B4RerrorLog where Date_Logged < SYSDATE-7;
    Delete from B4RsecurityLog where Date_Logged < SYSDATE-7;
    Delete from B4RadminAuditLog where Date_Logged < SYSDATE-7;
    Delete from B4RadminErrorLog where Date_Logged < SYSDATE-7;
COMMIT;
END B4R_ B4R_HOUSEKEEPING;
```

In a production Billing4Rent environment these records should be saved to a file in order to be archived

- **Audit: View All** – By selecting this option a user can view all entries in the Audit Log. The audit log stores all log entries from the B4R client prototype and serves as an ‘audit trail’ for each client as he / she interacts with Billing4Rent and executes various actions. When the log viewer page first loads no records are displayed. Instead the user can choose to display all records or just display certain records (Figure 5.14).



Figure 5.14: Filter Logger

Users view all records in the log, or filter the results by Client ID associated with the log message, the date the log entry was logged or the status of the log entry (*i.e.* 1 is a critical issue, 5 is informational). When the

'Set Filter' button is clicked, the appropriate records are retrieved from the B4R database (Figure 5.15).

The screenshot shows the B4R Admin console interface. At the top, there is a navigation bar with links for Home, User Admin, Client Admin, Statistics, Logger, and Logout. Below the navigation bar, there are three dropdown menus for filtering: User ID (set to 'All'), Date Logged (set to 'All'), and Status (set to 'All'). A 'Set Filter' button is located to the right of these dropdowns. The main content area is titled 'B4RauditLog' and contains a table with the following data:

User ID	Date	Status	Module	Logger Message
nkazna	05-Apr-06, 14:51:55 PM	5	B4RClient	Authenticate B4RClient [nkazna]
nkazna	05-Apr-06, 14:51:55 PM	5	RdbmsClientLoginModule	Validation of login details for B4RClient [nkazna]
nkazna	05-Apr-06, 14:51:55 PM	5	RdbmsClientLoginModule	Trying to connect to database!
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Connected to database!
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	SELECT * FROM B4RCLIENT where clientid='nkazna' and enabled='true'
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	SQL select complete
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Passwords do NOT match!
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Validation of login details for B4RClient [nkazna] failed
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Abort
nkazna	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Logout
nkazna	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	Connected to database!
nkazna	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	Passwords match!
nkazna	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	UPDATE B4RCLIENT set last_login_time = 114405116338 where CLIENTID = 'nkazna'
nkazna	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	SQL update complete
nkazna	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	Validation of login details for B4RClient [nkazna] succeeded

Below the table, there is a pagination control showing 'Pages: 1 2 3 4 5 6 7 8 9 10 [Next]'. At the bottom of the page, there is a footer with a privacy statement link and a copyright notice: '© 2005 Billing4Rent. All rights reserved.'

Figure 5.15: Display Log Records

- **Error: View All** – This log displays all error messages generated by the B4R prototype.
- **Security: View All** – This log displays all security messages generated by the B4R Prototype.
- **Admin Audit: View All** – By selecting this option, a user can view all entries in the Audit Log relating to the B4R Administrative Interface.
- **Admin Error: View All** – This log displays all error messages generated by the B4R Administrative Interface.
- **Audit: Delete Single** – When this option is selected, the user is displayed a dropdown box containing all Client Ids stored in the B4R Prototype audit

log. When a client is selected, the user will be prompted to confirm his / her intention to delete all logger entries for that specific client.

- **Error: Delete Single** – When this option is selected, the user is displayed a dropdown box containing all Client Ids stored in the B4R Prototype error log. When a client is selected, the user will be prompted to confirm his / her intention to delete all logger entries for that specific client.
- **Security: Delete Single** – When this option is selected, the user is displayed a dropdown box containing all Client Ids stored in the B4R Prototype security log. When a client is selected, the user will be prompted to confirm his / her intention to delete all logger entries for that specific client.
- **Admin Audit: Delete Single** – When this option is selected, the user is displayed a dropdown box containing all Client Ids stored in the B4R Administrative Interface audit log. When a client is selected, the user will be prompted to confirm his / her intention to delete all logger entries for that specific client.
- **Admin Error: Delete Single** – When this option is selected, the user is displayed a dropdown box containing all Client Ids stored in the B4R Administrative Interface error log. When a client is selected, the user will be prompted to confirm his / her intention to delete all logger entries for that specific client.
- **Audit: Delete All** – When this option is selected, the user is prompted to confirm his / her intention to delete all logger entries from the B4R Prototype audit log.
- **Error: Delete All** – When this option is selected, the user is prompted to confirm his / her intention to delete all logger entries from the B4R Prototype error log.

- **Security: Delete All** – When this option is selected, the user is prompted to confirm his / her intention to delete all logger entries from the B4R Prototype security log.
 - **Admin Audit: Delete All** – When this option is selected, the user is prompted to confirm his / her intention to delete all logger entries from the B4R Administrative Interface audit log.
 - **Admin Error: Delete All** – When this option is selected, the user is prompted to confirm his / her intention to delete all logger entries from the B4R Administrative Interface audit log.
- 5) **Logout** – When a B4R user selects the logout button their current session is invalidated.

5.4 Proposed Network Architecture

The architecture of a system always defines its broad outlines, and may define precise mechanisms as well. The term *architecture* can refer to either hardware or software or to a combination of hardware and software.

5.4.1 B4R Network Diagram

Figure 5.16 illustrates the Billing4Rent network architecture. Each network connection in the proposed architecture should include a redundant component. However, these redundant components are not evidenced in the diagram in order to aid clarity.

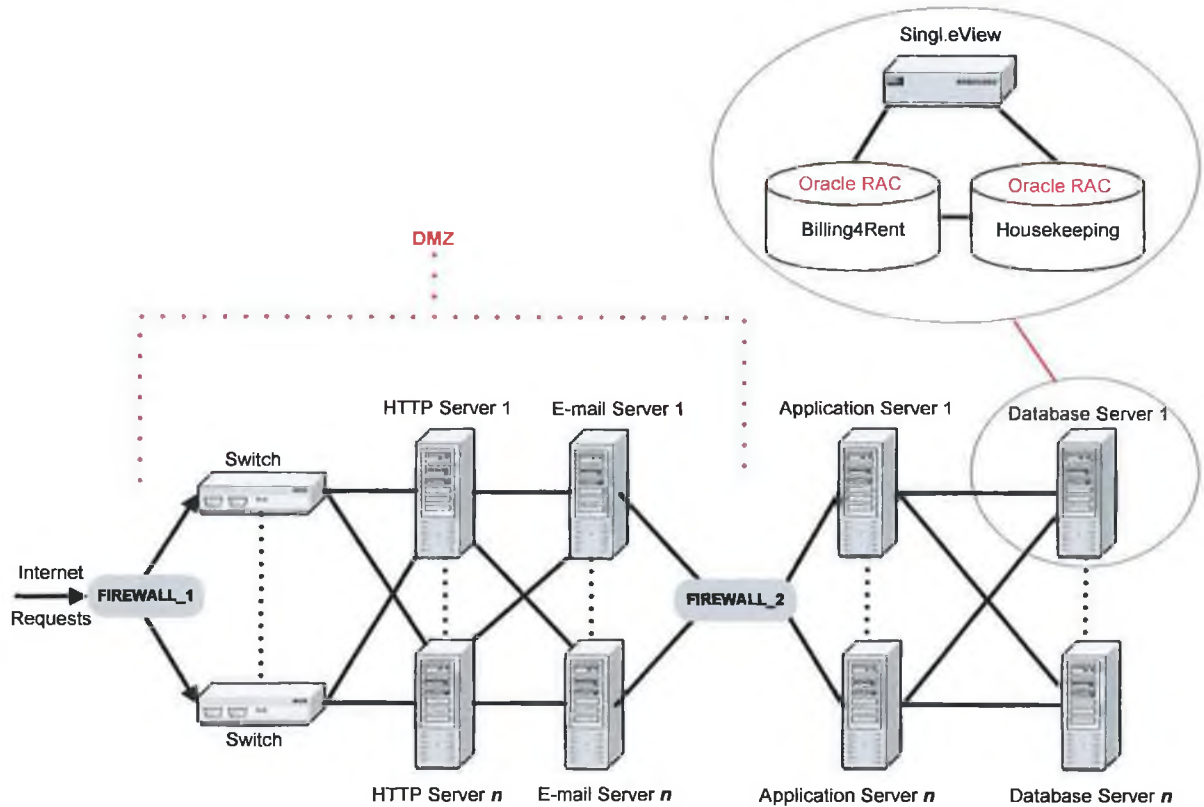


Figure 5.16: Billing4Rent Architecture

5.4.2 Architecture Breakdown

The following sections breakdown the various components of the Billing4Rent architecture:

5.4.2.1 Firewall_1

A firewall is a system designed to prevent unauthorised access to or from a private network. Firewalls can be implemented in either hardware or software, or a combination of both. Firewalls are frequently used to prevent unauthorised Internet users from accessing private networks connected to the Internet, especially intranets. All messages entering or leaving the intranet pass through the firewall, which examines each message and blocks those that do not meet the specified security criteria.

Firewall_1 will protect the Billing4Rent HTTP servers from external attacks, and is the first line of defence between the B4R network and any external threats. While the HTTP server may be using various ports to provide services, these

ports should not be accessible by external users. Consequently, *Firewall_1* will limit remote access to only the following ports:

- Port 21: FTP
- Port 80: HTTP
- Port 443: HTTPS
- Port 990: FTPS

Optional open ports:

- Port 22: SSH remote login
- Port 23: Telnet

5.4.2.2 Load Balancing Switch

A Switch is a device that filters and forwards packets between Local-Area Network (LAN) segments. Switches operate at the data link layer (layer 2) and occasionally the network layer (layer 3) of the OSI Reference Model and therefore support any packet protocols. A switch is commonly connected to at least two networks or devices and is usually located at gateways, the places where two or more networks connect. Switches determine the best path for forwarding packets. However very little filtering of data is done through switches. In the set-up depicted in Figure 5.16 the switch redirects Billing4Rent HTTP requests to an available HTTP server. This enables effective load balancing in periods of increased network traffic, as the switch will direct a stream of requests across all of the available HTTP servers and avoid sending requests to servers that are out of operation, or are otherwise busy. To address the Single Point of Failure (SPOF) at the switch, a second switch is implemented in a hot standby configuration in the event of the first switch being out of operation.

5.4.2.3 HTTP Server

A HTTP server is a server process running at a web site, which sends out web pages in response to HTTP requests from remote browsers. Every Web server has an IP address and possibly a domain name. For example, when a user enters the URL **http://www.billing4rent.com/index.html** in a browser, this sends a request to the server whose domain name is **billing4rent.com**. The server then fetches the page named **index.html** and returns it to the requesting browser. In many production environments, the HTTP server is co-located on a single

machine with either the application server, or database server, or both. However, by separating the HTTP server from both the application server and the database server, this alleviates any resource contention between the three sets of processes.

5.4.2.4 E-mail Server

An e-mail server is a process and device that provides 'post office' facilities. It stores incoming mail for distribution to users and forwards outgoing mail through the appropriate channel. The term may refer to just the software that performs this service, which can reside on a machine with other services. The Billing4Rent e-mail server will support both incoming e-mail (in the form of service cost inquiries, invoice inquiries *etc*) as well as outgoing e-mail (in the form of replies to client enquiries and customer invoice data in HTML or Portable Document Format (PDF)). Billing4Rent customers will have the facility to upload their entire client usage data in a single file, which will be imported and processed by Billing4Rent. However these files will be uploaded via FTP or FTPS rather than submitted via e-mail.

5.4.2.5 DMZ

In order to protect application servers from unauthorised access, the separation of the Web server from the application server using firewalls is often used to create a secure *DeMilitarised Zone* (DMZ) surrounding the Web server [79]. Application data, and business logic is protected by isolating the HTTP server in the DMZ, which restricts access from the public Internet. Machines with very limited, well-understood and logged services are placed in the DMZ. The two firewalls - one between the public Internet and the DMZ and the other between the DMZ and the Billing4Rent LAN - strictly limit traffic in and out of the DMZ. HTTP servers in general have limited capabilities and, because it is well understood how to protect these HTTP servers from attack, they are situated in the DMZ. Application servers are not placed in the DMZ, because if the outside firewall were compromised, it would expose B4R business logic as well as confidential client information. Additionally, application servers include a Java Virtual Machine (JVM), which could potentially be used to assist an attacker in further compromising the website.

5.4.2.6 Firewall_2

The second firewall in the DMZ configuration only allows traffic to pass between the HTTP server within the DMZ and the Billing4Rent Application Server situated in the trusted private Billing4Rent LAN.

5.4.2.7 Application Server

Application servers are typically used for complex transaction-based applications. The applications server is essentially where the ‘brains’ of Billing4Rent reside and it contains items such as B4R business rules and data manipulation. The application server handles all application operations between users and Billing4Rent’s business applications and databases.

5.4.2.8 Database Server

The Billing4Rent database server consists of an installation of *Singl.eView* from Intec. There are two databases administered by *Singl.eView*, a production Billing4Rent database as well as a Housekeeping database. The Billing4Rent database contains all customer details, their client details and billing information. The Housekeeping database is used for administrative purposes (for example to store user logins and passwords, user profiles *etc*). It is envisioned that Oracle 9i will be used as the underlying DBMS, deployed in an Oracle 9i Real Application Cluster (RAC) configuration.

5.4.3 Inter-tier Traffic Flows

Figure 5.17 depicts typical inter-tier network traffic flow as a result of a Web-based transaction.

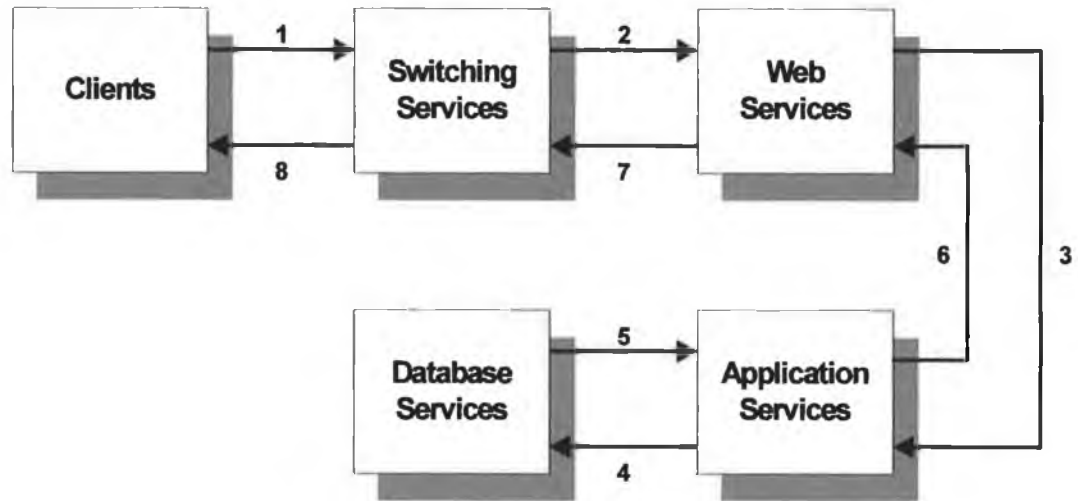


Figure 5.17: Inter-tier Traffic Flows

5.4.4 Inter-tier Traffic Flow Details

<p>Flow: 1 Interface1: Client Interface2: Switch Protocol: HTTP / HTTPS</p> <p>Description: Client initiates Web Request</p>	<p>Flow: 2 Interface1: Switch Interface2: Web Service Protocol: HTTP / HTTPS</p> <p>Description: Switch redirects client request to a particular Web server based on load-balancing algorithm.</p>
<p>Flow: 3 Interface1: Web Service Interface2: Application Service Protocol: LDAP</p> <p>Description: Web service 'talks' to the Application server through a Web connector.</p>	<p>Flow: 4 Interface1: Application Service Interface2: Database Service Protocol: SQL / PLSQL</p> <p>Description: Application service requests to retrieve or update a row/rows in the database table.</p>
<p>Flow: 5 Interface1: Database Service Interface2: Application Service Protocol: SQL / PLSQL</p> <p>Description: Database request completed.</p>	<p>Flow: 6 Interface1: Application Service Interface2: Web Service Protocol: RMI</p> <p>Description: Application server returns dynamic content to Web server.</p>

Flow:	7
Interface1:	Web Service
Interface2:	Switch
Protocol:	HTTP / HTTPS
Description: Switch receives reply from Web server.	

Flow:	8
Interface1:	Switch
Interface2:	Client
Protocol:	HTTP / HTTPS
Description: Switch rewrites IP header, and returns HTTP request to client.	

5.4.5 Vertical Scaling

In addition to horizontal scaling - where many machines are added to the B4R system to improve availability and performance - all existing or additional machines should support vertical scaling (*i.e.* many instances of an application on one machine). Figure 5.18 depicts an Application server set-up with n instances on each machine.

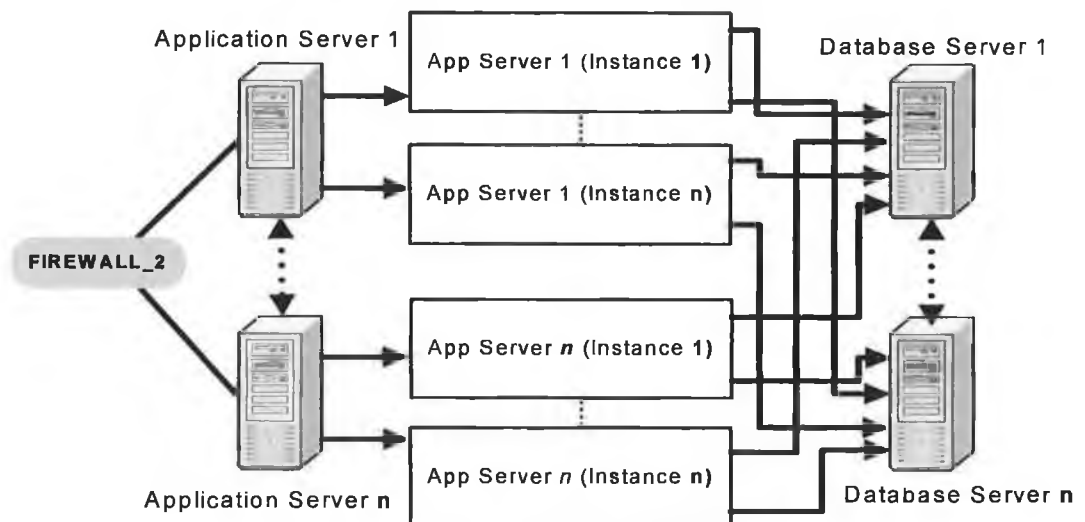


Figure 5.18: Application Server Vertical Scaling

Similarly, Figure 5.19 depicts a HTTP server set-up with n HTTP server instances on each server.

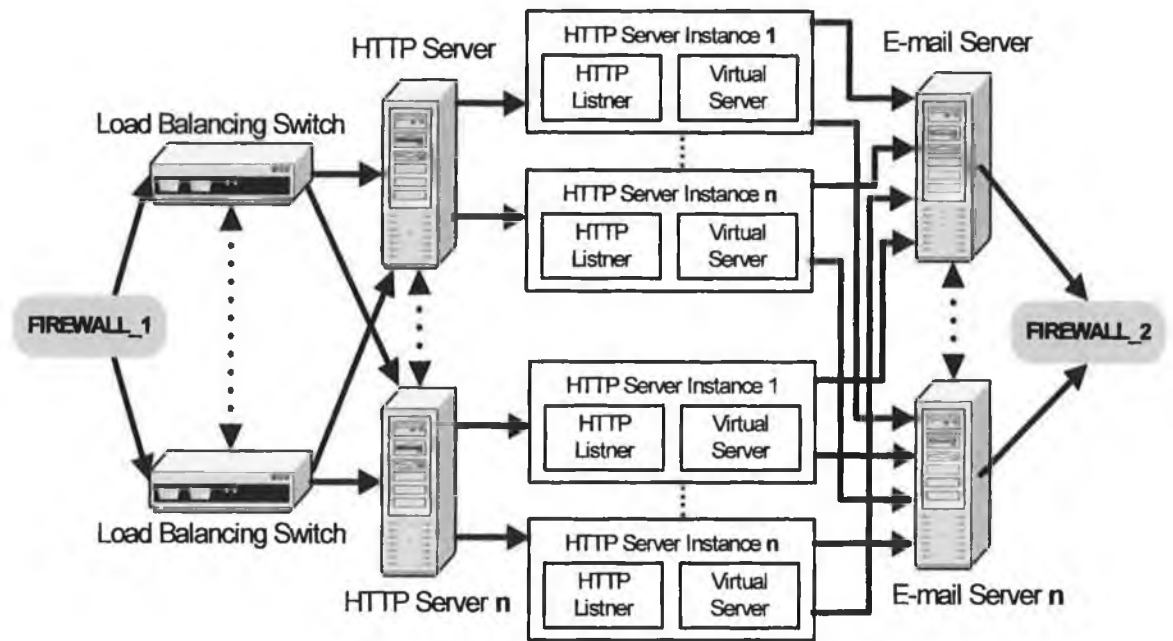


Figure 5.19: HTTP Server Vertical Scaling

5.4.6 Backup Internet Connection

There should be two separate, independent connections from the Internet to the Billing4Rent network in the event of one of the service providers' infrastructure becoming compromised, which would lead to the entire Billing4Rent network becoming isolated from the Internet. In that eventuality, Billing4Rent clients should still have access to the Billing4Rent service via a backup Internet service (Figure 5.20).

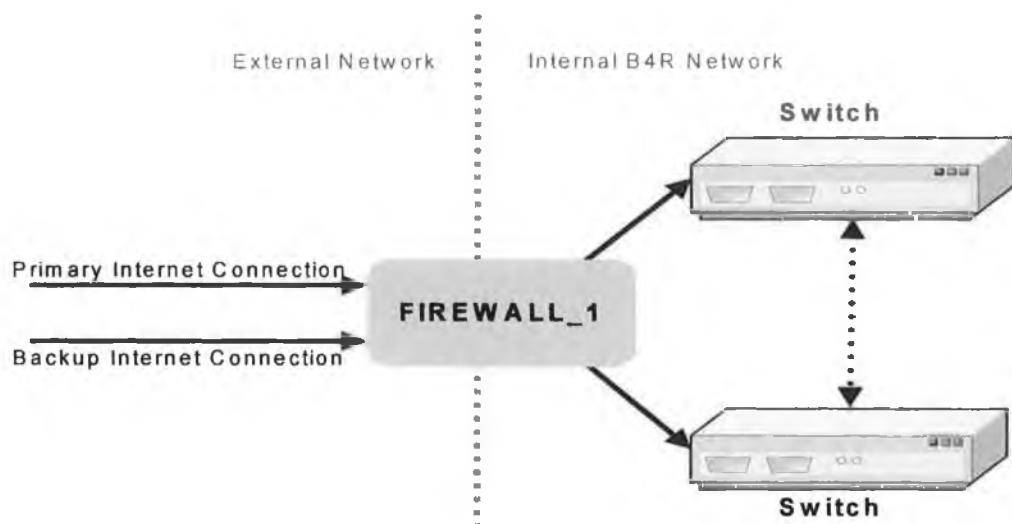


Figure 5.20: Backup Internet Connection

5.5 Code Optimisation for Improved Performance

Code optimisation involves writing code so that it runs as fast as possible on its host computer [83]. Many high-level language compilers offer options as to what type of code to generate at compile time (*i.e.* optimisation for run-time or for code size). Designing B4R with optimised code would provide improved performance. Billing4Rent was developed using Java, Servlets and JSP and the following recommendations (both general and Java specific) were implemented where possible:

5.5.1 Memory Usage

By limiting memory usage, the performance impact of memory management performed automatically by Java (Garbage Collector) is minimised. It is envisioned that Billing4Rent will run for long periods; therefore even a small memory leak can cause the Java Virtual Machine (JVM) to run out of free memory.

- Only create objects as needed. The more objects that are allocated, the more memory pressure this puts on a system, potentially resulting in more frequent, longer garbage collections.
- Delete references to objects no longer required *i.e.* 'loitering objects'. Loitering objects are objects that are allocated, not used, and not garbage collected. The effect of these objects is to increase the size of the JVM heap, causing excessive overhead on the garbage collector, as well as causing memory leaks, such as an 'out-of-memory error'.
- A common source of memory leaks in Java is due to not closing Java Database Connectivity (JDBC), Java Message Service (JMS) and Java Connector Architecture (JCA) resources when they are no longer required, particularly under error conditions.

5.5.2 Session Management

In a Web application, state information relating to each client is typically stored in an HTTP session, which is identified by a unique identifier that is associated with an HTTP cookie. The amount of data stored in a session should be minimised, as

session data is usually shared, and as such it must be serialised. Serialisation also involves serialising all objects that are reachable from the session. In Java, serialization is a memory intensive operation. If persistent sessions are implemented the serialized session data must be stored in a database, which introduces further overhead as session data is stored as a Binary Large Object (BLOB). The use of sessions can be avoided by implementing any of the following:

- Using hidden form fields, or cookies, to store session data.
- Storing data directly in a database, as using native data types instead of serialized BLOBs can produce better performance.
- Entity Enterprise Java Beans (EJBs) can be used to store session data.

5.5.3 Servlets and Java Server Pages (JSPs)

- Minimize the use of the “**<jsp: include>**” tag, since each included JSP is a separate servlet.
- The “**<jsp: usebean>**” tag should only be used to obtain a reference to an existing object, rather than for creating a new object. When a “**<jsp: usebean>**” tag is encountered and an existing Java bean object with the appropriate name does not exist, a new one is created. This is usually done by a call to the **Beans.instantiate()** method which is an expensive operation because the JVM checks the file system for a serialized bean
- When executing a JSP, a session object is normally created implicitly if one does not already exist. However, if the session is not required, creation can be avoided by the use of the “**<%@ page session="false" %>**” directive.

5.5.4 Logging

Logging on a system is an important tool, not only for isolating an action which caused a systems failure, or to assist in the recovery of that failure, but also to provide an ‘audit trail’ for a user to track his/her actions. When employing logging on a system, it is important to determine if the level of logging is adequate for system recovery. However excessive logging can utilise too much processing time.

Regardless of logging levels, applications should limit use of the *System.out.println()* command, as I/O provides excessive performance overhead.

5.5.5 Enterprise Java Beans (EJBs)

There are a number of performance considerations that need to be taken into account when using EJBs:

- Obtaining EJB references involves a lookup process, which can take a relatively long time. Caching any obtained references can improve performance on subsequent lookup operations.
- All access to Entity EJBs should be performed through stateless session beans, as this greatly reduces the number of remote method calls. Calls to EJB methods are implemented as remote method calls even if the EJB exists in a container that shares the same JVM as the Web container.
- Accessing entity beans from session beans can limit the number of transactions.
- If an entity bean has methods that do not update attributes (*i.e.* getter type methods), specify these methods as read-only in the deployment descriptor.
- The use of stateful session beans should be avoided. If they are to be used, they should they should be kept to a minimum if possible.

5.5.6 Database Access

When accessing a database, obtaining and closing a connection to a database using Java Database Connectivity (JDBC) can be a relatively expensive exercise.

- Using connection pools can significantly reduce overhead. A connection pool contains a (defined) number of connections to the database that have already been established. When a database operation is to be performed, a connection can be obtained from the pool. Similarly, when the connection is closed, it is returned to the pool and made available for reuse.

- JDBC resources should always be released once they are no longer required. Failure to properly close resources can cause memory leaks, and can cause slow response due to threads having to wait for a connection to become available from the pool.
- If an application repeatedly executes the same query, but with different input parameters, then performance can be improved by using a *java.sql.PreparedStatement* instead of *java.sql.Statement*.

5.5.7 General Coding Considerations

- The use of string concatenation, which involves the creation of new strings with the data copied from the original strings, *i.e.*

```
String concatString = origString + newString;
```

Concatenating strings is a slow process and it also creates more work for the garbage collector. Using *java.lang.StringBuffer* as an alternative to *java.lang.String* can improve performance, *e.g.*

```
String concatString = new StringBuffer(origString)
                      .append(newString).toString();
```

- When creating classes, the structure of the class should not be excessively complicated as there is a performance overhead in loading and instantiating these classes.
- Avoid excessive and repeated casting. Once an object has been cast, assign a variable of the correct type and reuse this reference.
- When iterating *n* items, iterating from *n-1* to 0 instead of 1 to *n* is quicker for most JVMs
- Avoid repeatedly calling the same method within a loop if the result is the same every time. As an alternative, store the value in a variable prior to entering the loop and use this stored value for each iteration of the loop.

- Use the *System.arraycopy()* method to copy the contents of one array to another rather than iterating across each array element and copying it individually.
- Exceptions should be mainly used for infrequent error conditions, and their overuse should be avoided, unless the exception checking is performed with the aid of *try / catch blocks*, as this is not particularly performance intensive.
- Input/output (I/O) to peripherals takes time and should be limited. A counter that says 'XX% complete' is inefficient and should not be used inside a loop. Should a warning or message to the user be required, general messages like 'please wait while process completes' will perform better.

5.6 Billing4Rent Disaster Recovery

When implementing a Disaster Recovery (DR) Plan for Billing4Rent it is worth acknowledging that the impact of a disaster situation would have a twofold effect: not only would B4R clients experience service interruption, any outage may also impact the customers of those clients. The ability to quickly recover client data after a disaster is an important component in delivering high levels of availability in an ASP environment. When implemented correctly, a proper DR plan allows for a quick restoration of an organisations' IT services, usually implemented by making a backup of servers and files critical to the organisation, and quickly restoring those files in the event of a disaster. However, simply making a backup of data does not constitute a complete disaster recovery plan. In the event of an organisations offices being destroyed, if the backups are held at the same location as the servers the data was backed up from, both the servers (and the backup) may be irrecoverable.

