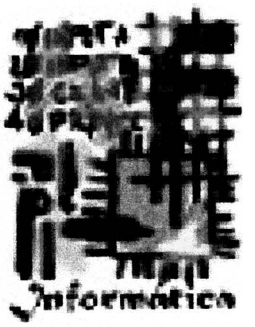
Universidade do Porto
Faculdade de Engenharia
FEUP

Pedro Alberto Pinto da Silva Pontes

Helpdesk Tools and the Common Package at Imagine Broadband SARL

LEIC
1/PONp

LEIC
Porto, 2002



Helpdesk Tools and the Common Package at Imagine Broadband SARL

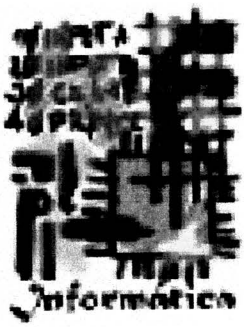
Pedro Alberto Pinto da Silva Pontes

CONFIDENTIAL REPORT

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

2002-10-07



Helpdesk Tools and the Common Package at Imagine Broadband SARL

Pedro Alberto Pinto da Silva Pontes

LEIC 2001/02 – Academic Internship Report

Coordinator at FEUP: Prof. André Puga

Coordinator at Imagine Broadband SARL: Eng. Henri Chabrier

CONFIDENTIAL REPORT

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

2002-10-07

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 65320
CDU 004(047.3)
Data 14 / 3 / 2003

À minha família,
por toda a ajuda e disponibilidade

Summary

The internship took place at Imagine Broadband SARL and was composed of two projects. The first consisted in the development of tools to support the IB Helpdesk, a new Technical Support service that the company offers to its clients. The second comprised the improvement and maintenance of a set of applications that belong to the IB Common Package, which is a series of programs whose functionality is required by two main IB products.

The first project started from a requirements specification. The new system was then designed and developed, although the development process was harmed by harsh time restrictions. Due to the requirements' characteristics, the system was developed as a set of *web* applications. Despite the troubled development process, the outcome was a success. The Helpdesk process is now entirely supported by the developed applications, although there is conscience that some details should be improved.

The work in the second project started with a discussion about items to improve in a program from the IB Common Package. A number of other programs from that package would also have to be maintained over time, answering new client's requirements, improvements in efficiency and debugging / correction tasks. All these programs have a strong networking component. The result was also very satisfactory - the improvements were all implemented and even surpassed and the new requirements were always corresponded.

The outcome of the internship is truly positive. The intern has always corresponded to the proposed tasks and has thus successfully integrated in a corporate working environment like the one at Imagine Broadband.

Acknowledgements

I would like to acknowledge the following persons, who made this internship possible and/or played an important role in its success:

At the company, Eng. Gregory Collins, Eng. Henri Chabrier and Eng. Beatriz Sanchez, with whom I worked directly.

At the college, Prof. André Puga and Prof. Raúl Vidal, the first for coordinating the internship and the second for valuable help to its attainment.

Finally, a strong thank you to my colleague and friend "Manel", for all the gossip during these six months and all the valuable tips on this report business.

Glossary

CMTS - A cable modem termination system is a component that exchanges digital signals with cable modems on a cable network. A cable modem termination system is located at the local office of a cable television company or ISP.

CNR – Short for Cisto Network Registrar (that acts as a DHCP server).

CPE - Short for customer premises equipment, communications equipment that resides on the customer's premises.

DOCSIS - Data Over Cable Service Interface Specifications is a standard interface for cable modems, the devices that handle incoming and outgoing data signals between a cable TV operator and a personal or business computer or television set.

DSL – Digital Subscriber Line, technology for bringing high-bandwidth information to homes and businesses over ordinary copper telephone lines.

DTV – Digital Television, the transmission of television signals using digital rather than conventional analog methods.

IIP – Short for Imagine Instant Provisioning, a product of Imagine Broadband.

ISA – Abbreviation for Imagine Service Assurance, an Imagine Broadband product.

JDBC – Java Database Connectivity, API and set of drivers that provide database access from Java applications to any database server, by a uniform method.

OSS – Operations Support System, a system that helps network operators.

PDC – Short for Product Development Center, the Imagine Broadband office at Sophia Antipolis, France.

RDBMS – Abbreviation for Relational Database Management System, a program that creates, updates and administers relational databases, using Structured Query Language (SQL) statements entered by a user or program.

SSH – Secure Shell, a protocol for securely getting access to a remote computer.

STB – Set-Top Box, a device that enables a television set to become a user interface to the Internet and also enables a television set to receive and decode digital television (DTV) broadcasts. A specialized computer that can "talk to" the Internet - that is, it contains a Web browser and TCP/IP support.

UBR - A hybrid fiber coaxial network is a telecommunication technology in which optical fiber cable and coaxial cable are used in different portions of a network to carry broadband content (such as video, data, and voice).

Walled Garden - A walled garden is an environment that controls the user's access to web content and services.

Table of Contents

1 Introduction 6
1.1 The Company: Imagine Broadband SARL 6
Area of Business 6
What Imagine Broadband Offers to its Clients 6
Activities and Skills 6
Company's Offices and the Internship Location 7
History of the Company 8
1.2 The Helpdesk and the Common Package Projects at Imagine Broadband SARL 9
The First Project – The Imagine Broadband Helpdesk 9
The Second Project – The Imagine Broadband Common Package 10
1.3 How this Document is Organized and What it Contains 11
2 The First Project: Imagine Broadband Technical Support 12
2.1 Description of the Problem 12
Support Plans 12
Trouble Tickets 12
Levels of Support 13
The Technical Support Helpdesk Process 13
2.2 Requirements of the Helpdesk Set of Applications 15
The Actors 15
The Helpdesk System Requirements Overview 16
The Helpdesk Client Front-end Requirements 16
The Helpdesk Technician Front-end Requirements 17
The Helpdesk Administration Front-end Requirements 19
2.3 Design of the Helpdesk Set of Applications 20
Support Technologies and Methodologies Used 20
System Structure 21
Logical Architecture 21
Physical Structure 25
2.4 The Implemented Products 27
Client Front-end 27
Technician Front-end 30
Administration Front-end 32
Helpdesk Keeper 33
2.5 Aspects to Improve 35
System Design 35
Graphical User Interfaces 35
2.6 Technologies, Methodologies and Tools Involved 35
PHP 35
GD Graphics Library 36
MySQL 36
Keystone 36
3 The Second Project: Imagine Broadband Common Package 38
3.1 Improving the Imagine Broadband OSS Database Populator 38

The Imagine Broadband OSS Database Populator 38
Improvements for the IB OSS DB Populator 46
The Implementation of the Improvements and Final Results 47
3.2 Maintaining the Lease Server 54
Correctly Parsing the Response DHCP Packet 55
3.3 Maintenance of the "Infrastructure" Applications 56
updateHFCIPHist 57
updateHFCScope 59
updateHFCTableLocationHSDQos 61
updateScopeUBRLink 62
3.4 Aspects to Improve 64
3.5 Technologies, Methodologies and Tools Involved 64
Object Oriented Programming 64
Java 64
JDBC 65
JavaHelp 65
Expect 65
Sybase PowerDesigner 66
Microsoft Visual Source Safe 66
Adobe Photoshop 66
4 Conclusions 68
4.1 The Helpdesk Project 68
4.2 The Common Package Project 68
4.3 Global Appreciation 68
5 Bibliography 70
Appendix A - Imagine Broadband Support Plans 71
A.1 Gold Service 71
A.2 Silver Service 71
A.3 Bronze Service 72
Appendix B - IB OSS DB Populator GUI User's Guide 74
B.1 Introduction to the Imagine Broadband DB Tool 1.2 74
B.2 Starting the Program 74
B.3 Main Window 74
Menu Bar 75
Toolbars 76
Scenario Panel 77
Database Configuration/Script Information / Edition Panel 77
Output Log Panel 78
Status Bar 79
Appendix C - Lease Server Packet Manipulation Specification 80
C.1 Introduction 80
C.2 Sending a Query to the CNR 80
Sending a Lease Query 80
Sending a Reverse Lease Query 80
C.3 Receiving the Response from the CNR 81
Response to a Lease Query 81

Response to a Reverse Lease Query 82

Appendix D - What is the "Imagine Instant Provisioning" Product? 84

D.1 Fundamentally Changing Marketing for Large Scale Network Operators 84

Speeds the Introduction of New Services 84

Enables Cost Effective Trial Marketing 84

Supports Multiple Provisioning 84

Drives Business Expansion 84

Cuts Costs Through Customer Self Provisioning 85

Eliminates Fraudulent Use 85

Supports Real Time Viewing of Service Standards 85

D.2 Comprehensive Core Functionality 85

IIP Kernel 85

Customer Service Provisioning 85

Customer Service Activation 86

Customer Service Representative Application 86

D.3 Wide Range Of Tools 86

Service Creation Toolkit 86

Network Provisioning Package 86

Self Provisioning Application 86

Service Packaging Application 86

Open Access Mediation Module 86

IP Virtual Private Networks (IP VPN) Module 87

Appendix E - What is the "Imagine Service Assurance" Product? 88

E.1 Real Time Monitoring and Control 88

Enhances Infrastructure Management 88

Streamlines Component Management 88

Simplifies Management and Monitoring 88

Reduces Operating Costs 88

Ensures Performance Quality 89

Maximizes Service Quality Achievement 89

Improves Customer Satisfaction Levels 89

Supports Business Development 89

E.2 Comprehensive Core Functionality 89

Service Monitoring Manager 89

Imagine Service Emulator Agent 89

Knowledge Management System 90

Master Management Console 90

Client Management Console 90

Correlation Engine 90

E.3 Wide Range of Tools 90

Imagine SLA Management Product 90

Imagine Inventory Management Product 90

Imagine Performance Management Product 90

Imagine Security Manager 90

Imagine Cable Modem Manager 90

Imagine DSL Modem Manager 91

Imagine Set Top Box Manager 91

Imagine IP Address Management Product 91
Imagine Integrated Element Manager System 91
Imagine Service Emulator Agents 91

Table of Figures

Figure 1-1 – *Imagine* Broadband Offices in the World 7
Figure 1-2 – The Periods of the Two Projects 9
Figure 2-1 - Technical Support Process Overview (source: Bibliography [1]) 14
Figure 2-2 – Overview of the Use Cases 16
Figure 2-3 – Use Cases for the Helpdesk Client Front-end 17
Figure 2-4 – Helpdesk Technician Front-end Use Cases 18
Figure 2-5 – Administration Front-end Use Cases 19
Figure 2-6 – The Imagine Broadband Helpdesk Structure 21
Figure 2-7 – Package Diagram of each Front-end Application 22
Figure 2-8 – Package Diagram of the Helpdesk Keeper Module 22
Figure 2-9 – Helpdesk Class Diagram 23
Figure 2-10 – Deployment Diagram of the Helpdesk Applications 25
Figure 2-11 – Package Component Diagram of the Helpdesk Applications 26
Figure 2-12 – Trouble Ticket Listing in the Client Front-end Website 27
Figure 2-13 – Activity Diagram for Trouble Ticket Insertion 28
Figure 2-14 – Graph Drawing Activity Diagram 29
Figure 2-15 – Screen Shot of the Graph Report Page 30
Figure 2-16 – New Phone Call Page at the Technician Front-end 31
Figure 2-17 – New Phone Log Entry Activity Diagram 32
Figure 2-18 – Screen Shot of the Administration Front-end 33
Figure 2-19 – Helpdesk Keeper Activity Diagram 34
Figure 2-20 – Ticket View in Keystone 37
Figure 3-1 – Example of a Data File in Excel Format 40
Figure 3-2 – IB OSS DB Populator Use Cases Diagram 41
Figure 3-3 – Package Diagram of the IB OSS DB Populator 42
Figure 3-4 – Class Diagram of the IB OSS DB Populator 43
Figure 3-5 – Deployment Diagram of the IB OSS DB Populator 44
Figure 3-6 – Overview of the Component Structure of the IB OSS DB Populator 44
Figure 3-7 – Activity Diagram of the IB OSS Database Populator Scenario Execution 45
Figure 3-8 – Screenshot of the Application's First Version GUI 46

Figure 3-9 – Class Diagram of the Improved Application 48
Figure 3-10 – Activity Diagram of the Improved Scenario Execution Algorithm 49
Figure 3-11 – Sequence Diagram of a Scenario Execution Context 50
Figure 3-12 – Main Window of the New IB OSS DB Populator 51
Figure 3-13 – Database Configuration Parameters Dialog Box 52
Figure 3-14 – Script Edition Window 53
Figure 3-15 – Online Help Window 54
Figure 3-16 – Use of Lease Server Sequence Diagram 55
Figure 3-17 – DHCP Parsing Activity Diagram 56
Figure 3-18 – Overview of the Infrastructure Applications Process 57
Figure 3-19 – Activity Diagram of updateHFCIPHist 59
Figure 3-20 – updateHFCScope Activity Diagram 61
Figure 3-21 – updateHFCTableLocationHSDQos Activity Diagram 62
Figure 3-22 – updateScopeUBRLink Activity Diagram 63
Figure 3-23 – IB OSS DB Populator Logo 67

1 Introduction

In this chapter, a general introduction to the internship is provided. We start by presenting the company at which it was held and then move on to an initial description of the projects that composed the internship.

Unlike the majority of the academic internships, this one was not limited to a specific single project. It also did not have a pre-specified plan of activities. The intern was integrated in the company staff and thus became a valid resource for what the company would need. This means that the treatment, conditions and responsibilities given to the intern were no different from those given to the other company employees at the same level. This has provided an approach as close as possible to the corporate working reality.

We end this introduction by explaining how this document is organized and the by referring the contents that can be found in each major section.

1.1 The Company: Imagine Broadband SARL

In this section, the company where the internship was held is presented. We start by defining its area of business and other general information. A brief historical retrospective is also provided.

Area of Business

Imagine Broadband develops and delivers operational solutions for organizations utilizing complex communications infrastructures such as broadband networks, IP networks, broadcast systems and related services and applications. It focuses on the technology challenges that the communications industry faces repeatedly and has built unrivalled expertise in many key areas including service quality delivery, operational and deployment efficiency, and the implementation of multi-faceted new services.

What Imagine Broadband Offers to its Clients

The main goal of Imagine Broadband is to make its clients maximize the profit they draw from their communications infrastructure. The company works closely to the clients and uses technology to create first class platforms that enable high quality commercial services which have the capability to generate revenue right from the beginning. It helps large scale network operators to gain marketplace advantage by being able to provide new and superior products, in a more efficient and cost effective way. It sees current technology as the potential rather than the constraint for progress and improvement of the client's services.

Activities and Skills

The activities of the company can be described as delivering two fundamental types of business solutions:

- Content Delivery Platforms - to enable convergence (voice, video and data) such as broadcast, video, games and voice delivered over different types of access technology.
- Operational Support Products - to give network operators full visibility and control over the quality of service delivered to customers.

Skills and services include:

- Complex large scale network design, implementation and operations;
- New generation operations support, systems design and implementation including network operations centers;
- Interactive application and services;
- End user device software development and implementation (STB[1], mobile, etc.);
- Short and long term operational support;
- Ongoing technology risk management.

Company's Offices and the Internship Location

Imagine Broadband is present in four continents:

- Europe, one office in London, England and the Product Development Center in Sophia Antipolis (near Nice), France;
- America, one office in Reston (near Washington), USA;
- Asia, one office in Hong-Kong, China;
- Oceania, with an office in Sydney, Australia.

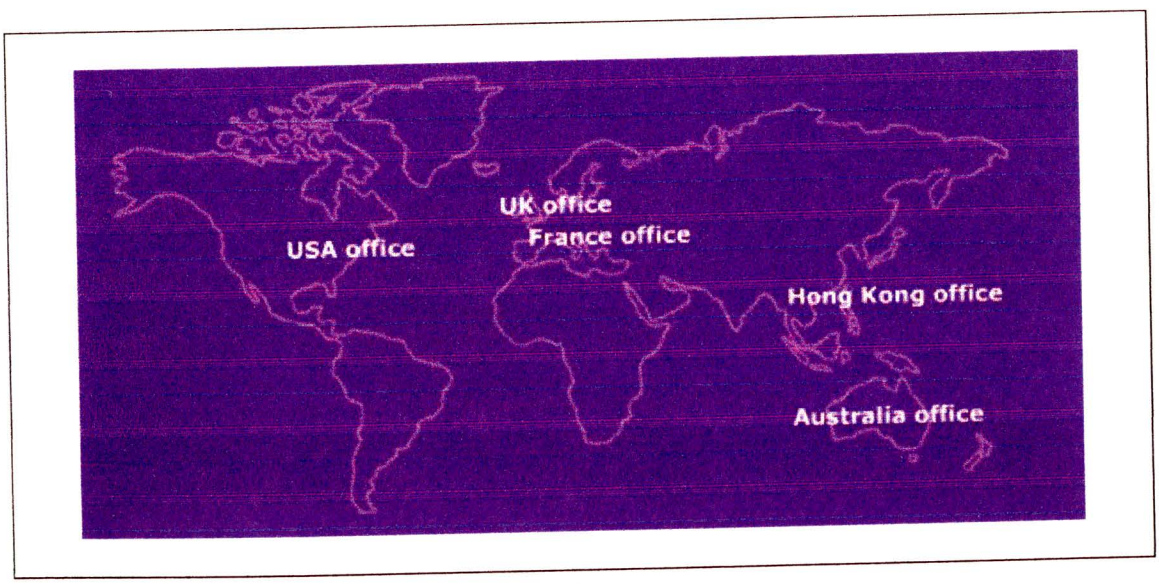


Figure 1-1 – Imagine Broadband Offices in the World

The internship took place at the Product Development Center (PDC), in Sophia Antipolis, France. The activity at the PDC is the development of advanced products and software for large scale networks, the broadband industry and interactive digital television.

[1] Set-Top Box, a device that enables a television set to become a user interface to the Internet and also enables a television set to receive and decode digital television (DTV) broadcasts. A specialized computer that can "talk to" the Internet - that is, it contains a Web browser and TCP/IP support.

History of the Company

Imagine Broadband was founded by Dr. Arjang Zadeh in March, 2000.

The Foundation

The very experienced Dr. Zadeh worked for many years in this business area. He conducted cutting edge research and development in the area of large-scale distributed multimedia systems for BT laboratories before working in numerous large scale communications clients at Accenture for 7 years. After several important achievements and when multiple requirements from the industry were arising, due to the increasing demand for large scale and complex methods of communication in the broadband consumer and large scale corporate networks, the foundation of an independent company was inevitable.

Imagine Broadband was thus born as an Accenture Technology Venture, inheriting 65 Accenture workers at its inception. Telewest Communications, a minority shareholder has also provided extensive support by signing a 3 year bulk purchase for investment into broadband through engineering consulting and large scale network delivery. The Telewest engineering organization provides extensive input into the Imagine Broadband strategy for new products and services.

Imagine Broadband is a leader in its area. Its position has attracted a great number of highly experienced industry experts, which has further strengthened the engineering organization.

To learn about two important Imagine Broadband products, please refer to Appendix D and Appendix E.

Greatest Achievements

The company's greatest achievements include:

- The first ever Broadband Digital Interactive TV Platform over Cable in the world, state of the art and scaleable true Video on Demand solution;
- Delivery of TV e-mail (IB product), Interactive Advertising and Walled Garden[2] applications and services over iDTV Platforms;
- Delivery of state of the art Network Operations Centers for multi-service platforms including DTV platforms;
- Delivery of interactive digital television projects on committed dates without one day of slippage;
- Delivery of large scale high speed data platforms for both the service provider and corporate sector in DOCSIS[3], EURODOCSIS and DSL;
- Delivery of the world's first digital interactive television platform with active cable modem giving PC broadband internet access through the set top box.

---

[2] A walled garden is an environment that controls the user's access to web content and services.

[3] DOCSIS (Data Over Cable Service Interface Specifications) is a standard interface for cable modems, the devices that handle incoming and outgoing data signals between a cable TV operator and a personal or business computer or television set.

1.2 The Helpdesk and the Common Package Projects at Imagine Broadband SARL

Here, we present the projects of the company that were involved in the internship. As referred above, the internship comprised not one, but two projects.

Work on the first project was undertaken from the beginning of the internship, at the 18th of March, to the 15th of May 2002. The second took the remaining time from the 16th of May to the end of the internship, at the 17th of September 2002. The following picture summarizes the timely placement of each project within the internship period.

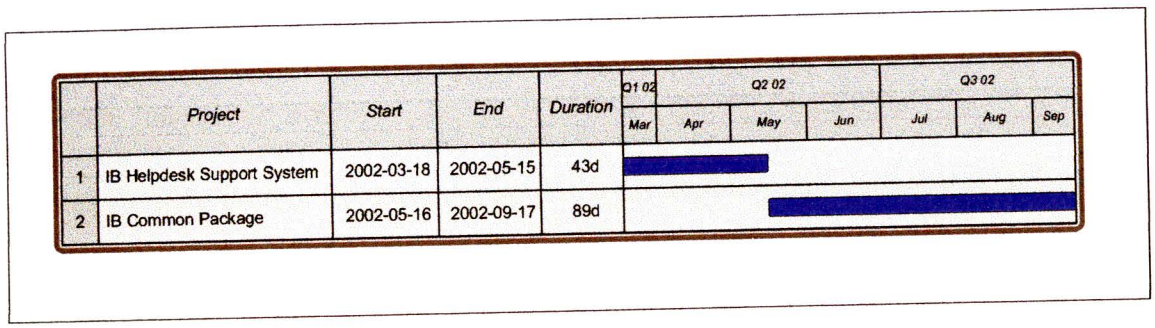


Figure 1-2 – The Periods of the Two Projects

The First Project – The Imagine Broadband Helpdesk

By the time of the beginning of the internship, Imagine Broadband was planning the creation and release of "Helpdesk" services, part of the company's technical support policy. A contract for technical support had already been signed with one of the clients, France Telecom, and more contracts with other clients, like Telewest and UPC France, were on the way.