5.6.1 Billing4Rent Backups

Backing up files means copying files to a second medium (*i.e.* a disk or a tape), as a precaution in case the first medium fails. Even the most reliable system will break down at some stage; therefore it is vitally important that files are backed up regularly. Regardless of the commercial software used for Disaster Recovery,

Billing4Rent should create two copies of each backup; one copy should be stored on-site in close proximity to the servers that are backed up. This ensures quick recovery in the case of a (relatively) minor disaster such as a hard disk crash. The second copy of the backup should be stored off-site, in an entirely different location to the Billing4Rent servers. If a critical failure occurs in which both the B4R servers and the on-site backups are destroyed, there will still exist a copy of the backup in a different geographic location.

It is important that the copies of backups are made daily and are up to date; as the nature of Billing4Rents clients means they would be unable to cope with losing a few days, or a week worth, of business data. A full backup should be made once a week, and daily incremental backups should be taken. A full backup of B4R systems would be a time, and processing intensive operation, whereas an incremental backup (which only backs up the data that have been modified since the previous backup) would not have the same impact to B4R service performance. Full backups are more time-intensive, however in the aftermath of a disaster, only one full restore operation needs to be carried out in order to completely re-establish an organisations data system. With incremental backups, the last full backup needs to be applied first, followed by each additional incremental backup in sequence. If the B4R system were to suffer a disaster on the sixth day of the week, to complete the restore process would require applying one full backup, and a maximum of six incremental backups.

5.6.1.1 Offsite DR Location

As was noted in the previous section, two copies of each backup should be made, with one copy stored in an off-site location in order to preserve B4R data in the event of a critical disaster. The second site should be in a different city to the original, to legislate against a large-scale disaster such as a power outage affecting an entire city. Due to the nature of Billing4Rent clients, only a short service interruption would be tolerated. Therefore Billing4Rent should, at a minimum, implement an active secondary site with the ability to resume B4R services as quickly as possible in the event of an outage in the primary B4R site. In order to implement such a recovery solution, a high bandwidth connection between the two sites would need to be established (see Figure 5.21).

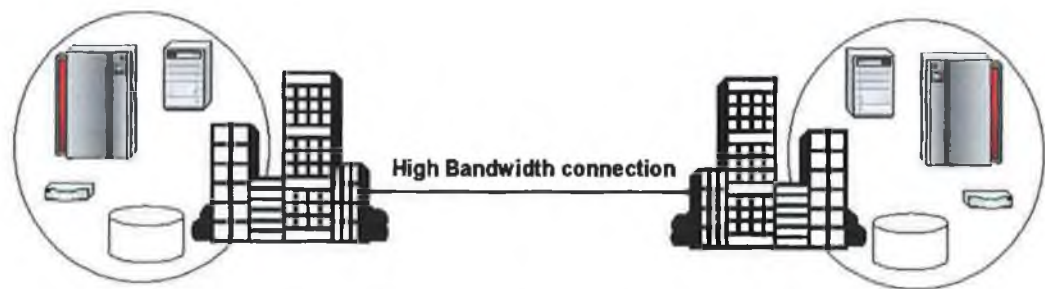


Figure 5.21: B4R Disaster Recovery Solution

Backups taken on the primary site could be quickly transferred to the secondary site and applied to the secondary sites systems. Such a solution would provide a Disaster Recovery solution with the ability to quickly recovery from a system outage, with a maximum of one day's data loss.

5.6.2 Disaster Recovery Policies

Within the Billing4Rent organisation, a person should be assigned as Disaster Recovery Manager. The Disaster Recovery Manager has the sole responsibility of designing and implementing disaster recovery plans and policies. The following tasks are the responsibility of the DR manager:

- **Create a Disaster Recovery Plan (DRP):** A Disaster Recovery Plan is a comprehensive set of processes to be implemented before, during and post disaster. The primary goal of any DRP is the restoration of normal system activities as quickly as possible. The plan should be thoroughly tested to ensure the continuity of operations and availability of critical resources in the event of a disaster. In order to create a DRP, the DR manager must have a clear understanding of the Billing4Rents infrastructure and how all of its resources are interconnected, as well as how B4R clients (and their customers) would be affected in the event of a disaster. Secondly the manager must assess B4R for vulnerabilities: contingency planning, operating procedures, physical space and equipment, data integrity.

- **Maintain and update DRP:** When a DRP has been created, it is important that the plan is re-evaluated annually, or semi-annually. As Billing4Rent evolves and changes, it is vitally important the DRP also evolves.
- **Create a DR team:** The DR manager should appoint a team, responsible for implementing the DR plan in the event of a service interruption. Each team member should be made familiar with the Policies and Procedures specified in the DRP. The DR manager should also train team members with regard to the plan if required.
- **Test the DRP:** The plan should be tested annually in order to test its effectiveness in the face of a disaster situation. All Procedures used to test the plan should also be documented. The goal of DR testing is to provide reassurance that all necessary steps are included in the plan. Testing will not only highlight areas of the plan that are inadequate, it may also demonstrate the ability of B4R to recover from a disaster situation (and provide a 'selling point' to attract new clients).

Any Disaster Recovery Plan should be evaluated and approved by B4R. In the event of a disaster, the DR manager will have the following responsibilities:

- **Damage Assessment:** In the aftermath of a disaster, the first task of the DR manager will be to assess the extent of the damage to B4R facilities and systems. The ability of B4R to continue to provide billing services to clients should be evaluated.
- **Notification:** The DR manager will be responsible for providing initial notification of disaster to:
 - 1) Disaster recovery team members.
 - 2) B4R Management
 - 3) B4R Clients. In the event of a disaster it may be appropriate to notify clients of the service interruption, in order for them to be able to further notify their own customers.

- 4) Admin team. If B4R implements a secondary DR site, it may be necessary for some DR team members to travel to the secondary site. Accordingly, travel and accommodation arrangements for designated teams members will have to be made.
- **Co-ordinate Recovery Teams:** The DR manager will co-ordinate all recovery processes and direct team members where appropriate. He / she may have to authorise any necessary purchases in order to complete the recovery process.

5.6.3 Disaster Recovery Prevention

A good preparation for Disaster is to implement policies to prevent a disaster situation in the first place. Accordingly, B4R should implement some, or all of the following:

- **Good Housekeeping:** The building where B4R is located should be kept clean and free of obstructions and fire hazards. Loose paper burns at a quicker rate than large, tightly bound books, directories *etc.* Therefore a 'tidy desk policy' should be implemented to remove loose paper from desktops to reduce losses due to fire. This will also help to protect documents from sprinkler discharge and other incidents.
- **Ban non-essential electrical items:** In order to eliminate overloaded electrical circuits, B4R employees should be prohibited from using non-business electrical appliances such as radios, heaters, fans, mobile phone chargers *etc.* These appliances could cause electrical fires by overloading circuits not designed for these appliances.
- **Security:** Security procedures should be implemented in the B4R facility in order to prevent unauthorised persons gaining access to B4R and purposely causing a system outage.

5.7 Chapter Summary

Chapter five proposed an ASP framework for Billing4Rent. The functional requirements for developing the Billing4Rent ASP were outlined. The prototype was to provide the platform on which all recommendations or proposals will be implemented and allows these proposals to be analysed and quantified in an objective manner. Accordingly, this chapter detailed the B4R prototype, which consists of a B4R client website, where clients can add, remove, update customers, products / services, and invoices. A second 'Administrative Interface' was created in order to manage and administer the B4R ASP prototype. This Administrative Interface incorporated various availability and scalability components and allows authorised B4R personnel to create and manage user accounts for clients. From the Administrative Interface, authorised personnel can also create users, view statistics relating to B4R storage media (*i.e.* Database or Filesystem) usage, as well as view audit and error logs. In order to Monitor B4R, users can launch a Monitoring program which collects statistics about Billing4Rent such as page hits, page completion times etc

This Chapter also detailed a Proposed Network Architecture for Billing4Rent, and de-composed the proposed architecture into its constituent parts. Various recommendations for those parts were also suggested, as well as source code optimisations in order to improve B4R performance were also detailed. Finally, documented in this chapter was a Disaster Recovery Plan suitable for restoring service not only to Billing4Rents clients, but also to their customers in the event of a disaster situation.

SECTION FOUR



Research Evaluation

Chapter Six: Research Evaluation

This chapter provides an evaluation of the proposed framework detailed in the previous chapter. The evaluation considers the real world implementation, Billing4Rent, and how it performs in comparison with accepted industry benchmarks. Also considered are the frameworks and recommendation for High Availability and Scalability detailed in chapter five. The appraisal will be conducted along the following criteria:

- Fulfilment of objectives.
- Comparison of the individual components of Billing4Rent with accepted industry benchmarks and / or related technologies.

This chapter also provides a recommendation for the deployment of Billing4Rent, as well as justification for this proposal.

6.1 Fulfilment of Objectives

The aim of this project is to develop a framework, including the architecture and system configuration required, to ensure High Availability (HA) and Optimum System Performance in an Application Service Provider (ASP) Environment. This thesis commenced with an investigation into the current state of Application Service Provision. Chapter Four examined High Availability and Scalability, while Chapter Five proposed a framework for Availability, Scalability and System Performance, which can be adopted by ASPs going forward. The prototype ASP, **Billing4Rent (B4R)**, provided the real world platform on which the proposals arising out of this body of research were implemented. The B4R prototype allows the research to be objectively analysed and quantified in a real world environment. In addition to the B4R prototype, an *Administrative Interface* was also created in order to administer Billing4Rent and to incorporate availability and scalability components.

6.1.1 Billing4Rent Architecture

The architecture of B4R defines its broad outlines, and can refer to either hardware or software, or to a combination of both. When designing a system architecture, there may be a trade-off between cost and performance. Figure 6.1 illustrates the Billing4Rent network architecture proposed in Chapter Five. Each network connection in the proposed architecture should include a redundant component. However, these redundant components are not evidenced in the diagram in order to aid clarity.

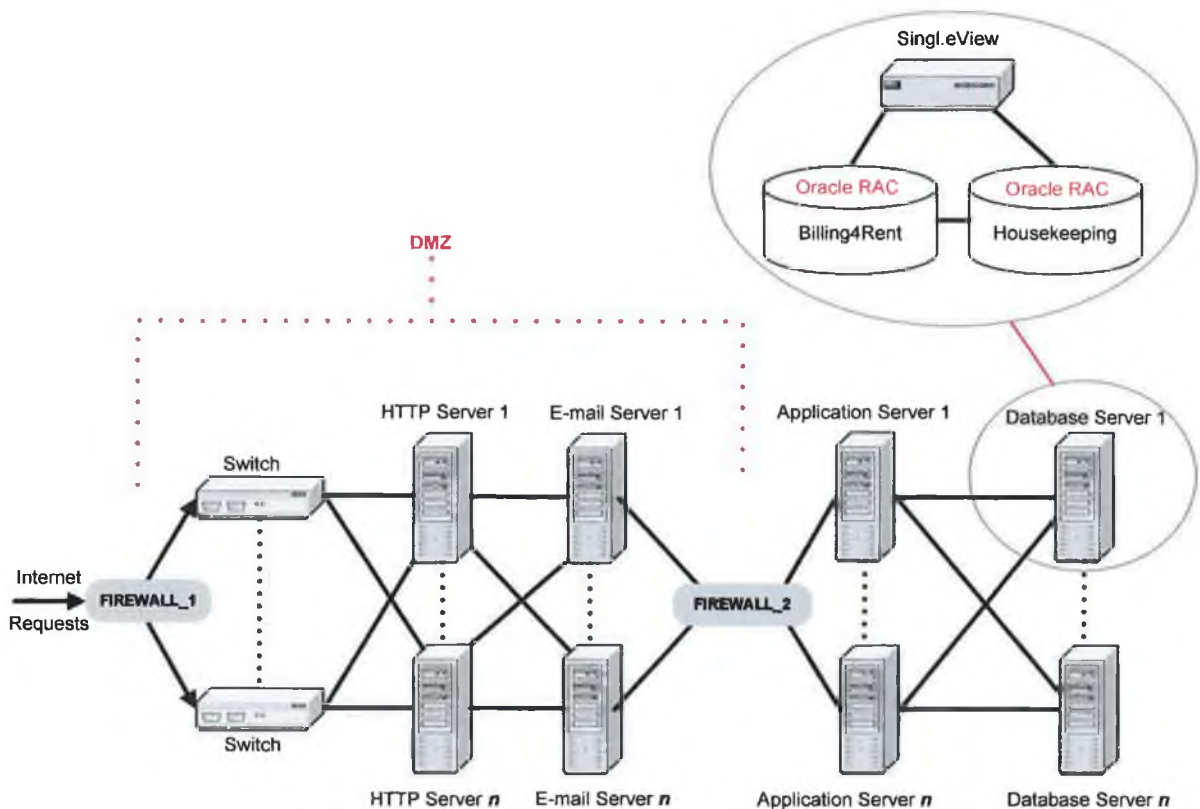


Figure 6.1: Billing4Rent Architecture

When designing a HA system, one of the most important considerations is removing Single Points of Failure (SPOF). As outlined by Marcus and Stern [53], a SPOF is a single component of a system (*i.e.* hardware, firmware, software or otherwise) whose failure will cause some degree of downtime. Equally, in their article “Architecture and Dependability of Large-Scale Internet Services”, Oppenheimer and Patterson [43] likened SPOFs to the weakest link in a system; when that link breaks the entire system fails. Most systems have obvious potential

SPOFs (*i.e.* servers, disks, network devices and cables). For most systems, a significant protection against SPOFs is achieved via redundancy. However, the cost involved in introducing redundancy into an entire system may be prohibitive. In the above architecture, all major components of the system have redundant components (Load Balancing Switch, HTTP Server, Application Server, E-mail Server and Database Server). There are, however, obvious SPOFs in the above configuration – there is no redundancy built into either Firewall_1 or Firewall_2. This design was intentional, as each firewall acts as a redundant component for the other. For example, if Firewall_1 were to stop functioning, the only section of the B4R enterprise exposed to the public Internet would be the HTTP server and the E-mail server in the De-Militarised Zone (DMZ). Only machines with very limited, well understood, and logged services are placed in the DMZ. Firewall_2 would still protect the business logic contained in the application server and client data stored on the database server. Similarly, should only Firewall_2 cease activities, the entire B4R enterprise would still be protected from the public Internet by Firewall_1. The B4R service would not experience any unnecessary downtime while either of the Firewalls are being replaced or repaired.

In the architecture outlined in Figure 6.1, the precise number of redundant components is not shown. Instead, the redundant component(s) are referred to as component *n*. While the precise budget available to Billing4Rent to provide high availability is unknown at this point, the proposed architecture is an optimal configuration, designed to scale up (or down) in order to respond to changing workloads. However, in the opinion of this author, the minimum number of redundant components should be one, to provide a *hot* standby in the case of the primary component developing a fault. This will allow the standby component to seamlessly assume the workload of the primary component without any noticeable loss of service to the end user. Should this be the case in Billing4Rent it is vitally important (as Gray and Siewiorek [41] point out) that the spare component be installed and configured in advance so that when one component fails the redundant component can replace it almost immediately. This allows the failed component to be repaired off-line while the system continues to deliver service. In the case of a database server, it may be necessary, in order to assure it is transactionally consistent, to copy data immediately to the backup database server.

While this may introduce further overhead on a system, its primary benefit will be a secondary server, which can be brought into operation quickly, minimising B4R downtime.

The proposed architecture for Billing4Rent has been designed with scalability in mind. In the words of André Bondi [61], scalability is “*the ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work, and/or to be susceptible to enlargement*”. Jogalekar and Woodside [60] further advance this definition by stating that the new configuration should not just operate, but operate efficiently and gracefully when dealing with increased capacity. For example if - in order to accommodate an increased workload - an additional HTTP server were required to supplement the HTTP servers currently deployed in the DMZ depicted in Figure 6.1. In order to integrate the new HTTP server (which has the same configuration as the existing server), the system administrator only has to update the routing table on both the primary and redundant load-balancing switch to *declare* the new HTTP server ready to handle client requests. Similarly, should a new Application server be introduced to B4R, it would simply require configuration changes on all the HTTP servers to reflect the additional Application server. Finally, should a redundant E-mail or Database server be required, the existing Application servers would have to be updated to reflect the newly added hardware.

6.1.2 Model / View / Controller and The B4R Prototype

The Billing4Rent prototype was developed using Java, Java Servlets and Java Server Pages (JSP). The prototype has an Oracle 9i RDBMS backend to store client and administrative data and runs in a Jakarta Tomcat web-container. The prototype was developed using the Model / View / Controller (MVC) design pattern, which consists of three components:

- **Model:** Holds all data, state and application logic.
- **View:** Provides a presentation of the model.
- **Controller:** Defines the way the *User Interface* reacts to user inputs.

The advantage of the MVC design pattern (as advocated by Yonglei Tao [84]), is that it is designed to provide multiple views of the same data. This means that the

various components of the Billing4Rent prototype can be interchangeable. Figure 6.2 illustrates how MVC works on an online Billing4Rent transaction (adapted from [85]).

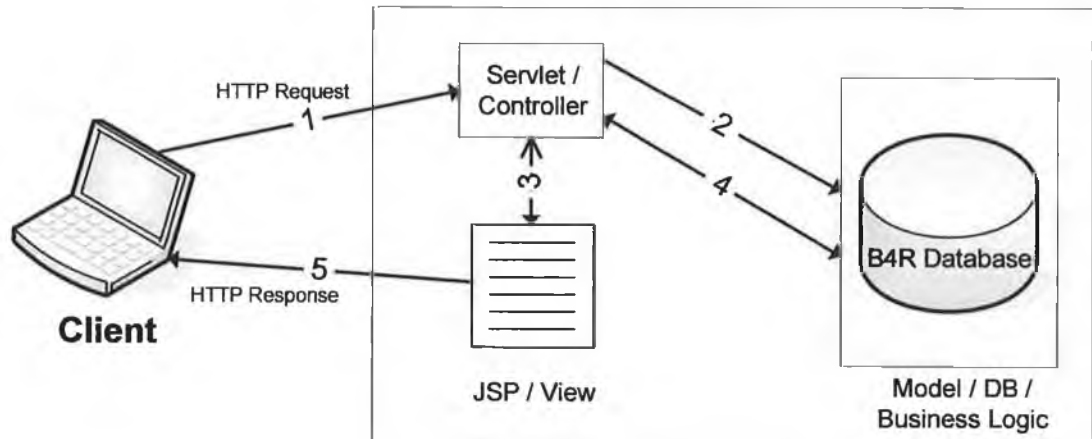


Figure 6.2: MVC in a B4R Transaction

- 1) A client, via a web browser makes a HTTP request. The request can commonly contain form data (such as usernames / passwords). A servlet receives the form data.
- 2) The servlet acts as the controller and processes the request. In most cases this processing involves making requests on the database (or model).
- 3) & 4) The controller obtains the data from the Database and forwards it to the JSP whose job is to generate the page representing the view of the model.
- 5) The view returns a page to the browser via a HTTP response.

The above transaction displays a separation of model, view and controller during the completion of a Billing4Rent service encounter. The B4R prototype was designed using MVC and Java Interfaces which allows any / all components to be interchangeable. This effectively, *future proofs* Billing4Rent by making it possible to change any component. For example, should a newer, more powerful, data storage medium with the capacity for larger numbers of transactions per second become available it would be relatively straight forward to change the way B4R

handles data storage to utilise the newer medium. Due to the fact that B4R used *Interfaces*, a class designed to interact with the new storage medium would only have to implement the **writeData()**, **retrieveData()**, **deleteData()** *etc* methods in order to harness the power of the new data storage mechanisms.

6.1.3 B4R and Code Optimisation

Code optimisation allows (according to Cutts *et al* [86]) “*an executing system to be incrementally improved*” in order for it to run as fast as possible. Designing Billing4Rent with optimised code will provide improved performance. As evidenced in the research contribution chapter (pp 79-123), various recommendations were made to optimise B4R (which was developed using Java, Servlets and JSP), and those recommendations (both Java specific and general) were implemented where possible.

6.1.3.1 Memory Usage

The *Garbage Collector* performs memory management automatically in Java in order to remove objects no longer required by B4R. The more objects that are allocated will potentially result in more frequent and longer garbage collections. Objects in Billing4Rent are only created when they are required. For example, calls to create new *invoice* objects...

```
B4RinvoiceDetails invoice = new rdbmsB4RinvoiceDetails(debug,  
B4RAuditDatabase, B4RAuditUser);
```

...are only made when B4R interacts with invoices and at no other time. Consequently, this displays

6.1.3.2 Session Management

In a Web application, state information relating to each client interaction is typically stored in an HTTP session. The amount of data stored in a session should be minimised, as session data is usually shared, and as such it must be serialised. One method proposed to store session data is the use of hidden form

fields in JSPs to store session data. Figure 6.3 (a code extract from the clientNewinvoice.jsp page) illustrates the use of hidden form fields. This proposal has been adopted throughout the Billing4Rent prototype.

Code Extract from: clientNewinvoice.jsp

```
<td width="25%" class="requiredinput">
  <%
    if (session.getAttribute("invoiceID") != null)
      {
        String invID = (String) session.getAttribute("invoiceID");
        %>
        <input type="TEXT" name="invoiceID" size="30" value="<%=invID%>"
              disabled="TRUE"></input>
        <input type="HIDDEN" name="invoiceID" size="30"
              value="<%=invID%>"></input>
      }
    else
      {
        %>
        <input type="TEXT" name="invoiceID" size="30" disabled="TRUE"/>
        <% } %>
  </td>
```

Figure 6.3: Hidden Form Fields used in B4R

6.1.3.3 Logging

Logging on a system is an important tool, and can be used for:

- Isolating an action which caused a system failure
- Assisting in the recovery of that failure
- Providing an 'audit trail' for a user to track his/her actions.

There are however disadvantages to excessive logging. For example, if the logging program is too complex, it may consume excessive CPU time and memory. Similarly, if the logging program blocks when writing the logs, there is a possibility of causing a denial of service on the system. When employing logging on a system, it is important to determine if the level of logging is adequate for system recovery. Billing4Rent provides for logging to be

configurable: logging can be turned on / off by setting a field (*b4r.debug*) in the B4R.properties file (Figure 6.4).

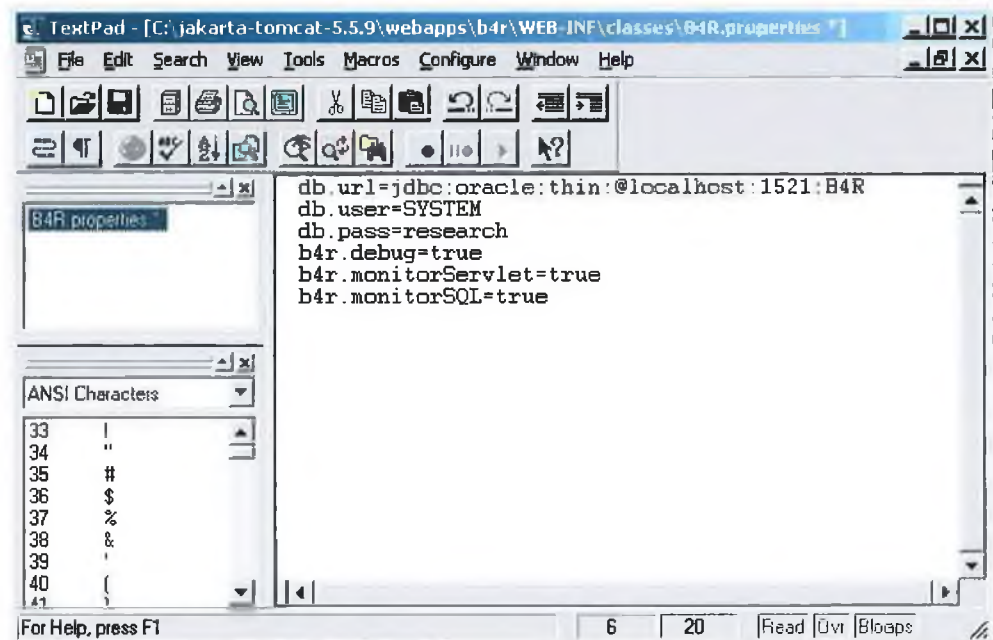


Figure 6.4: B4R.properties

The properties file is read by the B4R application and depending on the setting, logging is performed. When logging is performed, it can also be configured to log all messages or to just log messages with a high priority. This allows B4R to monitor the performance impact of logging and adjust the logging level as necessary - a significant design feature.

6.1.3.4 Database Access

In Java, accessing a database (obtaining and closing a connection to a database) using Java Database Connectivity (JDBC) can be a relatively computationally expensive exercise. Consequently, Chapter Five recommended that JDBC resources should always be released once they are no longer required, as failure to do so can cause memory leaks. The following code extract illustrates how this is implemented in practice...

Code Extract from: rdbmsB4RloggerDetails.java

```
// Connect to the B4R database
B4Rdb.connect();
```

```
// Execute the SQL statement
stmt = B4Rdb.getStatement();
stmt.executeUpdate(SQLstatement);

// Disconnect and close the B4R database connection
B4Rdb.disconnect();
```

Figure 6.5: Closing B4R resources

6.1.3.5 General Coding Considerations

It was recommended in Chapter five that string concatenation be avoided where possible as the use of string concatenation involves the creation of new strings containing the data copied from the original strings. Concatenating strings is a slow process and it also creates additional work for the garbage collector. Using `java.lang.StringBuffer` - as an alternative to `java.lang.String` - can improve performance, and the B4R source code reflects this by using this alternative *e.g.*

Code Extract from: B4RviewLogger.java

```
String errmsg = new StringBuffer("[B4RviewLogger] ")
                .append(e.getMessage())
                .toString();
```

6.1.4 Disaster Recovery

Disaster Recovery is a vitally important component in delivering a Highly Available system. As noted in the literature review (according to Jon William Toigo [63]), Disaster Recovery (DR) and disaster recovery planning, consists of a set of activities aimed at not only reducing the likelihood of a disaster on critical systems, but also minimising their impact. In an ASP environment, downtime will have a two-tier effect. The first tier affected are the clients directly receiving service from the ASP. The second tier are the customers of those clients who will experience the knock-on effect of the ASPs downtime. Typically, in a disaster situation, users are aware that an outage has occurred. However, due to the multi-tiered business model of an ASP, the second tier of customers may not be aware

that the service provider has been compromised. It was advocated in the research contribution (chapter five) that a role be created (*i.e.* a Disaster Recovery Manager) with the sole responsibility for Disaster Recovery and Disaster Recovery Planning. One of the tasks identified for this DR manager would be - in the event of a disaster - to notify clients of the service interruption. Due to the nature of an ASP business model, this allows Billing4Rents clients to notify their own customers of the service interruption, thereby minimising the 'bad press' that inevitably arises as a result of downtime.

Gregor Neuga, *et al* [50] identified two different measurements in which Disaster Recovery can be quantified: *data* loss and *service* loss. Data loss refers to the amount of data an organisation loses as a result of a disaster and is often expressed in how much work must be re-executed once the system has been restored to return all data to previous levels. Service loss refers to the loss of an organisations IT systems (usually from the moment of disaster, right up to the moment the system is restored to normal operation).

When deciding which Disaster Recovery Solution to implement it is important that Billing4Rent estimate the exact cost of downtime in monetary terms, which will allow B4R to establish a DR budget. Figure 6.6 presents a formula put forward by Ian Masters [87], and also referred to in NSI Software's whitepaper on business continuity [66], to estimate the cost of downtime.

Downtime Estimate Formula
<p>Productivity Impact + Revenue Impact = Downtime Estimate</p>
<p>Productivity Impact: Average worker rate or salary x estimated number of business hours the users would be impacted</p>
<p>Revenue Impact: Average monthly gross revenue for the critical application x number of business hours the application is impacted</p>

Figure 6.6: Downtime Estimate Formula

Productivity impact can be calculated on the basis of the average employee salary or rate multiplied by the number of business hours the users are likely to be impacted. Revenue impact can be calculated on the basis of the average monthly gross revenue for the critical application multiplied by the number of business hours that the application is affected. These are then added together to achieve the estimated cost of downtime. Consider the following estimates from the Billing4Rent Innovation Partnerships proposal document [1]:

Number of Clients subscribing to Billing4Rent:	500
Number of Customers per Client:	5000
Monthly rental fee per Customer:	€1
Potential monthly revenue for Billing4Rent:	€2,500,000

Should a system experience a service interruption of 10 minutes (which is all a system designed for 99.9% HA can tolerate on a weekly basis [41], [44], [45], [48]), the estimated cost of downtime would be €422,510:

Average worker salary (estimated):	€30,000
Length Users Impacted:	0.167 hours
B4R Monthly Revenue (estimated):	€2,500,000
Length Application Impacted:	0.167 hours

$$\text{Cost of Downtime: } 30,000 * 0.167 + 2,500,000 * 0.167 = \text{€422,510}$$

As is evidenced in the above figures, the potential revenue loss to Billing4Rent in the event of a service outage is considerable. However, it worth noting that the *Downtime Estimate Formula* is commonly used to estimate downtime in a 'real-time' system, where continuous operation is essential to an organisation (e.g. a mobile telephone operator). Due to the batch-processing nature of B4R transactions, it is conceivable that a 10-minute interruption to Billing4Rent services would not impact B4R clients to the same degree as a real-time system. Consequently, the estimated cost of downtime for Billing4Rent could be considerably lower than €422,510. Regardless of the estimated cost of downtime, the Disaster Recovery solution in place must be adequate to quickly restore service

operations. The DR solution proposed for Billing4Rent equates to level six in the *Seven Levels of Availability* proposed by the IBM associated SHARE organisation [68]. Level six in the Seven Levels of Availability guidelines applies database updates to both the local and remote copies of the databases with a single commit. A commit is not completed until both the primary and secondary locations are updated. Level six requires hardware on the secondary platform with the ability to accept the workload of the primary site during an outage. The typical recovery time associated with Level six is usually less than 12 hours.