An initial requirements study and respective system specification had been performed by the manager of the project, Eng. Gregory Collins. It was then planned to develop a specified set of essential functionalities, release the service, and then add new features and capabilities to the system. This first set of core requirements was called "Phase 1".

The intern was then assigned to study the "Phase 1" requirements and to design and develop the compliant system, which would be used to support the Helpdesk services. First, the development team was composed by two elements, the intern and one coworker. One month later, it was reduced to the intern only.

The applications would be developed using the *web* paradigm. The technologies PHP, for implementing the business logic, and MySQL, for data storage have been chosen. The application Keystone has also been chosen to initially support the "trouble ticket" system. More info on this is given throughout chapter 2, especially in section 2.3.

From all the project adversities, time restrictions were the ones that most affected its course. The date for the release of the Helpdesk services was already very close (just two weeks after, although at this phase, not all the requirements were essential and some of the work would be done manually) and no development work had been done whatsoever. This ended up to be a seriously negative point, as the initial urgent need for functionalities didn't left much room for careful system designing and planning. The practical result was a very quick development of functionalities at the cost of a badly structured system, hard to maintain and improve.

This has led to an unusual development process. Development and testing were often mingled, as new functionalities would impact on previous ones, due to the absence of initial structuring and planning.

At the end of Phase 1 development, when the first versions of the applications were already in use, there was some effort to modularize and structure the application. Great improvements have been done at this phase: the system got cleaner, easier to maintain, change or improve. One component of the system, that was considered to be off less importance at the time, was the Graphical User Interface. No human resources have been allocated for this area until now, and the GUI still lacks a professional look and feel at the present.

Despite the referred drawbacks, the project was driven to success. The helpdesk services are currently being fully supported by the developed applications, which meet the specified requirements with a great level of stability and efficiency.

Details about the project, its requirements and the developed system are provided in chapter 2.

The Second Project – The Imagine Broadband Common Package

The second project is, in all its components, different from the first one. This project consisted in the improvement and maintenance of a set of applications that belong to the Imagine Broadband Common Package.

Two of the products of the company are the Imagine Instant Provisioning and the Imagine Service Assurance. These two big products share a set of common functionalities. This set of common functionalities has been developed into what is called the Imagine Broadband Common Package which, in its turn, is used by both of the products. Further details of these functionalities may be found in chapter 3. To know more about these two products, please refer to Appendix D and Appendix E.

If the first project involved the design and development of a new system from scratch, this one involved the improvement and maintenance of an existing set of applications. What exactly is "improvement" and "maintenance" is explained next.

When the intern was made responsible for this package, a list of improvements to be performed was discussed with the internship coordinator, Eng. Henri Chabrier. The majority of these improvements had a particular application of the package as subject. This application, responsible for the creation and population of database instances, was designed to be operated manually and had a very raw user interface, with some flaws and many limitations. The others are destined to be run in a periodic base and without human interaction. The improvements to these last applications are mainly concerning efficiency issues.

The clients of the Imagine Instant Provisioning and Imagine Service Assurance products are large-scale network operators. In an area as complex as this, with all the different technologies it involves, it is easy to understand that each client has different networking infrastructure and thus different needs and also that these needs vary frequently. With "maintenance" of the Imagine Broadband Common Package we mean not only the correction of problems that eventually arise or the continuous improvement of the set applications, but also their adaptation to new client requirements.

The applications were developed in the Java language and make use of a number of technologies and applications like JDBC, Expect, FTP, Telnet and Secure FTP and SSH clients. This is detailed in section 3.4.

The outcome of this project was a success. Not only the discussed improvements were quickly implemented with perfection but also there was not a single maintenance task that hasn't been accomplished with success

The only difficulty that was felt had to do with the lack of existing comprehensive documentation about the applications in question. It was a bit hard to understand and get into them and this was achieved by experimentation, study and quick lessons by the coordinator and some coworkers.

Details about the project, its requirements and the developed system are provided in chapter 3.

1.3 How this Document is Organized and What it Contains

In this chapter we have presented an introduction to the company where the internship was held and to the two projects that it was composed of.

In chapter 2 you will find detailed information about the first project to which the intern was assigned. Its problem, requirements, specification, design and final products are described.

Chapter 3 has similar structure and contents, this time referring to the second project of the internship.

Finally, the conclusions that can be drawn from the internship are presented, in chapter 4.

This is followed by bibliographic references and the appendixes, where you can find additional information like quick introductions to the Imagine Instant Provisioning and Imagine Service Assurance products, among others.

2 The First Project: Imagine Broadband Technical Support

As referred in section 1.2, the first project of the internship consisted in the development of a set of tools to support the emerging Imagine Broadband Helpdesk, integrating the Imagine Broadband Technical Support system. The tools to be developed should meet all the Helpdesk Phase 1 Requirements. These requirements and a general description of the problem are presented in section 2.1.

2.1 Description of the Problem

Imagine Broadband not only sells products to its clients but also sells technical support plans. There are different kinds of support plans from which the companies may choose to sign a contract. Each company may have different support contracts with Imagine Broadband, based on what product is covered.

Support Plans

Each support plan provides the client with the following features:

- **Hours of service**, the period of each day and the number of days per week in which technical support is available to the client;
- **Response times**, the maximum delay in which Imagine Broadband will answer to a help request from the client and the time periods in which the company will provide updates on the status of the request. The response times are based in the nature of the request, for example, if it is critical or just a suggestion of enhancement;
- **Reviews and reports**, the time periods for operational reviews and activity reports;
- **Training**, the number of man-days provided for training, per maintenance year.

There are currently three support plans available for clients, although Imagine Broadband is open for a hybrid solution. For details on these plans, refer to Appendix A.

Trouble Tickets

A request can be a report of a bug, a call for help, a question or simply a suggestion for improvements. How does the technical support manage requests? By using the concept of "trouble tickets". Each "trouble ticket" has the following information:

- **Identification**, its internal and unique identification, by which it is distinguished from all the other tickets through all the workflow process;
- **Product**, the product to which the request is related;
- **Module**, the module of the product to which this request is related, if not to the product in general;
- **Summary**, a one line description of the request;
- **Problem**, the detailed description of the issue, question, suggestion, etc.
- **Instance impacted**, information about the program instance in which the error or issue occurred. For example, the server name and its location;

- **Assigned technician**, the person that is assigned to the processing of this request;
- **Status**, the status of this ticket, for example, new, assigned, closed, etc.;
- **Severity**, the severity of the reported problem or issue. This field determines several factors, as the response and update times and the process to which the ticket is submitted itself;
- **Date of creation**, the date in which the ticket was created;
- **Follow-ups**, a set of texts describing actions that were made in the process of solving the issues related in this ticket or simply containing additional information;
- **Resolution**, when closing a ticket, a description of the resolution should be stored with the closed ticket;
- **Attachment(s)**, file attachments like program logs or error messages;
- **Reported by**, the contact of the person who reported and created this ticket.

This is the fundamental piece of information that flows through the technical support workflow.

Levels of Support

Another concept that needs to be explained is the level of support. Imagine Broadband provides support in three levels. The first level is at customer's site. The second and third levels operate at Imagine Broadband's premises. It is the first level that takes action when a problem occurs. It determines its severity and if unable to solve it on site, submits the problem to level two. Level two personnel are responsible for receiving the tickets from level one and proceed to the first investigations at Imagine Broadband. They may contact the client for further information. They should also provide the first workarounds to the client, while the problem is being investigated. If level two finds out that the problem originates in a bug, then it submits it to level three, which is located at the Product Development Center. Level three then validates the bug, investigates and fixes it and finally tests the corrections and releases them.

The Technical Support Helpdesk Process

Having presented all the key concepts, it's time to show the processes and workflows that direct the technical support helpdesk functioning. The following flowchart shows an overview of the helpdesk process.

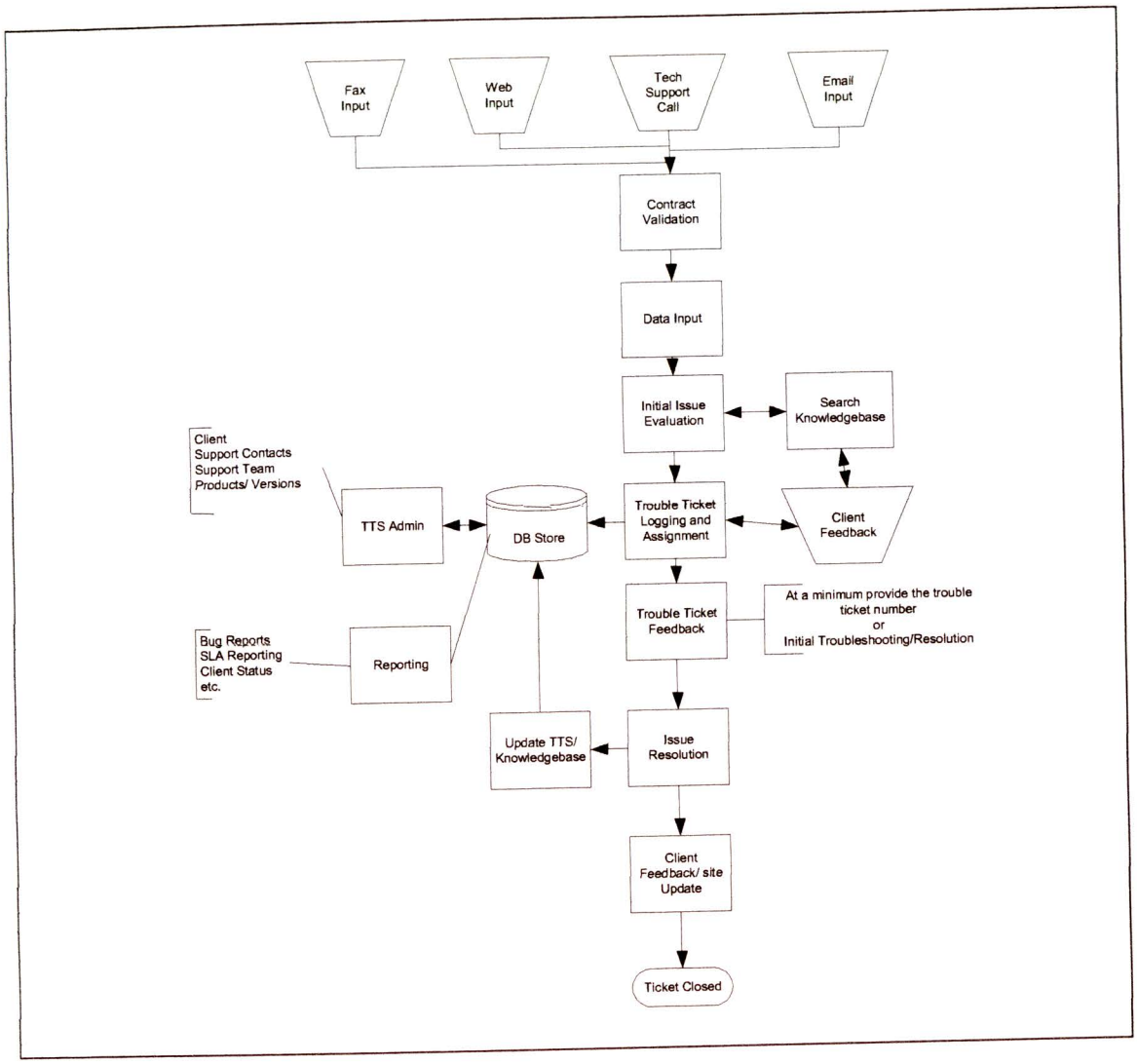


Figure 2-1 - Technical Support Process Overview (source: Bibliography [1])

Trouble tickets are input to the technical support system by four methods: fax, web, phone or email. Before any action is taken about each new help request, a validation of the support contract occurs. This validation is necessary to make sure the client has a valid support contract with Imagine Broadband for the product to which the trouble ticket is related. By valid, we mean a contract that has not expired, is not pending, is not suspended and contemplates the product in question.

If the contract is indeed valid, then the trouble ticket is effectively inserted into the technical support system, to be processed. Otherwise, the client is not able to provide the problem information.

Trouble tickets are then submitted to an initial evaluation, which consists in a research of a knowledgebase. This knowledgebase has an historical of problems and their most common resolution. The client is provided with any resolution from the knowledge base, if any, shortly after the submission of the trouble ticket. Feedback and additional information from the client may be available / required at this time.

After providing the client with eventual knowledgebase information, the ticket is stored in a database and a technician is assigned to it. The assigned technician is the person responsible for the investigation of the causes of the related problems and their resolution.

As referred above, each support plan specifies response and update times. After a ticket submission, a first response to the client must be given before the contracted response time. Also, counting from that moment, periodic updates must be provided to the client in the contracted periods of time.

Research for the cause(s) of the related problem(s) takes place. This process is managed by the assigned technician, who tries to solve the problem(s) or, if unable, asks for assistance from the development team responsible for the module(s) involved.

When a solution for the problem is found and tested, it is provided to the client who is expected to confirm the correction or provide additional feedback. When the encountered solution is proved to fix all the reported problems, the ticket is closed and the knowledgebase is updated with the new problem / solution pair(s).

The technical support process also considers the elaboration of reports per client, resuming information about ticket activity and technical support usage.

Administration tasks are also taken into consideration. The administration personnel is expected to manage all the data concerning clients, support contracts, support plans, products, modules, versions, technical staff, etc.

2.2 Requirements of the Helpdesk Set of Applications

It was then necessary to develop a set of applications that would support the helpdesk process flow. Details about these applications and their requirements are provided in this section.

As can be seen from the process described in section 2.1, the applications to develop must provide three main groups of functionality: trouble ticket insertion, trouble ticket manipulation and administrative / operational tasks. Three actors have also been identified.

The Actors

An actor is a role that some person or entity (like other system) performs when interacting with the system. The three identified actors are the following:

- **The client**, who inserts new trouble tickets and updates their information, among others;
- **The technician**, who consults the trouble tickets and updates them according to the investigation and resolution process;
- **The administrator**, who controls and manages all the information in the system, from company logins to technician information.

It was then clear that the best way to isolate each group of functionality for each actor, simplifying this way the interaction with the system and improving its security, would be by creating three separate system front ends, or applications, that would operate different tasks on the same data instances. Further details on the design of the system are provided in section 2.3.

The Helpdesk System Requirements Overview

Having introduced the roles that different kind of users will act when interacting with the system, i.e. the actors, we now move on to the specification of the requirements, by means of use cases. This specification of use cases is very important as it permits the establishment of functionalities of the system that are relevant to the user. They show the relevant tasks that the system allows or, in this case, must allow, the user to perform.

We will present the use cases by means of UML's use case diagrams. The following diagram shows the overview of the use cases of the system.

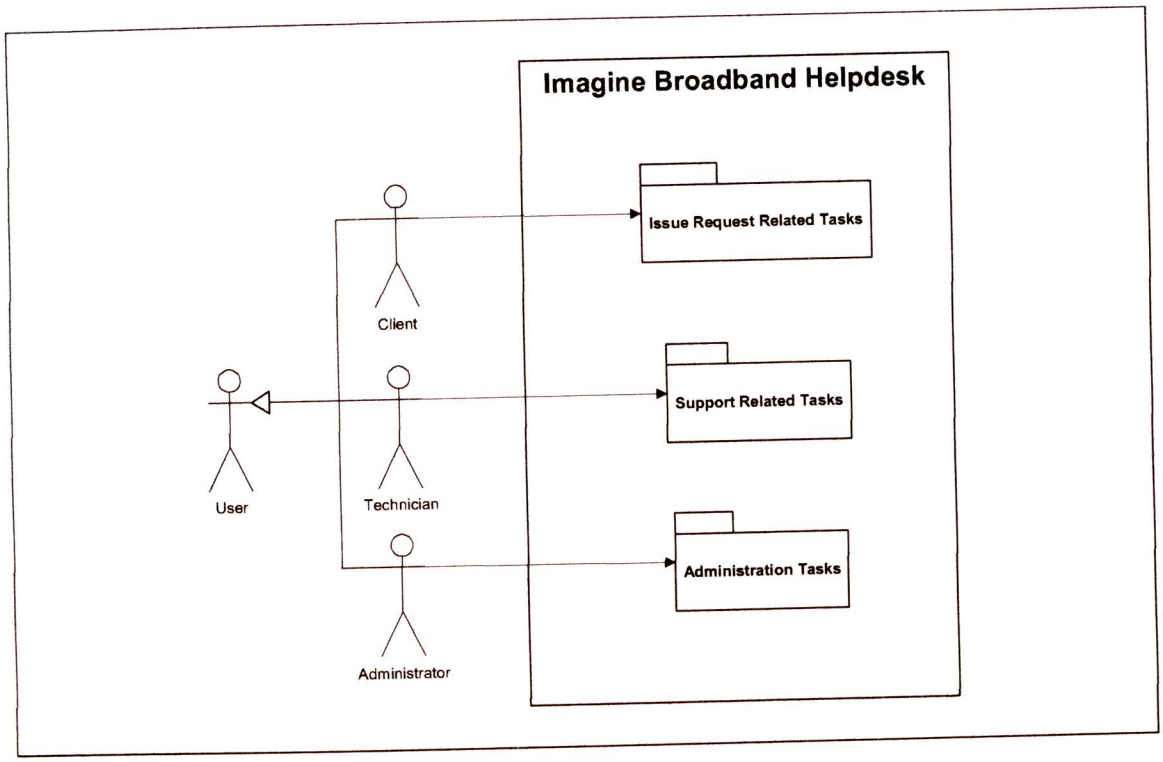


Figure 2-2 – Overview of the Use Cases

This diagram depicts the three actors that will interact with the system. The client users insert and update problems in the helpdesk system. Technicians update those tickets. Administrators manage all the information and perform administrative tasks.

In suit, we will detail each of the three packages presented in the previous figure. As we will see in the section 2.3, each package will be supported by its own front-end to the system. The package of use cases for the client will be supported by the "Helpdesk Client Front-end", the one with use cases for the technician, by the "Helpdesk Technician Front-end" and the one with the use cases for the administrator, by the "Helpdesk Administration Front-end". As referred before, these three front-ends are like three different operational tools over the same data system.

The Helpdesk Client Front-end Requirements

Here's the use case diagram for the Helpdesk Client Front-end:

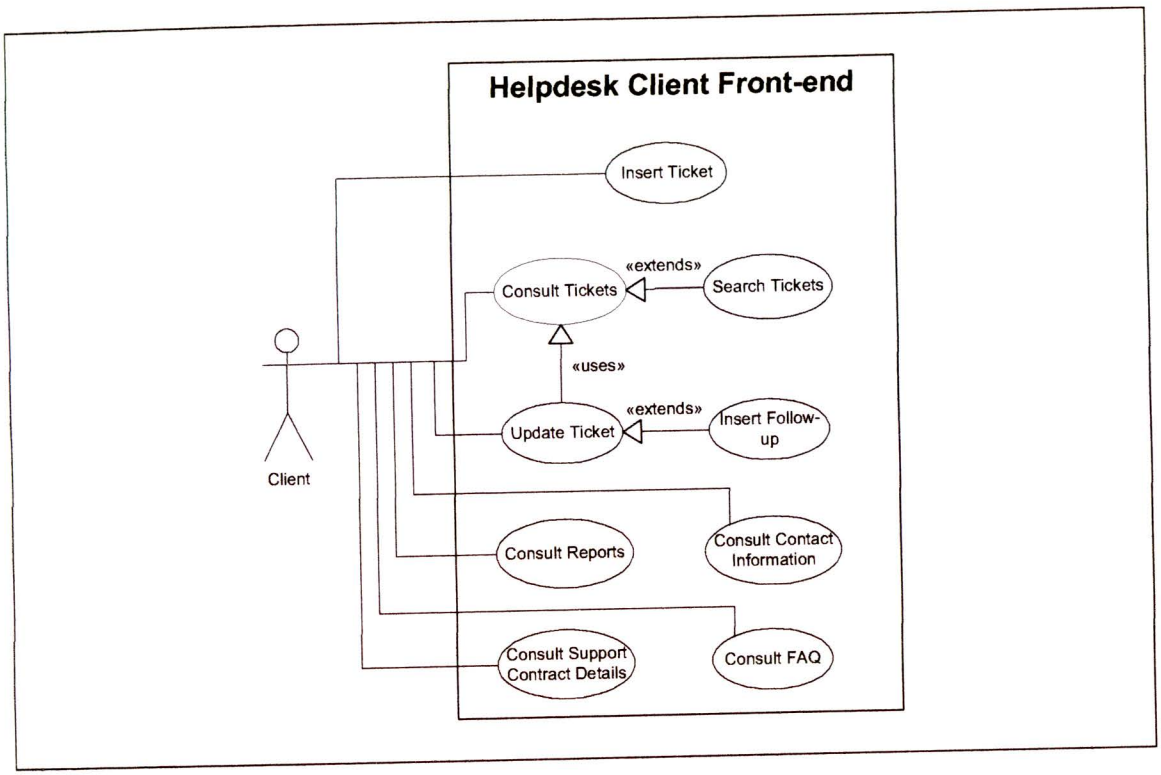


Figure 2-3 – Use Cases for the Helpdesk Client Front-end

The client must be able to introduce new trouble tickets into the system, using the Helpdesk Client Front-end. He / she must also be able to consult the tickets related to his / her company, eventually performing a search operation, and update their information. One extension to this update functionality is the insertion of a follow-up. Please refer to section 2.1 for details on trouble tickets and follow-ups. The client user may also consult reports and graphics, showing the use of the system by the company of the user. A section with frequent asked questions and their respective answers should also be provided to the client. Finally, the system must provide further contact information, like the telephone and fax numbers of the Imagine Broadband Helpdesk service.