In a worst-case scenario, should the Billing4Rent organisation be inoperable for 12 hours, the potential cost of downtime could be considerable. Therefore, it may be necessary to implement Level seven of the Seven Levels of Availability, which guarantees zero data loss. In the event of a disaster, an immediate and automatic transfer of operations is made to the secondary site, resulting in a typical recovery time of only a few minutes. The associated cost of implementing Level seven may prove prohibitive to many organisations, but with the potential cost of downtime resulting in €422,510 for every 10 minutes Billing4Rent is inoperable, it could well be, in the opinion of this author, an expenditure well worth making.

6.2 B4R Prototype and Accepted Benchmarks

The following section details accepted benchmarks for the various components of Billing4Rent. When taken separately, each component is a proven, industry-leading product, which delivers high levels of performance. Therefore, the use of these components in Billing4Rent guarantees a high level of performance for the B4R service as a whole.

6.2.1 Oracle Database

The Transaction Processing Council (TPC) [88] is a non-profit organisation founded to define transaction processing and database benchmarks. One of the benchmarks the TPC use to evaluate Database performance is known as **TPC-C**, where transactions are executed against a target database and their *throughput* is

measured. The TPC-C benchmark simulates the activities of an order-entry environment (*i.e.* transactions include entering and delivering orders, recording Payments, checking the status of orders, and monitoring the level of stock at the warehouses)[†]. Therefore the TPC define throughput as “*how many New-Order transactions per minute a system generates while the system is executing four other transactions types (Payment, Order-Status, Delivery, Stock-Level)*”. The primary metric produced is the transaction rate (tpmC). A system exhibiting a tpmC of 500 is capable of generating 500 *New-Order* transactions per minute, while simultaneously executing the rest of the TPC-C transaction workload. The database chosen for the B4R prototype is the Oracle database [75], version 9i. Similarly, the billing and transaction engine Singl.eView, to be incorporated into the B4R service also runs on an Oracle RDBMS. The current TPC benchmark for Oracle 9i, running on a HP-UX 11i UNIX machine, is 423,414 tpmC – 423,414 *New-Order* transactions while simultaneously executing the rest of the TPC-C transaction workload. This figure shows Oracle to be a reliable, industry leading RDBMS solution, capable of providing Billing4Rent with sufficiently high levels of performance.

Additionally, should Billing4Rent upgrade to Oracle 10g in the future, it is notable that Oracle 10g is capable of 1,601,784 tpmC – 1,601,784 *New-Order* transactions while simultaneously executing the rest of the TPC-C transaction workload, and was the first RDBMS to exceed one million transactions per second [89] in November 2003.

[†] The TPC stress that it is not their intent to specify how to best implement an Order-Entry system. While the benchmark simulates the activity of a wholesale supplier, TPC-C is not limited to the activity of this particular business segment.

6.2.2 Jakarta Tomcat Webserver

The Standard Performance Evaluation Corporation (SPEC) [90] is a non-profit corporation formed to establish and maintain standardised set of benchmarks, which can be applied in high-performance computing. One of the SPEC benchmarks is *SPECweb2005*, which can be used to evaluate the performance of Webserver's. SPECweb2005 measures the maximum number of simultaneous connections a secure web server is able to support while still meeting specific throughput and error rate requirements, *i.e.*

- Simultaneous user sessions
- Dynamic content (*i.e.* JSP and PHP)
- Page image requests using 2 parallel HTTP connections
- Simulates standard workloads –
 - Banking (HTTPS)
 - E-commerce (HTTP and HTTPS)
 - Support (HTTP)
- Simulates browser caching
- File accesses

SPECweb2005 results are measured in simultaneous user connections. The Webserver used for the Billing4Rent prototype is the Jakarta Tomcat [76] web-container. The overall SPEC benchmark for Jakarta Tomcat using Tomcat version 5.5.9, on a Dell PowerEdge 2850 server with a SuSE Linux Operating System (OS) is 7881. The benchmarked figure indicates that Tomcat is capable of 7881 operations per second.

6.2.3 Java Virtual Machine

Another benchmark provided by The Standard Performance Evaluation Corporation (SPEC) is used to measure the performance of Java runtime environments. *SPECjbb2005* is designed to emulate a 3-tier web application and, as in the previous SPEC benchmark - the system modelled is a wholesale company. The benchmark simulates a users interaction with the wholesale company (*i.e.*

placing new orders, requesting the status of an existing order). Additionally, the wholesale company itself also processes orders for delivery, entering customer Payments, checking stock levels, as well as the ability to request a report on any user. For the benchmark test, users map directly to Java threads, which execute operations in sequence. As the benchmark test runs, the number of threads increase. For Billing4Rent, the JVM used was Sun Microsystems Java Platform, Standard Edition (Java SE, formerly known as J2SE) [74], version 1.5.0_06. The performance of JVMs is measured in Business Operations Per Second (BOPS), and a Java Runtime Environment (version 1.5.0_06) running on a SuSE Linux Enterprise Server 9 was benchmarked at performing 26,698 operations per second. The following table (Figure 6.7) provides a summary of the benchmarks achieved for the various components used in the B4R prototype.

Component	Technology	Hardware	Metric	Date
Database	Oracle 9i	HP 9000 Superdome	423,414 tpmC	August 2002
	Oracle 10g	IBM eServer	1,601,784 tpmC	April 2005
Webserver	Jakarta Tomcat 5.5.9	Dell PowerEdge 2850	7881 simultaneous connections	September 2005
JVM	Java SE (1.5.0_06-b05)	Tyan S2865	26,698 BOPS	February 2006

Figure 6.7: Component Benchmark Summary

6.3 Outsourcing B4R Deployment

An organisation may outsource to another company (e.g. a consultancy firm or an Application Service Provider), to provide a service that - although the organisation has the ability to provide itself - can be accomplished more efficiently or more cost

effectively when performed using a third-party resource. According to Claver *et al* [91], in Information Technology (IT) terms, outsourcing “*means that the physical and / or human resources related to an organisations information technology needs are supplied and / or administered by an external specialised supplier*”. An outsourcing contract can be for a temporary period of time or can run indefinitely. Equally, organisations can outsource all of their IT needs or only parts of it. The primary reasons for outsourcing suggested by Lonsdale and Cox [92] include the following:

- **Focus resources on core activities:** for example, an insurance company which outsources its IT needs (*e.g.* e-mail or data storage) will have greater resources available to devote to activities critical to their business like customer interaction, and offering reduced premiums.
- **Cost reduction:** IT outsourcing can save an organisation 20% - 25% over maintaining the same applications in house (reports Kate Gerwig [93]), while Ravi Patnayakuni and Nainika Seth place the cost savings (upfront and total) at between 30% and 70% [22].
- **Innovation:** Patnayakuni and Seth also allude to the fact that outsourcing IT operations allows organisations to have access to state-of-the-art hardware and software from the outsourcer [22]. An outsourcing company can afford to provide up-to-date software and hardware as well as the newest upgrades as the total cost of providing new solutions is spread across multiple customers.
- **Availability, Scalability and Performance:** Organisations who provide IT outsourcing capabilities will generally invest heavily in backup and redundant systems in order to minimise service disruption. In most cases (as Walsh [20] notes), these safeguards are above and beyond what many small to midsize companies can afford. Tao [24] also subscribes to this view and suggests that providers do a better job of ensuring 24 / 7 application availability than customers could. Additionally, organisations can scale up and down based on actual usage [22].

6.3.1 Outsourcing Features & Benefits

The following list illustrates some of the features (co-location services) commonly provided by outsourcing providers:

- 24 x 7 x 365 operation
- Full systems redundancy
- Heating Ventilation Air-Conditioning (HVAC) systems
- Redundant Power with UPS and generator systems
- Communications with multiple fibre routes
- Security
 - 24-hour Security Guard in many cases
 - CCTV monitoring
 - Intruder-detection systems
- Data storage in fire-proof cages
- Comprehensive Service Level Agreements

6.3.2 B4R Outsourcing

While it is feasible for Billing4Rent to purchase and configure the equipment necessary to provide Availability, Scalability and Optimal system performance, the associated cost may make outsourcing the deployment of Billing4Rent a more attractive proposition. A hosting company able to deploy Billing4Rent in its data centre will be able to do so for a fraction of the cost involved in self deploying B4R. There are numerous hosting companies that guarantee 100% application availability, scalability, comprehensive Disaster Recovery policies, and are able to distribute the costs involved across multiple clients. Should there be a legal requirement that Billing4Rent be located physically on Irish soil there are a number of data centres available. An example of an Irish based data centre would be DataElectronics [94], located in Dublin. See Figure 6.8 for breakdown of DataElectronics Co-location package...

Data Electronics Co-location Package	Managed Services
<ul style="list-style-type: none"> • 19" standard cabinet (42Ux600x900) • Redundant power (A&B) • Power guaranteed at 100% availability • A constant temperature is maintained on the data centre floor by N+1 Close Control Units • All cooling charges (again up to 2.2KW) are included in the co-location charges • Each cabinet has front and rear-locking perforated doors • 24x7 Access to the data centre. • Internal & external CCTV security. • Dual Fire Detection. • Extensive leak detection systems. 	<ul style="list-style-type: none"> • Managed Internet (100% availability) • Managed Security (Managed Cisco PIX and Nokia Checkpoint) • Managed Load Balancing (Managed Alteon load balancing service) • Managed Storage (Storage Area Network on a High Availability Hitachi SAN) • Managed Back-Up (Real-time and encrypted Tape and SAN backup solution offerings) • Managed Platforms (Windows, Unix, Linux, Exchange, Citrix etc.) • Monitoring Services (All Server, Telecoms and Software Systems monitored)

Figure 6.8: DataElectronics Co-location package

A selection of other Irish based data centres, offering similar services:

- TeleCity [95]
- Hosting365 [96]
- Ketec [97]

Additionally, the following non-Irish based hosting companies could potentially host Billing4Rent while offering 100% service availability:

- FirstServ (London) [98]
- Below Zero (Edinbrough) [99]
- Verio (New York) [100]
- PowerTel (Melbourne) [101]
- PSINet (Paris) [102]

6.4 Chapter Summary

This chapter provided an evaluation of the proposed framework detailed in chapter five. The evaluation was conducted in order to determine if the framework proposed fulfilled its objectives. Secondly, the evaluation included a comparison of the individual components of Billing4Rent, against accepted industry benchmarks.

In order to determine if the framework fulfilled its objectives, the proposed B4R architecture was evaluated to ascertain if it displayed any Single Points of Failure, whose occurrence could potentially cause downtime to the entire B4R system. The design of the B4R prototype was examined and its use of the Model / View / Controller architecture was highlighted. The prototype was also appraised to determine if the recommendations for code optimisation outlined in chapter five were implemented. This chapter also evaluated the Disaster Recovery plan recommended for Billing4Rent and how the costs associated with downtime not only effect B4R clients but also their customers. Consequently, this evaluation highlighted the fact that a more comprehensive DR plan may be necessary.

Finally, this chapter also provided a recommendation for the deployment of Billing4Rent in an external data center. This recommendation is justified not only by the cost saving accruing from outsourcing B4R deployment but also from the guarantees hosting companies give with respect to application availability.

Chapter Seven: Research Conclusion

The primary focus of this project has been the study and development of a framework (including the architecture and system configuration required) to ensure High Availability and Optimum System Performance in an Application Service Provider (ASP) Environment. The ASP solution (Billing4Rent) provided the real world platform on which any recommendations or proposals arising out of the body of research were implemented and allowed this research to be objectively analysed and quantified in a real world environment. This thesis investigated the following areas:

- **Application Service Provision:** An Application Service Provider (ASP) is a third party entity that deploys, hosts and manages access to an application or service. The service is delivered to customers across a wide area network (usually the public Internet), and applications are typically provided on a subscription or rental basis. ASP has had many guises over the years, from 'Software-as-a-Service' (SaaS), 'on-demand' computing, as well as the current branding as 'hosted application management'. ASPs provide access to software on a one-to-many basis. Consequently, the cost of ownership and maintenance of the solution is shared by several clients, which introduces economies of scale for end users. Despite initial projections, user uptake of the ASP model has been slow to materialise. This is mostly due to the downturn of *dot.com* share prices in early 2000, which caused the closure of many on-line service providers, and which severely dented the confidence of many with regard to renting application access. In the past few years however, uptake of the ASP model is increasing – indeed current estimates on software-as-a-service spending predict revenue of \$15.2 billion by 2007.
- **High Availability:** High availability refers to a system (or system component) that remains continuously operational for prolonged periods of time. The goal of a fault-tolerant application is to be available 24 x 7 x 365,

which minimises potential losses that may be incurred if the application suffers downtime. Typically, availability is measured relative to '100% operational' availability. For many organisations a desired level of availability is 99.999%, commonly referred to as 'five nines'. This level of reliability, for a service running 24 x 7 x 365, equates to just over five minutes of downtime per year, or six seconds downtime per week.

- **Scalability:** Scalability is the ability of a product - either hardware or software - to continue to function appropriately when it is changed in size or volume, usually to facilitate increasing workloads. In the majority of cases, an application will scale to a larger size or volume in order to accommodate a larger number of users and processes on a system. When designing a scalable system, a common goal is to develop a system, which not only functions in the rescaled environment but also takes advantage of the additional resources in terms of increased performance.

7.1 Summary of Research Contribution

Chapter five detailed the proposed ASP framework. The functional requirements for developing the Billing4Rent ASP were also outlined and an online Billing 'prototype' was developed. The aim of developing this prototype was to provide a platform on which all generic recommendations or proposals could be implemented. The B4R prototype consists of a B4R client website where clients can add, remove, update customers, products / services, and invoices. Additionally, an 'Administrative Interface' was created in order to manage and administer the B4R ASP prototype. The Administrative Interface allows authorised B4R personnel to create and manage user accounts for clients. From the Administrative site, authorised personnel can also create users, view statistics relating to B4R storage media (*e.g.* Database or Filesystem) usage, as well as view audit and error logs. A monitoring program is also available and provides the ability to Monitor B4R and collect statistics relating to page hits, page completion times etc. The research contribution also detailed the Proposed Network Architecture for an ASP, and decomposed this proposed architecture into its constituent parts. Recommendations for each of these parts were

also suggested. Source code optimisations in order to improve the prototype system performance were also detailed. Finally, Chapter Five also documented the detail of a Disaster Recovery Plan suitable for restoring service to ASP tier-1 level clients and their respective clients in turn in the event of a disaster situation.

The research contribution was subsequently evaluated in order to determine if the proposed framework fulfilled its objectives. The proposed architecture was evaluated to ascertain if it displayed any *Single Points of Failure*, the occurrence of which could potentially cause downtime to the entire system. The design of the B4R prototype (including the use of the Model / View / Controller architecture) was highlighted and evaluated. The evaluation also considered the Disaster Recovery plan recommended for Billing4Rent and how the costs associated with downtime affect clients and their customers. In the case of B4R, it was recommended that the deployment of the service be outsourced to an external data center. Given the background business model, this recommendation is justified across cost and service level / availability criteria.

7.2 Research Conclusions

The Computing Industry has come full circle from *Mainframe Batch Orientated Processing* of the 1950's, 60's and 70's, through to batch orientated centralised processing ASP's such as B4R and others. Although the ASP business model came to prominence during the late 1990's, application hosting dates back to the 1960s. Indeed, Electronic Data Systems (EDS) was one of the first ASPs, renting time on their mainframe computers to customers who either could not afford a mainframe or who did not have enough usage to justify the purchase of a dedicated mainframe [103]. Known as the first wave of outsourcing - or *time-sharing* [104] - commercial organisations and educational institutions took turns renting mainframe-processing capabilities. The mainframe performed all the processing and all memory and storage resided on the mainframe. Access to the mainframe was via a 'dumb terminal'. The terminal, which was not capable of any processing, but relayed messages to and from the mainframe. The time-sharing approach allowed organisations to access applications online on a pay-per-use basis rather than purchasing and maintaining

hardware and / or software at the organisations physical site. The 1980's and 90's saw the emergence of the Personal Computer (PC), which allowed applications to be installed on individual machines. However, towards the end of the 1990's, organisations began outsourcing various applications to Application Service Providers, who would charge them on a per-usage basis. While the user interface, processing speed and the speed of the network connection have all increased exponentially, many of the principals of time-sharing remain essentially the same.

Scalability and High Availability are becoming increasingly important components of overall system architectures. For many applications, their function is well understood, well coded, and well documented for the end user. In many application domains, there are numerous competing software products essentially delivering the *same* level of features to their user base. Consequently, the ability of the application to deliver high performance levels regardless of user load, or to scale gracefully is becoming a decisive selling point. Application features are becoming more a *given*, application performance and cost-effective scalable platform configurations are emerging as the distinguishing features for software solutions. For example, in terms of B4R, the concept of *billing* is well understood by potential users of the service, and, there are numerous billing solutions for them to choose from in the market place. The innovation of Billing4Rent lies in the ASP solution architecture; the billing application will be hosted and available to end-users on a subscription / rental basis. This hosting innovation is heavily reliant on scalability and availability considerations in order to deliver a service with high (expected) levels of performance and with minimum service interruption. As was evidenced in the literature review chapter on high availability and scalability (pp 26-77), a study conducted by Compaq found that 29% of enterprise customers can tolerate only 0-3 seconds of downtime per outage of their critical applications. A further 37% of these customers can tolerate only up to 3 minutes of downtime per outage of their critical applications. These figures clearly indicate that two out of every three respondents will only tolerate up to 3 minutes of downtime per outage. Similarly, Kern *et al* [9] report that 85% of potential ASP customers rate quality of service as being one of the key factors in ASP satisfaction. Various sources - including Nozar Daylami *et al* [37], Barry Jaruzelski *et al* [7] and Bhavini Desai *et al* [17] - cite performance

considerations in terms of high-availability, scalability and reliability as the major inhibitors to the widespread uptake of the ASP model.

With revenues set to rise in the coming years, it is worth noting the emergence into the ASP marketplace of many of the larger players in the software industry. This trend would suggest that many organisations view delivery of software-as-a-service as being a potentially rich source of income. Consequently, the ASP delivery model may be utilised on a more regular basis for future software releases. For example, some of the worlds largest technology companies have ASP offerings: Oracle On Demand [105] is a hosted Customer Relationship Management (CRM) solution designed to manage an organisations IT infrastructure, software, security, service levels *etc.* Similarly, Office Live [80] from Microsoft provides an organisation with e-mail accounts, business management applications, as well as a Web site managed and maintained by Microsoft. Additionally, Google have recently (2006) added a word-processing package to their suite of products. 'Wrightly' [82] will provide an alternative to Microsoft Word, and will be delivered to users via the web.

7.2.1 Future Research Potential

The author identifies significant further research potential in the following areas:

- Formal benchmarking of ASP solutions. In the research evaluation chapter of this thesis (pp 125-143), a comparison of accepted benchmarks for the various components of Billing4Rent, for example, was documented. Each component, when taken separately, is a proven, industry-leading product, capable of delivering high levels of performance. The use of these components in Billing4Rent should guarantee a high level of performance for the B4R service as a whole. Future research could consider issues surrounding the benchmarking of individual components in a system versus the benchmarking of the system as a whole.
- The influence of the ASP and software-as-a-service paradigm on application design and development methodologies. If the premise is correct that application features are essentially expected / given, and that

application performance and cost-effective scalable platform configurations are emerging as the distinguishing features of software solutions, to what extent does that affect current design and development methodologies going forward. Many accepted methodologies are 'features' driven and tend to focus on iterative prototype development, testing, integration etc....

- The formal integration of Singl.eView and the B4R prototype. The Billing4Rent prototype was designed using the Model / View / Controller design pattern, in order to allow interchangeability between various components of the Billing4Rent prototype. This design allows the Singl.eView rating and billing engine to be *plugged-in* to the B4R prototype, to provide B4R billing capabilities. Currently, the prototype *back-end* is an Oracle RDBMS.
- Enhancement of the *logging* capabilities of Billing4Rent. Currently, the B4R prototype allows logging to be either enabled or disabled. If the logging is enabled, different logging levels can be specified to log all messages or to just log messages with a high priority. This allows B4R to monitor the performance impact of logging and adjust the logging level as necessary. A future enhancement would be the setting of logging levels on an individual basis for each client.

Bibliography

- [1] Innovation Partnerships – 2005,
“Billing4Rent – The ASP Hosted Billing Service”, *Enterprise Ireland, ADC, WIT, GMIT partnership documentation*.
- [2] Tim O’Reilly – 2005,
“What Is Web 2.0”, *Design Patterns and Business Models for the Next Generation of Software*,
<http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1>,
Accessed: 25th January 2006
- [4] Dr. Philip Seltsikas and Prof. Wendy L. Currie – 2002,
“Evaluating the Application Service Provider (ASP) Business Model: The Challenge of Integration”,
Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35 2002)
- [3] Information Technology Association of America (ITTA) – 2000,
“ITAA’s Application Service Provider (ASP) Service Level Agreements (SLA) Guidelines”,
<http://www.itaa.org/isecc/pubs/e200009-2.pdf>,
Accessed: 8th March 2005
- [5] Ben Pring – 2004,
“ASP Transforms Into ‘On Demand’ and ‘Software-as-a-Service’”, *Gartner Research – ID Number: G00124529*,
http://www.gartner.com/DisplayDocument?ref=g_search&id=459608,
Accessed: 7th March 2005.
- [6] Barry M. Rubenstein, *et al* – 2005,
“From the Ashes and Beyond: The Evolution of ASPs into Hosted Application Management Providers”, *IDC Insight – Document Number: 33408*
- [7] Barry Jaruzelski, Frank Riberio, Randy Lake – 2001,
“ASP 101: Understanding the Application Service Provider Model”, *Booz, Allen, Hamilton Consulting*,
<http://extfile.bah.com/livelink/livelink/61813/?func=doc.fetch&nodeid=61813>,
Accessed: 11th March 2005
- [8] David Greschler, Tim Mangan – 2002,
“Networking lessons in delivering ‘Software as a Service’ – Part 2”, *International Journal of Network Management 2002; 12: 339 (DOI: 10.1002/nem.447)*
- [9] Thomas Kern, Mary Cecelia Lacity, Leslie Willcocks, Mary Lacity – 2002,
“Netsourcing: Renting Business Applications and Services Over a Network”,
ISBN: 0-13-092355-9, *Pretence Hall Publishing Inc.*
- [10] David Greschler, Tim Mangan – 2002,
“Networking lessons in delivering ‘Software as a Service’ – Part 1”, *International Journal of Network Management 2002; 12: 317 (DOI: 10.1002/nem.446)*
- [11] Lixin Tao – 2001,
“Shifting Paradigms with the Application Service Provider Model”, *IEEE Computer Society*, October 2001 (Vol. 34, No. 10) pp. 32-39
- [12] Bill. Vassiliadis, K. Giotopoulos, K. Votis, S. Sioutas, N. Bogonikolos, S. Likothanassis – 2004,
“Application Service Provision through the Grid: Business models and Architectures”, *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC’04)*

- [13] Bandula Jayatilaka, Andrew Schwarz, Rudolf Hirschheim – 2002,
 “Determinants of ASP Choice: an Integrated Perspective”, *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35 2002)*
- [14] David J. Roddy – 1999,
 “The Internet-based ASP Marketplace”, *Deloitte Consulting and Deloitte & Touche Research*
- [15] Anjana Susarla, Anitesh Barua, Andrew B. Whinston – 2001,
 “Making the Most out of an ASP Relationship”, *IEEE Computer Society*
<http://csdl.computer.org/comp/mags/it/2001/06/16063abs.htm>,
 Accessed: 7th March 2005
- [16] Mark Clancy – 2001,
 “The Insidious Resistance to ASPs”, *Internet News Article*,
<http://www.internetnews.com/bus-news/article.php/738321>,
 Accessed: 6th March 2005
- [17] Bhavini Desai, Vishanth Weerakkody, Wendy Currie, D. E. Sofiane Tebboune, Naureen Khan – 2002,
 “Market Entry Strategies of Application Service Providers: Identifying Strategic Differentiation”,
Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36 2003).
- [18] Bhavini Desai, Prof. Wendy L. Currie – 2003,
 “Application Service Providers: A model in Evolution”, *The Fifth International Conference on Electronic Commerce, ICEC 2003*
- [19] Rob Garretson – 2005,
 “The ASP reincarnation”, *NetworkWorld Print Edition*, 29th August 2005
- [20] Kenneth R. Walsh – 2003,
 “Analysing the Application ASP Concept: Technologies, Economies, and Strategies”,
Communications of the ACM, August 2003/Vol. 46, No. 8
- [21] Thomas Kern, Jeroen Kreijger, Leslie Willcocks – 2002,
 “Exploring ASP as sourcing strategy: theoretical perspectives, propositions for practice”, *Journal of Strategic Information Systems 11 (2002) 153-177*
- [22] Ravi Patnayakuni, Nainika Seth – 2001,
 “Why Licence When You Can Rent? Risks and Rewards of the Application Service Provider Model”,
Proceedings of the 2001 ACM SIGCPR conference on Computer personnel research.
- [23] Rivka Ladin, Barbara Liskov, Liuba Shrira, Sanjay Ghemawat – 1992,
 “Providing High Availability Using Lazy Replication”, *ACM Transactions on Computer Systems, Volume 10, No. 4*
- [24] Thomas Kern, Jeroen Kreijger – 2001,
 “An Exploration of the Application Service Provision Outsourcing Option”, *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34 2001)*
- [25] Beth Cohen – 2004,
 “Consider Purchasing Real-Time Collaboration Applications Through An ASP”, *InformationWeek Advisory Council*,
<http://www.informationweek.com/story/showArticle.ihtml?articleID=55800253>,
 Accessed: 10th March 2005
- [26] The Economist – 2006,
 “Economies of Scale”, *Economics A-Z*,
<http://www.economist.com/research/Economics/alphabetic.cfm?TERM=ECONOMIES%20OF%20SCALE#ECONOMIES%20OF%20SCALE>,
 Accessed: 23rd January 2006

- [27] Michael Miley – 2000,
“Reinventing Business: Application Service Providers”, *Oracle Magazine*, November/December
Edition 2000
- [28] Tinabeth Burton – 2002,
“TAA Survey Shows ASP Customers Achieve Real Benefits from Outsourcing”, *Information
Technology Association of America*
<http://www.itaa.org/news/pr/PressRelease.cfm?ReleaseID=1017252264>,
Accessed: 8th March 2005
- [29] Sybex – 2002,
“Networking Complete – 3rd Edition”,
ISBN: 0-7821-4143-9, *Network Press*
- [30] Salesforce.com,
<http://www.salesforce.com/>
- [31] IBM’s Applications on Demand (formerly Corio),
<http://www-1.ibm.com/services/us/index.wss/itservice/aod/a1011244>
- [32] Korbi.net,
<http://www.korbi.net>
- [33] LearningStation.com,
<http://www.learningstation.com/>
- [34] Usinternetworking,
<http://www.usi.net/>
- [35] The Economist – 2005,
“Microsoft – The Latest in Memoware”, *The Economist North American Print Edition*, 19th November
2005
- [36] Kathleen Simpson – 2002,
“Data/Voice Convergence”, *Gartner Research – ID Number: M-14-5991*,
http://www.gartner.com/DisplayDocument?doc_cd=104600&ref=g_fromdoc,
Accessed: 19th April 2005
- [37] Nozar Daylami, Terry Ryan, Lorne Olfman, Conrad Shayo – 2005,
“Determinants of Application Service Provider (ASP) Adoption as an Innovation”, *Proceedings of the
38th Hawaii International Conference on System Sciences (HICSS-38 2005)*
- [38] Hewlett-Packard – 2000,
“High Availability Overview”, *Hewlett-Packard: High Performance Computing White Paper*
- [39] Christopher G. Willard – 2005,
“Assessing the Costs of Computer System Failures in Technical Environments”, *IDC Customer
Survey 2000*
- [40] Cyberguard White Paper – 2002,
“The Business Case For High Availability (HA)”,
http://www.cyberguard.com/download/white_paper/en_cg_business_case.pdf,
Accessed: 6th April 2005
- [41] Jim Gray, Daniel P. Siewirok – 1991,
“High-Availability Computer Systems”, *IEEE Computer Archive Volume 24, Issue 9*

- [42] IEEE Task Force on Cluster Computing – 1999,
 “High Availability (HA)”, *IEEE Task Force on Cluster Computing*,
<http://www.ieccetfcc.org/high-availability.html>,
 Accessed: 30th March 2005
- [43] David Oppenheimer, David A. Patterson – 2002,
 “Architecture and Dependability of Large-Scale Internet Services”, *IEEE Internet Computing, Volume 6, Issue 5, Sept – Oct 2002*
- [44] Han Wang, Hao Wang, and Jinmei Shen – 2004,
 “Architectural Design and Implementation of Highly Available and Scalable Medical System with IBM Middleware”, *Proceedings of the 17th IEEE Symposium on Computer-Based Medical Systems (CBMS’04)*
- [45] Gene J. Schroeder, Marvin M. Johnson – 1988,
 “Simulation: The correct approach to complex availability problem”, *Proceedings of the 1988 Winter Simulation Conference*.