The Helpdesk Technician Front-end Requirements

The following diagram depicts the use cases required for the Helpdesk Technician Front-end.

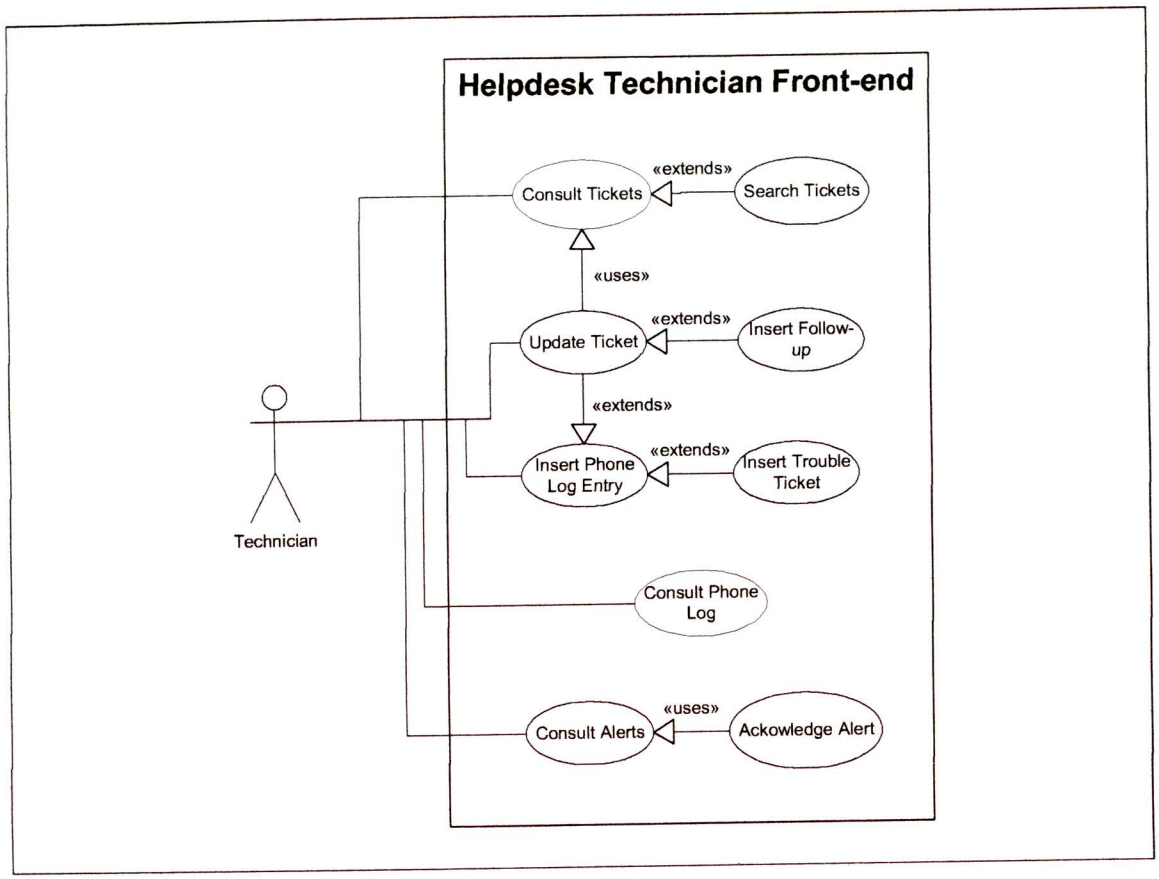


Figure 2-4 – Helpdesk Technician Front-end Use Cases

As depicted in the last diagram, the technician must be able to conduct the lifecycle of a trouble ticket to its end. This means that the system must allow the technician to consult the existing tickets, eventually performing a search operation, and to update them, which could also be inserting a new follow-up or closing the ticket.

The technician is also expected to answer phone calls from the client. In all of those cases, he must be able to insert a phone call log entry into the system. Eventually, a phone call may result in a new trouble ticket, associated to the caller client, or in the update of an existing one. The system must permit the technician to insert the new trouble ticket, in this case. For reference, the whole log of phone calls should be available to the technician.

As it was described before, each trouble ticket has a first response time and periodic updates time period. We will call these "response" and "update" by "events". In each one of those events, the client must be sent some information about the progress of the problem investigation. In an environment where there are many active trouble tickets, it would be very hard for the technical support staff to be aware and remember the exact dates of those events for each ticket. This way, the system is required to monitor this information and issue alerts when any of those types of events are to be executed soon. The system alerts should be accessible by the technicians in the Technician Front-end, who must be able to acknowledge them. By acknowledging a ticket, the technician makes him / herself responsible of performing the event in question and, this way, other technicians won't lose time analyzing the acknowledged alert.

The Helpdesk Administration Front-end Requirements

Here we present the uses cases diagram for the Administration Front-end:

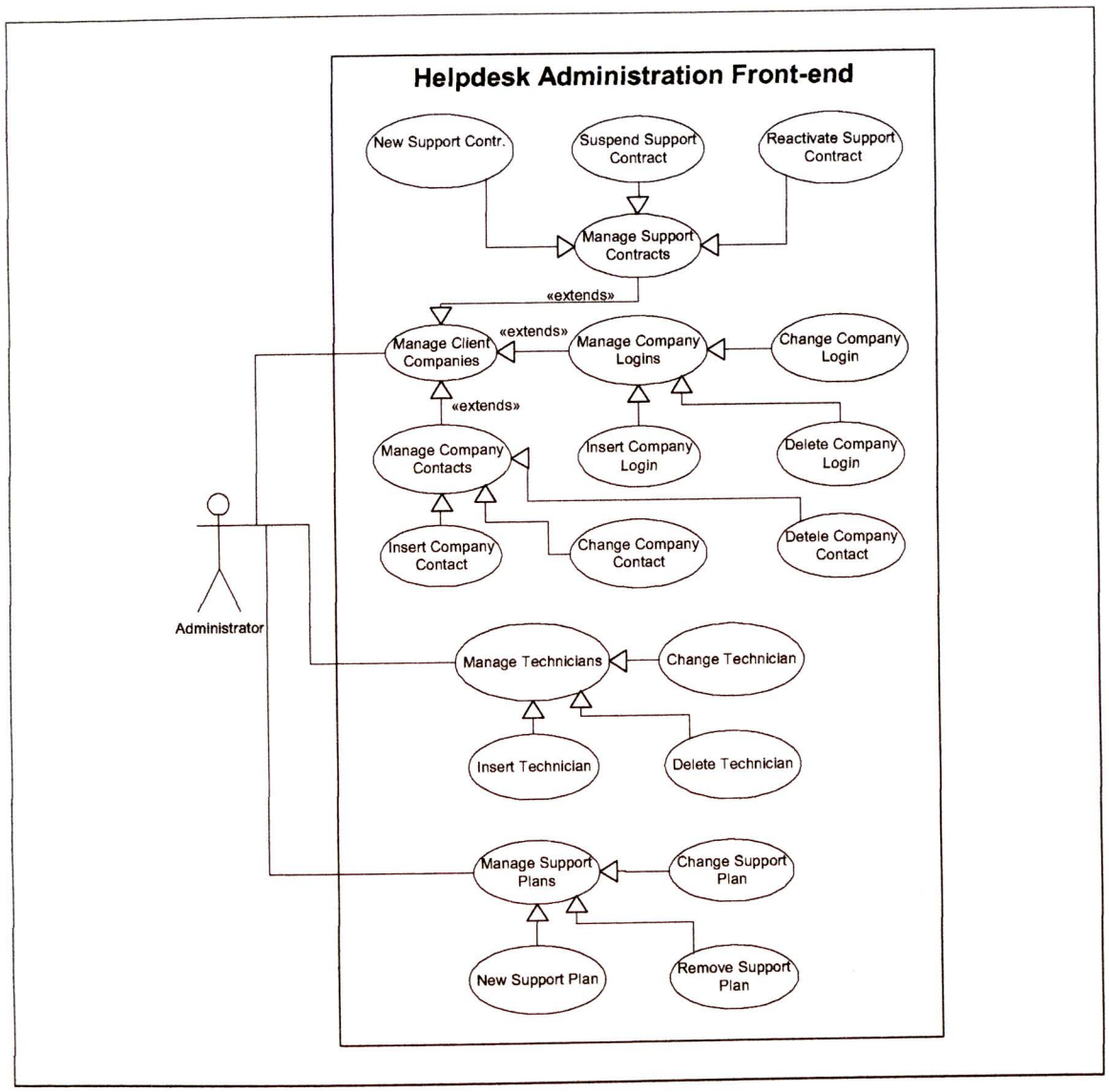


Figure 2-5 – Administration Front-end Use Cases

The administrator seems a busy person. As the actor that controls all the information that supports the functioning of the helpdesk system, he or she has a lot of tasks that must be supported by the system, more precisely, the Administration Front-end. Fortunately, he or she is not expected to perform these actions as often as the technicians perform theirs, for example.

Fundamental for the system is the management of clients and their logins and contacts. The administrator must be able to insert, update and delete companies. For each client company, the administrator must also be able to add, change and delete logins and contacts. Client company logins are pairs login / password that the personnel from that company use to gain access to the Imagine Broadband helpdesk system. Client company contacts are contact information that can be associated with a trouble ticket. Initial responses and update

information are sent to this contact information. For example, one client company has decided to ask for only one login and password to access the Helpdesk system, for simplicity. Nevertheless, it does not want that trouble ticket responses and updates are sent to all its personnel, but only to the employee that inserted the trouble ticket. This way, we'll have one company login and several company contacts, one for each of its employees that are responsible for submitting issue requests to Imagine Broadband. For each client company, it is also necessary that the administrator is able to manage the support contracts that such company has signed with Imagine Broadband. The system must permit him or her to insert new contracts, as well as suspend and reactivate them. Contracts must be inserted before the date from which they are valid. The system is responsible of controlling its state (pending, active, expired), based in the current date and the directions given by the administrator.

The Helpdesk Administration Front-end must offer the administrator the possibility of managing technicians and their related information, like their login and password.

Support plan management must also be available to the administrator. He or she should be able to insert, modify and remover support plans, but the system must ensure that no active support contract with any company is impacted.

2.3 Design of the Helpdesk Set of Applications

In this section, we present the designed solution for the required system that was described in section 2.2 to support the processes presented in section 2.1.

Support Technologies and Methodologies Used

The Helpdesk system should be accessible by all the clients of Imagine Broadband, who can be in any part of the world. Also, the technicians should not have physical location restrictions to access the system. Only the internet and the web paradigm provide this freedom of access. It was clear that the front-ends should be websites, which operate on a database management system server.

It was initially decided that, in the first phase, a third-party tool would be used to manage the trouble tickets, namely Keystone. Keystone is a web application that manages trouble tickets and technicians. Although it supported client contacts, it made no distinction between them, so a client would always see all the tickets in the system, for all the companies. Nevertheless, it was a helpful tool and the initial planning consisted in developing the Client Front-end while the technicians and administrators used Keystone. The Keystone relational database model would also be the starting point for the Imagine Broadband Helpdesk relational database model. The IB model would maintain backward compatibility so that Keystone would not stop working. As we will see later, the expectations were exceeded and Phase 1 comprised fully functional Technician and Administration Front-ends that relegated the use of Keystone to ticket assignment[4] and occasional control checks on the data. Keystone has been developed in the PHP scripting language and can be configured to use the most common Relational Database Management System (RDBMS), like Oracle, MySQL and PGSQL.

[4] Phase 2 of the Helpdesk System development, which is completed, already provides ticket assignment functionalities.

Like Keystone, the Helpdesk Front-ends were also to be developed using the PHP scripting language. For the first phases, MySQL would the RDBMS to use. The HTTP server would be Apache.

A small overview on the key advantages that these technologies offer to our particular case is given in section 2.6.

System Structure

As it was advanced in section 2.2, this system was idealized to consist in three main separate modules that act upon the same data. These components provide different functionalities and each of them is suitable for each one of the three identified types of users. They are nothing more than three front-ends to the system. The following diagram illustrates the system structure:

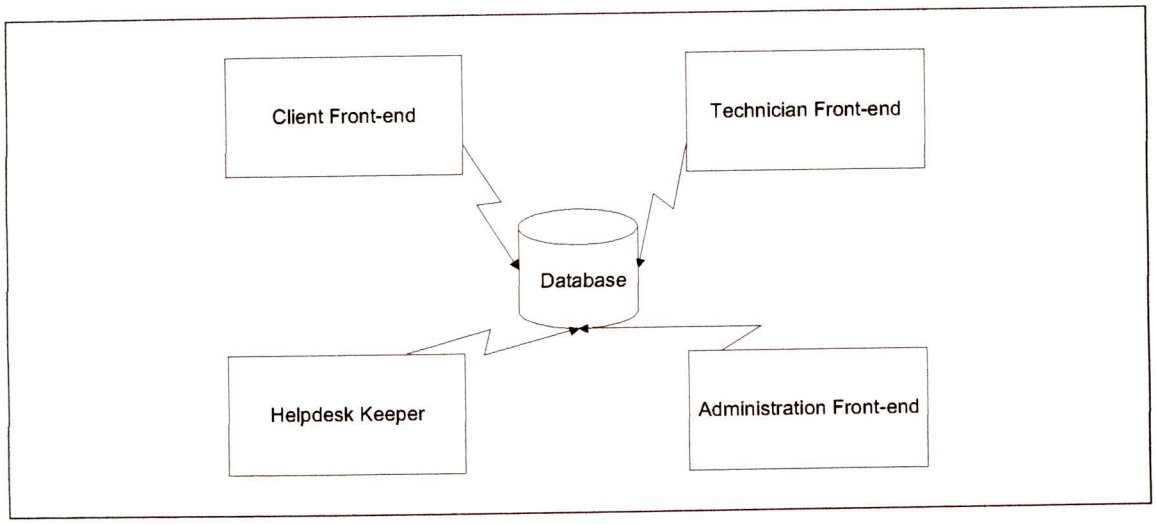


Figure 2-6 – The Imagine Broadband Helpdesk Structure

In addition to these three modules, which are operated by human users, an extra small (but important) component has been idealized. This module, which has been named the "Helpdesk Keeper", is independent of any user interaction and is responsible for the alerting system and further database manipulation. It periodically checks the next dates of each ticket's response or update events and eventually raises alerts in the database. These alerts will then be "read" by the Technician Front-end and presented to the technicians. It also keeps the status of all the support contracts (pending, active, expired, etc.) updated.

Logical Architecture

The four system components described are then well defined, separated applications. The only common aspect in their operations is that they are executed over the same data. That's how they cooperate to form the whole system functionality.

The package architecture of the front-ends is identical:

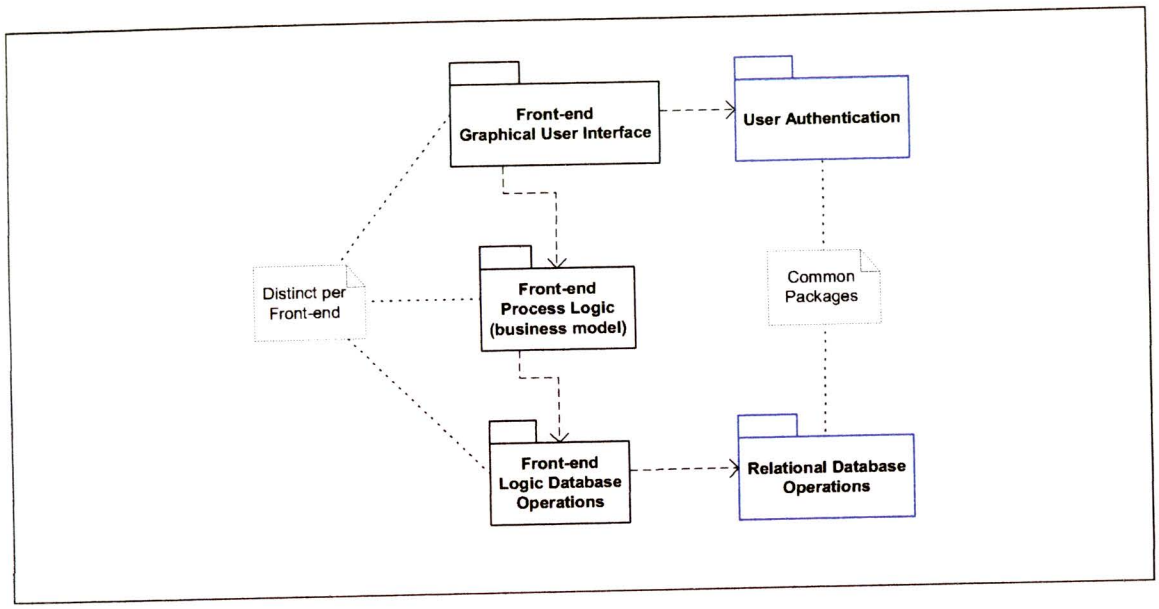


Figure 2-7 – Package Diagram of each Front-end Application

Each front-end has its own graphical user interface. It is the most exterior part of the system, with which the user interacts. As each front-end comprehends distinct functionalities, it is natural that their user interfaces are also different. In the same way, each front-end has its very own process logic, as they support different sub-processes or phases of the Helpdesk Support process. Each front-end deals with different data entities, besides the common ones. Thus, it also contains different logical database operations (for example, the Technician Front-end has the logical database operation "insert phone call log", which obviously is not needed nor used by the Client Front-end, as client users don't have access to the Helpdesk Phone Call log). The common packages are the "User Authentication" and "Relational Database Operations". These packages provide common functionalities that are used by all the components of the system. Operations like authenticating a user or updating a table row in the database are valid and independent from the process logic or goal of an application.

The package architecture of the Helpdesk Keeper is a reduced version of the front-end's one, like the following figure shows:

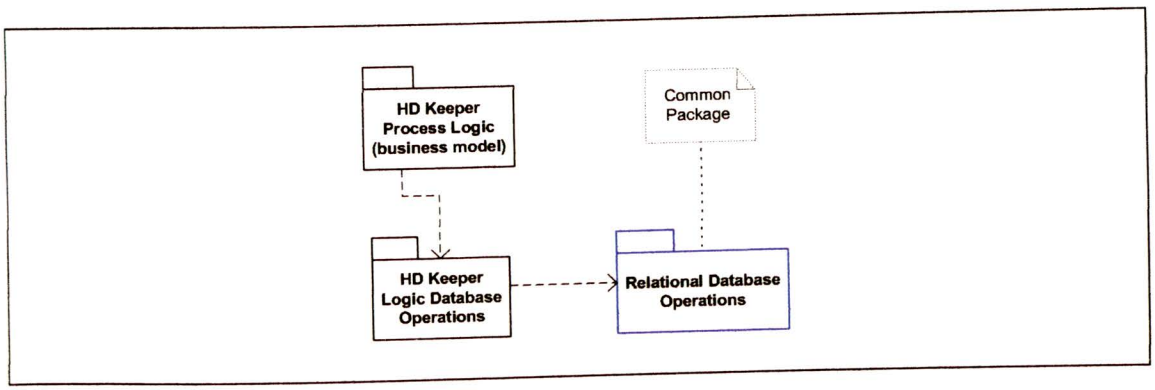


Figure 2-8 – Package Diagram of the Helpdesk Keeper Module

Next, we increase the level of detail and explore the contents of the presented packages. The packages are tightly connected, which makes a presentation of class diagrams per package

unadvisable, as we would be missing the many connections between them, which are essential to the understanding of the system. Also, presenting the class structure by module is not suitable in this case, as it is similar in all of them. Concluding, the best way to present the class structure of this system is by means of a single diagram containing all the classes of the system domain, while marking the package to which they belong. Here it is:

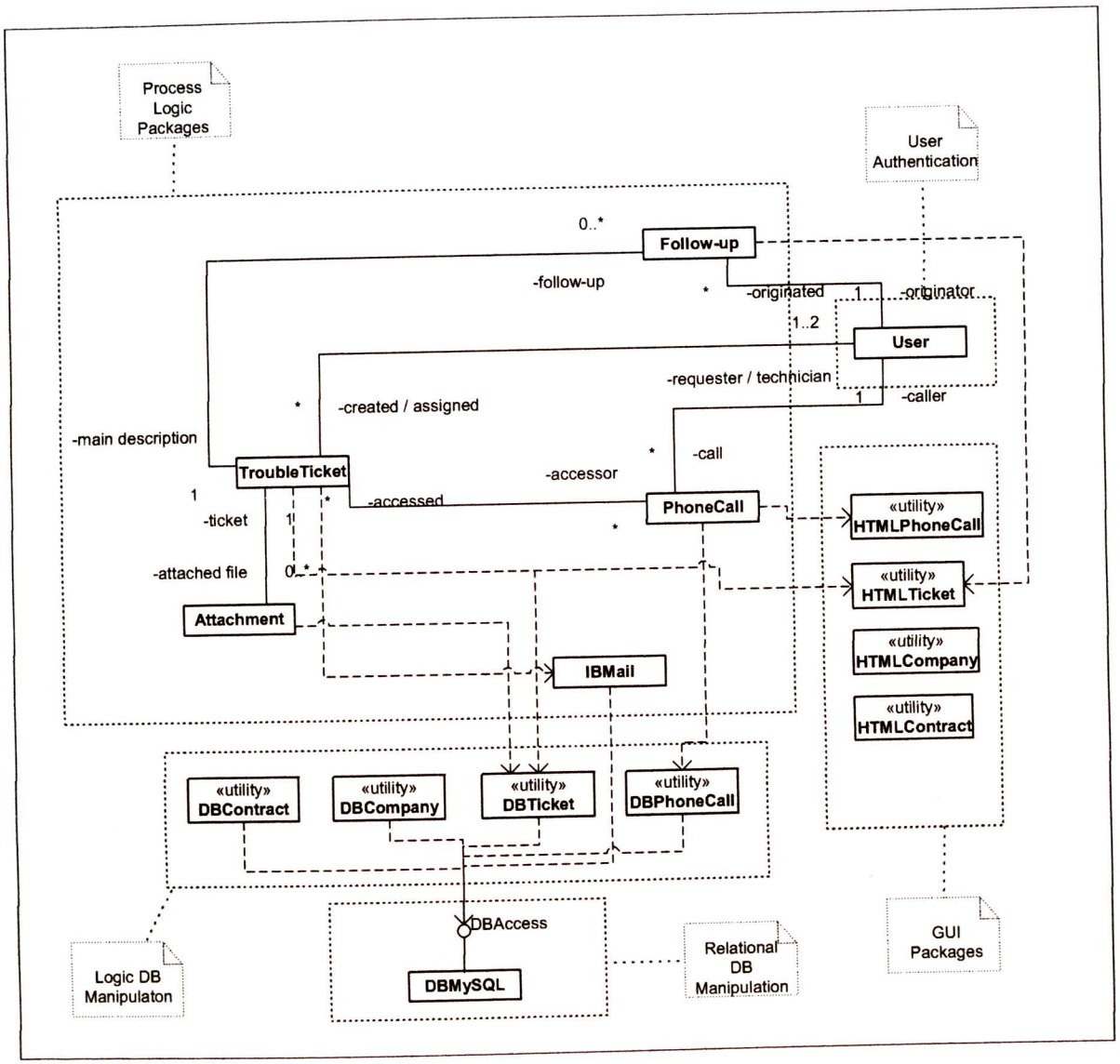


Figure 2-9 – Helpdesk Class Diagram

To keep things simple, class attributes and methods have been suppressed in this diagram.