- [46] Kai Hwang, Zhiwei Xu – 1998,
 “Scalable Parallel Computing”,
 ISBN: 0-07-031798-4, *WCB/McGraw-Hill Companies Inc.*
- [47] Compaq – 1997,
 “Five Levels of High Availability”, *Compaq High Availability White Papers*
- [48] Jane Dallaway – 2003,
 “Microsoft Seminar on SQL Server 2000 High Availability”,
<http://www.ieccetfcc.org/high-availability.html>,
 Accessed: 5th April 2005
- [49] Vasfi Gucer, Satoko Egawa, David Oswald, Geoff Pusey, John Webb, Anthony Yen – 1999,
 “High Availability Scenarios with IBM Tivoli Workload Scheduler and IBM Tivoli Framework”,
 ISBN: 0-738-49887-4, *IBM Redbooks*
- [50] Gregor Neaga, Pierluigi Buratti, Christian Labauve, Gordon Raffel, Allan Sarlo, Maryela Weihrauch – 1998,
 “Continuous Availability S/390 Technology Guide”,
 ISBN: 0-738-41231-7, *IBM Redbooks*
- [51] Steve Russell – 2001,
 “High Availability without Clustering”,
 ISBN: 0-738-41966-4, *IBM Redbooks*
- [52] The Provident Bank – 2002,
 “The Value of Customer Loyalty”, *Business Finance Articles*,
<http://hasslefree.providentconnect.com/contentmanager/www/business/BFList.aspx?ArticleTypeID=2>
 Accessed: 17th May 2005
- [53] Evan Marcus, Hal Stern – 2003,
 “Blueprints for High Availability – 2nd Edition”,
 ISBN: 0-471-43026-9, *Wiley Publishing Inc.*
- [54] Sun Microsystems – 2003,
 “Seven Steps to Highly Available Systems”, *Sun Inner Circle*,
<http://www.sun.com/emrkt/innercircle/newsletter/feature0203.html>,
 Accessed: 15th April 2005
- [55] Hilary H. Hosmer – 1996,
 “Availability Policies in an Adversarial Environment”, *Proceedings of the 1996 Workshop on New Security Paradigms*

- [56] Hewlett-Packard – 2003,
 “Increasing availability with ProLiant Essentials Workload Management Pack”, *Hewlett-Packard: High Availability Computing White Papers*
- [57] Tim Burke – 2000,
 “High Availability Cluster Checklist”, *ACM Linux Journal Vol 2000, Issue 80es*
- [58] BEA – 2001,
 “Achieving Scalability and High Availability for E-Business”, *BEA Systems Inc. White Paper*
- [59] The Internet Engineering Task Force – 2006,
 “RFC 1034”, *Domain Names – Concepts and Facilities*,
<http://www.ietf.org/rfc/rfc1034.txt>,
 Accessed: 23rd January 2006
- [60] Prasad Jogalekar, Murray Woodside – 2000,
 “Evaluating the Scalability of Distributed Systems”, *IEEE Transactions on Parallel and Distributed Systems Volume 11, No. 6 (June 2000)*
- [61] André B. Bondi – 2000,
 “Characteristics of Scalability and Their Impact on Performance”, *Proceedings of the 2nd International Workshop on Software and Performance*
- [62] H. Edward Donley – 2005,
 “Amdahl’s Law”, *Indiana University Mathematics Department*,
<http://www.ma.iup.edu/~hedonley/ma451/amdahl.html>,
 Accessed: 7th June 2005
- [63] Jon William Toigo – 2003,
 “Disaster Recovery Planning: preparing for the unthinkable”,
 ISBN: 0-13-046282-9, *Pearson Education, Inc.*
- [64] Jon William Toigo – 1996,
 “Disaster Recovery Planning: For Computers and Communication Resources”,
 ISBN: 0-471-12175-4, *John Wiley & Sons, Inc.*
- [65] Stephen Pritchard – 2002,
 “Alarm bells ring over plans for disaster recovery”, *Financial Times IT Specials*,
<http://specials.ft.com/spdocs/FT333ULZXAD.pdf>
 Accessed: 6th February 2006
- [66] NSI Software – 2003,
 “6 Tips Small and Midsize Businesses Can Use to Protect Their Critical Data”, *NSI Software: White Papers*,
<http://www.nsisoftware.com/pdf/MSNSI%20SMB%20white%20paper%20FINAL%20104%20Formatted.pdf>,
 Accessed: 8th June 2005
- [67] Hewlett-Packard – 2004,
 “Better backups through Replication”, *HP White Papers*,
http://i.i.com.com/cnwk.1d/html/itp/66_better_backups_replication.pdf,
 Accessed: 8th June 2005
- [68] SHARE – 2005,
 “About Us”, *Share Inc*,
<http://www.share.org/About/>,
 Accessed: 13th June 2005

- [69] SingleView,
<http://www.intec-telecom-systems.com/its/productservices/singleview/>
- [70] Intec – 2005,
“Intec wins \$15 million SingleView billing contract with MTN, Africa’s leading wireless operator”,
Intec Telecom Systems Press Release,
<http://www.intec-telecom-systems.com/its/pressroom/pressreleases/pr2005/2005-02-28/>
Accessed: 4th April 2006
- [71] Intec – 2005,
“SingleView – Creating Profitable Customers”, *Intec Telecom Systems Product Brochure*,
<http://www.intec-telecom-systems.com/its/siteservices/downloads>
Accessed: 21st December 2005
- [72] Barb Lundhild– 2005,
“Oracle Real Application Clusters 10Gg”, *An Oracle Technical Whitepaper – May 2005*,
http://www.oracle.com/technology/products/database/clustering/pdf/TWP_RAC10gR2.pdf,
Accessed: 3rd May 2006
- [73] Hugo Haas – 2005,
“Web Services Activity Statement”, *World Wide Web Consortium (W3C) Activity Statement*,
<http://www.w3.org/2002/ws/Activity>
Accessed: 24th February 2006
- [74] Java, Java Servlets and Java Server Pages Specifications,
<http://java.sun.com/>
- [75] Oracle Database
<http://www.oracle.com/database/index.html>
- [76] The Apache Jakarta Project
<http://jakarta.apache.org/>
- [77] Eric Gamma, – 1995,
“Design Patterns: Elements of Reusable Object-Orientated Software”,
ISBN: 0-201-63361-2, *Addison-Wesley Publishing*.
- [78] JAMon – 2006,
“A Monitoring API”, *JAMon (Java Application Monitor) Users Guide*,
<http://jamonapi.sourceforge.net>
Accessed: 4th April 2005
- [79] Birgit Roehm, Balazs Csepregi-Horvath, Pingze Gao, Thomas Hikade, Miroslav Holec, Tom Hyland, Namie Satoh, Rohit Rana, Hao Wang– 2004,
“IBM WebSphere V5.1, Performance, Scalability, and High Availability – 2nd Edition”,
ISBN: 0-738-49780-0, *IBM Redbooks*
- [80] Microsoft Office Live
<http://officelive.microsoft.com/>
- [81] L. Haber – 2004,
“ASPs Still Alive and Kicking”, *ASPnews.com – Trends*
<http://www.aspnews.com/trends/article.php/3306221>, 7th December 2005
- [82] Wrightly,
<http://www.writely.com/>

- [83] Software Compass – 2001,
“Optimising Code for Faster Processors”, *Developer IQ Magazine, February 5th 2001*,
<http://www.developeriq.com/Magazinestories/02jul3|optimisation.php3>
Accessed: 19th April 2006
- [84] Yonglei Tao – 2002,
“Component Vs. Application-Level MVC Architecture”, *Proceedings of the 32nd ASEE/IEEE
Frontiers in Education Conference 2002, Volume 1, Pages: T2G-7*
ISSN: 0190-5848
- [85] Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra – 2004,
“Head First: Design Patterns”,
ISBN: 0-596-00712-4, *O'Reilly Media Inc*
- [86] Quintin Cutts, Richard Connor, Graham Kirby, Ron Morrison - 1994,
“An Execution Driven Approach to Code Optimisation”, *Proceedings of the 17th Australasian
Computer Science Conference, Christchurch, New Zealand, 1994, Pages: 83-92.*
- [87] Ian Masters – 2004,
“Surviving unplanned downtime...”, *Continuity Central's Business Continuity e-Journal*,
<http://www.continuitycentral.com/feature0115.htm>,
Accessed: 9th May 2006
- [88] Transaction Processing Council,
<http://www.tpc.org>
- [89] Hewlett-Packard – 2003,
“HP and Oracle Set World Record Performance Mark -First to Top 1 Million Transactions per
Minute”, *Hewlett-Packard Press Release*,
<http://www.hp.com/hpinfo/newsroom/press/2003/031105b.html>,
Accessed: 10th May 2006
- [90] Standard Performance Evaluation Corporation,
<http://www.spec.org>
- [91] Enrique Claver, Reyes Gonzáles, José Gascó, Juan Llopis – 2002,
“Information systems outsourcing: reason, reservations and success factors”, *Journal of Logistics
Information Management 2002, Vol 15; Part 4, Pages: 294-308*
ISSN: 0957-6053, *MCB University Press*
- [92] Chris Lonsdale, Andrew Cox – 2000,
“The historical development of outsourcing: the latest fad?”, *Journal of Industrial Management And
Data Systems 2000, Part 8/9, Pages: 444-450*
ISSN: 0263-5577, *MCB University Press*
- [93] Kate Gerwig – 1999,
“Apps on Tap: Outsourcing Hits the Web”, *netWorker Archive, Volume 3 Issue 3, Pages: 13 - 16*
ISSN: 1091-3556, *ACM Press*
- [94] DataElectronics.
<http://www.dataelectronics.ie>
- [95] TeleCity,
<http://www.telecit.com>
- [96] Hosting365,
<http://www.hosting365.com>
- [97] Ketec,
<http://www.ketec.ie>

[98] FirstServ,
<http://www.firstserv.com>

[99] Below Zero,
<http://belowzero.biz>

[100] Verio,
<http://www.verio.com>

[101] PowerTel,
<http://www.powertel.com.au>

[102] PSINet,
<http://www.psinet.fr>

[103] Alan Earnshaw – 2000,
“ASP Success Factors: An Analysis of the Critical Areas Required for Success as an Application Service Provider”, Information Management Systems Whitepaper,
<http://www.infoms.com/wp-asp.html>,
Accessed: 8th March 2005

[104] Adam Braunstein – 1999,
“The State of ASPs”, Robert Francis Group Research,
<http://www.rfgonline.com/research/getnote.html?docid=121699nt>,
Accessed: 11th March 2005

[105] Oracle On Demand
<http://www.oracle.com/ondemand/index.html>

List of Acronyms Used

API	Application Program Interface
ASP	Application Service Provision
ASPs	Application Service Providers
B4R	Billing4Rent
BCP	Business Continuity Plan
BiDi	Bi-Directional
BLOB	Binary Large Object
BOPS	Business Operations Per Second
CompTIA	Computing Technology Industry Association
CRM	Customer Relationship Management
CPU	Central Processing Unit
DBMS	Database Management System
DBCS	Double Byte Character Set
DMZ	DeMilitarised Zone
DNS	Domain Name System
DR	Disaster Recovery
DRP	Disaster Recovery Plan
EJBs	Enterprise Java Beans
ESP	External Service Provider
FMS	Failover Management System
GDC	Geographically Dispersed Clusters
GL	General Ledger
GMIT	Galway-Mayo Institute of Technology
HA	High Availability
IDC	International Data Corporation
I/O	Input / Output
IPSec	IP Security
ISV	Independent Software Vendor
IT	Information Technology
ITAA	Information Technology Association of America
JAMon	Java Application Monitor
JCA	Java Connector Architecture
JDBC	Java Database Connectivity
JMS	Java Message Service
JSP	Java Server Pages
JVM	Java Virtual Machine
LAN	Local-Area Network
NYBT	New York Board of Trade
MVC	Model / View / Controller
MTBF	Mean Time between Failures
MTTR	Mean Time to Repair
NAT	Network Address Translation
NIC	Network Identification Card
OS	Operating System
PC	Personal Computer
PDF	Portable Document Format
PO	Purchase Order

QoS	Quality of Service
RAID	Redundant Array of Independent Disks
(Oracle) RAC	Real Application Clusters
RDBMS	Relational Database Management System
SaaS	Software as a Service
SCSI	Small Computer System Interface
SLA	Service Level Agreement
SME	Small and Medium-sized Enterprise
SPEC	Standard Performance Evaluation Corporation
SPOF	Single Point of Failure
TPC	Transaction Processing Council
TSSG	The Telecommunications Software & Systems Group
UI	User Interface
UTP	Unshielded Twisted Pair
VPN	Virtual Private Network
W3C	World Wide Web Consortium

Appendices



- **Appendix 1: Related Research**
- **Appendix 2: B4R Prototype Screenshots**
- **Appendix 3: B4R Prototype CD-ROM**

Appendix 1 – Related Research

Paper Title:	Perception: The Real Inhibitor to ASP Adoption?
Authors:	Informatics Research Group, GMIT ----- Kenneth Kirrane Sean Duignan
	Sabrina McNeely John Healy
Presented at:	5th Annual Information Technology & Telecommunications (IT&T) Conference, Cork, Ireland. (October 2005).

The below paper was presented at the Industry Track Session 1(Critical Issues in Software 2006 – 2010: Development & Sourcing) at The 5th Annual Information Technology & Telecommunications (IT&T) Conference in Cork on October 26th, 2005. The paper was also published in the conference proceedings (ISSN: 1649 – 1246).

Perception: The Real Inhibitor to ASP Adoption?

Sabrina McNeely¹, Kenneth Kirrane²,
John Healy³, Sean Duignan⁴
Department of Maths & Computing,
Galway-Mayo Institute of Technology,
Galway, Ireland.

Tel: +353 (0) 91 753161

E-mail¹: sabrina.mcneely@gmit.ie

E-mail²: kenneth.kirrane@gmit.ie

E-mail³: john.healy@gmit.ie

E-mail⁴: sean.duignan@gmit.ie

Abstract: Upon its inception, many heralded the ASP paradigm as the death knell of software-as-a-product, and the birth of software-as-a-service. Despite the hype however, uptake is struggling to reach the levels many analysts predicted for the ASP model. This paper identifies and examines the key factors influencing the adoption of ASP, and highlights the inconsistencies in the available literature. We identify several questions that remain unanswered, which may be adversely influencing user perceptions of the model. In order to address these questions, areas of further research are proposed to deconstruct the inhibitors of the ASP paradigm, and ultimately answer the most burning question: *Is perception the primary inhibitor to the uptake of the ASP model?*

1. Introduction

The Application Service Provision (ASP) model has had many guises over the years including software-as-a-service, on-demand computing and utility computing. However, its underlying premise remains unchanged: Application Service Providers (ASPs) offer multiple users a subscription-based access model via the Internet to centrally managed applications [1]. ASPs provide access to software on a one-to-many basis and thus the cost of ownership and maintenance of the solution is shared by several clients. Service level agreements (SLAs) assist in ensuring client expectations are met with regard to the performance of the ASP solution.

Despite the initial hype, user uptake of the ASP model has been slow to materialise. In 2001 the International Data Corporation (IDC) Group forecast that spending on ASPs would grow to \$24 billion by 2005 [2]. By 2002/2003, the ASP market seemed all but dead, with a 90 percent failure rate according to industry analysis [3]. IDC reports that the ASP market had only reached \$5 billion by 2003 falling far short of that which was first envisaged [4]. Current estimates indicate worldwide spending on software as a service and associated software license revenue will reach \$15.2 billion by 2007, much lower than earlier predictions but substantial nonetheless [4] [5]. The above statistics suggest that ASP has been given a new lease of life.

Although many papers are quick to quote statistics and outline the determinants of ASP adoption, few delve into the reasoning behind these determinants, be they positive or negative. The objective of this paper is to identify and examine the key factors influencing the adoption of the ASP model. To accomplish this objective, this paper explores the existing literature in order to formulate a consensus on the reasons pertaining to the uptake or otherwise of the ASP model. Our analysis is divided into three distinct sections. First, we establish the key factors that individually lead to either an affirmative or adverse decision with regard to the uptake of the ASP model. Second, we attempt to expand on the research to date by examining each of these factors and their relevance to the adoption of the ASP as a whole. Finally, we conclude the paper by highlighting the required direction for further research of the ASP model.

2. Identification of the key factors influencing ASP adoption

In depth analysis of the literature highlights economies of scale as the key driver of the ASP paradigm [1], [6], [7], [8], [9], [10]. ASPs exhibit economies of scale as the cost of the solution is distributed among its customers on a one-to-many basis. In addition to economies of scale, Walsh [10] interestingly highlights security and reliability as major benefits of the ASP model, and states that for small or midsize organisations, ASPs can provide greater levels of security and reliability than the customers own organisation [10]. Walsh's point is contrary to the norm, as uncertainties with regard to security and privacy as well as performance concerns in the form of availability, scalability and reliability, are cited as the main inhibitors to the uptake of the ASP model [5], [11], [12]. This section is dedicated to examining each of the above factors in order to assess the benefits or threats they potentially pose to the adoption of the ASP model.

2.1 Economies of scale

In economic terms, economies of scale are achieved when the average cost of producing a product diminishes as each additional product is produced, as the fixed costs are shared over an increasing number of products. The economies of scale model is equally viable with regard to service provisioning. ASPs achieve economies of scale by lowering the average cost of the service through sharing fixed costs among many users. A survey conducted by the Information Technology Association of America (ITAA) in 2002 investigated key user expectations in selecting ASPs. Results of that survey indicate that 39 percent of respondents estimate their return on investment of between 10 and 50 percent, while an additional 14 percent of

respondents placed it between 51 and 100 percent [13]. The cost savings highlighted by the ITAA survey offer some solace to companies burdened by an increased reliance on IT, and its associated costs. ASPs can alleviate this burden, thus allowing a company to focus on other core areas of their enterprise. By contrast, a survey of 250 IT managers conducted by *Informationweek.com* highlights a high degree of scepticism with regard to the claimed cost advantages / economies of scale [14].

2.2 Performance (high availability, scalability and reliability)

Due to ASPs network-centric delivery model, performance considerations in terms of high-availability, scalability and reliability are often cited as the major inhibitors to the widespread uptake of the ASP model [5], [11], [12], [15]. High Availability (HA) requires systems designed to tolerate faults – to detect a fault, report it, mask it, and then continue service while the faulty component is repaired offline [16]. In the majority of cases, availability is expressed as a percentage of system up time, with “five nines” or 99.999% availability a desired level of availability for most ASPs. Scalability refers to the ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement [17]. Systems should not only adapt to their new configurations, they should be able to operate with the same level of efficiency and to the same standard of service. Reliability is defined as the assurance a product will perform its intended function for the required duration within a given environment. Reliability is best described as product performance over time [18]. A reliable system should consistently produce the same results, while meeting, or exceeding customer expectations.

Kern et al (2002) report that 85% of potential ASP customers rate quality of service as being one of the key factors in ASP satisfaction. The majority of potential ASP customers also rate scalability and flexibility as being very important [19]. A large factor in service quality, availability and scalability, for web-hosted applications is the quality and speed of the underlying network in delivering the service offering to its customers. Many factors influence network quality, such as bandwidth limitations, network latency and reliability of the Internet. This is especially true in the ASP paradigm as all the application processing takes place on the application server, with the results returned to geographically dispersed users over the network in a thin-client model. These findings are corroborated by ITAA’s (2001) survey of key user expectations with respect to ASP – over 80% of respondents claimed that guarantees on network reliability were a very important feature of Service Level Agreements (SLAs) between ASP and clients [2].

Availability and performance are probably two of the most important characteristics of an ASP. Consequently, ASPs will generally invest heavily in backup and redundant systems in order to minimise service disruption. Walsh notes that these safeguards go beyond what many small to midsize companies can afford, and are thus seen as a benefit of the ASP model [6]. Tao also suggests that most online service providers do a better job of ensuring 24/7 application availability than customers could [20]. Walsh and Tao’s position is further strengthened by various other references in the literature pertaining to availability, scalability and reliability as benefits of the ASP model [8], [11], [21], [22].

2.3 Security

Several researchers refer to security and privacy of data as one of the primary areas for concern with regard to the realisation of an ASP solution [6], [8], [5], [12], [23]. Fears of compromised security and privacy have prevented many firms from fully investigating and integrating the ASP business model [12]. Although both security and privacy are concerned with guarding the clients sensitive data, they can be distinguished as follows:

- Security is used to refer to protection of the ASP solution and the data exchanged or stored as part of the ASP solution. ASP security can be broken into three distinct considerations: physical security, solution security and security and integrity of client data [24].
- Privacy is exclusively concerned with ensuring the protection and integrity of the client data exchanged or stored as part of the ASP solution from unauthorised access.

Linthicum [23] outlines three possible security issues, which can be used to collaborate the above definition of security. Poor network security may leave the hosted solution open to external intrusion. Second, an unsatisfactory physical security policy may result in an internal attack. Finally, there are concerns around the security firewalls that are placed between the hosted application domains [23].

While the majority of research literature focus on the negative aspect of security, Walsh [10] looks at security from a different perspective concentrating on the security benefits that can be leveraged from an ASP solution. ASPs are responsible for defining and adhering to a security policy, which meets the needs of their clients. Walsh [10] states that often the security and reliability safeguards implemented by ASPs go beyond what many small to midsize companies can afford and thus are a benefit of the ASP model.

3. Analysis of the key factors influencing ASP adoption

Based on an in-depth analysis of the available literature, *Economies of Scale*, *Performance* and *Security* have been identified as the key factors that influence the uptake of the ASP model. The aim of this section is to expand on the research to date by examining each of these factors and to assess their relevance to the success of the ASP model in greater detail.

By operating a one to many business model ASPs can achieve economies of scale in terms of applications, network costs, server technology and implementation expertise [8]. It is argued however that clients who demand a high degree of customisation destroy much of the value that economies of scale provide [25]. This results in the need to pay higher fees for customised solutions. *Do client requirements for customisation need to adversely affect the benefits obtained through economies of scale?* If the ASP offering is based on open standards, then high levels of customisation may not equate to higher costs. For example, many dedicated concert and entertainment venues see economies of scale inherent in outsourcing their ticketing operations, an illustration that customisation can be accommodated within a centralised environment – the ticketing solution provided by “ASPs” can be customised with regards to venue, artist, date, etc, while the underlying service offering remains the same.

Much of the literature suggests that performance considerations (and in particular the issues of high-availability, scalability and reliability) are significant inhibitors to ASP adoption. Notwithstanding that, other researchers suggest that

given the sizeable investments undertaken by ASP solution providers, the ASP model may actually offer enhanced availability, scalability and reliability to the solution adopter. Indeed, many mature and industry proven solutions can be cited that offer support to this notion. Hewlett Packard's flagship version of the popular Unix operating system (HP-UX 11i) is a case in point. Like many other product offerings in the marketplace, this operating system is implemented on systems ranging from workstations and access servers to application servers and data center servers - systems where high availability is of paramount importance. HP-UX 11i scales easily to 64 processors and is designed to allow for future scaling to 256 processors in a single system [26]. Other players in the operating systems marketplace provide similar functionality in their products. Sun Microsystems, for instance, offers Sun Fire E25K Server; a massively scalable, highly available data center server that scales to 72 UltraSPARC IV processors. A key factor in the design of the Sun Fire E25K Server is the ability to consistently deliver high levels of reliability and availability [27]. *Why then is performance perceived as an inhibitor to ASP uptake, when industry proven solutions exist that support high-availability, scalability and reliability?*

Time and time again security and privacy are cited as major drawbacks to the uptake of the ASP model. As previously outlined by Walsh [10], ASPs often have the ability and resources to provide a higher level of security than many small to midsize companies [10]. The above begs the question: *Is trust the key to viewing security as a benefit or a threat to the uptake of the ASP model?* In an attempt to answer this question we examine security and specifically consider physical security, solution security and security and integrity of client data.

Information Technology (IT) organisations have trusted data centers with the security of their solution hardware for decades. Data centers have gained customer trust by implementing strict physical security policies, ensuring access is restricted to authorised personnel through the use of biometric scanners such as fingerprint or IRIS identification, in addition to the use of passwords and armed guard protection of facilities [6]. *Surely ASPs have a vested interest in ensuring the physical security of their hardware?* Authentication, authorization and encryption all fall under the solution security umbrella. Organisations have made significant progress in securing systems through authentication by enforcing the use of strong passwords. Restriction of access (both local and network) is achieved through authorisation policies. Advances in network security have alleviated the fear of transferring sensitive data such as bank and credit card details. HyperText Transfer Protocol over Secure Sockets Layer (HTTPS) ensures the privacy, integrity and consistency of data through the use of the encryption. Secure Sockets Layer (SSL) has been widely implemented and is now the de facto standard for providing secure e-commerce transactions over the web [28]. *Are ASPs not equally dedicated to preventing unauthorised access to machines on their network as their clients?* Although appropriate levels of physical and solution security assist in ensuring security and privacy of data it is also essential that proven encryption techniques are used and redundant hardware is disposed of in an appropriate manner. Software and data that is no longer needed should be uninstalled and erased to ensure that it is not accessible to unauthorized individuals. Yet again by implementing an appropriate security policy ASPs could overcome this perceived problem. *Surely ASPs strive to meet or even exceed customer expectations?*

Economies of scale, Performance and Security have been identified as the key factors that influence the uptake of the ASP model. However our analysis of

these factors suggests than 'perception' may in fact be the factor most relevant to the success of the ASP model. *Is it possible that 'bad press' has influenced perception of the ASP model and ultimately its uptake?*

4. Conclusion

Our analysis of the factors influencing the uptake of the ASP model suggests that the research conducted to date is at best, incomplete, or at worst, vague and ambiguous. Further research is required to fully clarify the relevance of Economies of Scale, Performance and Security, and most importantly *Stakeholder Perceptions* (Users, IT Managers, Software Developers, Industry Analysts and Academics) to the adoption of the ASP model as a whole. *Perceptions* have in the past greatly influenced the emergence or otherwise of new paradigms and / or the rate of adoption of new products. Global adoption of the automobile for instance was initially predicted to be in the low thousands due to the *fact* that not enough people would work as chauffeurs. History tells us that this *fact* was in the end an ill-informed and poorly validated *perception*. Closer to the world of ASP there are similar examples of perceptions influencing critical thinking – consider some of the IT industry's view of the potential market for personal computing some 20 - 25 years ago! We suggest that *perception* is in fact a key inhibitor to the uptake of the ASP model and seek to further this hypothesis. *Highlighting* the issue through this paper is an initial step. Furthering the body of research focusing on ASP adoption is another step in assessing our hypothesis. To begin to achieve this second step then, we propose a survey of IT organisations that specifically explores all the factors pertinent to ASP adoption at a much finer level of granularity. The primary focus of such a survey is to explore "perceptions" in particular and how they influence ASP adoption rates. Interviews should supplement the survey where appropriate to clarify any ambiguities that arise. It is intended to finalise the design of the survey and target a representative population in Ireland in 2005. An analysis of the research methodology and of the survey results will be the subject of a later paper.

5. References

[1] T. Kern et al – 2002,
"Exploring ASP as sourcing strategy: theoretical perspectives, propositions for practice", *Journal of Strategic Information Systems* 11 (2002) 153-177

[2] A. Suasaria et al – 2001,
"Making the Most out of an ASP Relationship", *IEEE Computer Society*
<http://csdl.computer.org/comp/mags/it/2001/06/f6063abs.htm>, 7th March 2005

[3] L. Haber – 2004,
"ASPs Still Alive and Kicking", *ASPnews.com – Trends*
<http://www.aspnews.com/trends/article.php/3306221>, 7th December 2005

[4] A. M. Konary - 2004,
"Presentation to the ITAA", IDC Analyst Presentation to the ITAA - February 12, 2004, <http://www.IDC.com>

- [5] N. Daylami et al – 2005,
“Determinants of Application Service Provider (ASP) Adoption as an Innovation”,
Proceedings of the 38th Hawaii International Conference on System Sciences
(HICSS-38 2005)
- [6] K. R. Walsh – 2003,
“Analysing the Application ASP Concept: Technologies, Economies, and
Strategies”, *Communications of the ACM, August 2003 Vol. 46, No. 8*
- [7] B. Desai and W. L. Currie – 2003,
“Application Service Providers: A model in Evolution”, *The Fifth International
Conference on Electronic Commerce, ICEC 2003*
- [8] T. Kern and J. Kreijger – 2001,
“An Exploration of the Application Service Provision Outsourcing Option”,
Proceedings of the 34th Hawaii International Conference on System Sciences
(HICSS-34 2001)
- [9] B. Vassiliadis et al – 2004,
“Application Service Provision through the Grid: Business models and
Architectures”, *Proceedings of the International Conference on Information
Technology: Coding and Computing (ITCC'04)*
- [10] K. R. Walsh – 2003,
“Analysing the Application ASP Concept: Technologies, Economies, and
Strategies”, *Communications of the ACM, August 2003 Vol. 46, No. 8*
- [11] B. Jaruzelski et al – 2001,
“ASP 101: Understanding the Application Service Provider Model”, *Booz, Allen,
Hamilton Consulting*,
<http://extfile.bah.com/livelink/livelink/61813/?func=doc.Fetch&nodeid=61813>, 11th
March 2005
- [12] B. Desai – 2002,
“Market Entry Strategies of Application Service Providers: Identifying Strategic
Differentiation”, *Proceedings of the 36th Hawaii International Conference on System
Sciences (HICSS-36 2003)*.
- [13] T. Burton – 2002,
“ITAA Survey Shows ASP Customers Achieve Real Benefits from Outsourcing”,
Information Technology Association of America,
<http://www.ita.org/news/pr/PressRelease.cfm?ReleaseID=1017252264>, 8th March
2005
- [14] J. Mateyaschuk - 1999,
“Leave the Apps to us! ASPs offer benefits through economies of scale”,
Informationweek.com, <http://www.informationweek.com/756/asp.htm>, 25th April
2005

- [15] D. Greschler and T. Mangan – 2002,
“Networking lessons in delivering ‘Software as a Service’ – Part 1”, *International Journal of Network Management* 2002; 12: 317 (DOI: 10.1002/nem.446)
- [17] A. B. Bondi – 2000,
“Characteristics of Scalability and Their Impact on Performance”, *Proceedings of the 2nd International Workshop on Software and Performance*
- [16] J. Gray and D. P. Siewiok – 1991,
“High-Availability Computer Systems”, *IEEE Computer archive Volume 24, Issue 9*
- [18] IEEE Reliability Society – 2000,
“Reliability Engineering”, *IEEE Reliability Society*,
http://www.ewh.ieee.org/soc/rs/Reliability_Engineering/index.html, 25th April 2005
- [19] T. Kern et al – 2002,
“Netsourcing: Renting Business Applications and Services Over a Network”, ISBN: 0-13-092355-9, *Pretence Hall Publishing Inc.*
- [20] L. Tao – 2001,
“Shifting Paradigms with the Application Service Provider Model”, *IEEE Computer Society*, <http://csdl.computer.org/comp/mags/co/2001/10/rx032abs.htm>, 08th Dec 2004
- [21] B. Cohen – 2004,
“Smart Advice: Consider Purchasing Real-Time Collaboration Applications through an ASP”, *InformationWeek Advisory Council*,
<http://www.informationweek.com/shared/printableArticleSrc.jhtml?articleID=55800253>, 25th April 2005
- [22] D. Sovie and J. Hanson – 2001,
“Application Service Providers: Where are the real profit zones? ”, *Mercer Management Consulting*,
http://www.mercermc.com/Perspectives/Perspectives_pdfs/hanson_sovie-ASPprofit.pdf, 20th January 2005
- [23] D. Linthicum – 2002,
“To ASP or Not to ASP?”,
<http://www.softwaremag.com/L.cfm?Doc=archive/2000apr/ASPorNot.html>, 27th April 2005
- [24] T. Anderson – 1994,
“Management Guidelines for PC Security”, *ACM SIGICE Bulletin - July 1994 - Volume 20, Issue 1, Pages 7-14*
- [25] P. Bendor-Samuel – 2001,
“Maximizing the Benefits of Economies of Scale”, *Outsourcing Journal Insights*, Outsourcing Journal May 2001

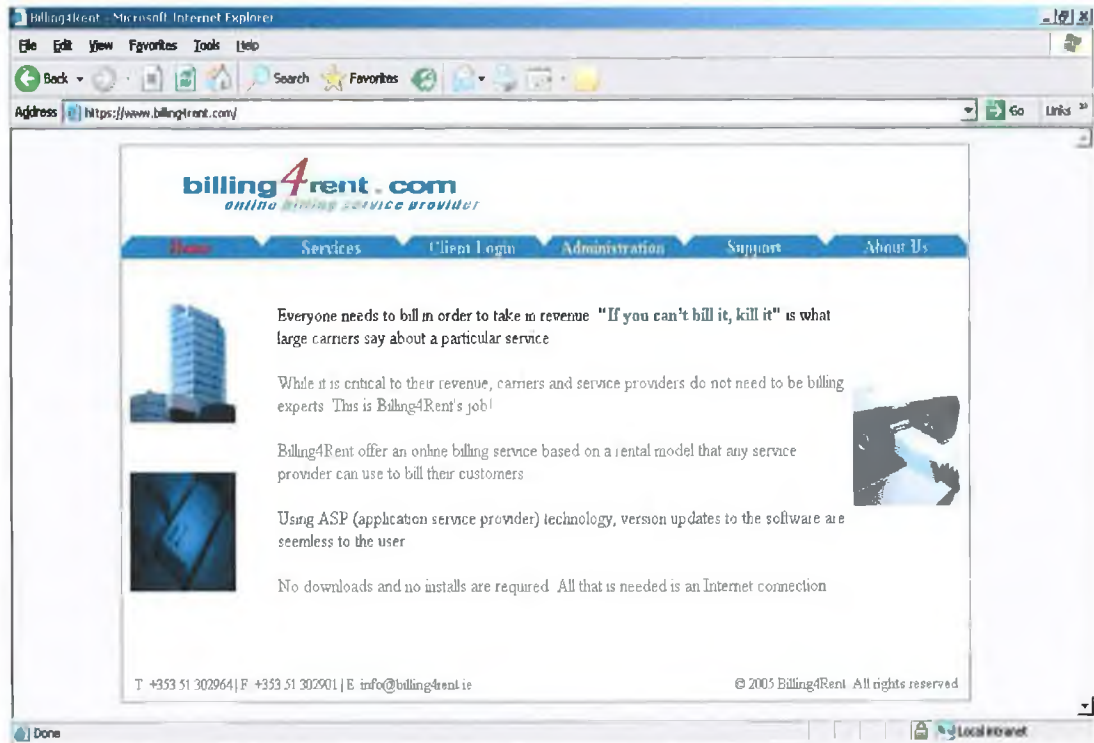
[26] Hewlett-Packard – 2005,
“HP-UX 11i, the Proven Foundation for the Adaptive Enterprise”, *Hewlett-Packard*,
<http://www.hp.com/products1/unix/operating/index.html>, 2nd May 2005

[27] Sun Microsystems – 2005,
“HP-UX 11i, the Proven Foundation for the Adaptive Enterprise”, *Sun High End Servers*, <http://www.sun.com/servers/highend>, 2nd May 2005

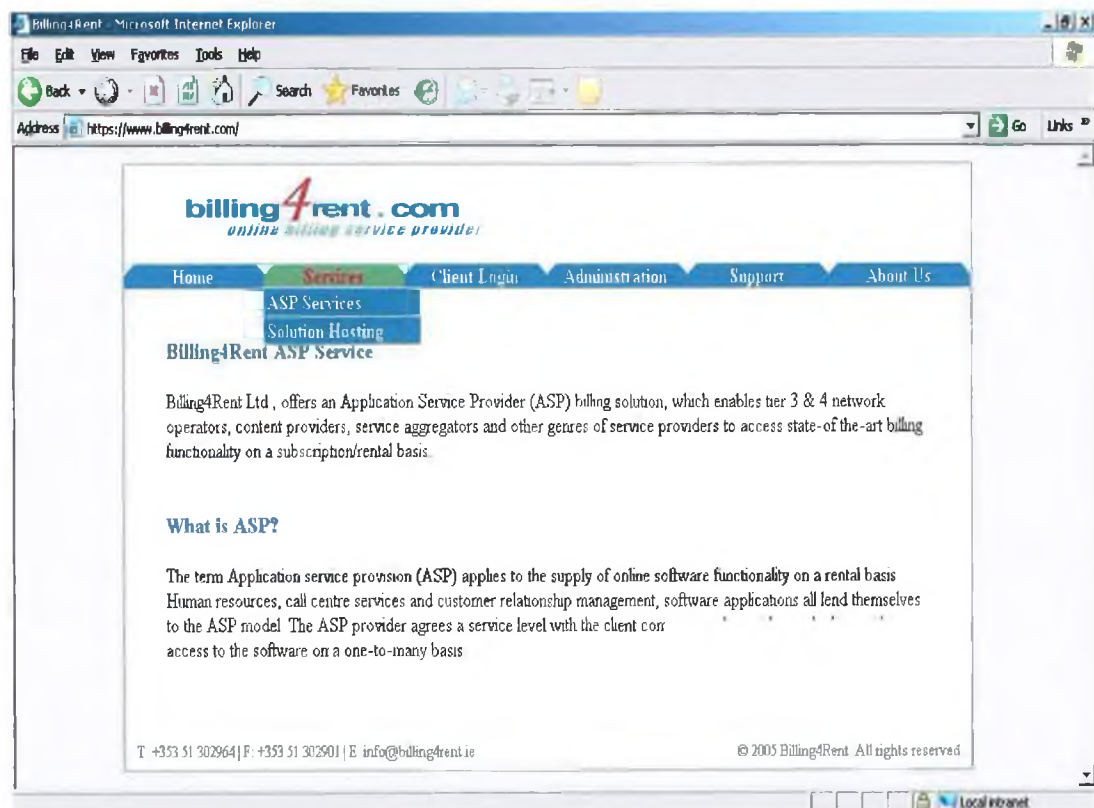
[28] W. Chou
“Inside SSL: The Secure Sockets Layer Protocol”, *IT Professional* – *July/August 2002 Vol. 4, No. 4 Pages 47-52*

Appendix 2 – B4R Prototype Screenshots

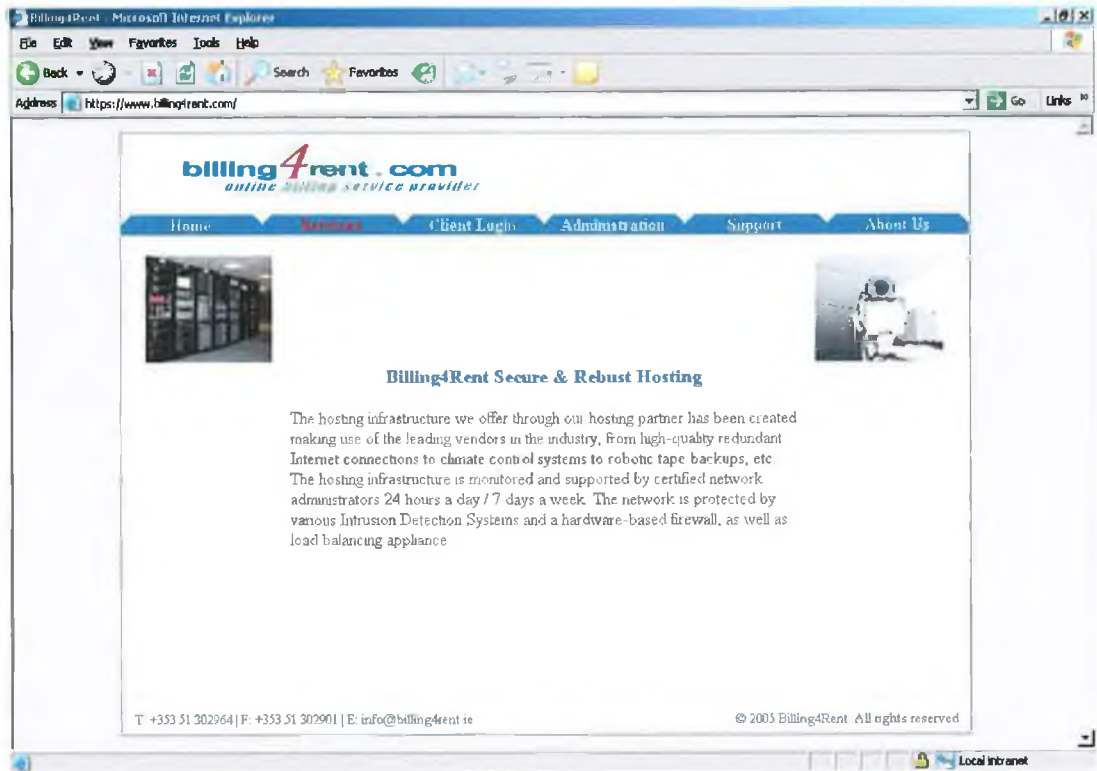
1) Client Site: Home Page



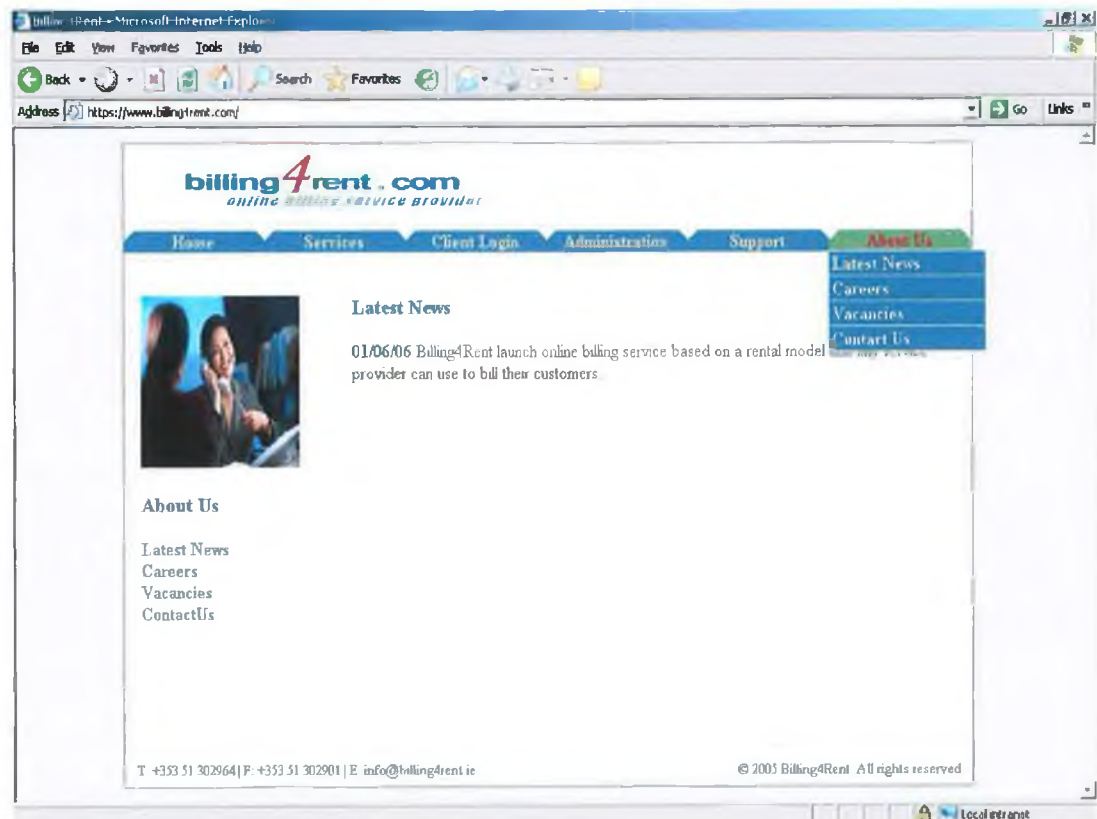
2) Client Site: Services



3) Client Site: B4R Secure & Robust Hosting



4) Client Site: Latest News



5) Client Site: Careers

The screenshot shows the Billing4Rent website in Microsoft Internet Explorer. The browser's address bar displays <https://www.billing4rent.com/>. The website's logo is "billing4rent.com online billing service provider". The navigation menu includes Home, Services, Client Login, Administration, Support, and About Us. The main content area is titled "Careers" and features a photograph of three people working at computers. The text describes the company as a busy energetic organization offering challenging roles. It lists the following positions: Account Managers, Software Engineers, Systems Architects, Web Developers, Quality Assurance Experts, and Helpdesk Administrators. Contact information for sending a CV is provided as careers@billing4rent.ie. The footer contains contact details: T +353 51 302964 | F +353 51 302901 | E info@billing4rent.ie and the copyright notice © 2005 Billing4Rent All rights reserved.

6) Client Site: Current Vacancies

The screenshot shows the Billing4Rent website in Microsoft Internet Explorer. The browser's address bar displays <https://www.billing4rent.com/>. The website's logo is "billing4rent.com online billing service provider". The navigation menu includes Home, Services, Client Login, Administration, Support, and About Us. The main content area is titled "Current Vacancies" and features a photograph of two people working at a computer. Below the photo is a table with the following headers: Description, Location, Salary, and Closing Date. The text below the table states "There are currently no vacancies." The footer contains contact details: T +353 51 302964 | F +353 51 302901 | E info@billing4rent.ie and the copyright notice © 2005 Billing4Rent All rights reserved.

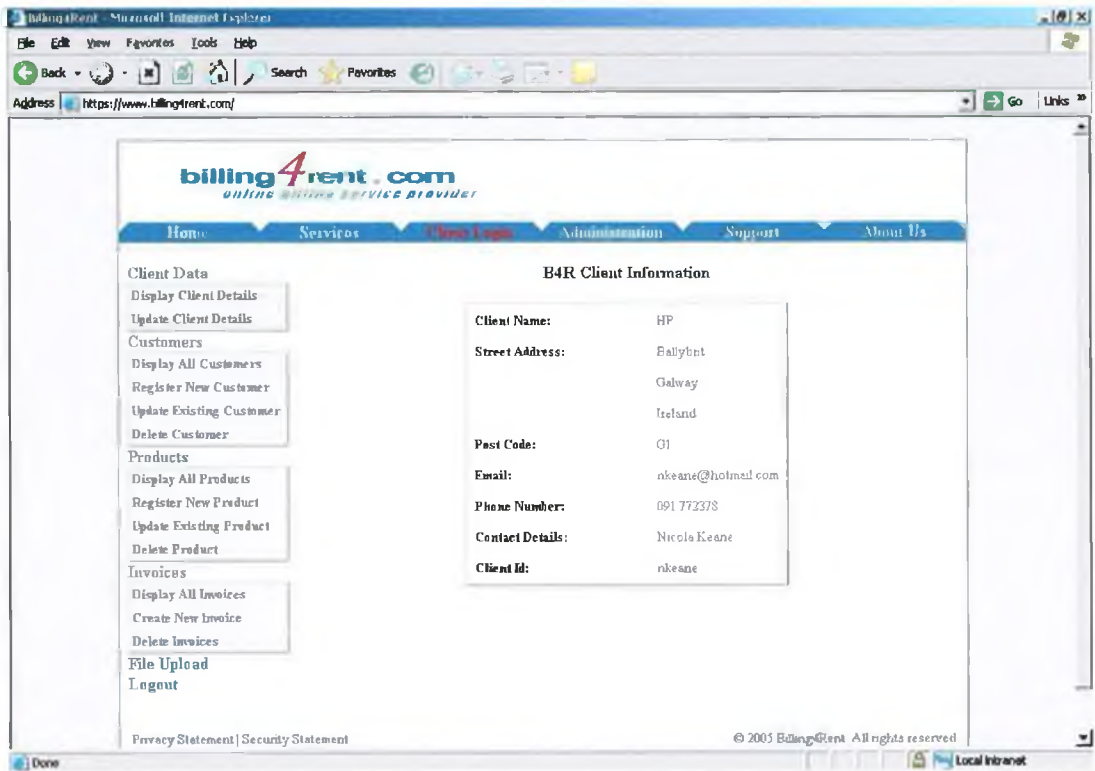
7) Client Site: Contact Us

The screenshot shows the 'Contact Us' page of the Billing4Rent website. The browser's address bar displays 'https://www.billing4rent.com/'. The website header includes the logo 'billing4rent.com online billing service provider' and a navigation menu with links for Home, Services, Client Login, Administration, Support, and About Us. The main content area features a 'Contact Us' section with a photograph of two people. To the right of the photo, contact details are listed: Address (Billing4Rent, TSSG, WIT, Cork Road, Waterford, Ireland), Telephone (+353 51 302964), Fax (+353 51 302901), and Email (info@billing4rent.ie). A sidebar on the left contains an 'About Us' section and a 'Latest News' menu with links for Careers, Vacancies, and Contact Us. The footer includes contact information (T +353 51 302964 | F: +353 51 302901 | E: info@billing4rent.ie) and a copyright notice (© 2005 Billing4Rent. All rights reserved).

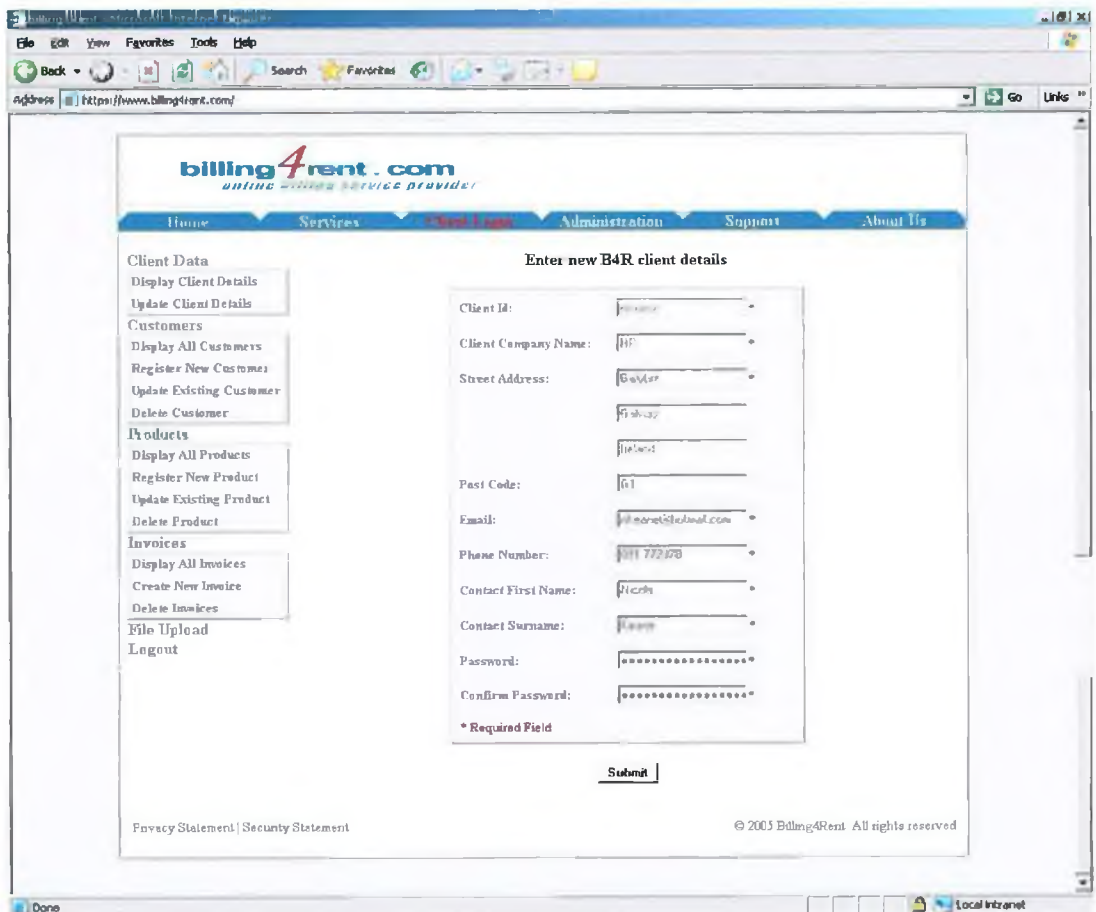
8) Client Site: Client Logon

The screenshot shows the 'Client Logon' page of the Billing4Rent website. The browser's address bar displays 'https://www.billing4rent.com/'. The website header is identical to the previous page, but the 'Client Login' link in the navigation menu is highlighted in red. The main content area features a 'Billing4Rent Client Login' section with two input fields for 'Client Id:' and 'Password:', and a 'login' button. Below the login fields, a message states: 'New Users: Please contact Billing4Rent for details on how to sign up and receive your client id and password'. The footer includes a copyright notice (© 2005 Billing4Rent. All rights reserved).