The most important entity is the "TroubleTicket". This class models the trouble ticket concept, the information it holds, like the problem description for example, and process logic associated, as the update of its information and its consequences, like sending an email to the update requester. Each trouble ticket can have associated file attachments. These are modeled by the class "Attachment", which holds the filename and its URL. A trouble ticket may also have follow-ups, which are modeled in the system by the class "Follow-up", containing the follow-up text and a reference to the user that created or originated it.

The class "PhoneCall" models real support phone calls that are answered by the technicians. It contains several information, like the summary of the call and a reference to the caller user (every caller must identify him / herself as a contact that exists in the Helpdesk system database). Also, it refers the tickets that were consulted, edited or inserted following the caller directions. This is important for keeping logs of all the calls and what changes they originated in the system.

The class "IBMail" implements and encapsulates the Helpdesk mailing functionalities. There are several events that trigger the sending on an email. For example, when a client user creates a new ticket, a confirmation email is sent to its email address and a "new ticket" email is sent to the Helpdesk email address. Similar mails are sent when a user updates a ticket. Also, when alerts are raised, the technicians and the Helpdesk email address are automatically mailed. This class encapsulates not only the action of sending a mail but also higher level functionalities, like getting a mail template from the database and updating it with real values before sending.

All kinds of users are managed using the class "User", whatever their type (client, technician or administrator) is. This class contains all the common information about a user of the system, like its login, name and contacts. It also contains validation and authentication functionalities, to restrict the access to the Helpdesk system to authorized users only.

An object is independent of any representation and storage method that may be used to visualize and record it. For example, a dog and a photo of that dog are two different things. The process of developing the photograph of a dog is not part of a dog's definition. Following this logic, efforts were made to isolate the objects definitions, in the Helpdesk system, from the way they are presented to the user (HTML) and the way they are manipulated in the database. For this purpose, the *utility* classes in the package "GUI" and "Logic DB Manipulation" were created.

The classes that compose the "GUI" packages of the modules of the system take care of the visual presentation of the system data entities in a web browser. Also, they provide visual forms that enable the users to manipulate or enter new data entities to the system, like tickets. "HTMLPhoneCall" provides HTML facilities to present phone call information and the phone call log, as well a form to insert a new phone call entry. "HTMLTicket" offers methods for presenting tickets and all their related information in an HTML page. Forms for inserting, editing and searching tickets are also available. Additionally, this *utility* class has several methods for filling in visual combo boxes with trouble ticket related options, like severities and status, for example. Company related visual combo boxes are filled by methods of the class "HTMLCompany", whereas "HTMLContract" provides methods to fill support contract and plans related combo boxes.

Logical operations in the database, like "inserting a ticket" or "finding a valid support contract for some client company" are implemented by the *utility* classes in the "Logic DB Manipulation" packages. They are used by the business logic packages classes, which depend on them. "DBContract" provides database operations related to support contracts. "DBCompany" supports operations related to companies and the contacts of their employees that interact with the system. "DBTicket" provides the whole lot of ticket related database operations, like inserting, updating or searching for tickets as well as follow-ups and attachments. The "DBPhoneCall" *utility* class comprehends phone call manipulation in the database, like insertion, fetching and searching facilities.

These classes that compose the "Logic DB Manipulation" packages isolate the objects they manipulate from their database representation, as referred. They act upon real database instances, performing a set of real database operations to achieve the desired logical action. However, one extra level of isolation from the database itself was needed, for the system to be independent of the particular RDBMS used. This independence was achieved by designing an interface that defines the relational database operations that a database access class must implement. This way, the program may use any class that implements this interface and does its job with any RDBMS. Like this, in the event of having to change the RDBMS used, the only task that needs to be performed is the creation of the proper database access class that must conform to the designed interface. This change is transparent to the logic part of the system. The only database access class that was developed works with the MySQL RDBMS, which was the selected one to support the Helpdesk data in phase 1[5]. The database access class to MySQL RDBMS's is the "DBMySQL" depicted in the diagram belonging to the "Relational DB Manipulation" package. It provides functionalities like connecting to a database server, selecting a database instance and performing the most common relational database operations.

The current logical structure of the Helpdesk system, that has just been presented, has yet several aspects that should be improved. As advanced in section 1.2, harsh initial time restrictions prevented a good and thoughtful system design. Also, the current structure is the result of later efforts to structure the system, considerable time after the beginning of its development. Some improvements that are in order are referred in section 2.5.

Physical Structure

In this section, the physical structure of the Helpdesk set of applications is presented.

As web applications, the Helpdesk tools require an HTTP server running with the PHP scripting language interpreter. The system also needs a database server. Presently it uses a MySQL RDBMS. Here's the deployment diagram of the Helpdesk:

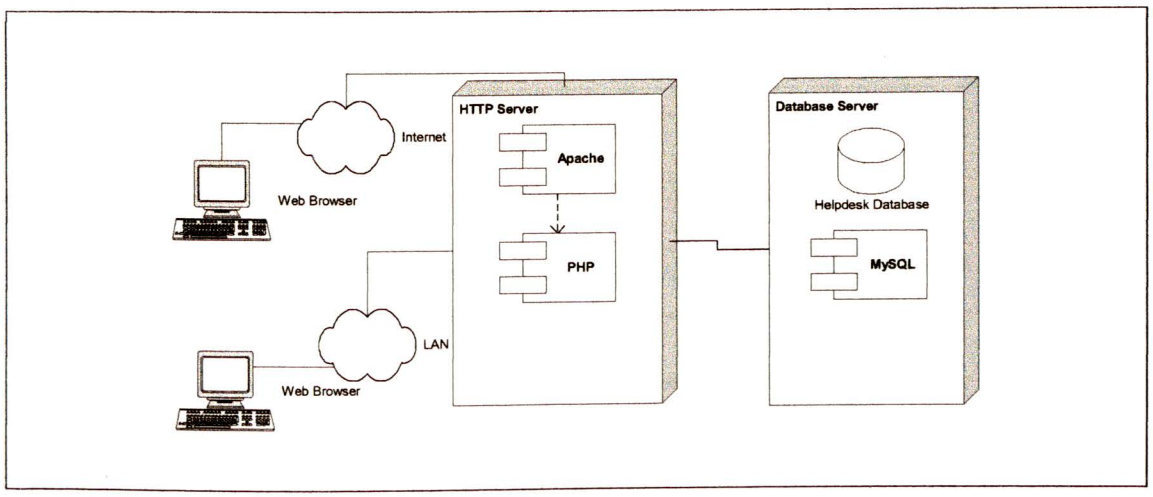


Figure 2-10 – Deployment Diagram of the Helpdesk Applications

[5] There are already plans to change the RDBMS used to Oracle in the future. This database isolation characteristic of the system design will surely prove itself advantageous by that time.

As can be seen in the diagram, two servers are needed: one HTTP server and a database server. Note that these two server instances may be deployed in the same physical server, as they are at the present in the Imagine Broadband production environment. The Helpdesk components must be accessible by the HTTP server so they should be deployed in the same physical machine. The same goes for the database instance, which should be deployed in the same physical server as the RDBMS server. This may not be a restriction, but it surely benefits performance. The system may then be accessed with a web browser, from any location using the internet. Technicians and administrators will most often use the local area network to access the system, when at office.

Having presented the deployment architecture of the Helpdesk applications, we now proceed to their component structure. The following component diagram is an overview of the component structure of the Helpdesk applications.

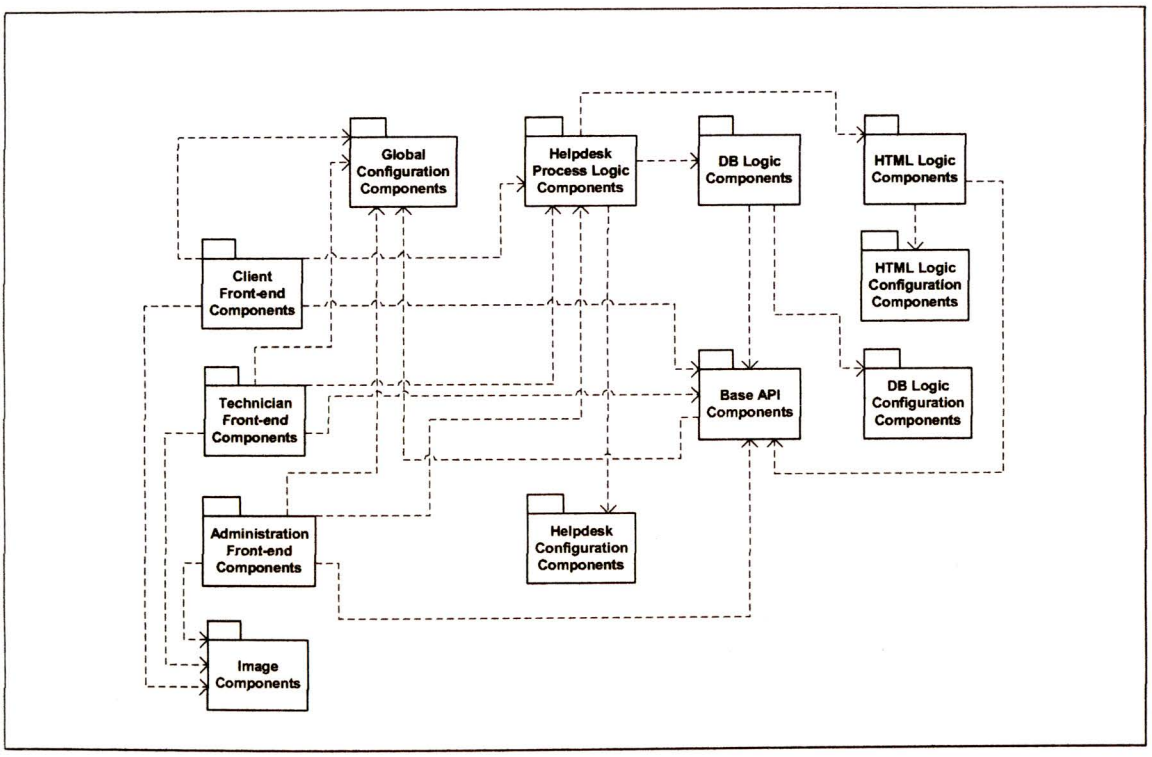


Figure 2-11 – Package Component Diagram of the Helpdesk Applications

The components of the system are grouped according to their functionality and nature. There are the components of the front-ends, which are called directly by the web server. Those components may require image components for the graphical interface and surely require common configuration components, process logic components and eventually database and HTML logic components for the system to work as planned. These last two packages are most commonly used by the process logic components. The process logic components get configuration information from the Helpdesk configuration components (that are specific to the Helpdesk system, where the global configuration components define parameters common to any website). The logic operations components use, as expected, the basic API components, like the database access object and HTML functionalities, among others.

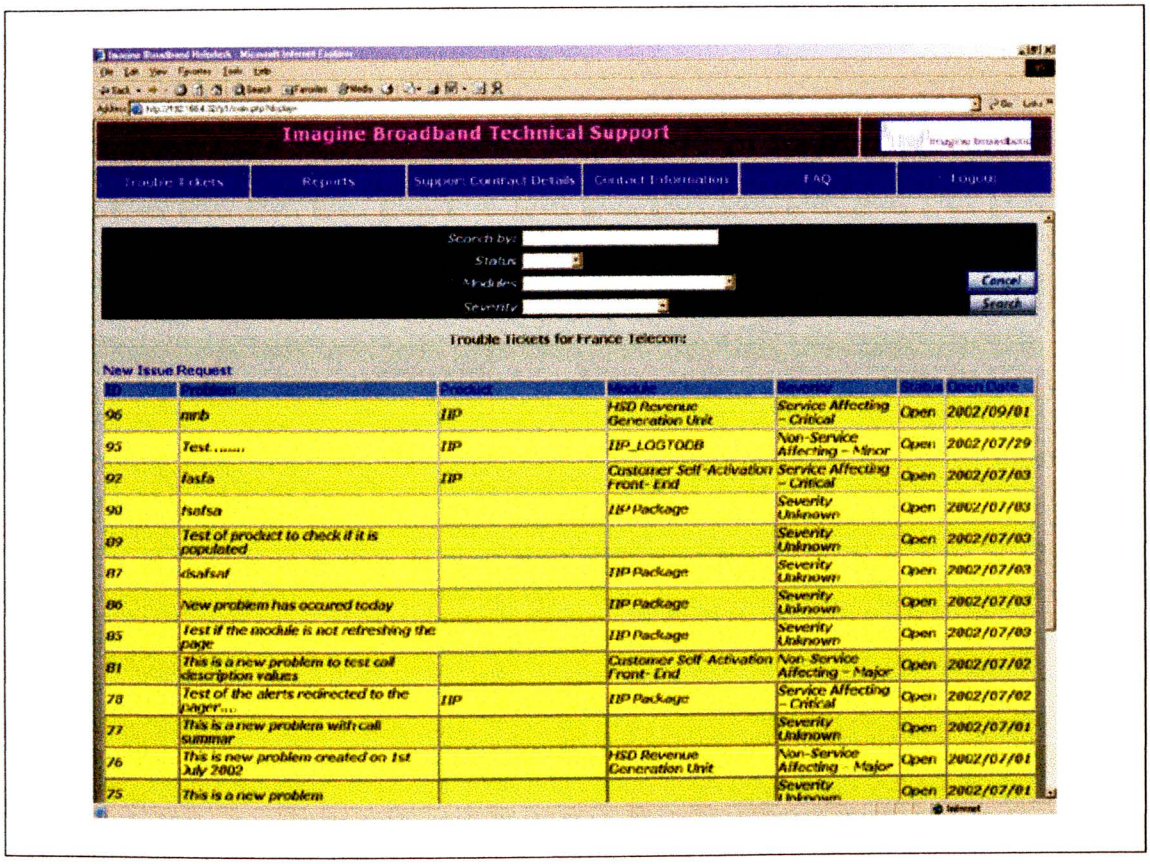
2.4 The Implemented Products

In this section, the developed applications are presented. For each front-end, the most important algorithms are described as well as screenshots of their graphical user interface.

Client Front-end

The Client Front-end application was developed to enable client users to interact with the helpdesk system, that is, insert trouble tickets, follow their evolution and have access to several kinds of information. The requirements of the Client Front-end are specified in section 2.2.

The following image is a screenshot of the main area of the Client Front-end website:



ID	Problem	Product	Module	Severity	Status	Open Date
96	mrb	IIP	HSD Revenue Generation Unit	Service Affecting - Critical	Open	2002/09/01
95	Test.....	IIP	IIP_LOGTODB	Non-Service Affecting - Minor	Open	2002/07/29
92	fasfa	IIP	Customer Self-Activation Front-End	Service Affecting - Critical	Open	2002/07/03
90	fsafsa		IIP Package	Severity Unknown	Open	2002/07/03
89	Test of product to check if it is populated			Severity Unknown	Open	2002/07/03
87	dsafsnf		IIP Package	Severity Unknown	Open	2002/07/03
86	New problem has occured today		IIP Package	Severity Unknown	Open	2002/07/03
85	Test if the module is not refreshing the page		IIP Package	Severity Unknown	Open	2002/07/03
81	This is a new problem to test call description values		Customer Self-Activation Front-End	Non-Service Affecting - Major	Open	2002/07/02
78	Test of the alerts redirected to the pager....	IIP	IIP Package	Service Affecting - Critical	Open	2002/07/02
77	This is a new problem with call summar			Severity Unknown	Open	2002/07/01
76	This is new problem created on 1st July 2002		HSD Revenue Generation Unit	Non-Service Affecting - Major	Open	2002/07/01
75	This is a new problem			Severity Unknown	Open	2002/07/01

Figure 2-12 – Trouble Ticket Listing in the Client Front-end Website

In the upper menu bar, we can see all the functionality groups that the site offers. Trouble ticket related operations, reports visualization, support contract data, additional support contacts and a FAQ section.

The trouble ticket and the report sections required most part of the work spent on the Client front-end. Next, we present the most important algorithms of the client front-end, starting by the trouble ticket insertion.

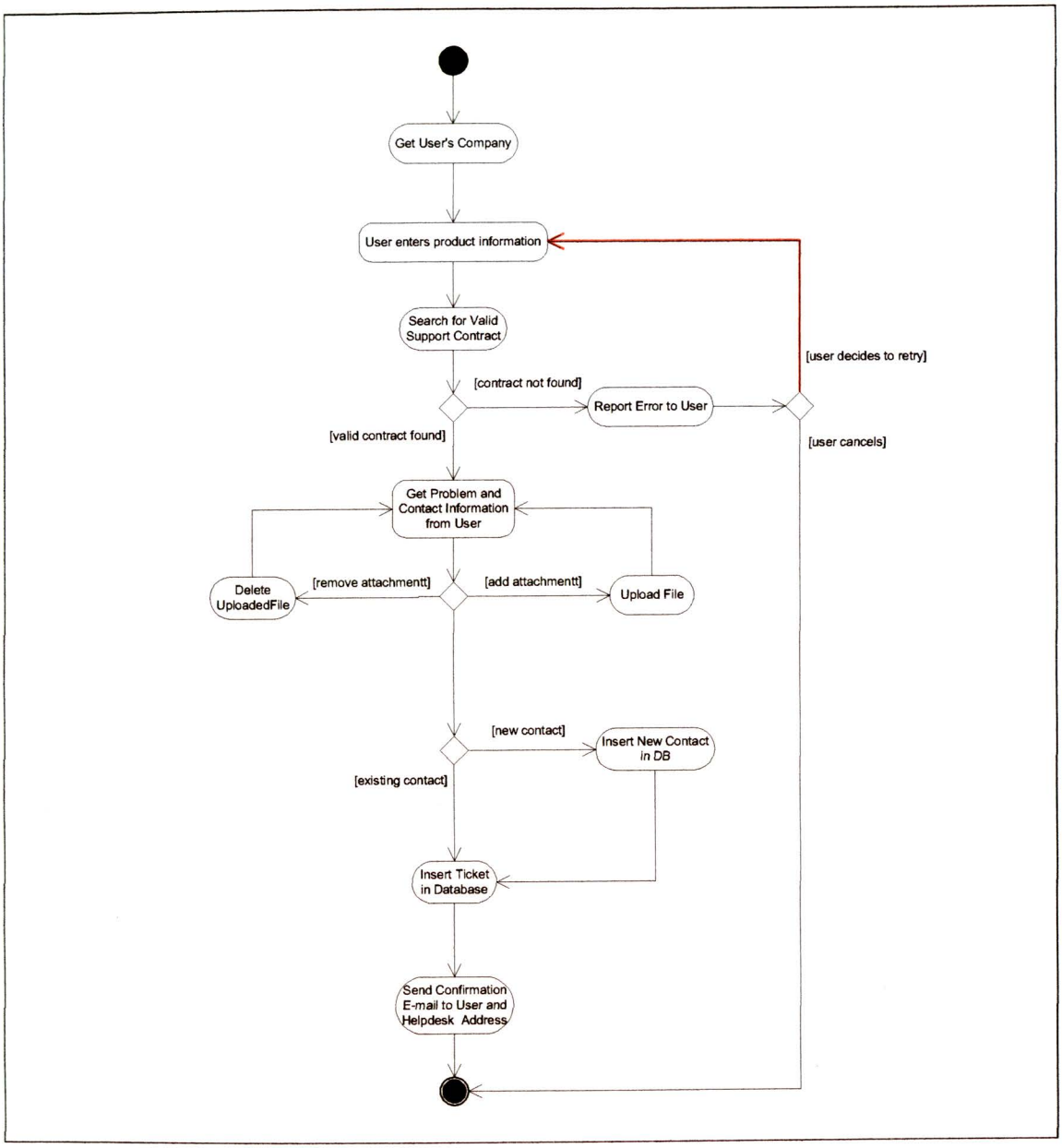


Figure 2-13 – Activity Diagram for Trouble Ticket Insertion

When inserting a new trouble ticket, the client user must start by indicating the product to which the new ticket refers. The company information is automatically retrieved by the login used. Then, the system must check if the company of the user has a valid support contract for the product in question. This is done by querying the database, searching all the active support contracts of the company for one that includes the referred product. If no contract has been found, then the user is not allowed to proceed in the new ticket insertion process. This fact is reported and the user may choose between specifying another product and canceling the process. On the other hand, if such a contract is found, then the user may continue with the process, indicating the problem information and eventually adding file attachments. To finalize the process, the user specifies its contact information, choosing from a list of contacts for his or her company or adding a new one. Further information requests and updates in the tickets will be notified to the contact associated with the new ticket. In the end, the ticket is

inserted in the database, a confirmation mail is sent to the contact that just inserted it and one warning email is sent to the Helpdesk email address.