9) Client Site: Display Client Details



10) Client Site: Update Client Details



11) Client Site: Display All Customers

billing4rent.com
online billing service provider

Home Services **Client Login** Administration Support About Us

Client Data
[Display Client Details](#)
[Update Client Details](#)

Customers
[Display All Customers](#)
[Register New Customer](#)
[Update Existing Customer](#)
[Delete Customer](#)

Products
[Display All Products](#)
[Register New Product](#)
[Update Existing Product](#)
[Delete Product](#)

Invoices
[Display All Invoices](#)
[Create New Invoice](#)
[Delete Invoices](#)

[File Upload](#)
[Logout](#)

Privacy Statement | Security Statement

© 2005 Billing4Rent All rights reserved

Done Local intranet

B4R Customer Information

Customer Id	Company Details	Contact	Profile
Imagine	Imagine Broadband	Jane Bloggs	enabled
Smart	Smart Telecom	Joe Bloggs	enabled

12) Client Site: Customer Details

billing4rent.com
online billing service provider

Home Services **Client Login** Administration Support About Us

Client Data
[Display Client Details](#)
[Update Client Details](#)

Customers
[Display All Customers](#)
[Register New Customer](#)
[Update Existing Customer](#)
[Delete Customer](#)

Products
[Display All Products](#)
[Register New Product](#)
[Update Existing Product](#)
[Delete Product](#)

Invoices
[Display All Invoices](#)
[Create New Invoice](#)
[Delete Invoices](#)

[File Upload](#)
[Logout](#)

Privacy Statement | Security Statement

© 2005 Billing4Rent All rights reserved

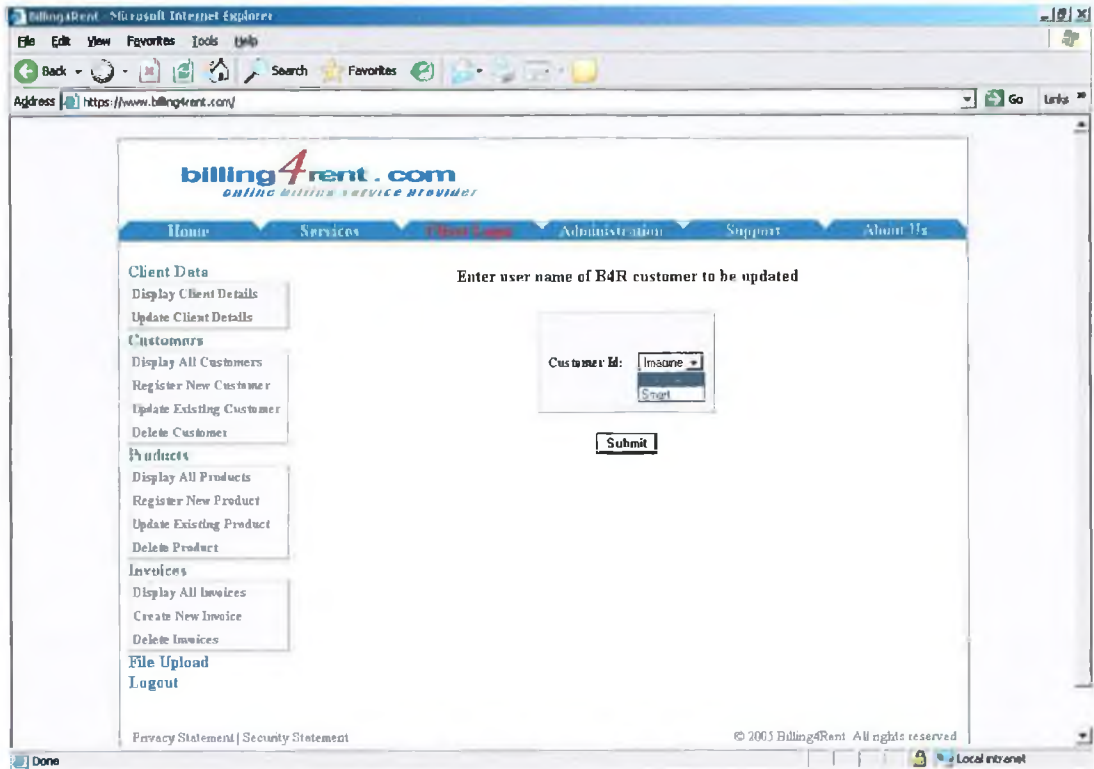
Done Local intranet

B4R company details

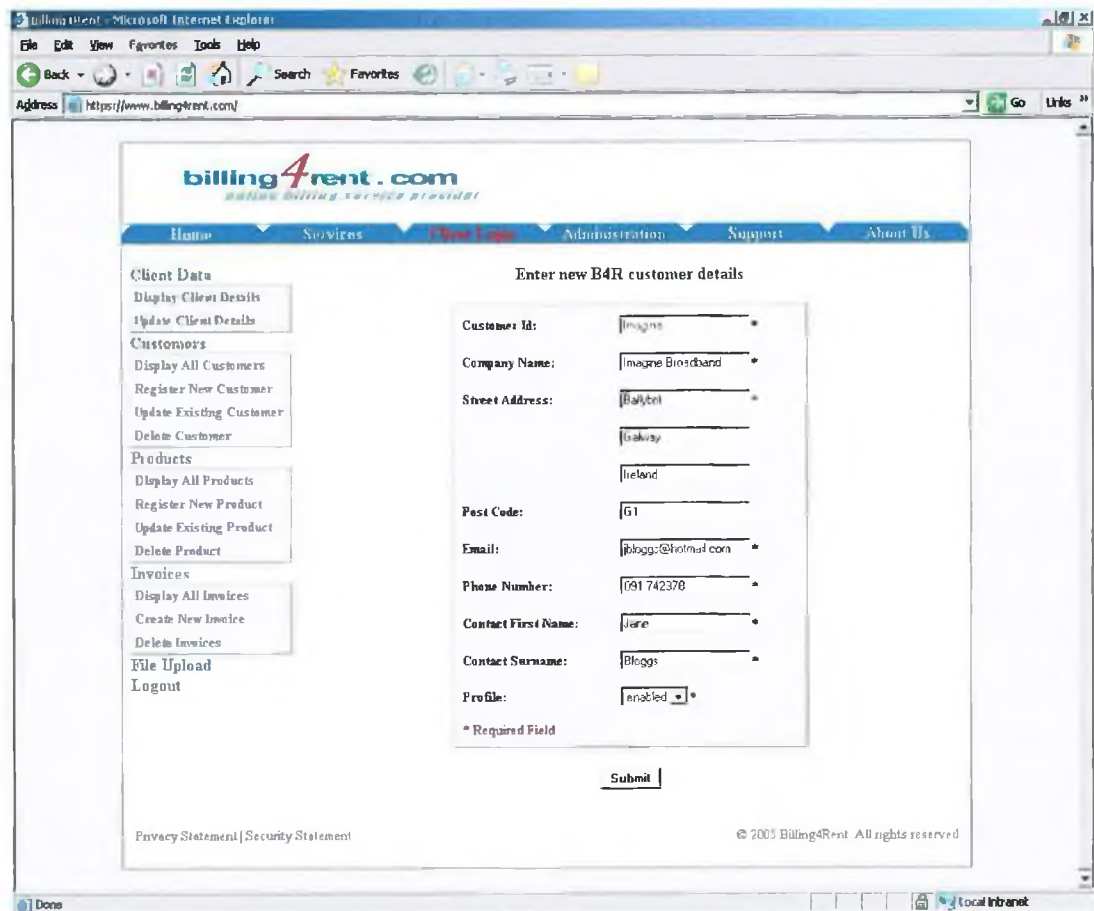
Company Name: Imagine Broadband
 Street Address: Ballybont
 Galway
 Ireland
 Post Code: G1
 Email: jbloggs@hotmail.com
 Phone Number: 091 742378

[Back](#)

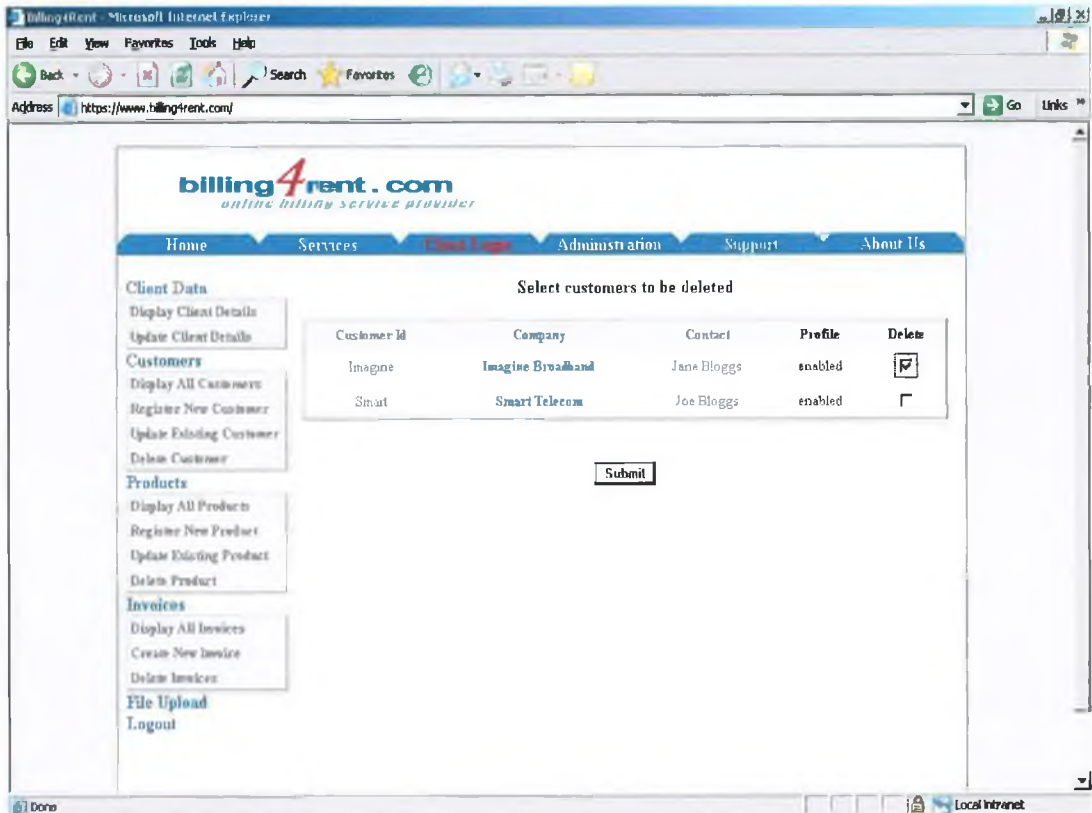
13) Client Site: Update Customer Details – Select Customer to update



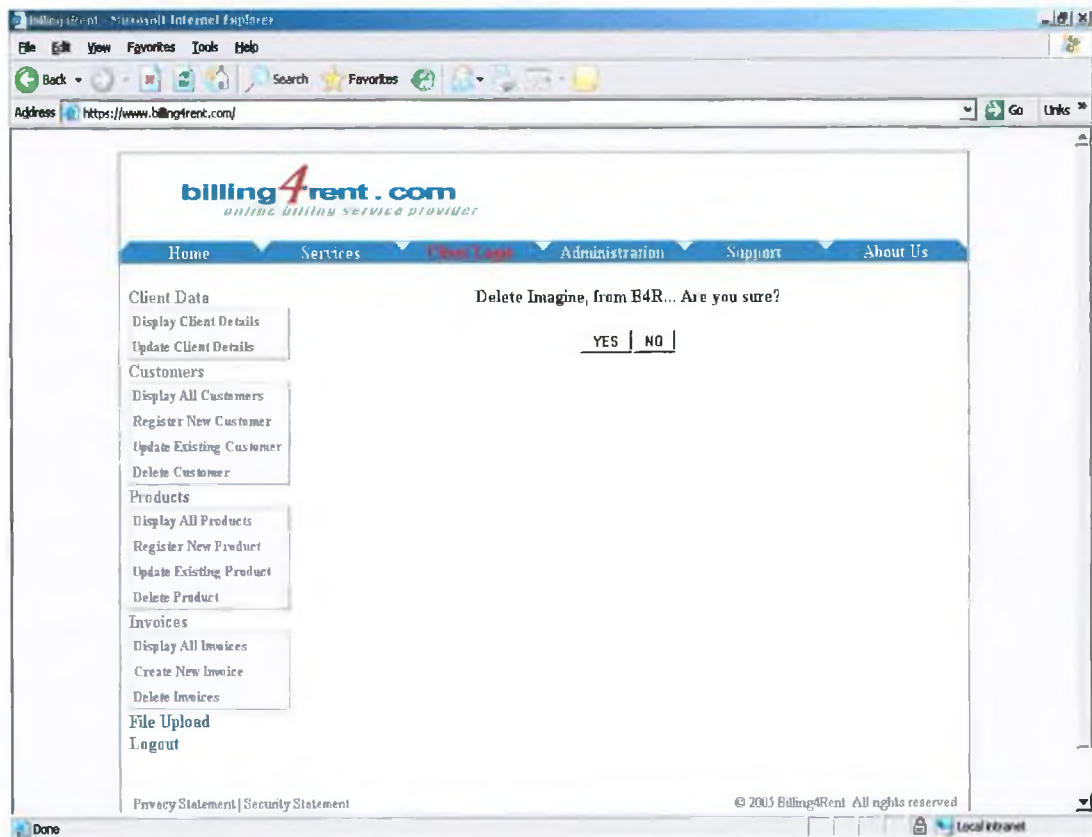
14) Client Site: Update Customer Details



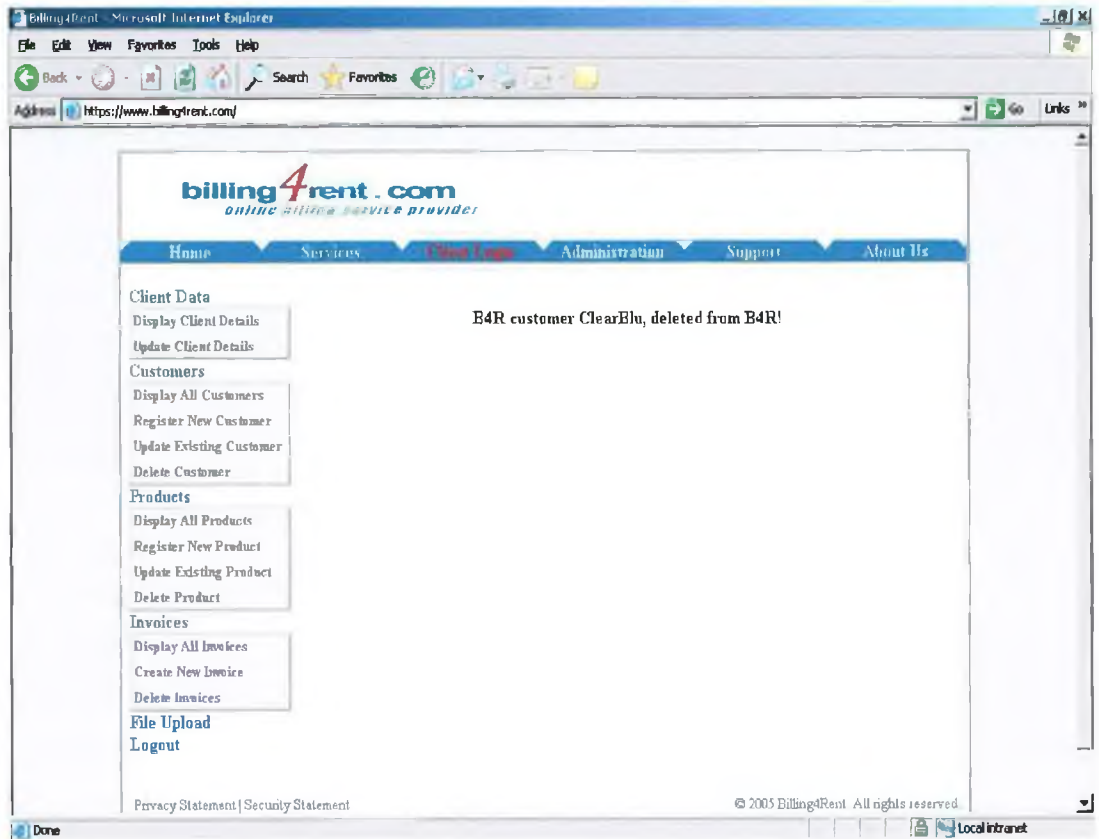
15) Client Site: Delete Customers



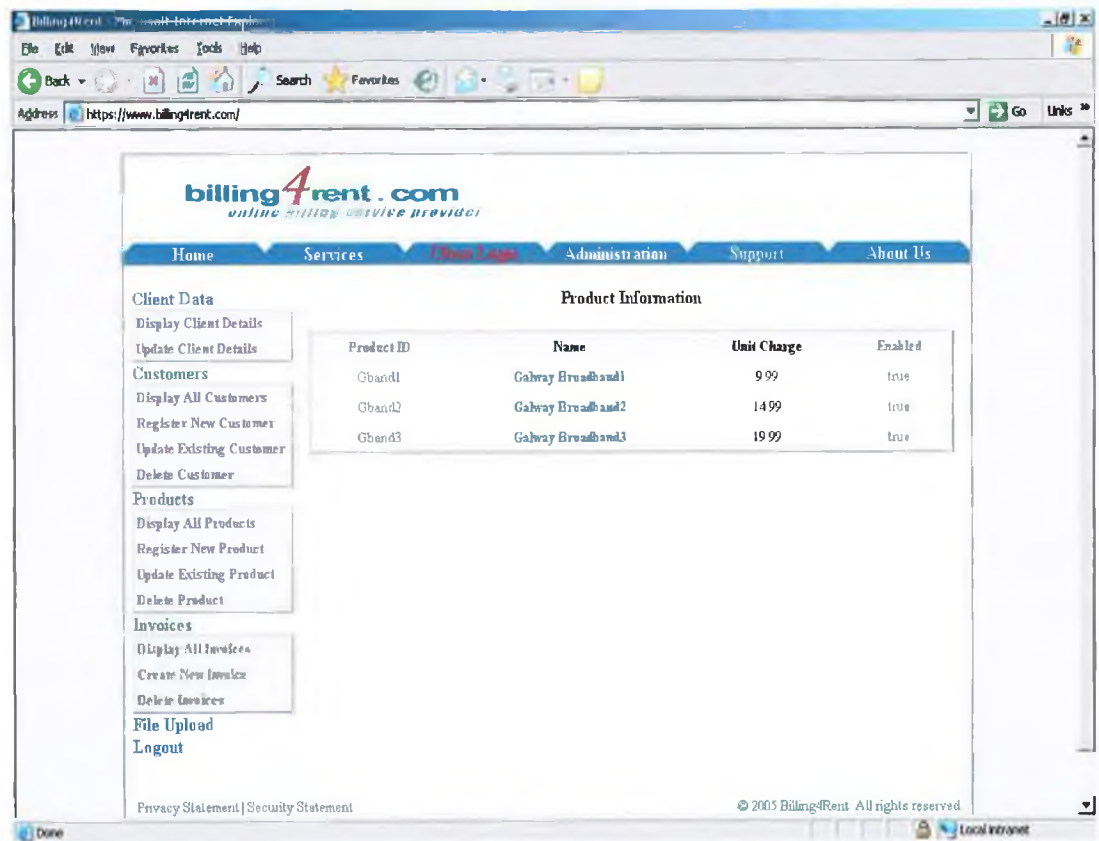
16) Client Site: Delete Customers Confirmation



17) Client Site: Confirmation of Deleted Customers



18) Client Site: Display All Products



19) Client Site: Product Details

The screenshot shows a Microsoft Internet Explorer browser window displaying the Billing4Rent website. The address bar shows <https://www.billing4rent.com/>. The website header includes the logo "billing4rent.com" and the tagline "online billing service provider". A navigation menu contains links for Home, Services, Client Login, Administration, Support, and About Us. On the left side, there are several menu categories: Client Data, Customers, Products, Invoices, and File Upload, each with a list of sub-links. The main content area is titled "B4R Product Details" and contains a table with the following information:

Product Name:	Galway Broadband1
Description:	1MB Broadband for Galway
Invoice String:	1MB Gband
Tax %:	10.00 %
G.L. Code:	3000

Below the table is a "Back" button. At the bottom of the page, there are links for "Privacy Statement" and "Security Statement", and a copyright notice: "© 2005 Billing4Rent. All rights reserved." The browser's status bar shows "Done" and "Local intranet".

20) Client Site: Update Product – Select Product to update

The screenshot shows the same Microsoft Internet Explorer browser window displaying the Billing4Rent website. The address bar shows <https://www.billing4rent.com/>. The website header and navigation menu are identical to the previous screenshot. The main content area is titled "Select the B4R product to be updated:" and contains a form with a "Product Name:" label and a dropdown menu. The dropdown menu is open, showing the following options: "Galway Broadband1", "Galway Broadband2", "Galway Broadband3", and "Galway Broadband4". Below the dropdown menu is a "Submit" button. At the bottom of the page, there are links for "Privacy Statement" and "Security Statement", and a copyright notice: "© 2005 Billing4Rent. All rights reserved." The browser's status bar shows "Done" and "Local intranet".

21) Client Site: Update Product Details

The screenshot shows the Billing4Rent website interface in Microsoft Internet Explorer. The browser address bar displays <https://www.billing4rent.com/>. The website header includes the logo "billing4rent.com" and the tagline "online billing service provider". A navigation menu contains links for Home, Services, **Client Tools**, Administration, Support, and About Us.

The main content area is titled "Enter updated Product details". On the left, there is a sidebar menu with categories: Client Data, Customers, Products, Invoices, File Upload, and Logout. The "Products" category is expanded, showing options like "Display All Products", "Register New Product", "Update Existing Product", and "Delete Product".

The main form contains the following fields:

- Product ID:
- Product Name:
- Description:
- Invoice String:
- Unit Charge:
- Tax %:
- G.L. Code:
- Profile:

A "Submit" button is located at the bottom right of the form. A note at the bottom of the form indicates "* Required Field".

At the bottom of the page, there are links for "Privacy Statement" and "Security Statement", and a copyright notice: "© 2005 Billing4Rent All rights reserved".

22) Client Site: Delete Products

The screenshot shows the Billing4Rent website interface in Microsoft Internet Explorer. The browser address bar displays <https://www.billing4rent.com/>. The website header and navigation menu are identical to the previous screenshot.

The main content area is titled "Select a Product to Delete". The sidebar menu is the same as in the previous screenshot.

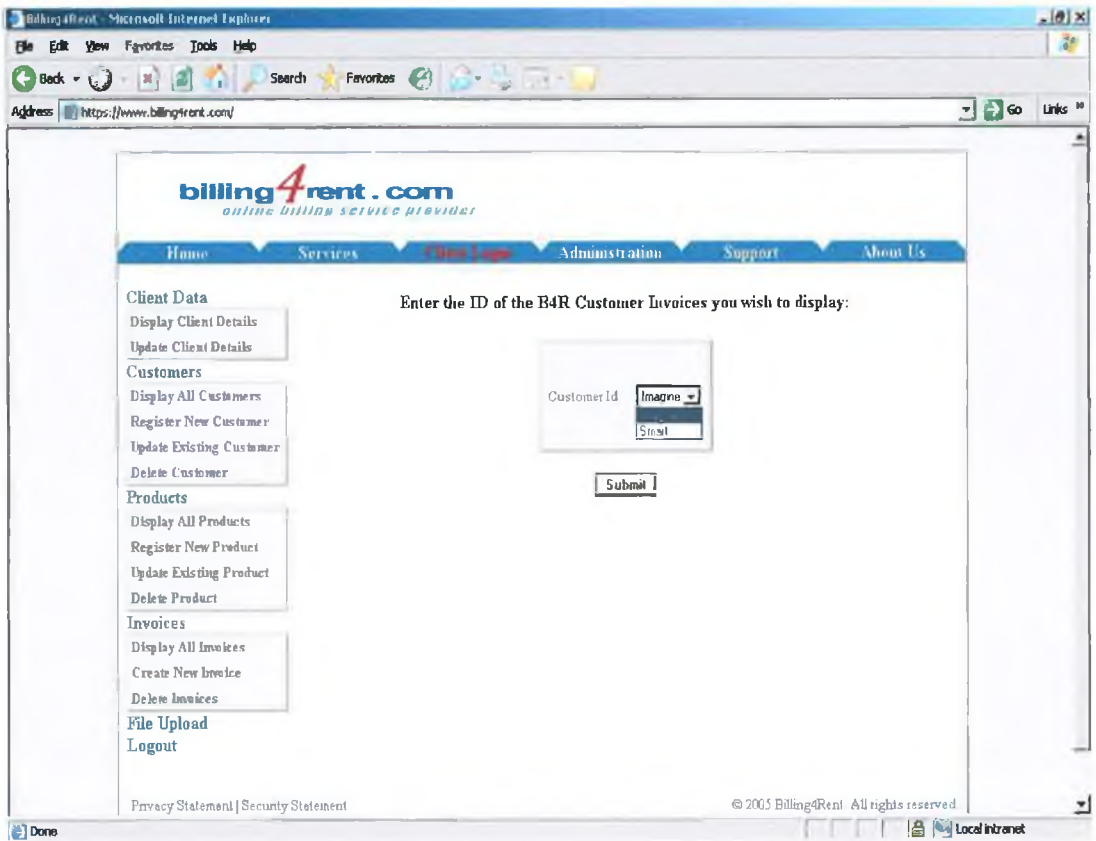
The main form displays a table of products:

Product ID	Name	Unit Charge	Enabled	Delete
Gband1	Galaxy Breadband1	9.99	true	<input type="checkbox"/>
Gband2	Galaxy Breadband2	14.99	true	<input type="checkbox"/>
Gband3	Galaxy Breadband3	19.99	true	<input type="checkbox"/>

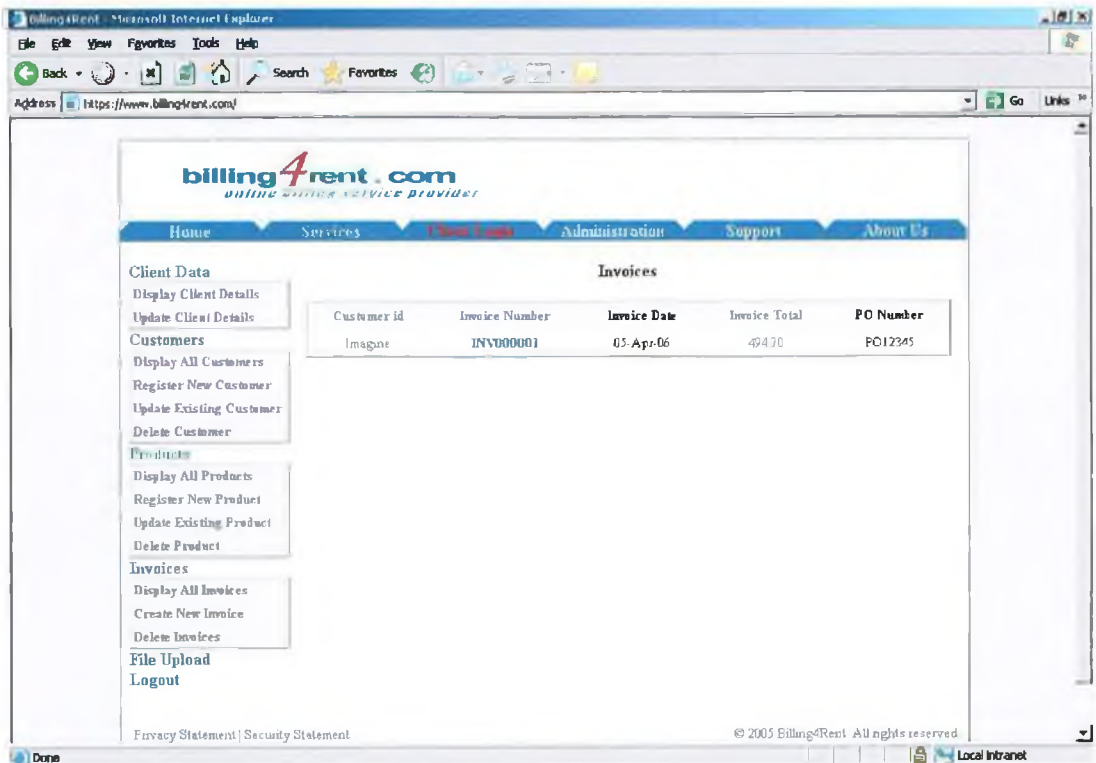
A "Submit" button is located below the table.

At the bottom of the page, there are links for "Privacy Statement" and "Security Statement", and a copyright notice: "© 2005 Billing4Rent All rights reserved".

23) Client Site: Display invoices – Select Customer



24) Client Site: Display invoice Summary



25) Client Site: Services

The screenshot shows the Billing4Rent website interface. The main content area displays an invoice with the following details:

INVOICE

Name: 107
 Address: Dalyton
 City: Chelms
 Postal: 107

Customer Name: Images Data-Rent
 Customer Address: Dalyton
 City: Chelms
 Postal: 107

Invoice ID: 15700001
 Invoice Date: 05-Apr-06
 PO Number: 1012345

Product ID	Product Name	Description	Quantity	Cost	
010001	Chelms Data-Rent	1MB Data-Rent for Chelms	10	99.90	
010002	Chelms Data-Rent	1MB Data-Rent for Chelms	10	149.90	
010003	Chelms Data-Rent	1MB Data-Rent for Chelms	10	199.90	
				Subtotal:	449.70
				Tax:	45.00
				Total:	494.70

Navigation menu: Home, Administration, Support, About Us

Left sidebar menu: Client Data, Customers, Products, Invoices, File Upload, Logout

Footer: Privacy Statement | Security Statement | © 2003 Billing4Rent. All rights reserved.

26) Client Site: Generate invoice – Select Customer

The screenshot shows the Billing4Rent website interface. The main content area displays the 'Generate Invoice: Select Customer' form:

Generate Invoice: Select Customer

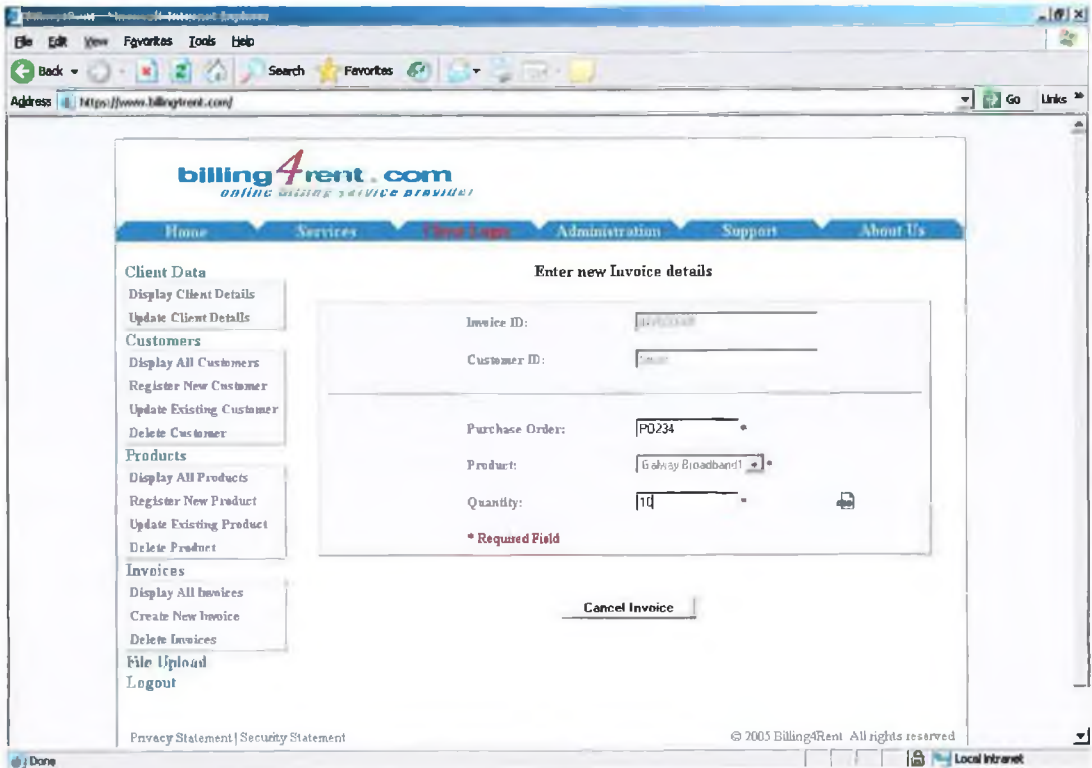
Customer ID:

Navigation menu: Home, Services, Client Tools, Administration, Support, About Us

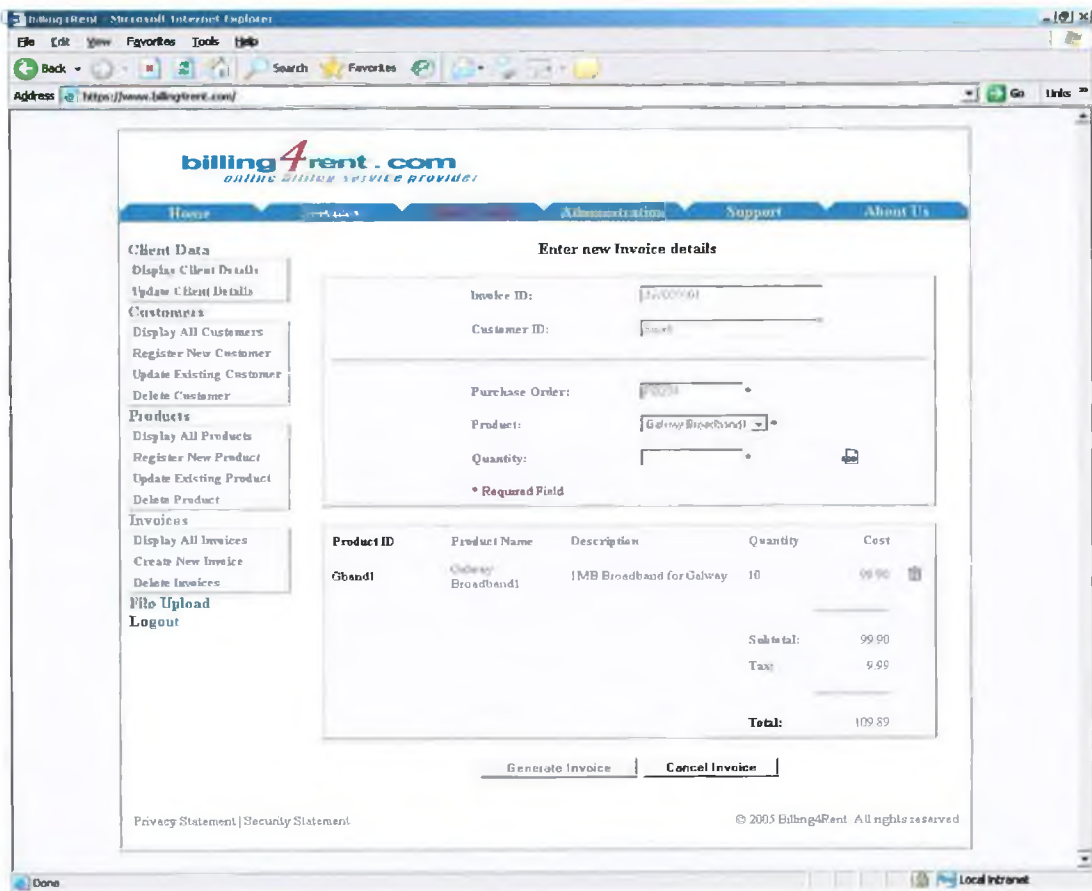
Left sidebar menu: Client Data, Customers, Products, Invoices, File Upload, Logout

Footer: Privacy Statement | Security Statement | © 2003 Billing4Rent. All rights reserved.