The trouble ticket updating algorithm is similar to the trouble ticket insertion one. The only differences are that, in the edition process, there is no need for contract validation and a follow-up is created automatically reporting that the user has updated the changed trouble ticket fields. The exception to this automatic follow-up procedure is when the user itself adds a follow-up. In these cases, there's obviously no need to create an automatic follow-up reporting that the user added a follow-up to the ticket.

The development of the usage reports section was also an interesting challenge. Two types of reports were required - one that showed the amount of active trouble tickets by severity and status, and other that consisted in line and bar graphs showing the total open issues by severity over time. If the first was a matter of creating database queries with dynamic restrictions (the user should be able to filter the results by product and time periods), the second was peculiar in web development as drawing decent line and bar graphs required dynamic image creation.

Next, the algorithm for graph drawing is presented. For the actual rendering and creation of the image, the 2D graphics library GD has been used. The PHP scripting language provides functions to control GD and this library produces images in the PNG format (among others), which is indicated for web development.

The following figure depicts the graph drawing algorithm:

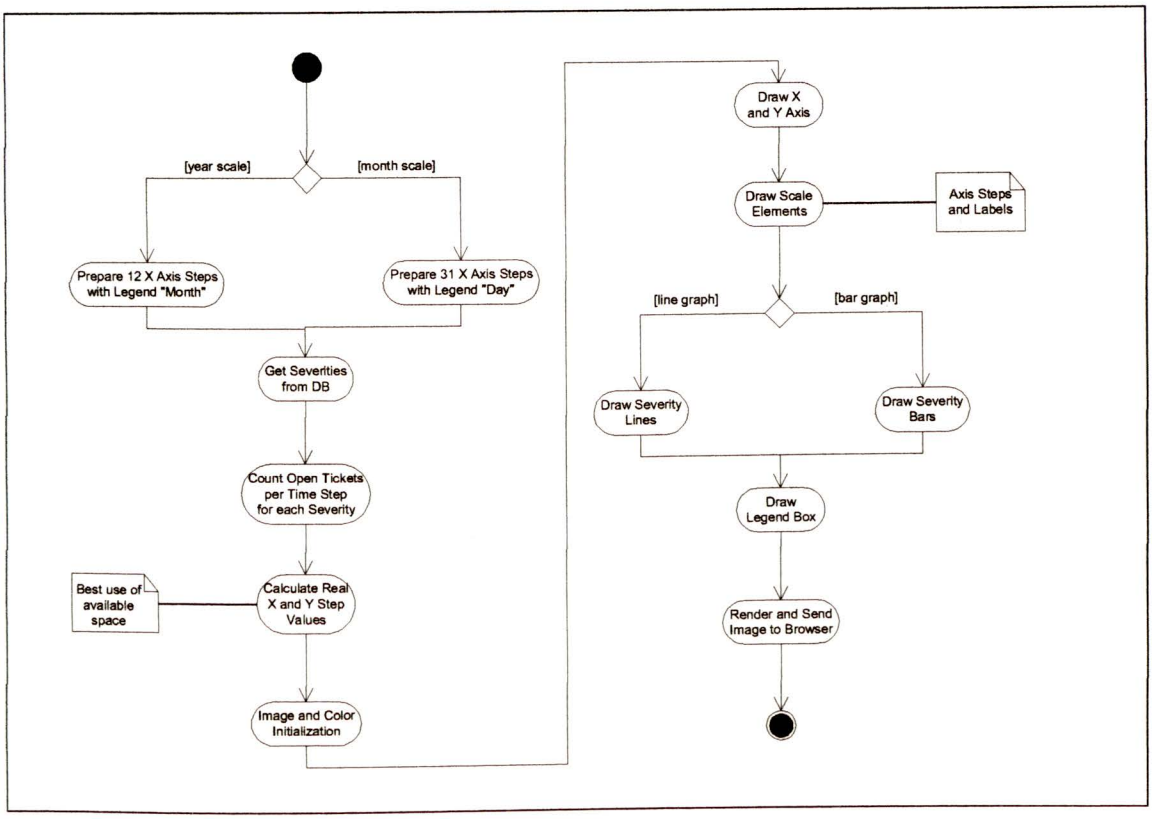


Figure 2-14 – Graph Drawing Activity Diagram

For this algorithm to work, important functions have been created. One of them converts logic coordinates into real image coordinates. It does so by multiplying the logic value by the respective axis' real step value. Other functions include drawing lines and bars. These functions take an array of Y values and draw a line connecting them or bars for each one of them. The next image is a sample of a graph:

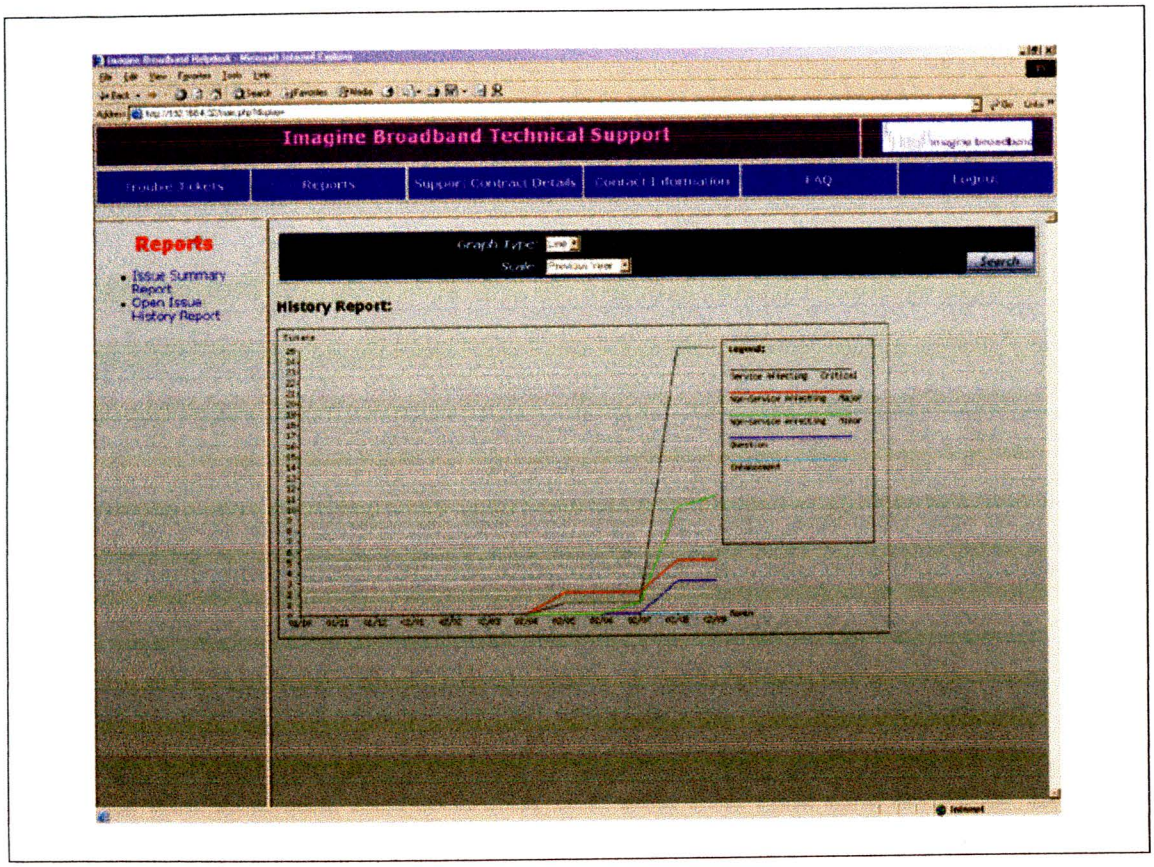


Figure 2-15 – Screen Shot of the Graph Report Page

Technician Front-end

The Technician Front-end provides technician users with the ability of updating tickets in their lifecycle, register and consult support phone calls and keep an eye on the system alerts.

The following image shows one of the sections of the Technician Front-end:

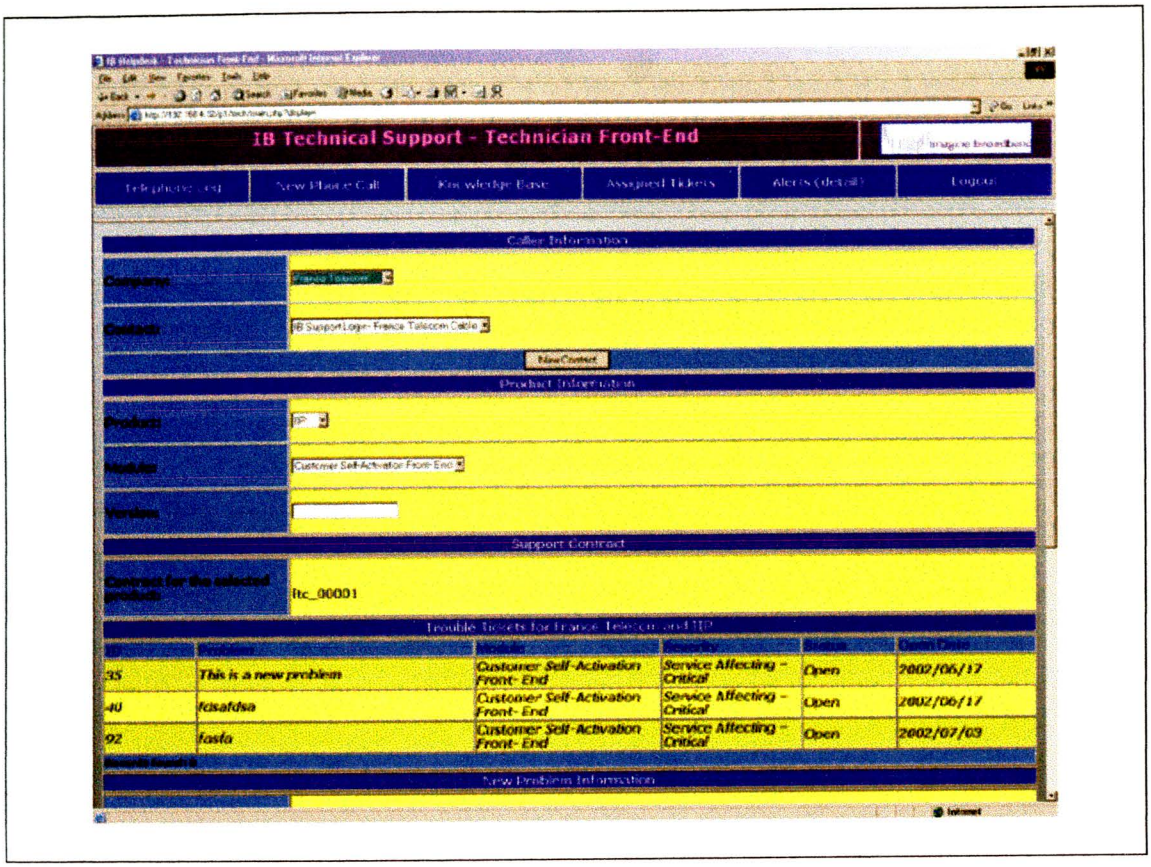


Figure 2-16 – New Phone Call Page at the Technician Front-end

In this application, there is one process that is worth describing – the new phone call algorithm. This functionality of the system is critical, as the technician user will be interacting with the caller and with the Helpdesk system simultaneously. Therefore, it was designed to be the most direct and easy to use possible, combining several functionalities in one single page. The layout of the graphical components of the page follows the order by which the call should be conducted by the technician. The next activity diagram shows how the system behaves when inserting a new phone call log entry.

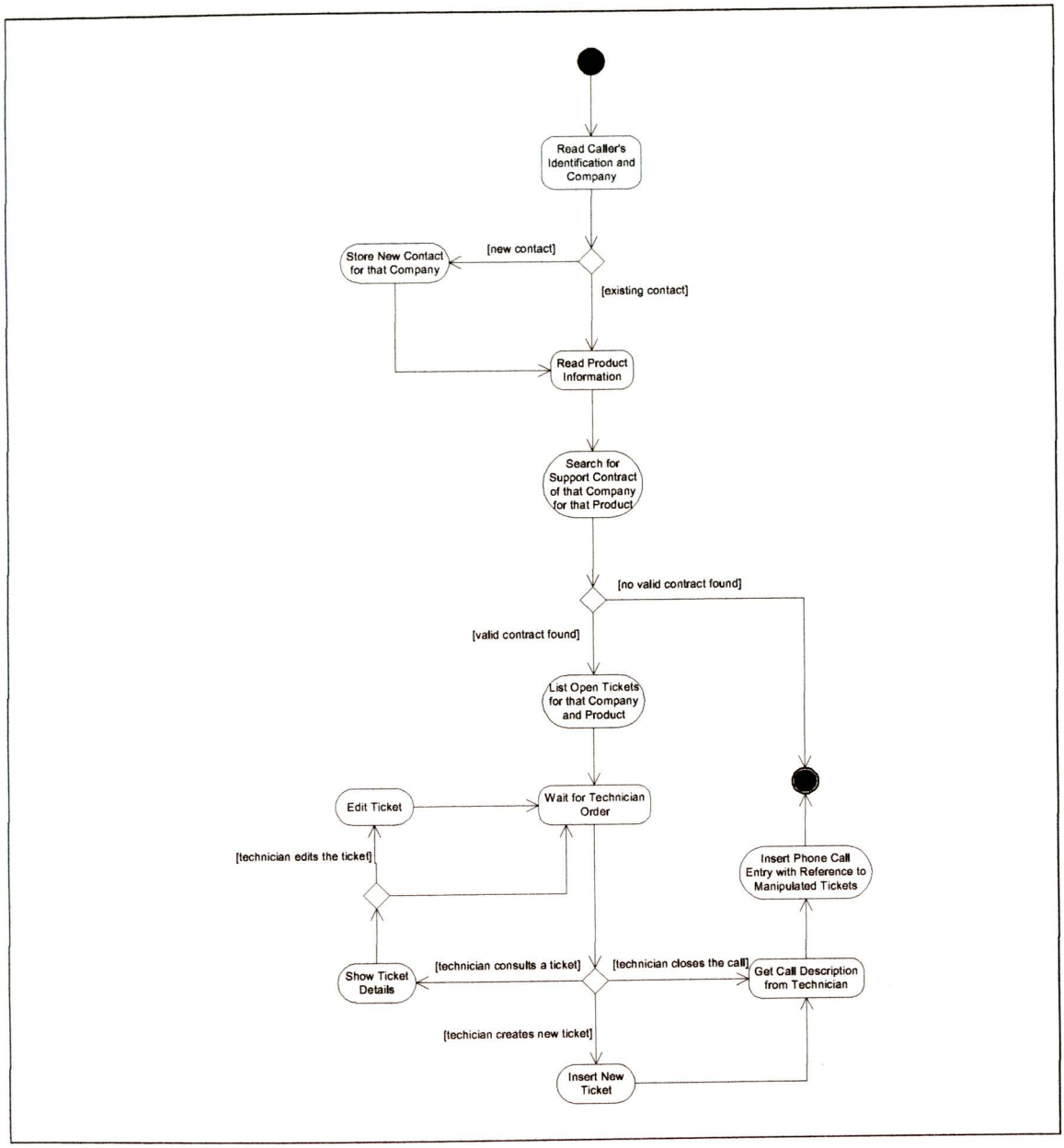


Figure 2-17 – New Phone Log Entry Activity Diagram

Administration Front-end

In the Administration Front-end, the administrator user is able to perform all the administrative tasks required by the Helpdesk system, like support plan and contracts management, company and client logins management, technicians, etc.

There is no special procedure or algorithm to highlight in this front-end. It all boils down to data management and database operations. The following image shows one section of the Administration Front-end.

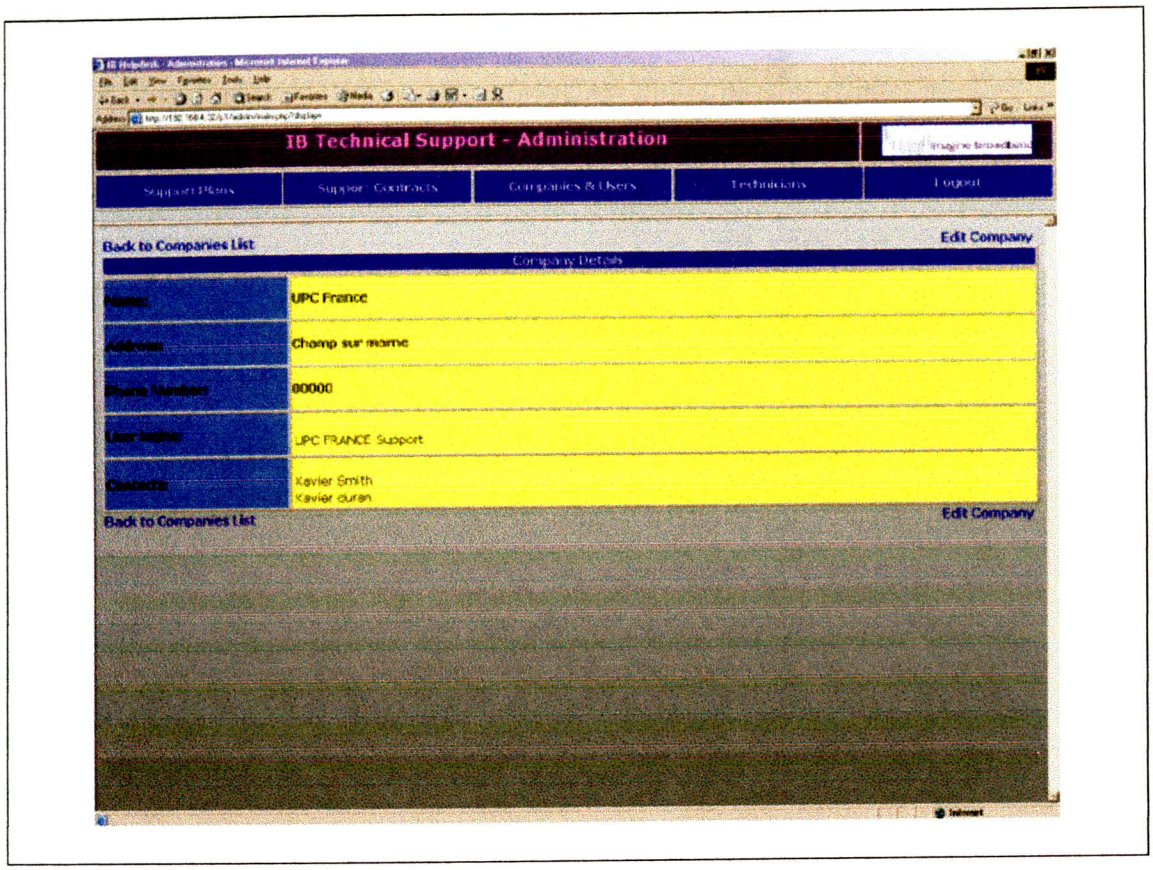


Figure 2-18 – Screen Shot of the Administration Front-end

One important fact, however, is that this Administration front-end, as with the Technician Front-end, exceeded all the expectations for the first phase of the Helpdesk development, providing all the functionalities needed by the Helpdesk system administrators, that were first supposed to be trusted to the Keystone issue tracking tool. With a quick and effective development, the Administration Front-end relegated the use of Keystone to some occasional data control checks, i.e. as a test tool.

Helpdesk Keeper

The Helpdesk Keeper maintains the contracts' status updated and raises alerts when a response or update event date for some trouble ticket is near. In these events, Imagine Broadband must provide some information about the status of the trouble ticket to the client. This application is run periodically (currently once per minute). The following picture describes the functioning of the Helpdesk Keeper each time it is run.

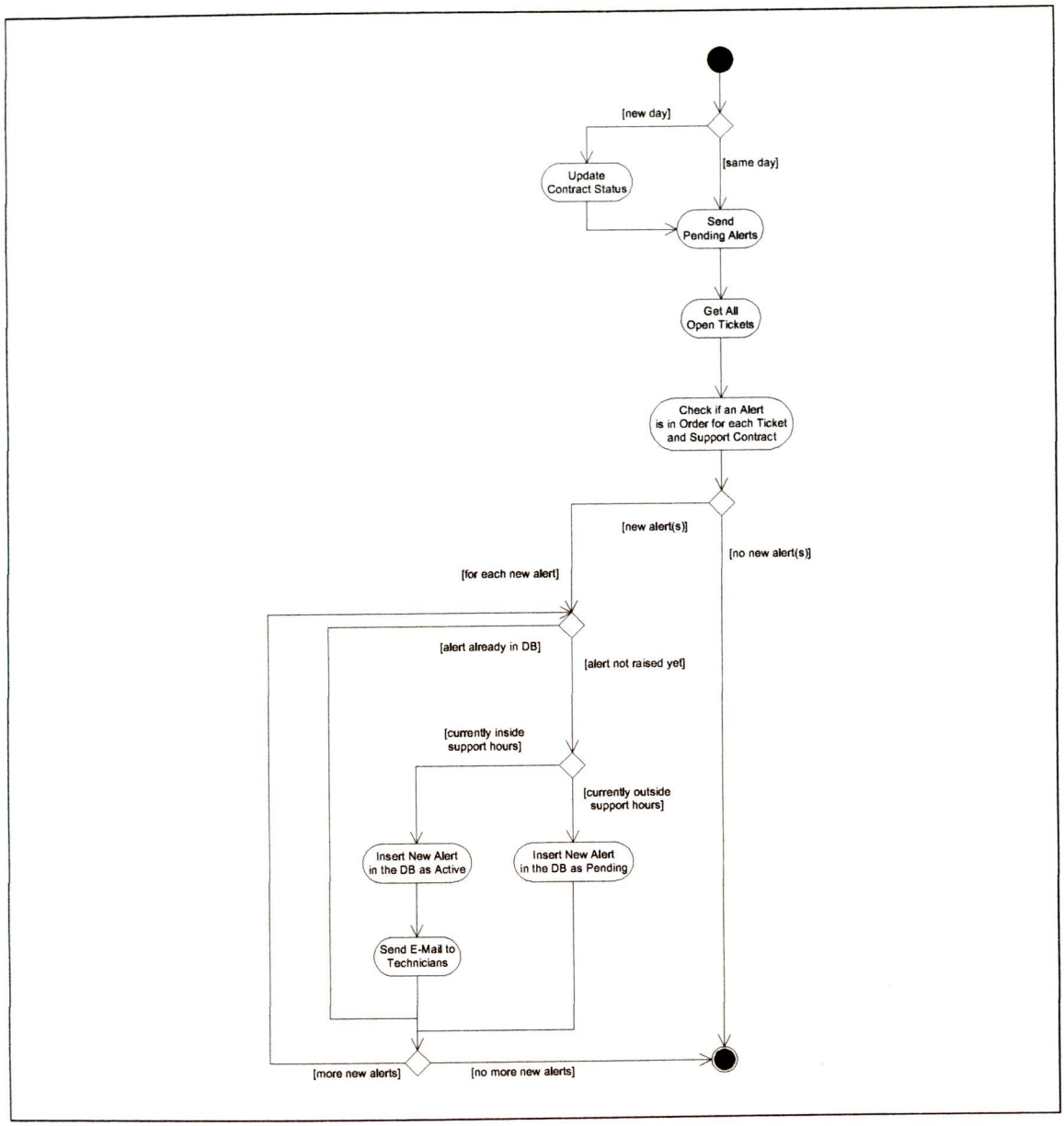


Figure 2-19 – Helpdesk Keeper Activity Diagram

The contracts' status should only be updated once per day, as their start dates and durations specify days and not detail into hours or minutes. Alerts should only be effectively raised and dispatched (by email to the technicians, for example) when we are at within support hours for the support contract associated to the ticket. If we're not, then the alert is created as "pending" and it waits until a new support period begins until being effectively dispatched.

Technicians will then receive emails (and in Phase 2, pager notifications) alerting for the proximity of a response or update event. They can also consult the active alerts in the Technician Front-end, where they can acknowledge them. That acknowledgement can also be performed directly by following a web-link present in the alert email.

2.5 Aspects to Improve

There is room for big improvements in two areas of the system. The system logical design and structure and the graphical user interfaces.

System Design

As it was first referred in section 2.1, serious initial time constraints mined a careful and thoughtful first design and structuring of the system. The urgent need for functionalities and met requirements rushed the implementation phase to start way before it should. With no time given to plan and design the system, it is no surprise that the current system presents flaws in this area. Besides, it would present even more flaws if it wasn't for some later efforts to structure the site and make it more maintainable, when the hunger for functionalities was being satisfied.

The system logical architecture can and should be improved a lot. If we look at Figure 2-9, we surely miss concepts such as "technician", "companies", "contacts", "support plans", "support contracts", etc. Concepts that exist in the real Helpdesk "world" and have not been correctly modeled in the Helpdesk support system. They are currently just groups of information that circulate through the system, without any kind of entity identification, behavior and other characteristics of an object. Creating such classes would simplify system understanding and improvements.

Graphical User Interfaces

The current graphical user interfaces of the three front-ends of the system are, as the reader could confirm from the screenshots presented, raw, unattractive and unprofessional. They sure are thought to be precise and user friendly, they do provide all the required functionality in an "easy to use" method, but they do it in an ugly way. Lots of improvements are in order here. In fact, there are plans since the beginning to improve the GUI's to match the Imagine Broadband's official website look and feel. However, there was never any specialized resource allocated for this task.

2.6 Technologies, Methodologies and Tools Involved

In this section, we list the key points of the technologies, methodologies and tools involved in this first project. The purpose is simply to describe their aspects or characteristics that were fundamental to the project.

PHP

PHP, which stands for "PHP: Hypertext Preprocessor" is a very efficient scripting language. It was born to allow web developers to write dynamically generated web-pages quickly. However, it has become so popular that it is used for several different purposes, including standalone applications. One of its key advantages, besides its robustness and efficiency, is the complete API that provides all kinds of functionalities for web development, like e-mail sending, database access and online payment function libraries.

In a PHP enabled web-server, the PHP interpreter is called by the HTTP server when a visitor requests a page. The PHP engine interprets (executes) the called PHP script and outputs the

result in HTML to the HTTP server that sends it to the client's browser. To make things easier for simpler tasks, PHP code can be embedded in the HTML code.

The reasons for choosing PHP for the Helpdesk Tools were not only the advantages referred above, but also the fact that the application that was planned to initially support the Helpdesk process, Keystone, was developed in PHP.

Official website: http://www.php.net

GD Graphics Library

GD is a 2D graphics library very popular for the UNIX community. It can draw many basic 2D forms and generates PNG and JPEG image file formats, which are very suitable for web development.

This library has been used to dynamically create the line and bar graphs in the "Historical Reports" section of the Client Front-end. The graphs are drawn, according to the data in the database, using the GD library to which PHP provides interface functions. After the image is drawn, a PNG image is created and sent to the client's browser via the HTTP server.

The GD library has been selected for this task due to the fact that PHP provides interface functions that make the use of this external library very easy. It has also proved to be a good choice by the functionalities it provides and by being simple to use.

Official website: http://www.boutell.com/gd

MySQL

The MySQL database server is proud of being extremely efficient, simple, robust and simple to deploy. It is "Open Source" and now supports a good set of standard database functionalities, among which is the support for transactions.

This RDBMS is being used to store the Helpdesk database. It was initially chosen as a good cost-free solution for the moderate requirements of the Helpdesk system. Posterior plans to switch to an Oracle server have appeared, but currently this is still the RDBMS in use

Official website: http://www.mysql.com

Keystone

Stonekeep Keystone is a web-based application that helps a development team to keep track of problems and issues in their systems. It has been developed in PHP and was initially planned to support the IB Helpdesk functionalities that were not provided in the first phase of the Helpdesk tools.

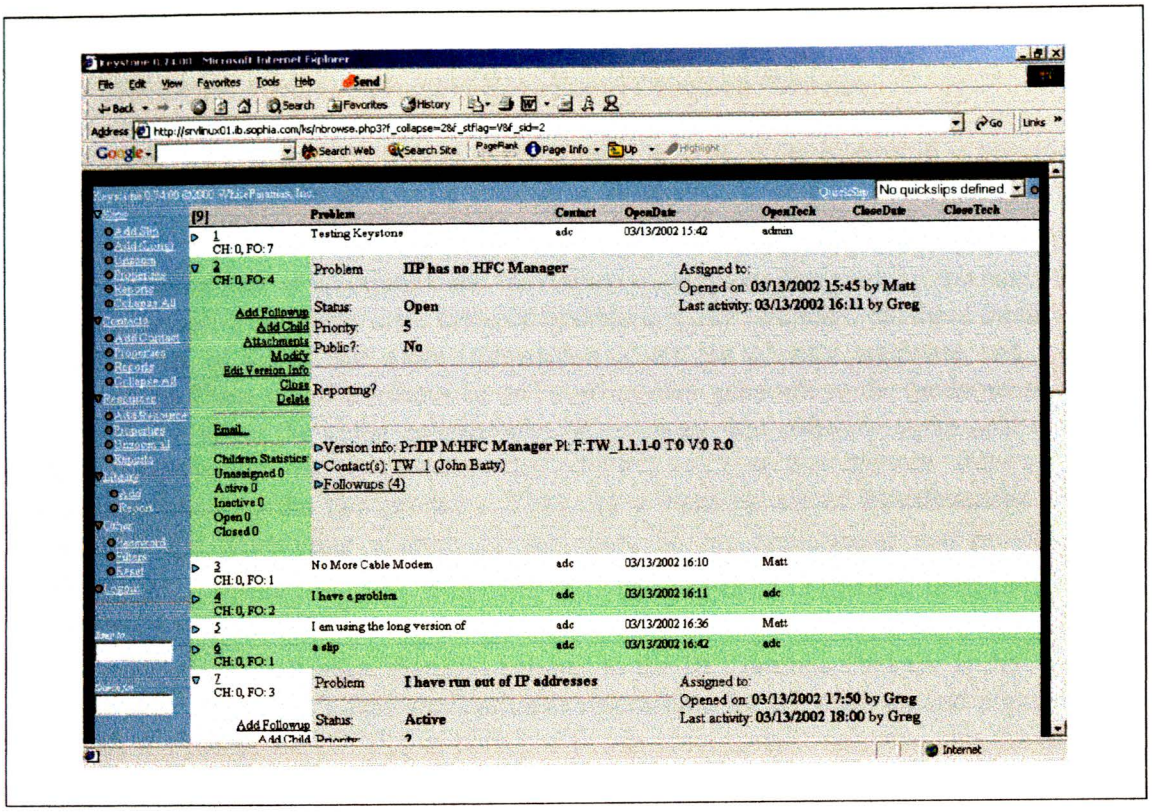


Figure 2-20 – Ticket View in Keystone

Official website: http://keystone.whitepj.net

3 The Second Project: Imagine Broadband Common Package

In this chapter, a detailed description of the second project of the internship is provided. An introduction to this project has been presented in section 1.2.

The Imagine Instant Provisioning (IIP) and Imagine Service Assurance (ISA) products provide different sets of functionalities. Nevertheless, they are designed for the same type of client environment, i.e. large scale network operators. Their relational database models share a common set of tables that store information about the client's hardware and software infrastructure. These tables are required and used by both products as the fundamental source of information about the client environment of hardware and software equipment and their state. It is then natural that the processes that populate and keep the common tables up to date are also required by the two products. They are a common set of functionalities that are essential to the functioning of both IIP and ISA. So the applications that provide those functionalities belong to the denominated "IB Common Package" and are called "Infrastructure applications".

Another important application that also belongs to the IB Common Package is the "Imagine OSS Database Populator". This application is responsible for the creation and population of the database instance for IIP, ISA or both (the two product specific schemas plus the common part). Unlike the Infrastructure applications, the IB OSS DB Populator is not run periodically but whenever a database instance needs to be created, initialized or changed. It requires user interaction to start and control the process. As it is a generic tool that is able to create and manage any database schema, it was included in the IB Common Package.

Also part of the IB Common Package is the Lease Server. The Lease Server acts as an intermediary between a Cisco Network Registrar (CNR) server, which acts as a DHCP server, and any client program that wants to query a CNR. Instead of caring about creating and processing DHCP packets, the client programs only need to send a request to the Lease Server and the latter will take care of the communication with the DHCP server.

3.1 Improving the Imagine Broadband OSS Database Populator

In this section, we start by describing the initial state of the application. Then we move to the specification of the defined improvements, their execution and the final results.

The Imagine Broadband OSS Database Populator

The main purpose of this program is to create a database schema instance and then populate it with initial data. It has been designed to be highly flexible in the database operations it performs and the source data it uses, so it offers a lot of additional functionalities, like being able to populate, delete or change only some number of tables.

Scripts

To maximize the flexibility of the operations to perform and the order by which they are performed, the IB OSS DB Populator provides a simple scripting language that allows the user to create scripts, which will then direct the application's operations. The user creates scripts and stores them in a text file and then the application executes the commands specified by those scripts.

These are the most relevant commands that the scripting language comprises:

- **CALL**, calls and executes another script;
- **DELETE**, deletes all the rows in a table;
- **ECHO**, prints the value of an environment variable to the *standard output* of the program;
- **EXTRACT TABLE**, reads from a table in a database and creates a text file with the contents read;
- **POPULATE**, populates a table in the database with the data found in a (specified) text file;
- **READ**, displays a dialog box asking the user to enter some value. The data from the user is then stored in an environment variable;
- **REM**, indicates that the current line is a comment and not to be processed;
- **SET**, stores a value in an environment variable;
- **SQLPLUS**, executes an SQL script file;
- **EXECUTEPLSQL**, executes a PL/SQL script file;
- **POPULATECLOB**, inserts the contents of a file in a column of type CLOB (Oracle specific column type for large amounts of data).

Scenarios

A "scenario" is a set of scripts to be run in a particular order. A scenario purpose might be "to create an instance of a particular database schema" or "to populate the database with the initial data for some company", for example. Instead of writing all the commands for these purposes in one script file, the user will usually divide the process in many scripts, having each one of them a sub-purpose, like "to populate the tables for some program module", for example. This way, the process is modularized and each part can be analyzed and executed independently. The scripts that compose a global purpose are then grouped in a "scenario". Physically, a scenario is a text file with the path to a script filename in each line.

Data Text Files

The data to insert into the database tables in population operations is stored in text files. These text files are the conversion of Microsoft Excel Spreadsheets into tab-delimited text files. For the program to be able to parse them, they follow a common format. For example, the format of a data file for a table with four columns would be:

Column_1_Name	**Column_2_Name**	**Column_3_Name**	**Column_4_Name**	***ENDLINE***
Data_1_Row_1	Data_2_Row_1	Data_3_Row_1	Data_4_Row_1	*ENDLINE*
Data_1_Row_2	Data_2_Row_2	Data_3_Row_2	Data_4_Row_2	*ENDLINE*
Data_1_Row_3	Data_2_Row_3	Data_3_Row_3	Data_4_Row_3	*ENDLINE*

Table 3-1 – Format of a Data Text File

These files may be created and edited using Microsoft Excel, then exported as "tab-delimited text file". The following is an example of a data file:

	A	B	C	D
1	**CODE**	**Name**	**Description**	**ENDLINE**
2	IA	Internet Access	Internet access service	ENDLINE
3	PROV	Provisioning	IP provisioning for cable modem and CPE	ENDLINE
4	ISP	ISP	Internet Service Provider (Email)	ENDLINE
5	SEC	Security	Security violation	ENDLINE
6	PERF	Performance	Performance issue	ENDLINE
7	MGT	Management	Errors coming from system management itself	ENDLINE

Figure 3-1 – Example of a Data File in Excel Format

As you can see in Figure 3-1, no numeric unique row identifier is provided for the example table, although most database management servers require such column. This issue is managed by the application itself. The RDBMS is responsible for generating a new numeric unique row identifier for each newly inserted row. How does the user create relationships between the data in different tables? By using the special column "CODE" (present in all the tables and, thus, in all data files), which has a similar use to that of a numeric unique row identifier, but is more user friendly and independent of any RDBMS used. To replicate the relationships in the actual database instance, the program then takes note of the numeric identifiers attributed to the newly inserted rows and uses them in the linking column of the other tables that link to them.

Generated Data Text Files

A correctly designed database model is not suitable for direct data manipulation. Data about the same entity is split over several tables, to avoid data redundancy among others. As seen before, the user is expected to create Excel files with the data to be inserted in a column. This process could become cumbersome if the user had to create different files (and manually take care of the links between them) for all the tables that hold information regarding some data entity to be inserted in the database.

This way, the IB OSS DB Populator offers "analysis" functionalities that enable the user to specify one data entity entirely in one single file. The program then analyses the fields in that file and automatically generates the corresponding data files for each of the tables that hold the information about the data entity, taking care of the links between them.

Functionalities

The IB OSS DB Populator offers a wide range of database manipulation options. From the complex creation of a database instance to the simple execution of some SQL or PL/SQL script. Here, we refer the most common ones.

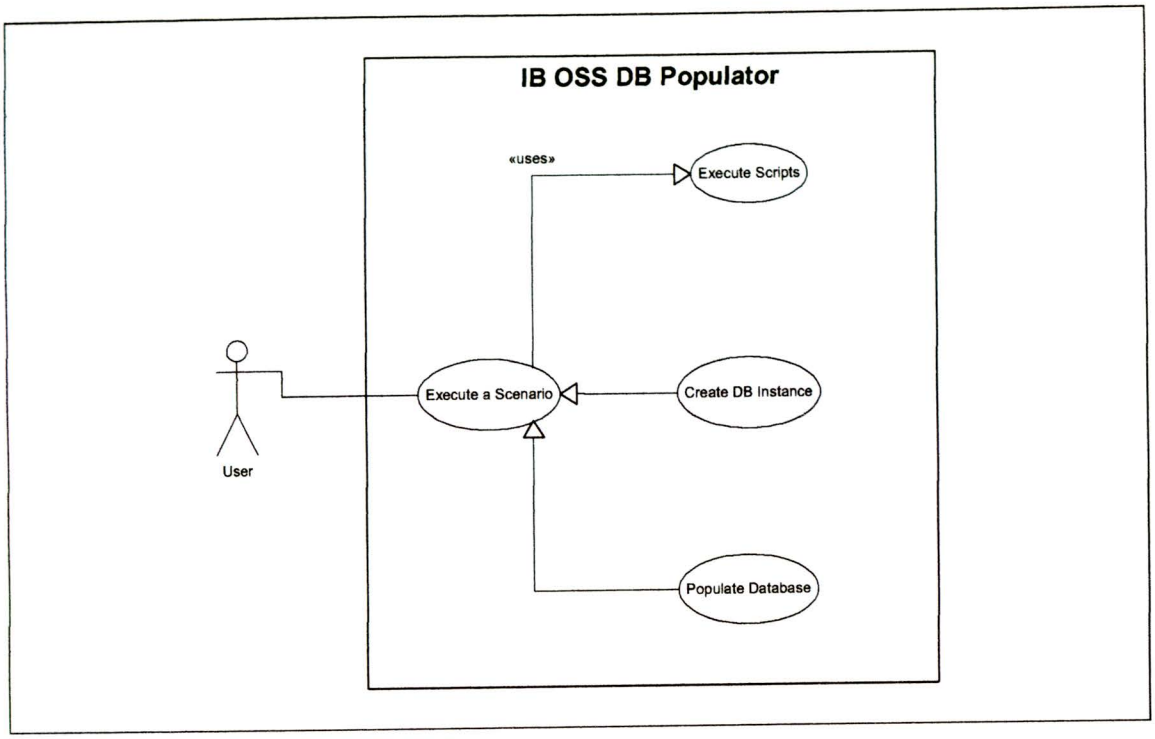


Figure 3-2 – IB OSS DB Populator Use Cases Diagram

Upon specifying a scenario, the user can execute whichever scripts from that scenario. Executing a scenario is nothing more than executing the application scripts. These scripts hold the operations to be executed, that can range from an automatic process like "cleaning a table" to the execution of a manually created SQL or PL/SQL script. For more information about the available commands, please check the subsection "Scripts".

Of course, creating a database instance and populating it with initial data are nothing more that particular cases to the general action of executing a scenario. These use cases are referred here because they are by far the most common use of this application.

Logical Architecture

Here, we specify how the application is logically organized and structured. The following picture shows the logical package structure of the system.

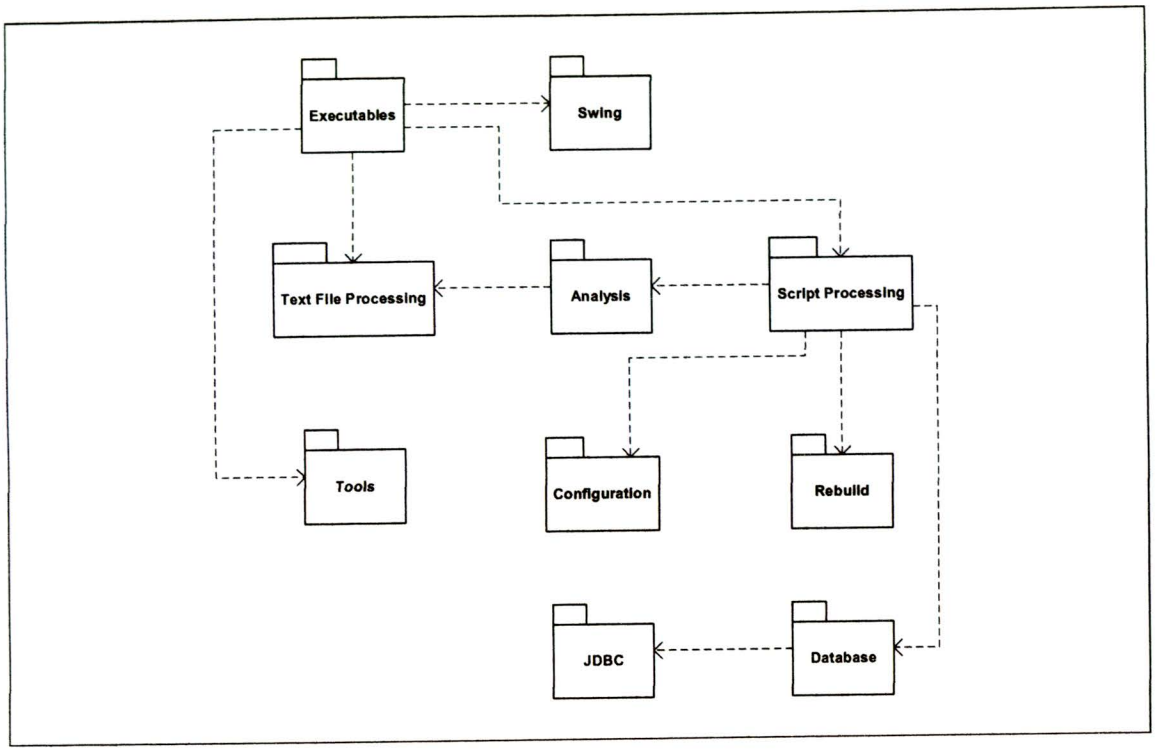


Figure 3-3 – Package Diagram of the IB OSS DB Populator

The user interacts with the executable objects of the classes that compose the "Executables" package. One of those classes provides a graphical user interface, which was built using Swing. The executables create and activate the script processing commands and tools in the package "Script Processing". The classes in the "Script Processing" package perform the core operations of the system, by parsing the script files, creating the respective command for each script file line and then executing that command. They get configuration parameters from the package "Configuration" and may trigger some additional analysis on data files before these are populated, using the functionalities provided by the package "Analysis". To execute the database operations, the package "Database" is used, that provides the needed database functionalities. In its turn, the "Database" classes use JDBC to connect and communicate with an Oracle RDBMS. The packages "Tools", "Rebuild" and "Text File Processing" provide auxiliary utilities and functionalities to the main operations.