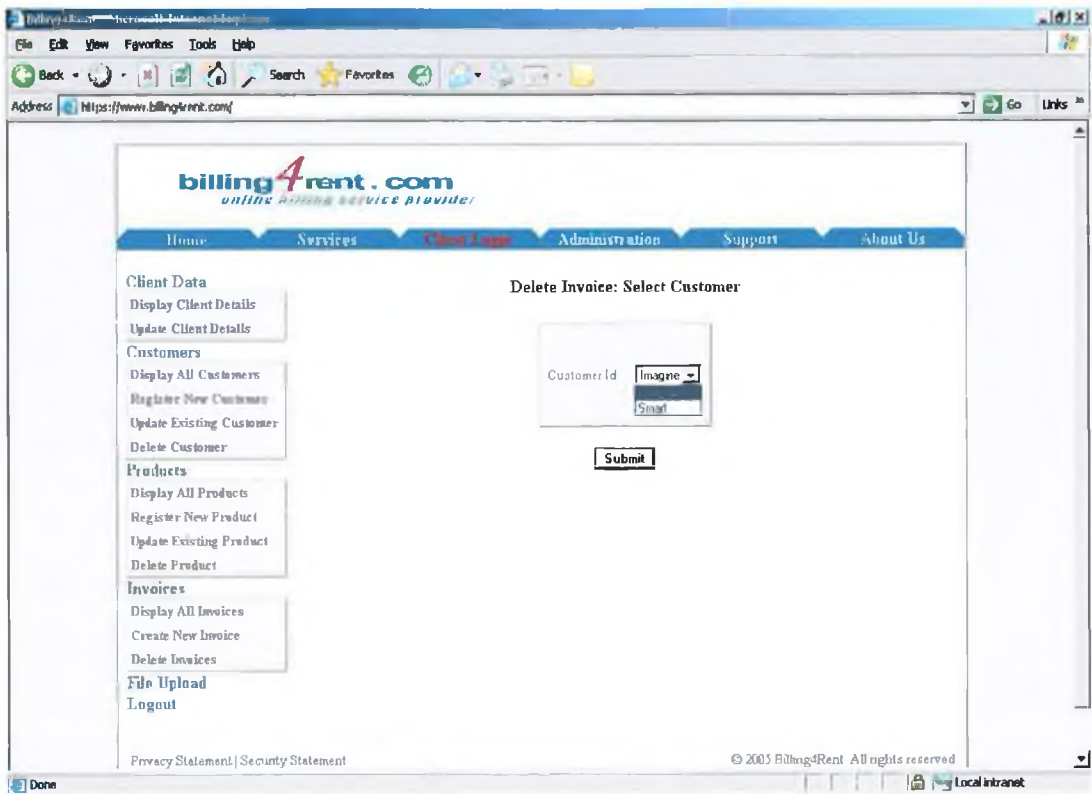
27) Client Site: New invoice – Enter Details



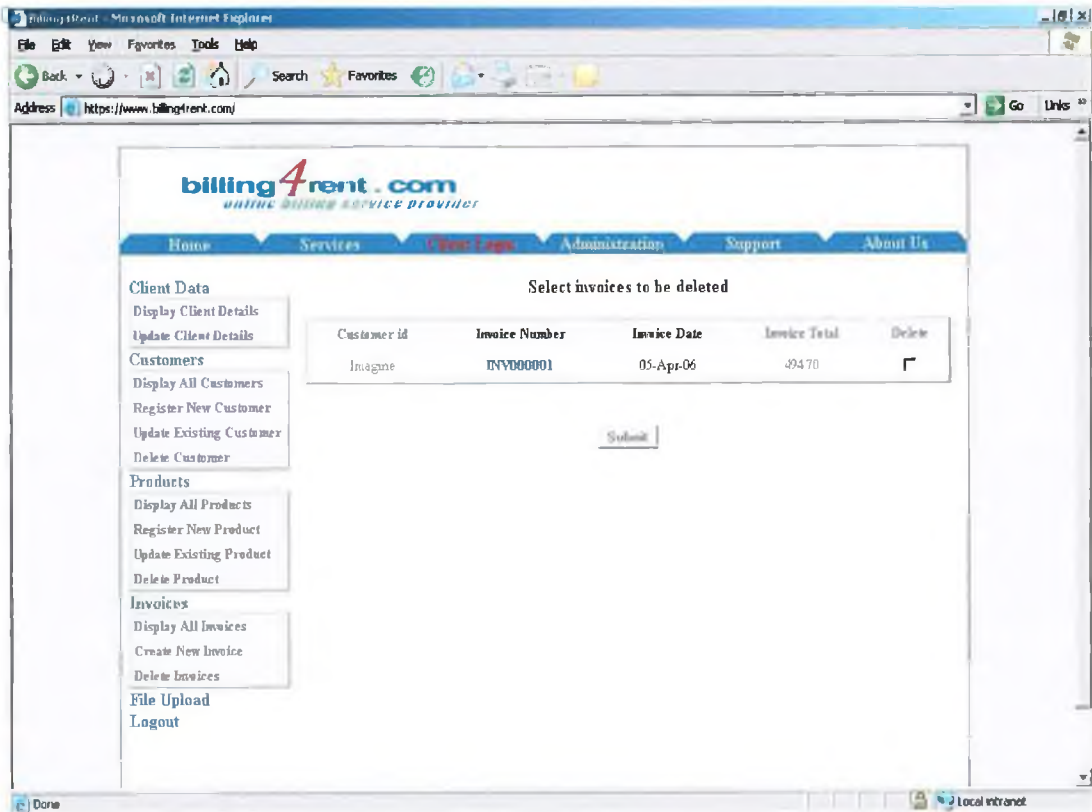
28) Client Site: New invoice – Add Products to invoice



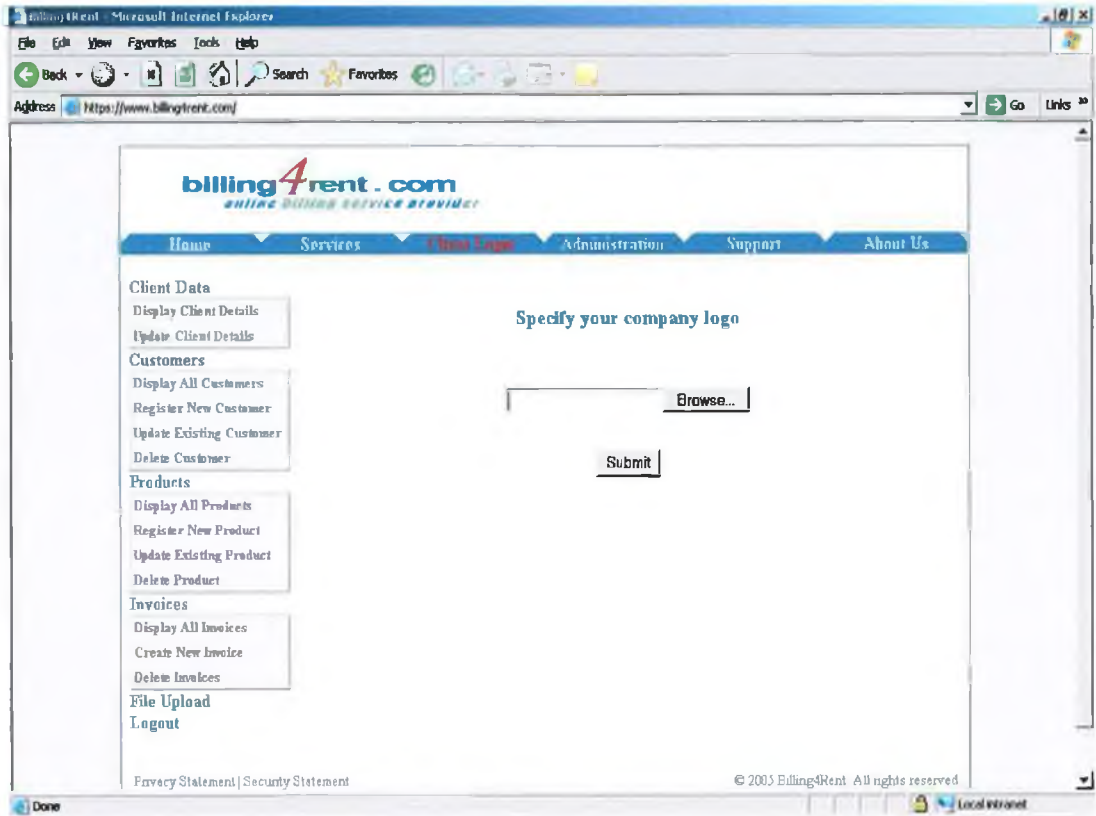
29) Client Site: Delete invoice – Select Customer



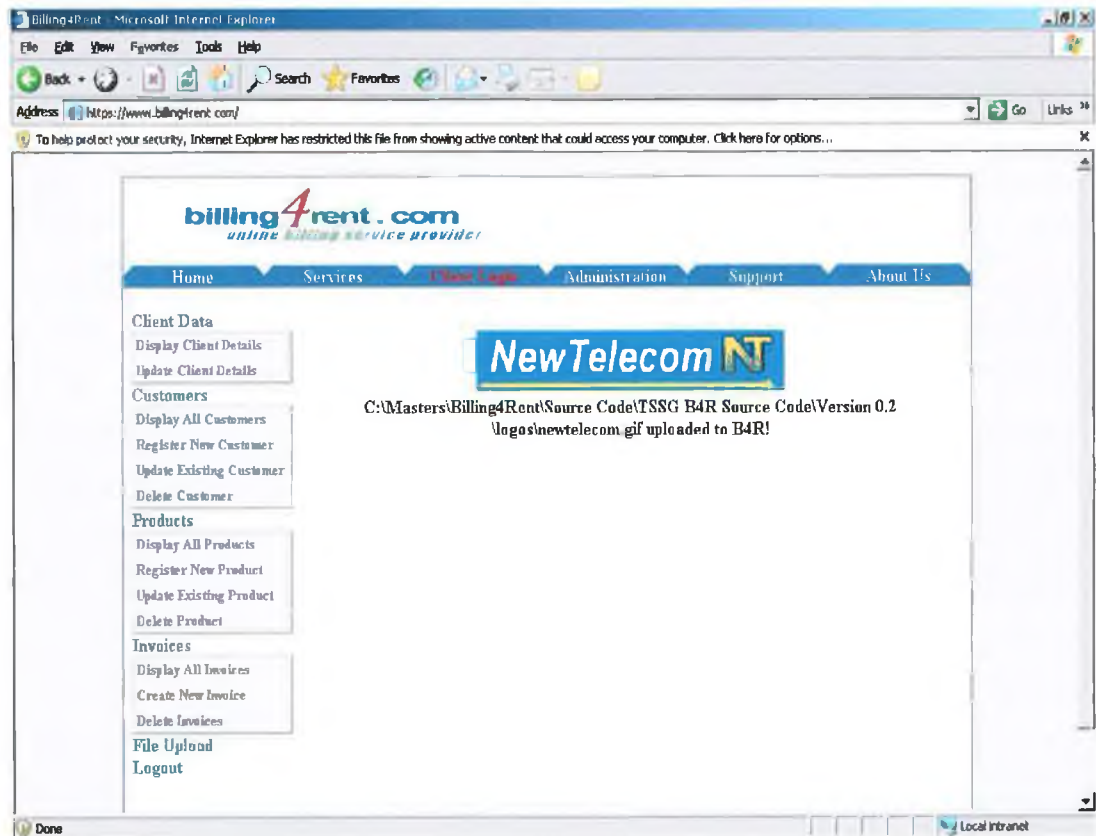
30) Client Site: Delete invoice



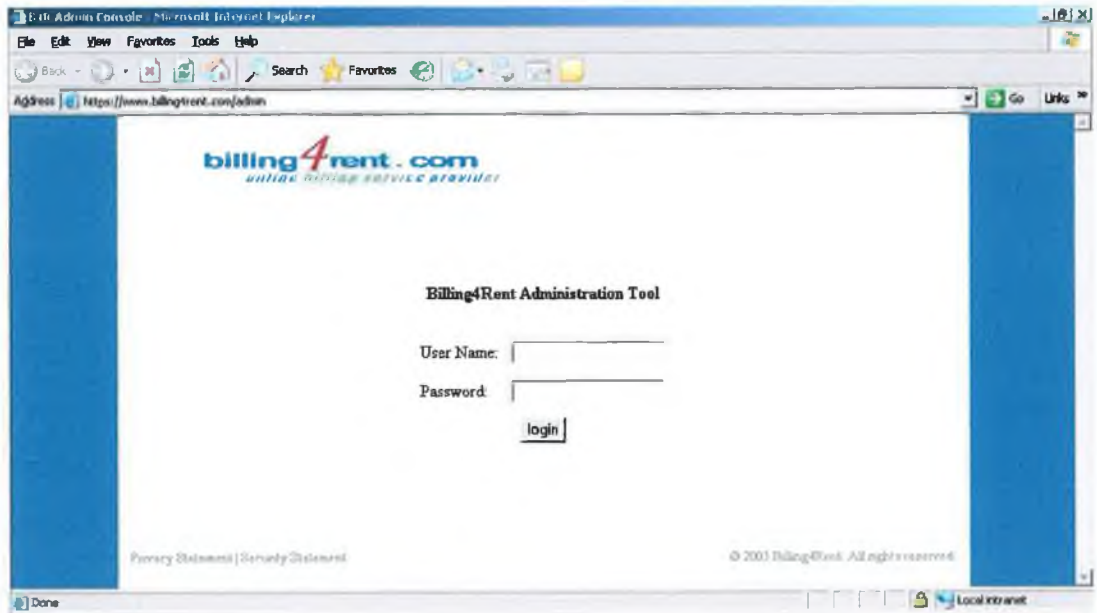
31) Client Site: File Upload – Specify location of Company Logo



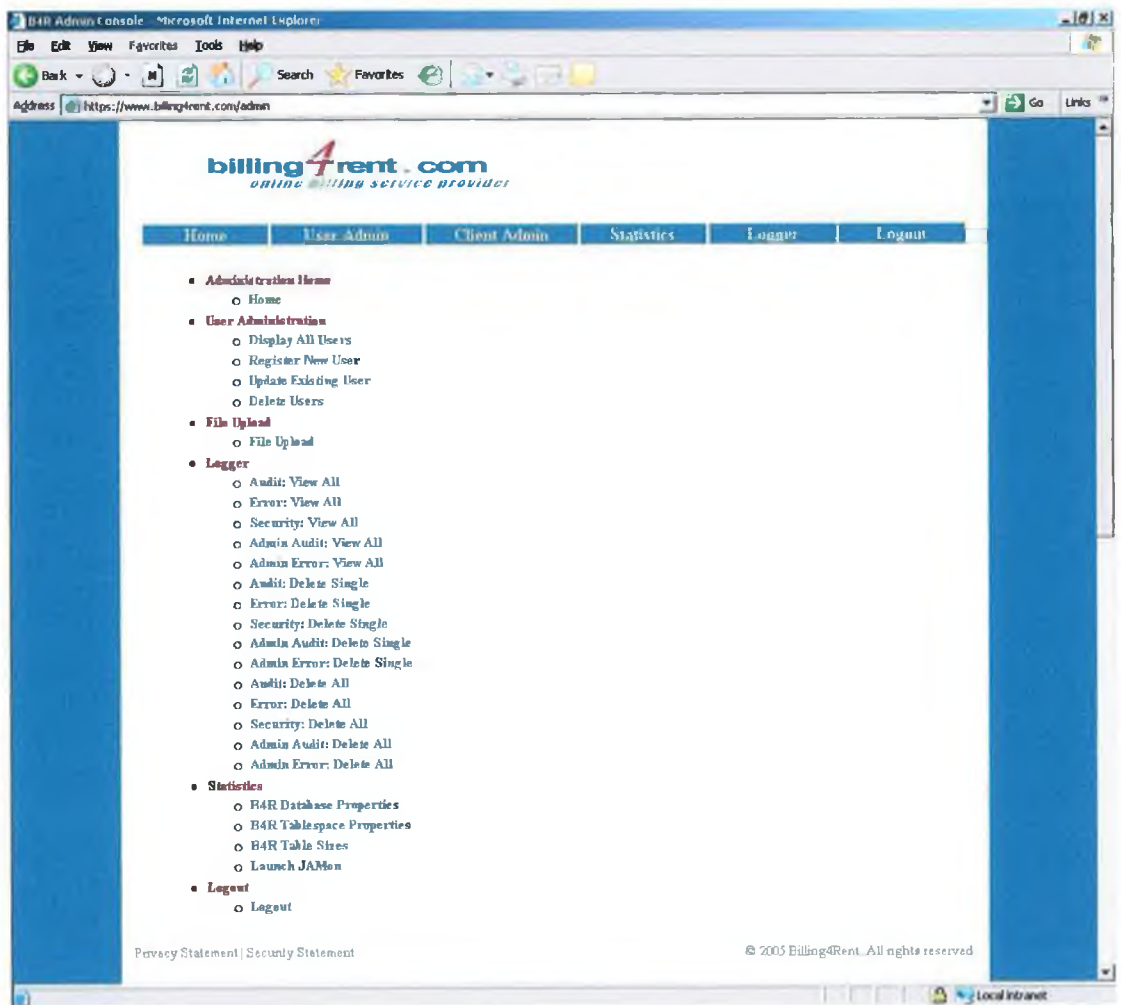
32) Client Site: File Upload – Display Uploaded Logo



33) Administrative Interface: Login



34) Administrative Interface: Home Page



35) Administrative Interface: Display All Users

billing4rent.com
enabling billing account activities

Home User Admin Client Admin Statistics Logger Logout

B4R User Information

Name	Username	Role	Credentials	Profile
Kenneth Kirrane	kkirrane	user	test	enabled
Sabine McNeely	smcneely	admin	test	enabled

Privacy Statement | Security Statement

© 2005 Billing4Rent. All rights reserved.

36) Administrative Interface: Register New User

billing4rent.com
enabling billing account activities

Home User Admin Client Admin Statistics Logger Logout

Enter new B4R user details

First Name:

Surname:

User Name:

Password:

Confirm Password:

Role:

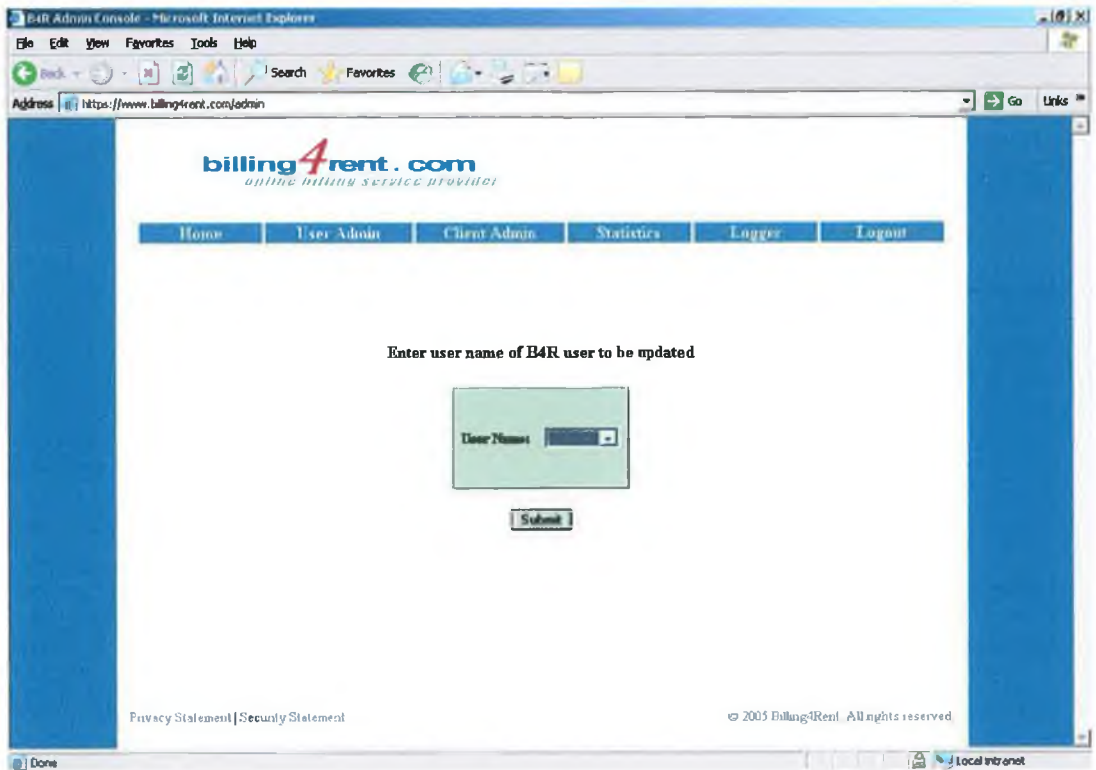
Credentials:

Profile:

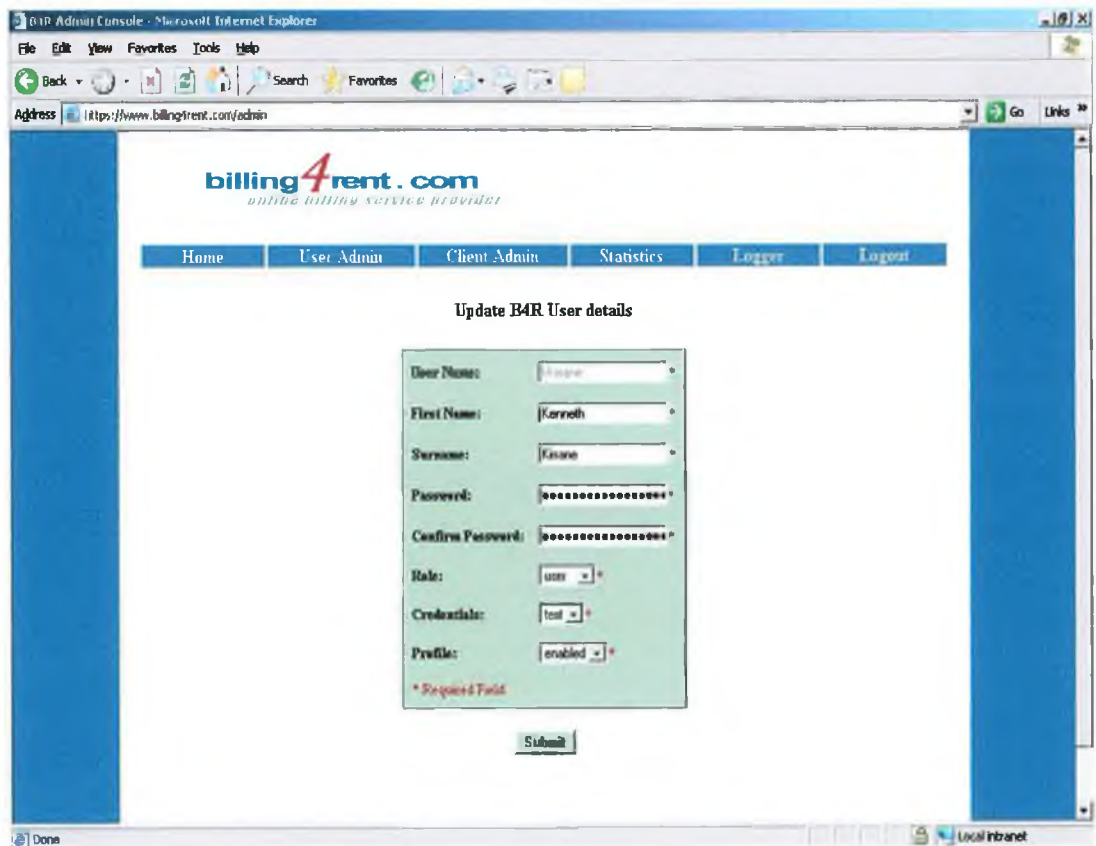
* Required Field

Submit

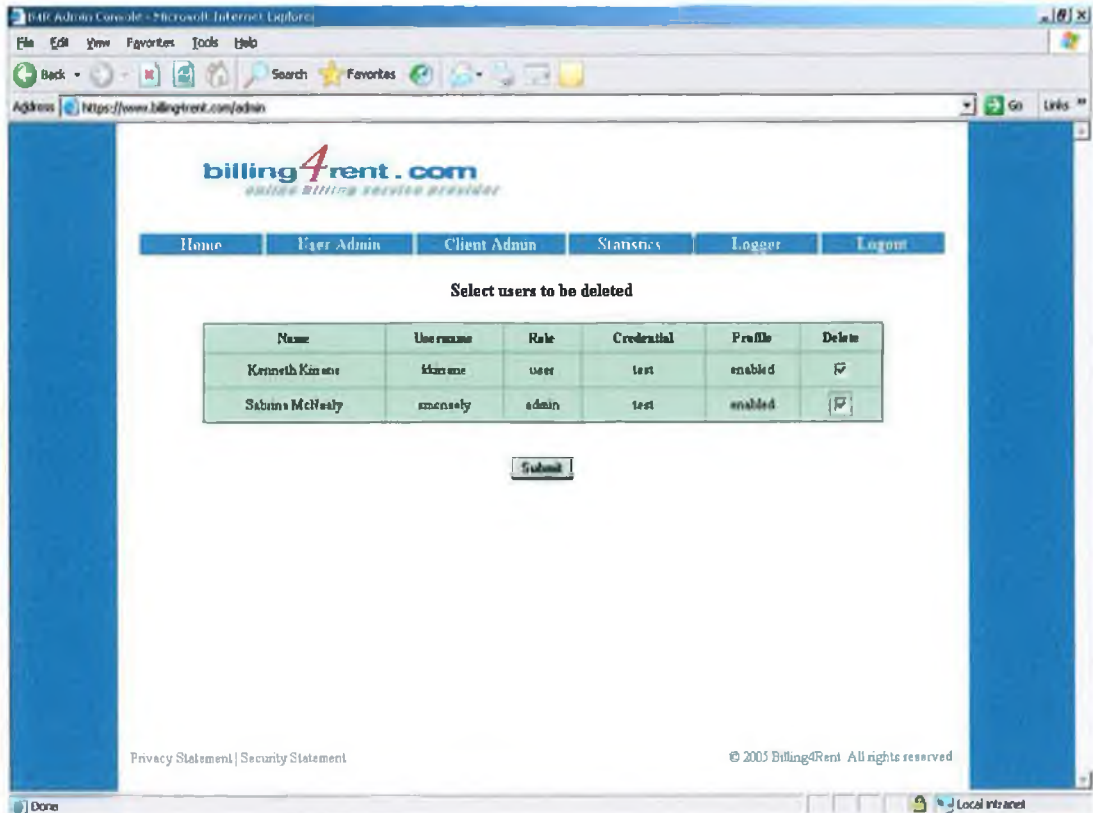
37) Administrative Interface: Update User Details – Select User to update



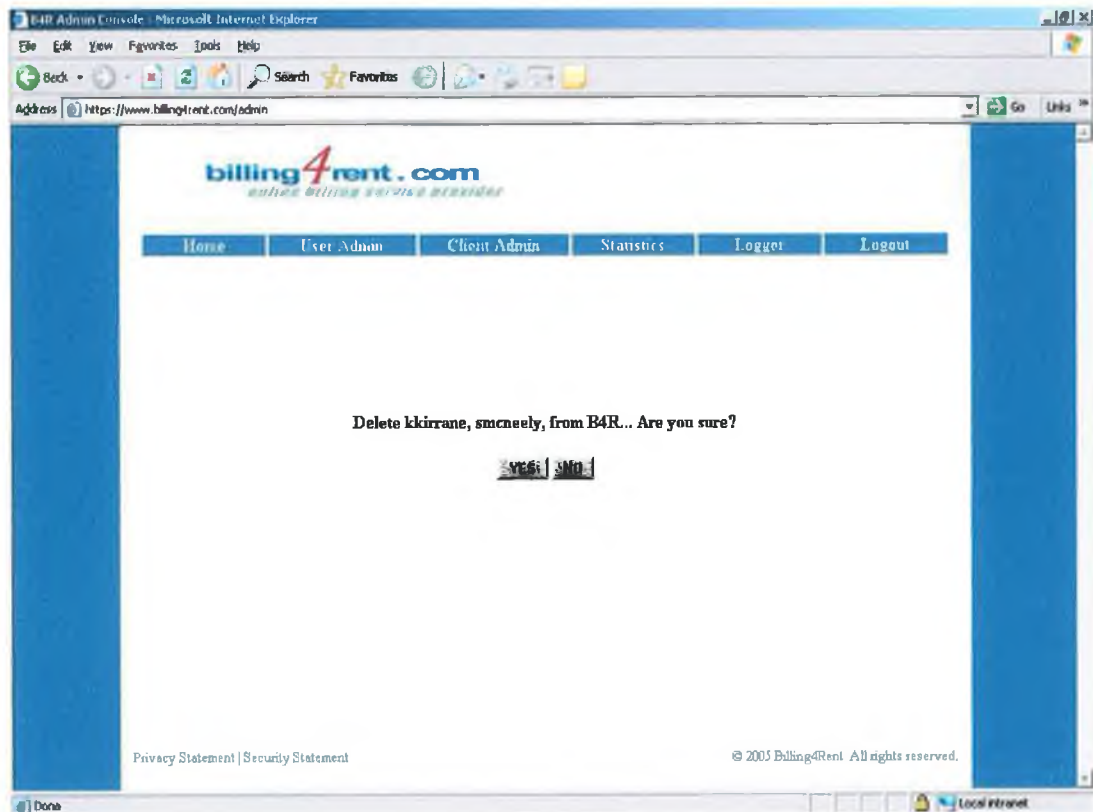
38) Administrative Interface: Update User Details



39) Administrative Interface: Select User to Delete



40) Administrative Interface: Delete Users Confirmation



41) Administrative Interface: Display All Clients

The screenshot shows the B4R Admin Console in Microsoft Internet Explorer. The address bar displays <https://www.billing4rent.com/admin>. The page features a navigation menu with links for Home, User Admin, Client Admin, Statistics, Logger, and Logout. The main content area is titled "B4R Client Information" and contains a table with the following data:

Name	Username	Company Details	Role	Credentials	Profile
Nicole Keene	nkeene	HP	admin	test	enabled
Evalyn Lynch	elynch	Noriel	admin	test	enabled

At the bottom of the page, there are links for Privacy Statement and Security Statement, and a copyright notice: © 2005 Billing4Rent. All rights reserved.

42) Administrative Interface: Display Client Details

The screenshot shows the B4R Admin Console displaying the details for a client. The address bar displays <https://www.billing4rent.com/admin>. The navigation menu is the same as in the previous screenshot. The main content area is titled "B4R client company details" and contains a table with the following information:

Company Name:	HP
Street Address:	Ballybrent
	Galway
	Ireland
Post Code:	G1
Email:	nkeene@hotmail.com
Phone Number:	091 772378

Below the table is a "Back" button. At the bottom of the page, there are links for Privacy Statement and Security Statement, and a copyright notice: © 2005 Billing4Rent. All rights reserved.

43) Administrative Interface: Register New Client

The screenshot shows a web browser window with the address bar displaying `https://www.billing4rent.com/admin`. The page header includes the logo for **billing4rent.com** and a navigation menu with links for Home, User Admin, Client Admin, Statistics, Logger, and Logout. The main content area is titled "Enter new B4R client details" and contains a form with the following fields:

- Client Company Name:
- Street Address:
- Post Code:
- Email:
- Phone Number:
- First Name:
- Surname:
- Client ID:
- Password:
- Confirm Password:
- Role:
- Credentiale:
- Profile:

A red asterisk indicates that fields with an asterisk are required. A "Submit" button is located at the bottom of the form. The footer contains links for "Privacy Statement" and "Security Statement", and a copyright notice: "© 2005 Billing4Rent. All rights reserved."