As we will be able to see in section 3.3, the improvements on this application were centered in the graphical user interface, belonging to the package "Executables", and the execution control, belonging to the packages "Executables" and "Script Processing". This way, we will now specify only the class structure of those packages that were later worked on and improved.

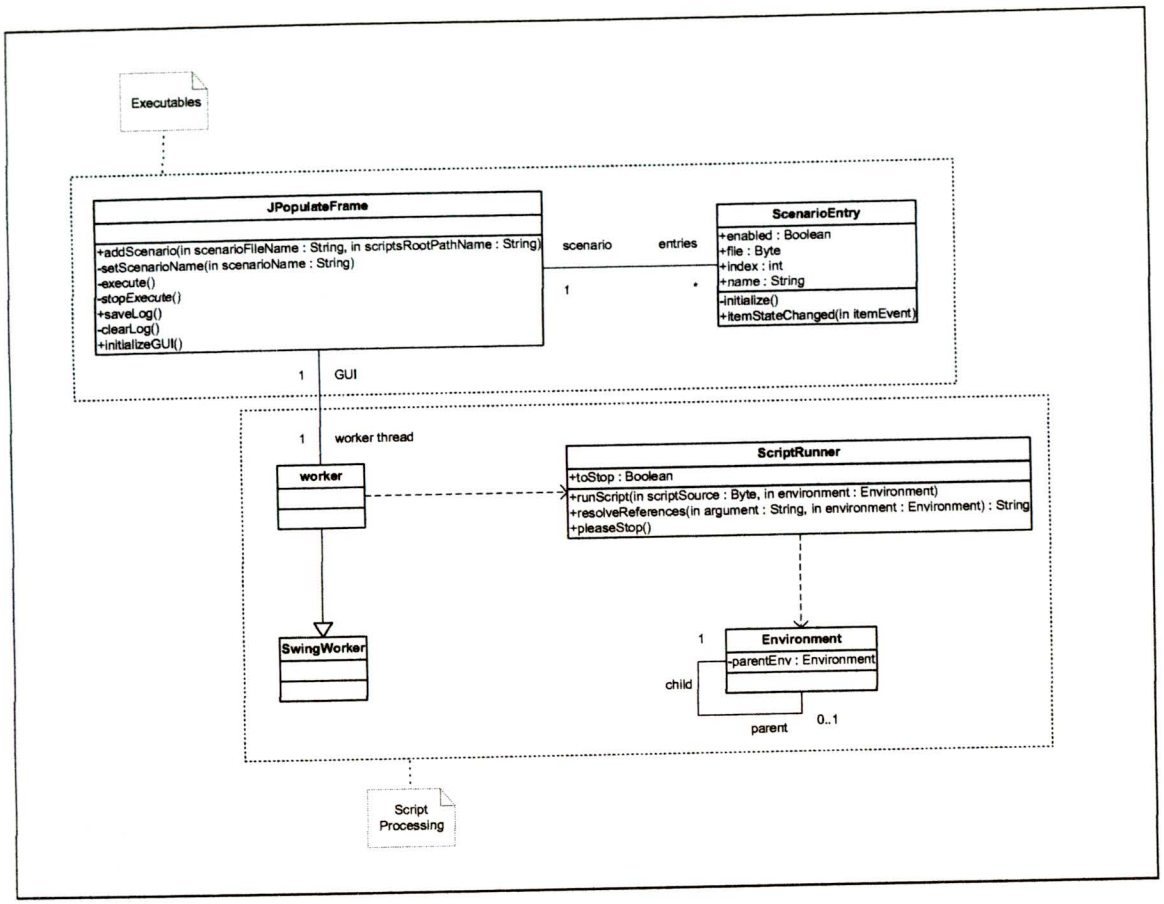


Figure 3-4 – Class Diagram of the IB OSS DB Populator

The class "JPopulateFrame" provides a very basic graphical user interface application. Each instance of that application (i.e. each program execution) has a scenario associated, which is nothing more than a list of scripts to be executed in a particular order (refer to subsection "Scenarios" for more details on scenarios and scripts).

The user may then start the execution process, which is performed by a new thread. That thread and respective functionalities are encapsulated by the class "Worker". "Worker", in its turn, extends the class "SwingWorker" (provided by Sun), using its base functionalities and adding the script processing algorithm to be executed by the thread.

The script processing algorithm makes use of the class "ScriptRunner". This class is responsible for running one script, taking each of its lines and executing the respective command. The "Environment" class is an entity that stores environment variables that are supported by the application's scripting language. Each "Environment" object may have a parent "Environment". This occurs when calling a script from another script. In this case, the called script will inherit the environment variables from the caller script, being also able to set new ones in its scope.

Physical Architecture

The physical architecture of the IB OSS DB Populator is pretty simple. The following is the deployment diagram of the application.

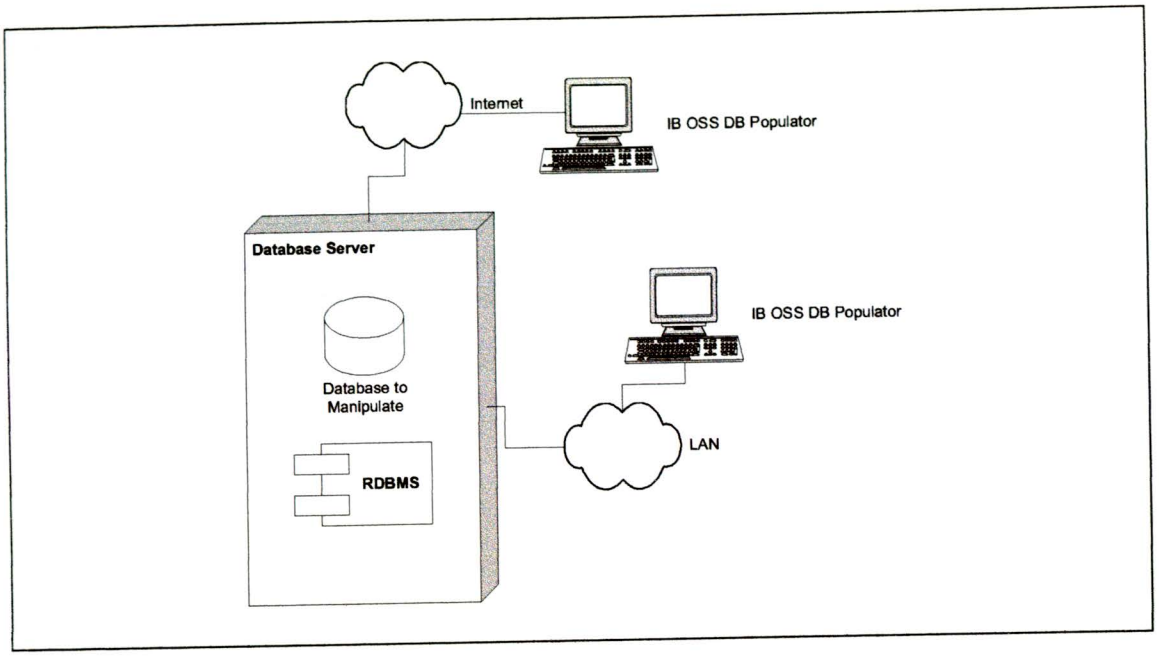


Figure 3-5 – Deployment Diagram of the IB OSS DB Populator

The application is always a client of a RDBMS. That RDBMS will most commonly be running in a remote server. The application can access that remote server using the Internet or a local area network.

The component structure is extensive and not relevant at all to our purposes. A package overview of the component organization is provided next.

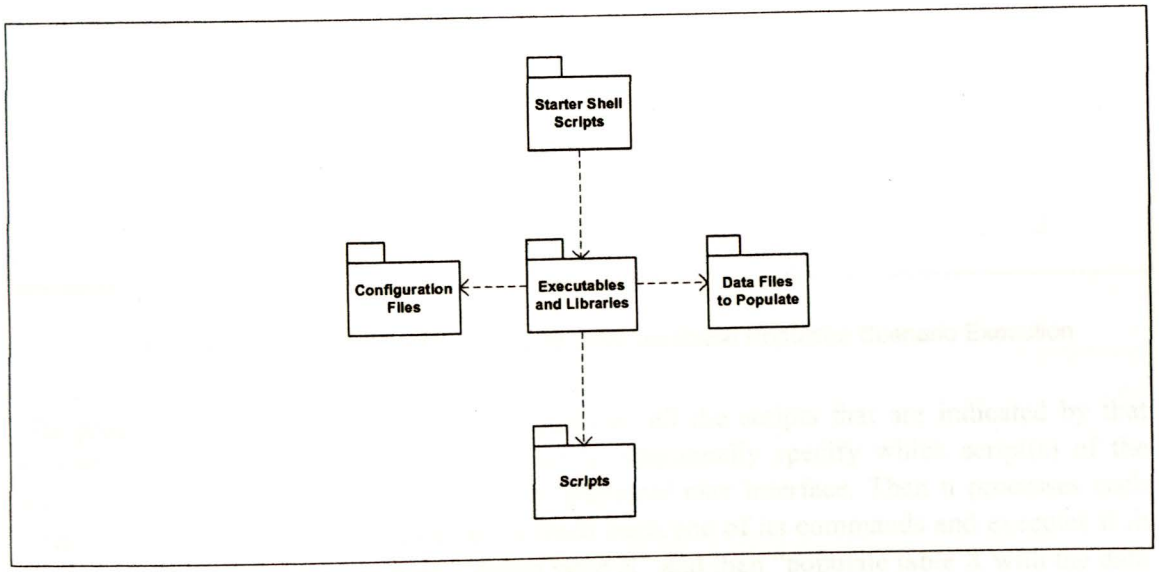


Figure 3-6 – Overview of the Component Structure of the IB OSS DB Populator

The application's physical components comprise starter shell scripts, that start the executable files with preset parameters and configuration values; the executables and libraries on which they depend; the configuration files that mold the program behavior; the script files that

specify the commands to execute; and the data files containing the data to be inserted in the database.

Execution Process

After the user has created the text-delimited data files, the scripts containing the operations and a scenario comprising the scripts to execute, he or she can start the program and trigger the execution of the scenario. The following diagram shows the core functioning of the Imagine Broadband OSS Database Populator.

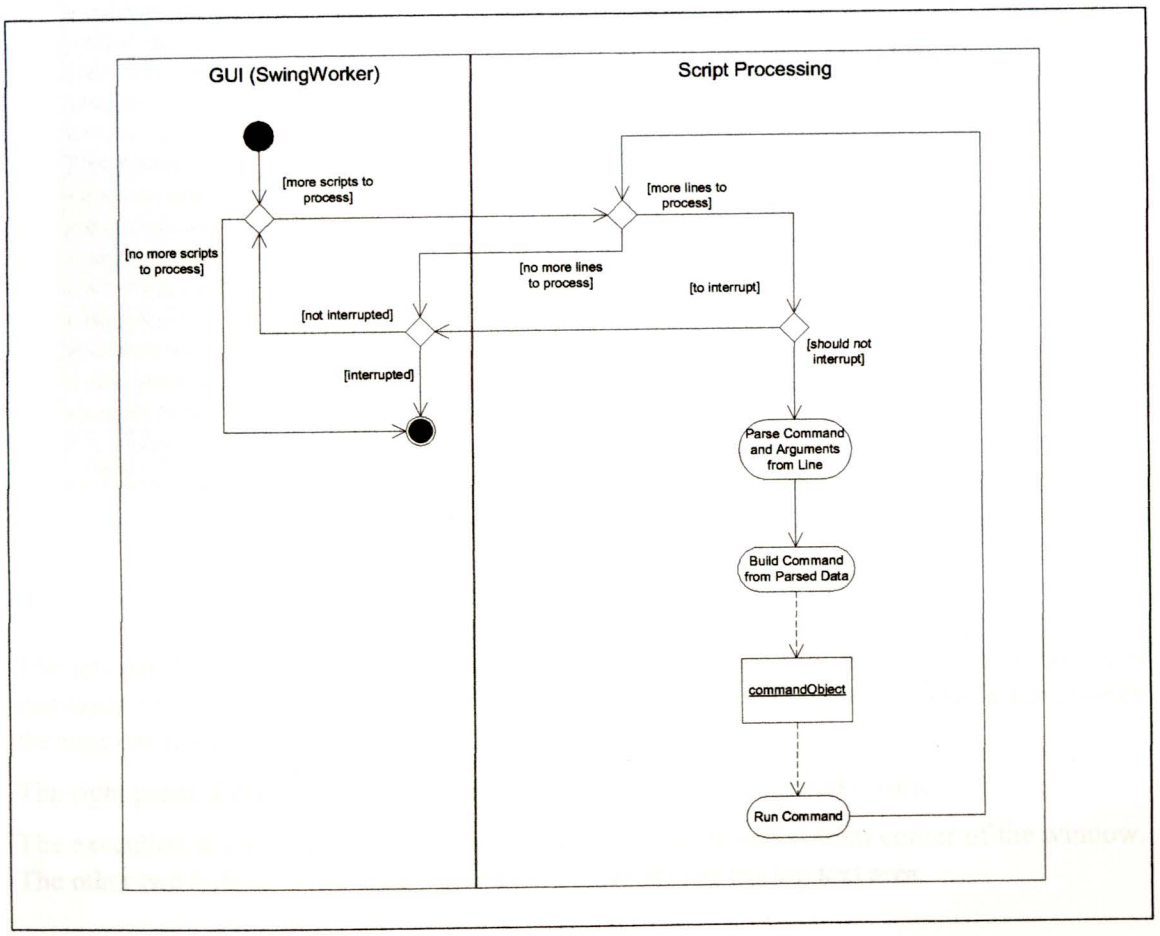


Figure 3-7 – Activity Diagram of the IB OSS Database Populator Scenario Execution

The program reads the scenario file and accesses all the scripts that are indicated by that scenario. The program also allows the user to additionally specify which script(s) of the scenario are effectively to be run, using its graphical user interface. Then it processes each script to be run – for each one of them, it reads each one of its commands and executes it in the specified database, for example, "delete table X" and then "populate table X with the data in file X.txt".

Graphical User Interface

The following image shows the graphical window of the application.

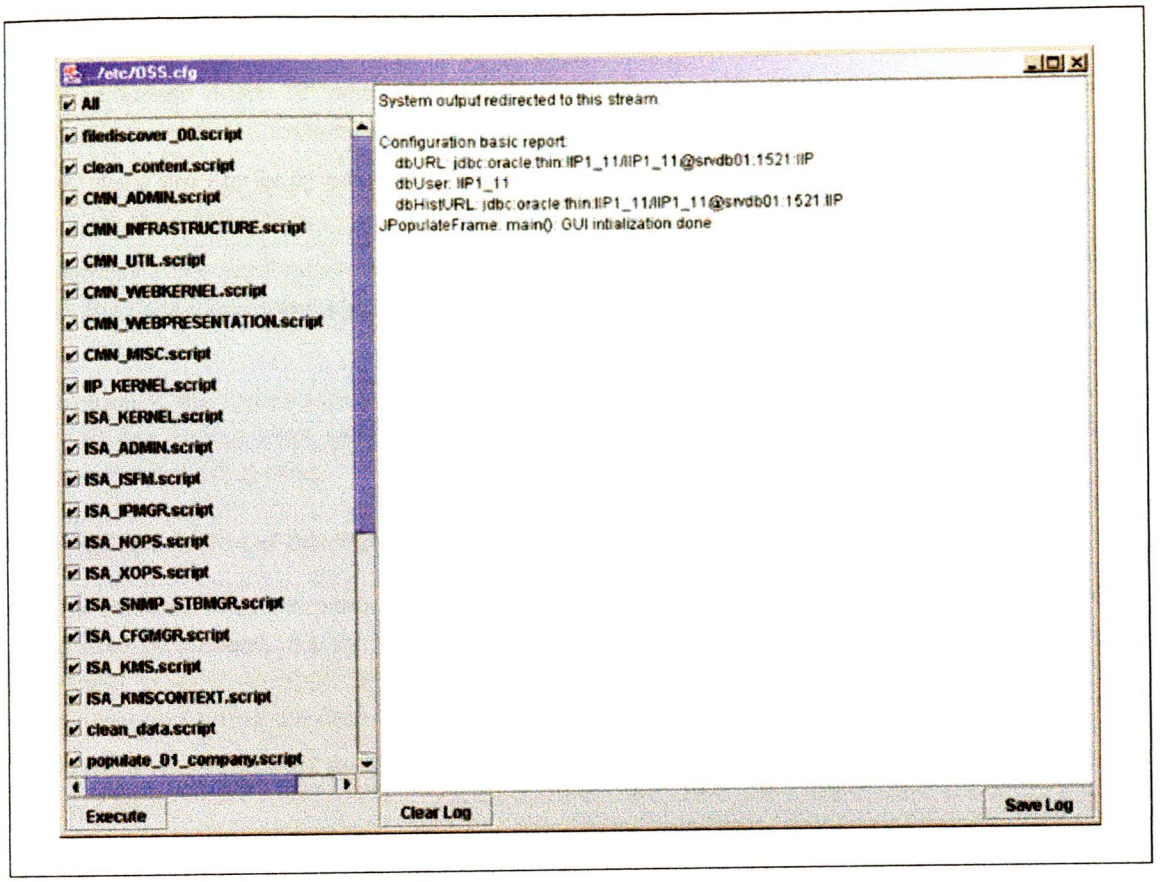


Figure 3-8 – Screenshot of the Application's First Version GUI

The left panel presents the scenario associated to this program execution. The scenario file is displayed in the title bar. Each script of the scenario has its corresponding check box, which the user can use to specify if that script should be executed or not.

The right panel shows the execution progress log output and eventual errors.

The execution is started and stopped using the button at the left-bottom corner of the window. The other two buttons allow clearing and saving the text in the log text area.

Improvements for the IB OSS DB Populator

The first version of the program presented many flaws. It had a very basic and raw graphical user interface (Figure 3-8), missing many common components and functionalities that make programs easy to use and intuitive to the user.

The following improvements for the GUI have been established:

- Add standard GUI components, like the common menus, toolbar and status bar;
- Add confirmation dialog when erasing the log and exiting the program;
- Improve script presentation: instead of the name of the script file, display the script name and a small description, which would be stored in the script itself, as special comments;
- Change the title to include the application's name and icon;

- Make the split bar movable;
- Change color of executed scripts depending on the outcome of their execution (successful or erroneous);
- Last but not least, write and attach an online help document, with searching facilities, to the application.

Furthermore, the application was considerably unstable, casting several exceptions, especially when the user interrupted an execution process. There was some work to be done in this area also.

Additionally, any other aspect that offered room for improvements should be considered. As we will see in the next section, there were many additional improvements performed, in addition to the listed ones.

The Implementation of the Improvements and Final Results

All the planned improvements have been implemented successfully. Additional improvements were also performed. As we will be able to see in this section, the Imagine OSS Database Populator is now an application that looks and behaves professionally, being easy to use, stable and presenting the data to the user in a clear and elegant way.

The physical structure of the program has not suffered any alteration, so we will not repeat it in this section.

Logical Architecture

The logical structure of the application has barely changed. Mainly, three classes have been effectively worked on: "JPopulateFrame", now called "PopulateDB", "ScenarioEntry" and "ScriptRunner". The first two have been changed to provide better GUI and more functionalities; the second to improve the execution control (with pausing and stopping capabilities) and increase program stability.