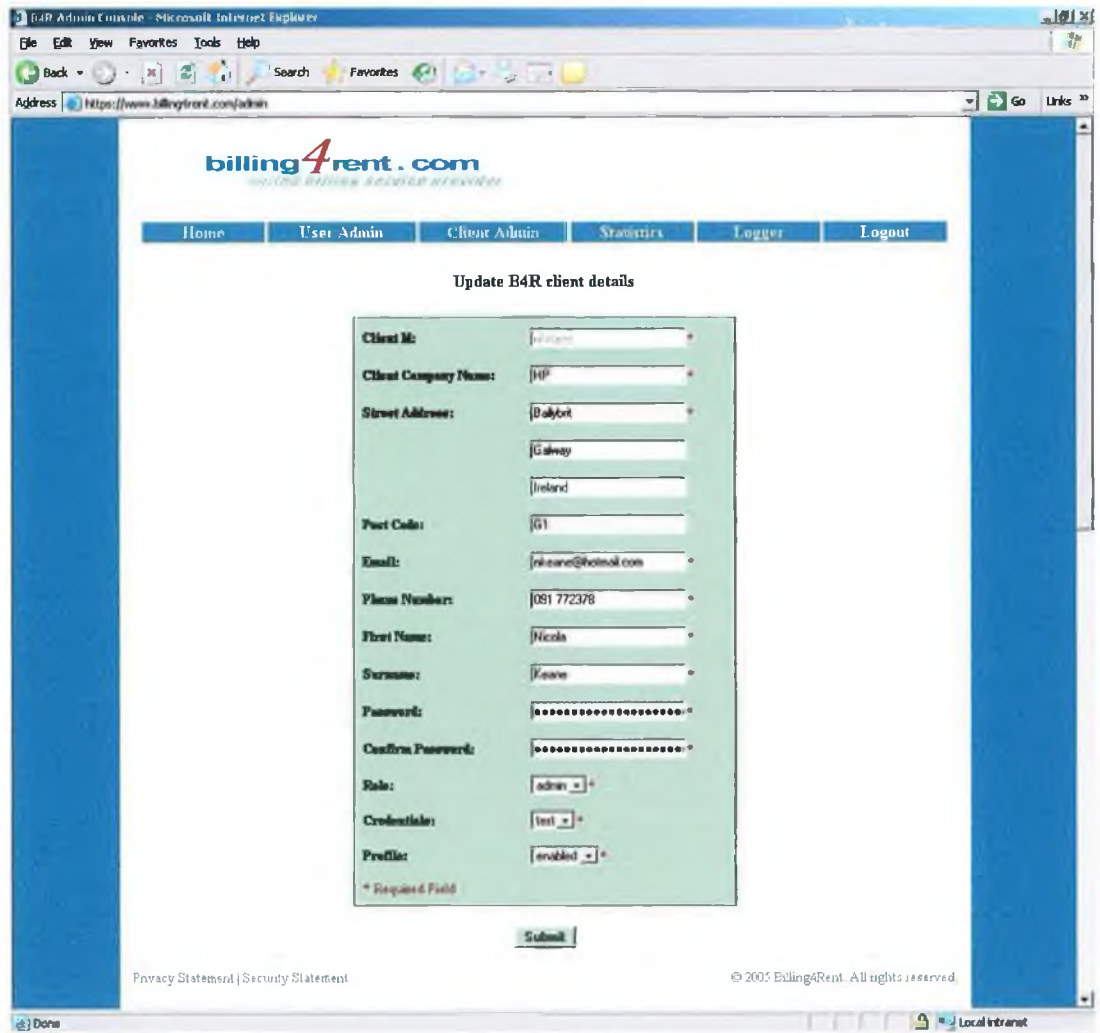
44) Administrative Interface: Update Client – Select Client to be Updated

The screenshot shows a web browser window with the address bar displaying `https://www.billing4rent.com/admin`. The page header includes the logo for **billing4rent.com** and a navigation menu with links for Home, User Admin, Client Admin, Statistics, Logger, and Logout. The main content area is titled "Enter user name of B4R client to be updated" and contains a form with the following fields:

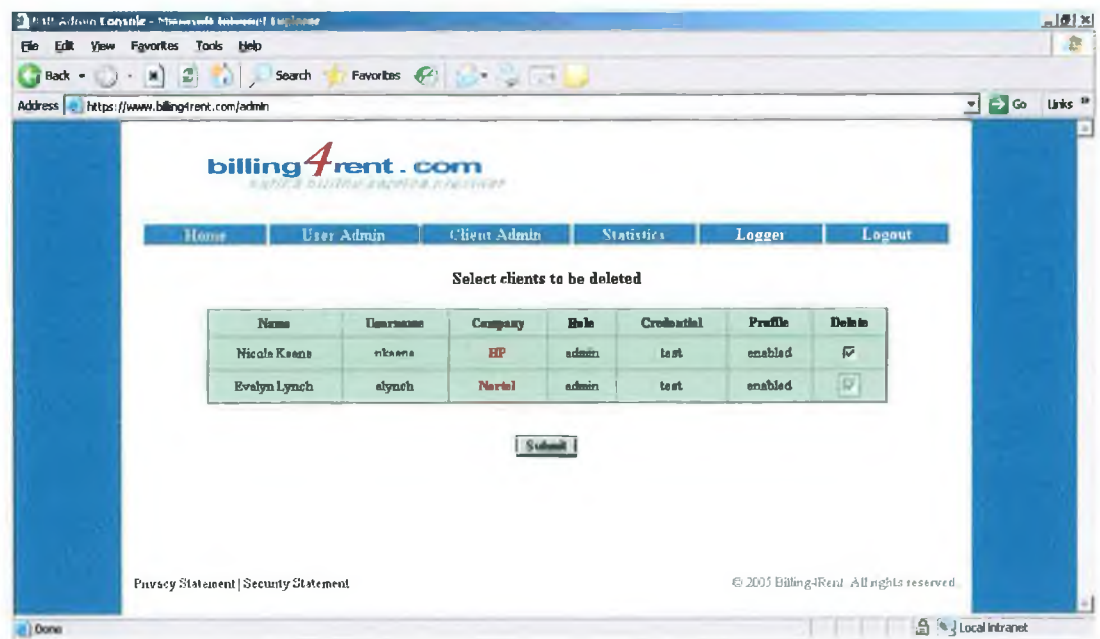
- User Name:

A "Submit" button is located below the form. The footer contains links for "Privacy Statement" and "Security Statement", and a copyright notice: "© 2005 Billing4Rent. All rights reserved."

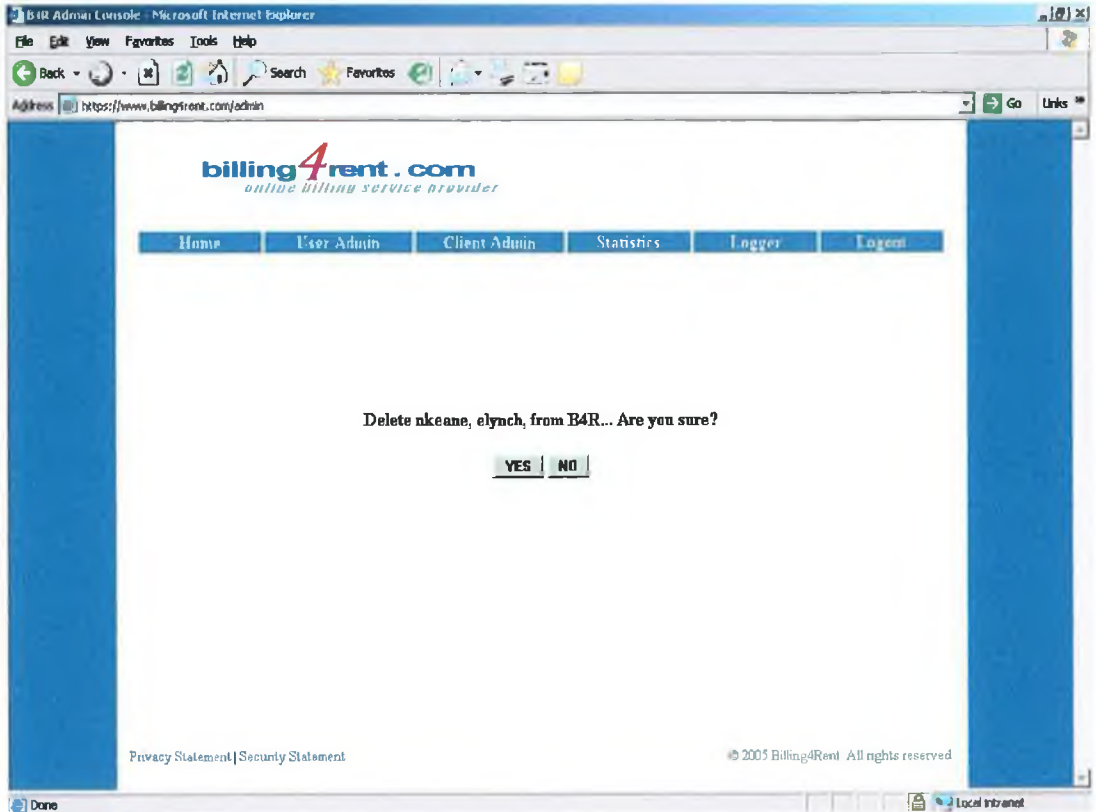
45) Administrative Interface: Update Client Details



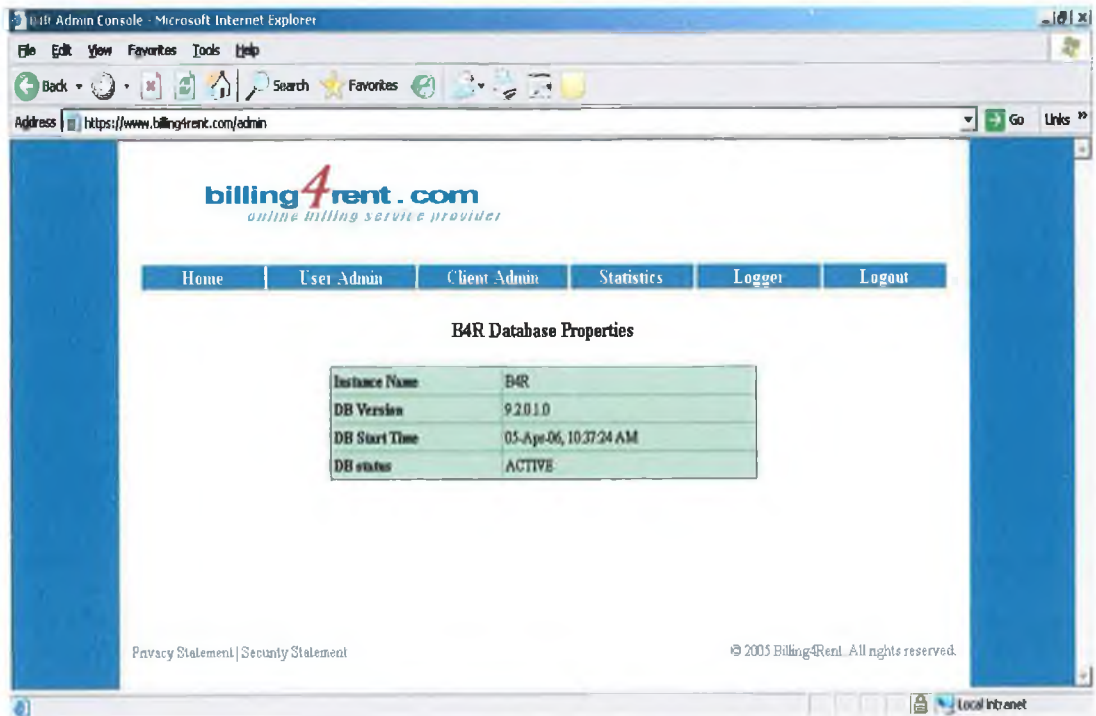
46) Administrative Interface: Delete Clients



47) Administrative Interface: Delete Clients Confirmed



48) Administrative Interface: Statistics – Database Properties



49) Administrative Interface: Statistics – Tablespace Properties

billing4rent.com
[Home](#) [User Admin](#) [Client Admin](#) [Statistics](#) [Logger](#) [Logout](#)

B4R Tablespace Properties

Tablespace Name	B4R
Tablespace Size	15.00 MB
Tablespace Used	0.69 MB
Tablespace Free	14.31 MB
% of Tablespace Used	4.58%

Privacy Statement | Security Statement © 2005 Billing4Rent. All rights reserved.

50) Administrative Interface: Statistics – Table Sizes

billing4rent.com
[Home](#) [User Admin](#) [Client Admin](#) [Statistics](#) [Logger](#) [Logout](#)

B4R Tables Sizes

B4R User	Number of Rows in Table: 2
B4R Client	Number of Rows in Table: 2
B4R Customer	Number of Rows in Table: 2
B4R Product	Number of Rows in Table: 7
B4R Invoice Header	Number of Rows in Table: 2
B4R Invoice Details	Number of Rows in Table: 4
B4R Error Log	Number of Rows in Table: 2
B4R Audit Log	Number of Rows in Table: 189
B4R Security Log	Number of Rows in Table: 5
B4R Admin Error Log	Number of Rows in Table: 2
B4R Admin Audit Log	Number of Rows in Table: 139

Privacy Statement | Security Statement © 2005 Billing4Rent. All rights reserved.

51) Administrative Interface: Billing4Rent Performance Monitor

Monitor Label	Hits	Avg. Hits	Total Hits	Std Dev	Min. Hits	Max. Hits	Active	Avg. Active	Max. Active	First access	Last access	Priority	Q. Hits	11.29ms
JSP . b4Home.jsp	1	10	10	0	10	10	0	1	1	06/04/06 15:30:37	06/04/06 15:30:37		1/0 (1/0/1)	
JSP . clientCustomerDetails.jsp	1	10	10	0	10	10	0	1	1	06/04/06 15:31:17	06/04/06 15:31:17		1/0 (1/0/2)	
JSP . clientDisplayClient.jsp	1	0	0	0	0	0	0	1	1	06/04/06 15:31:09	06/04/06 15:31:09		1/0 (1/0/2)	
JSP . clientDisplayCustomers.jsp	2	25	50	21	10	40	0	1	1	06/04/06 15:31:15	06/04/06 15:31:18		1/0 (1/0/2)	
JSP . clientHome.jsp	1	0	0	0	0	0	0	1	1	06/04/06 15:31:08	06/04/06 15:31:08		1/0 (1/0/2)	
JSP . clientLogin.jsp	1	10	10	0	10	10	0	1	1	06/04/06 15:30:57	06/04/06 15:30:57		1/0 (1/0/1)	
JSP . clientMsgDisplay.jsp	1	30	30	0	30	30	0	1	1	06/04/06 15:31:13	06/04/06 15:31:13			
JSP . clientUpdateClient.jsp	1	10	10	0	10	10	0	1	1	06/04/06 15:31:11	06/04/06 15:31:11		1/0 (1/0/2)	
SQL . SELECT * from B4RCUSTOMER where CLIENTID	2	15	30	21	0	30	0	1	1	06/04/06 15:31:15	06/04/06 15:31:18		1/0 (1/0/2)	
SQL . SELECT * from B4RCUSTOMER where CUSTOMERID	1	20	20	0	20	20	0	1	1	06/04/06 15:31:17	06/04/06 15:31:17			1/0 (1/0/2)
SQL . SELECT * from B4RCUSTOMER where CLIENTID	2	0	0	0	0	0	0	1	1	06/04/06 15:31:09	06/04/06 15:31:11		2/0 (1/0/2)	
SQL . Update B4Rclient	1	40	40	0	40	40	0	1	1	06/04/06 15:31:13	06/04/06 15:31:13			
Servlet . ClientCheckLoggedIn	1	0	0	0	0	0	0	1	1	06/04/06 15:31:13	06/04/06 15:31:13		1/0 (1/0/3)	
Servlet . ClientCheckPrivilegedAction	13	3	40	4	0	10	0	1	1	06/04/06 15:31:08	06/04/06 15:31:18		13/0 (1/0/1/9)	
Servlet . clientCustomerDetails	1	300	300	0	300	300	0	1	1	06/04/06 15:31:18	06/04/06 15:31:17			
Servlet . clientDisplayClient	1	341	341	0	341	341	0	1	1	06/04/06 15:31:09	06/04/06 15:31:09			

52) Administrative Interface: Filter Logger Dropdowns

billing4rent.com

Home User Admin Client Admin Statistics Logger Logout

User ID: All
 Date Logged: All
 Status: All
 Set Filter

53) Administrative Interface: Display Logger

billing4rent.com
online billing service provider

Home | User Admin | Client Admin | Statistics | **Logger** | Logout

User ID: Date Logged: Status:

B4RauditLog

User ID	Date	Status	Module	Logger Message
nikene	05-Apr-06, 14:51:55 PM	5	B4RClient	Authenticate B4RClient [nikene]
nikene	05-Apr-06, 14:51:55 PM	5	RdbmsClientLoginModule	Validation of login details for B4RClient [nikene]
nikene	05-Apr-06, 14:51:55 PM	5	RdbmsClientLoginModule	Trying to connect to database..
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Connected to database!
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	SELECT * FROM B4RCLIENT where clientid='nikene' and enabled = 'true'
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	SQL select complete
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Passwords do NOT match!
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Validation of login details for B4RClient [nikene] failed
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Abort
nikene	05-Apr-06, 14:51:56 PM	5	RdbmsClientLoginModule	Logout
nikene	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	Connected to database!
nikene	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	Passwords match!
nikene	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	UPDATE B4RCLIENT set last_login_time = 1144245116388 where CLIENTID = 'nikene'
nikene	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	SQL update complete
nikene	05-Apr-06, 14:52:06 PM	5	RdbmsClientLoginModule	Validation of login details for B4RClient [nikene] succeeded

Pages: 1 2 3 4 5 6 7 8 9 10 [Next]

Privacy Statement | Security Statement © 2005 Billing4Rent. All rights reserved.

54) Administrative Interface: Delete Single – Select User to be Deleted

billing4rent.com
online billing service provider

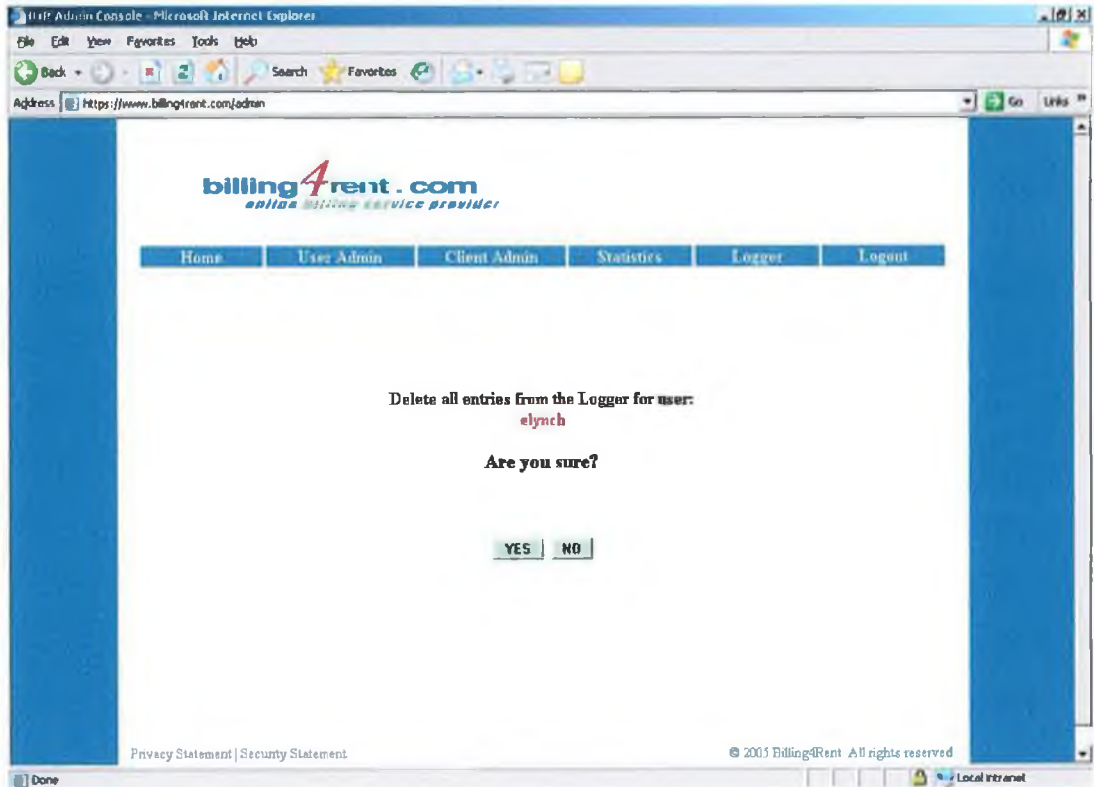
Home | User Admin | Client Admin | Statistics | **Logger** | Logout

Submit the ID of the User to be deleted from the B4RauditLog:

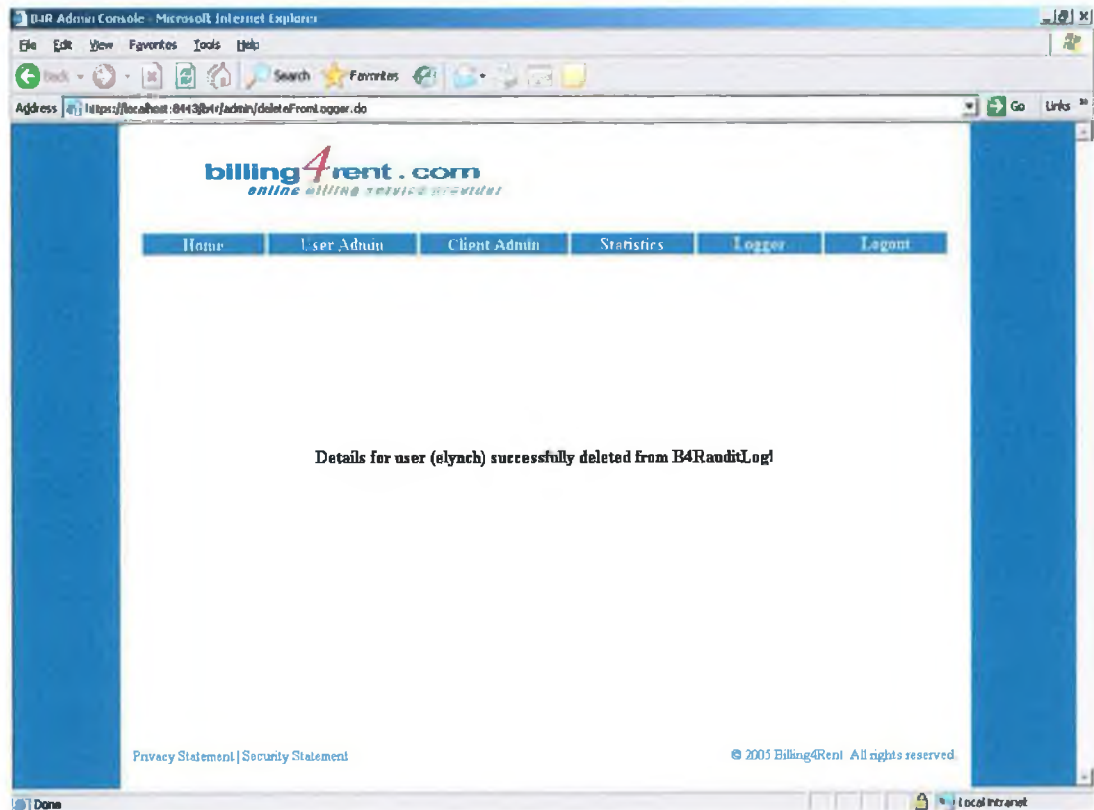
User ID:

Privacy Statement | Security Statement © 2005 Billing4Rent. All rights reserved.

55) Administrative Interface: Delete Single Confirmation



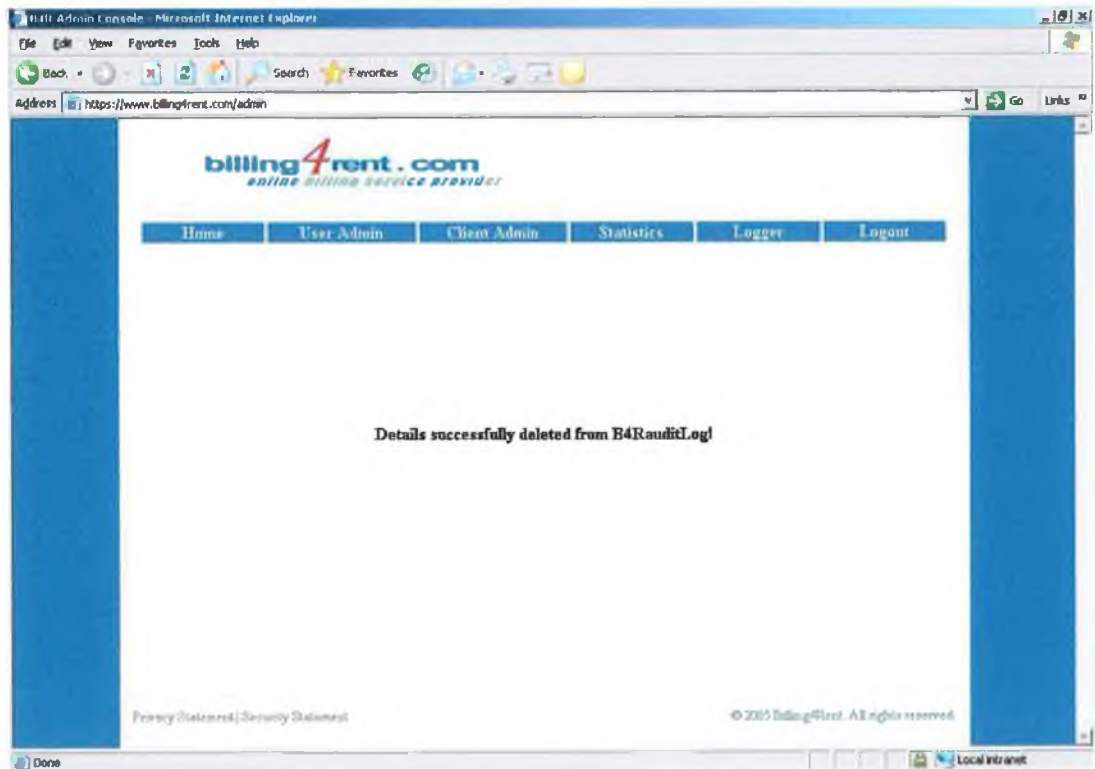
56) Administrative Interface: Delete Single Confirmed



57) Administrative Interface: Delete All Confirmation



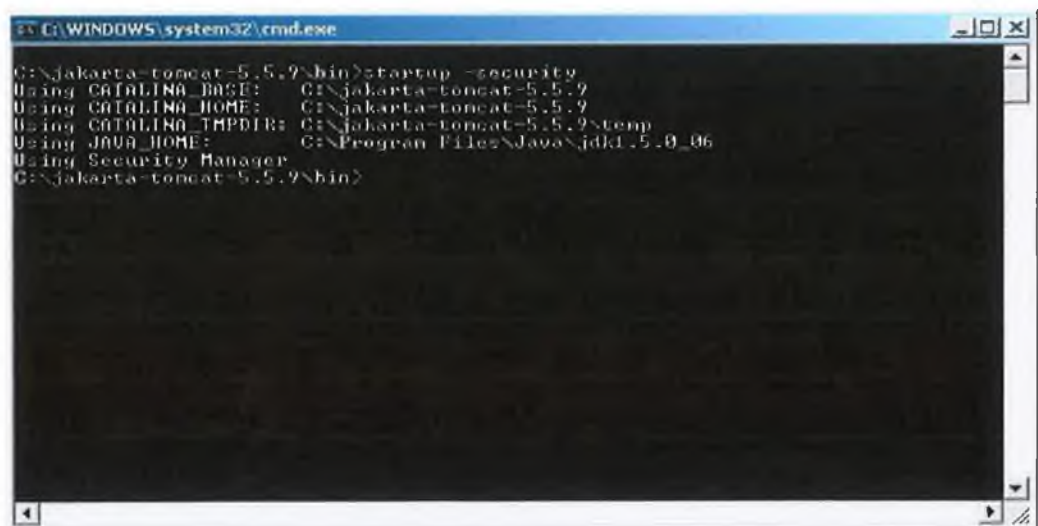
58) Administrative Interface: Delete All Confirmed



Appendix 3 – B4R Prototype CD-ROM

The B4R Prototype CD-ROM also contains all B4R Prototype source code.

- 1) Install the following pre-requisite software (the remainder of the installation process assumes the successful installation of all of the below):
 - (a) JAVA Runtime Environment
 - (b) Apache (Jakarta) Tomcat Web Server
 - (c) Oracle 9i
- 2) Create a directory called **b4r** under the <DRIVE>/jakarta-tomcat-5.5.9\webapps directory (e.g. C:\jakarta-tomcat-5.5.9\webapps\b4r)
- 3) Copy the contents of the **B4R-Prototype** directory on the below CD to the newly created **b4r** directory on the hard disk.
- 4) Copy the following files from the **B4R-Supporting Files** directory to the directory specified:
 - (a) keystore - copy to the home directory of the currently logged in user
 - (b) catalina.policy copy to the “C:\jakarta-tomcat-5.5.9\conf” directory
- 5) From the “C:\jakarta-tomcat-5.5.9\webapps\b4r\B4R Scripts” directory, run the included SQL scripts in the following order:
 - (a) B4R.sql
 - (b) Create Procedure.sql
 - (c) Create DBMS_JOB.sql
- 6) Start the TOMCAT web server using the following command:



```
C:\WINDOWS\system32\cmd.exe
C:\jakarta-tomcat-5.5.9\bin>startup -security
Using CATALINA_BASE:   C:\jakarta-tomcat-5.5.9
Using CATALINA_HOME:   C:\jakarta-tomcat-5.5.9
Using CATALINA_TMPDIR: C:\jakarta-tomcat-5.5.9\temp
Using JAVA_HOME:       C:\Program Files\Java\jdk1.5.0_06
Using Security Manager
C:\jakarta-tomcat-5.5.9\bin>
```

7) (This step assumes the successful completion of Step 5). Use a Web Browser to view the B4R Prototype Client Site, by using the following address: <http://<hostname>/b4r/b4rHome.jsp>, and the below login details:

(a) B4R Client Site

Username: nkeane

Password: R3s3arch!

(b) B4R Administrative Interface

Username: kkirrane

Password: R3s3arch!