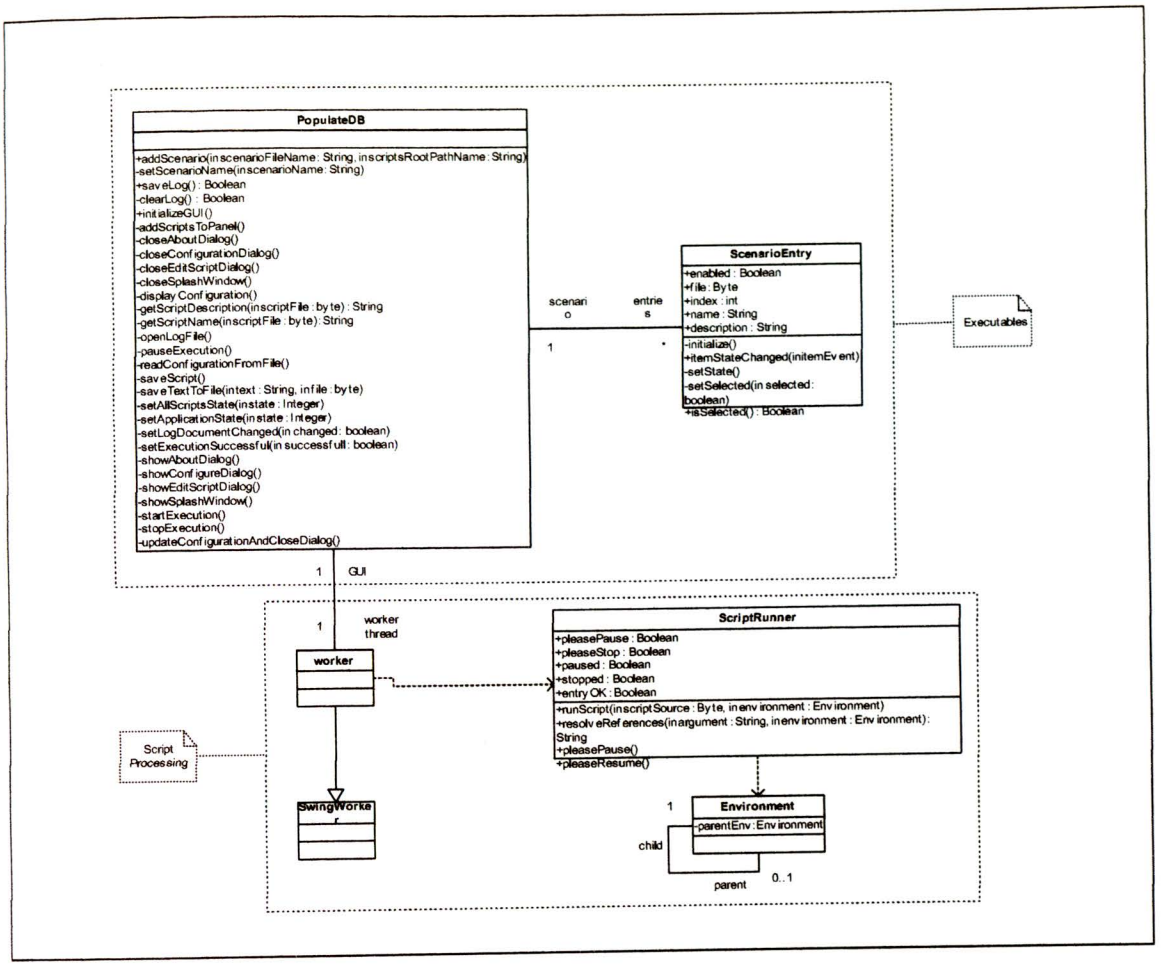


Figure 3-9 – Class Diagram of the Improved Application

As we can see in the diagram above, the class "PopulateDB" (former "JPopulateFrame") presents several new methods. Being the class that implements the main window of the application's GUI, these new methods reflect new functionalities directly "visible" to the user. For example, the new method "openLogFile" shows an "Open File" dialog box and copies the contents of the selected file to the log text area. This new functionality enables the user to open previously saved log files, containing the textual output of some execution. More important functionalities, that will be described later, can already be sneaked, for example, "showEditScriptDialog" and "saveScript", which enable the user to edit scripts directly in the application. The class "ScenarioEntry" also reflects the GUI improvements, now including a data member to hold the textual description of each script.

The "ScriptRunner" class has the new methods and data members for execution pausing and resuming, functionalities that did not exist in the first version of the application.

Execution Process

One of the areas of improvement was the program stability and execution process control. As referred, the application only allowed to abort an execution process and would do it with many errors. Useful pausing / resuming functionalities were necessary. These functionalities have been implemented and the program stability improved. Here's the new scenario execution algorithm:

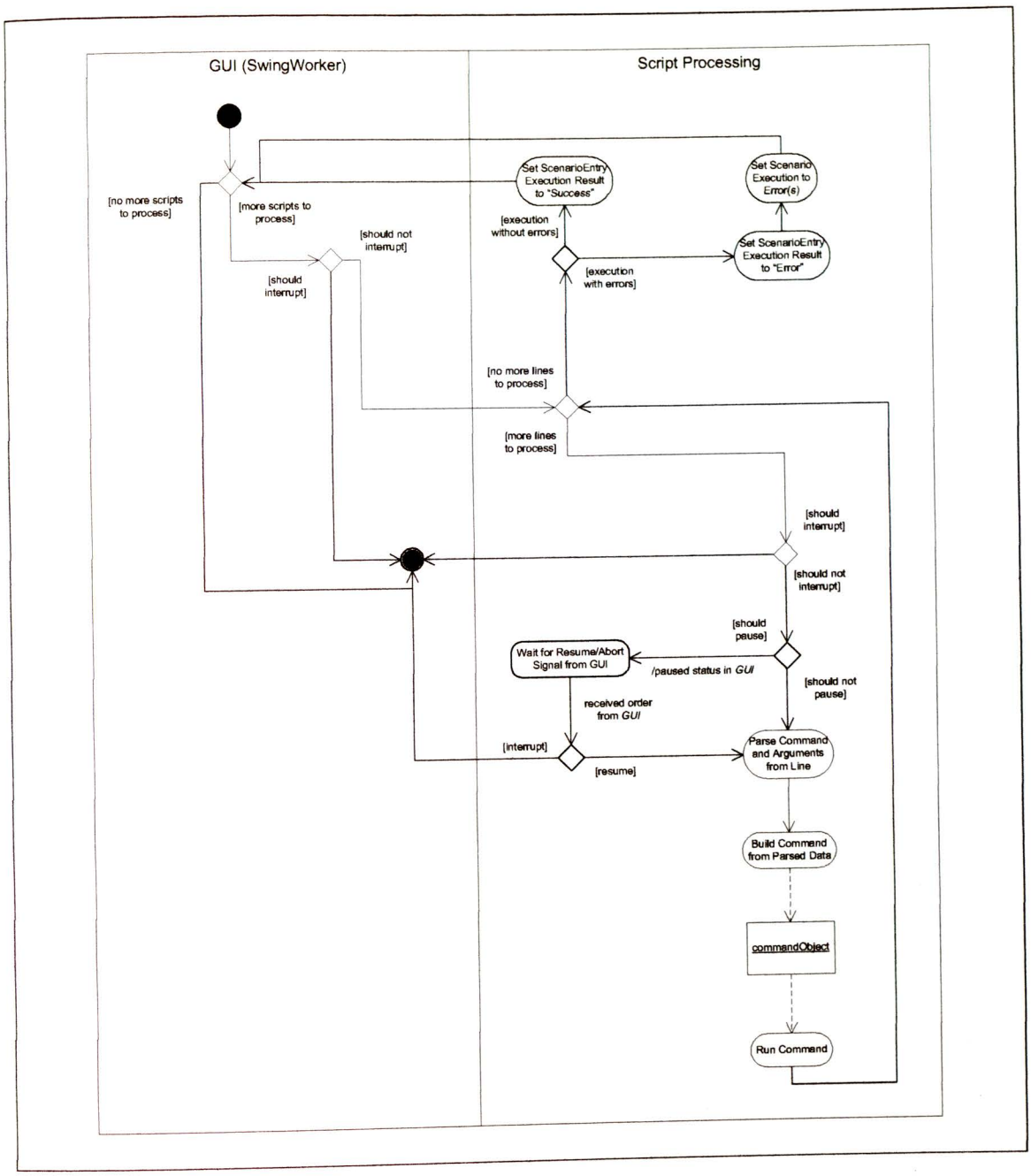


Figure 3-10 – Activity Diagram of the Improved Scenario Execution Algorithm

The program processes all the scripts of the scenario that are set to be executed. For each one of them, it parses each line, creates the command object for the current line and executes it. The user may pause the execution process at any time. For safety reasons, the pause or interruption will only take place between the executions of two commands. Pausing and aborting in the middle of a command execution may lead to undesired resource locks and inconsistencies in the database. Also reflected in this algorithm, is a more informative GUI that will report visually if a script has been correctly executed or not and alert the user in the case of errors.

Additionally, the implementation of this algorithm fixed many technical problems that were present in the process control system. There were redundant and unsafe thread control

operations, among others. The scenario execution system is now secure, stable and more functional.

The following sequence diagram illustrates the roles and the interaction between the most important objects involved in a scenario execution.

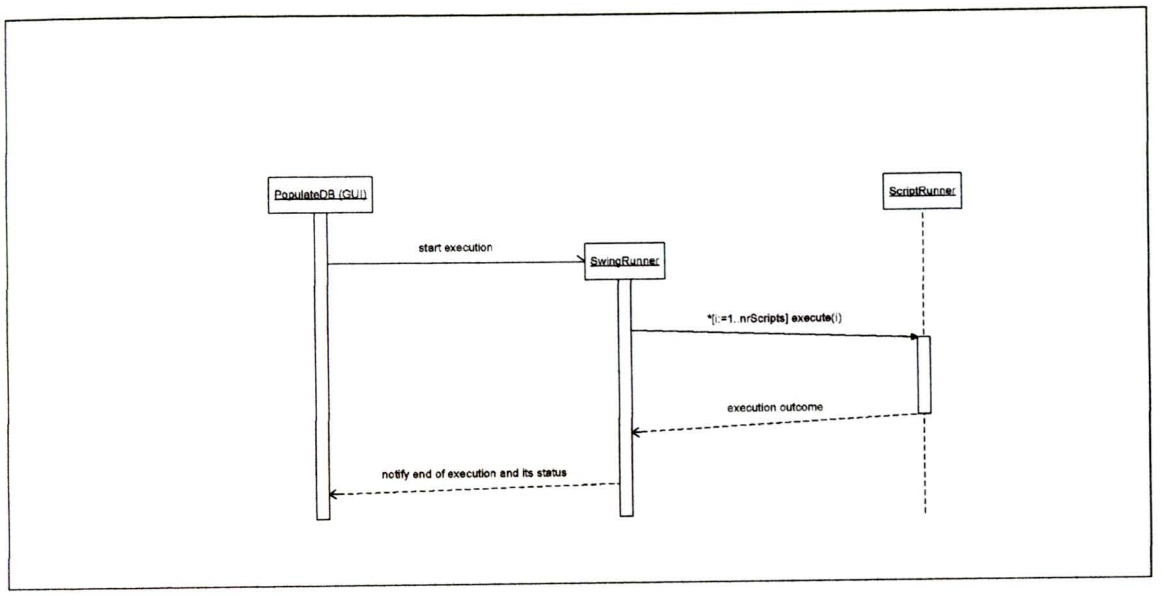


Figure 3-11 – Sequence Diagram of a Scenario Execution Context

The "PopulateDB" object encapsulates the GUI of the application. It starts the execution process, by creating a "SwingRunner" object and executing it. The "SwingRunner" object has its own thread which it uses to do its work. It runs though all the scenario entries, i.e. through all the scripts of the scenario, telling the "ScriptRunner" object to execute each one of them (that are not deselected by the user). The results of the execution are then propagated back from the "ScriptRunner" to the GUI.

Graphical User Interface and Online Help

Due to the enormous amount of improvements to be made in the program's GUI, a new one was developed from scratch. This time, the NetBeans IDE was used to support the development, not only accelerating the process but also ensuring that further improvements and alterations would again be quicker, easier and less error-prone.

The following figure shows the main window of the application's new GUI.

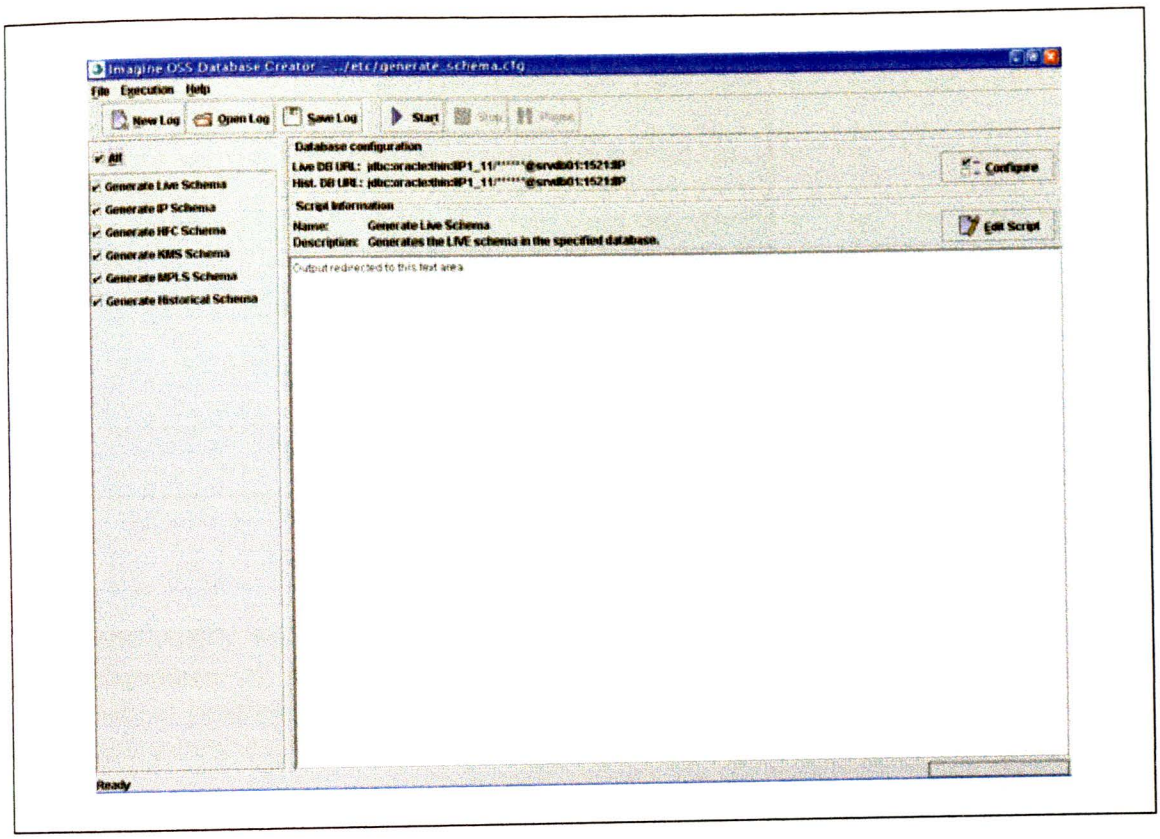


Figure 3-12 – Main Window of the New IB OSS DB Populator

This window contains a menu bar, whose menus provide access to functionalities of the program. The most important commands are also accessible by the toolbars right bellow the menu bar. These contain intuitive buttons with image icons and tool tips to help the user understand their meaning right away. In the left panel, we see the scripts that compose the scenario. Unlike before, they are now conveniently presented using comprehensive names (instead of the previous script file names). For each script, the user can still read a small description of its purpose. The database parameters that are being used by the application are always present in the main window. The execution process output text area keeps its previous functionality, although it does not show technical program errors or uncaught exceptions anymore - these are now always sent to the standard error output stream. A status bar has also been included, telling the user what the program is doing and its status and showing a progress bar, which provides a visual idea of the progress of some scenario execution.

The database configuration parameters may now be altered from within the program (the initial application didn't have this capability). The button "Configure" is enabled whenever an execution isn't taking place.

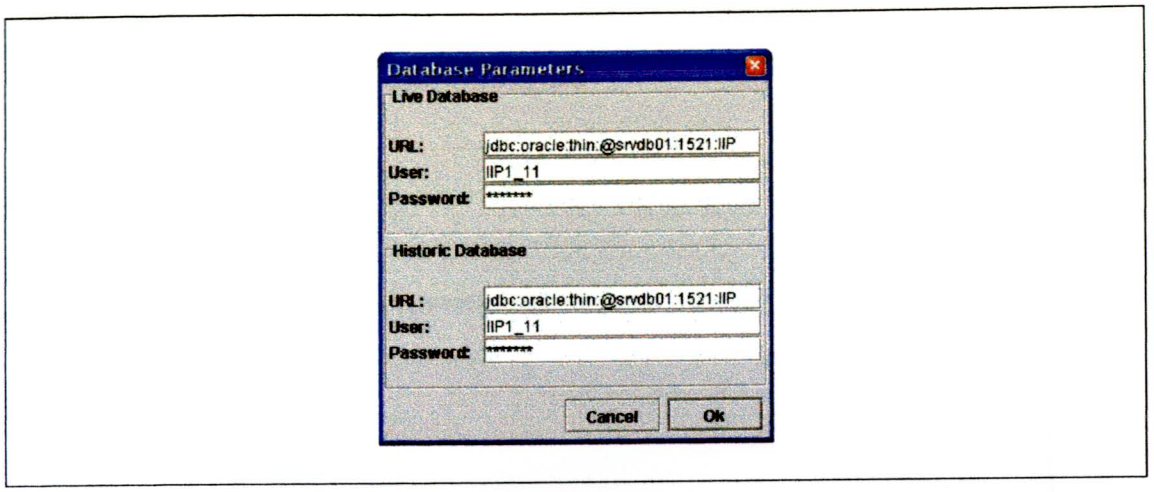


Figure 3-13 – Database Configuration Parameters Dialog Box

This functionality has proved itself to be very useful. However, in order to avoid the misconfiguration of the system in the client site, this dialog box only changes the configuration parameters for the current program session (execution) only. The next time the program is executed, the default configuration defined at the installation phase is used.

When the user needed to make a small alteration to a script, he or she had to open an external text editor to open the script file and perform that alteration. This was a bit time consuming, so this functionality was also added to the IB OSS DB Populator. Upon selecting a script in the left panel, the user can simply hit the button "Edit Script" and a window with the script contents will appear, where the user can edit the script and save or cancel the changes. The following figure shows this window.

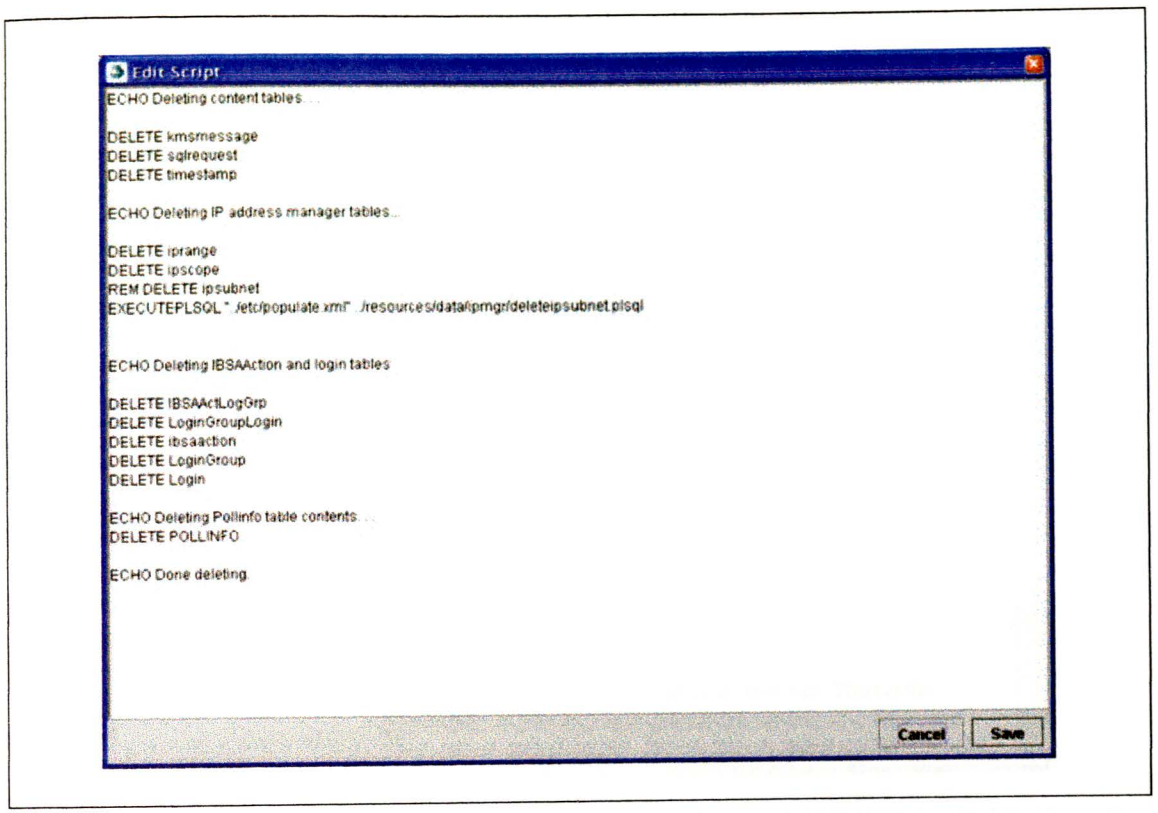


Figure 3-14 – Script Edition Window

This way, the user does not need to know where the script file is and open it with an external text editor. Here, another restriction applies. So that the client user does not introduce mistakes in the scripts by means of using the IB OSS DB Populator, this functionality is only available when the program is run in "advanced" mode. This configuration parameter is defined in the application's XML configuration file and set to "normal" mode by default. Normally, this "advanced" functionality will only be used by Imagine Broadband personnel in debugging or tweaking tasks.

The last two improvements are not present in the improvement list of subsection "Improvements for the IB OSS DB Populator". They were later proposed by the intern and respectively approved by the internship coordinator, with the referred restrictions.

The online help was written and then included in the program in a flexible and convenient way. The user can call the help index at any time, by using the "Help" menu. Context-sensitive help is also available – the user may access it by pressing the F1 key or selecting "Help on..." from the "Help" menu and then clicking in the component he or she needs help on.

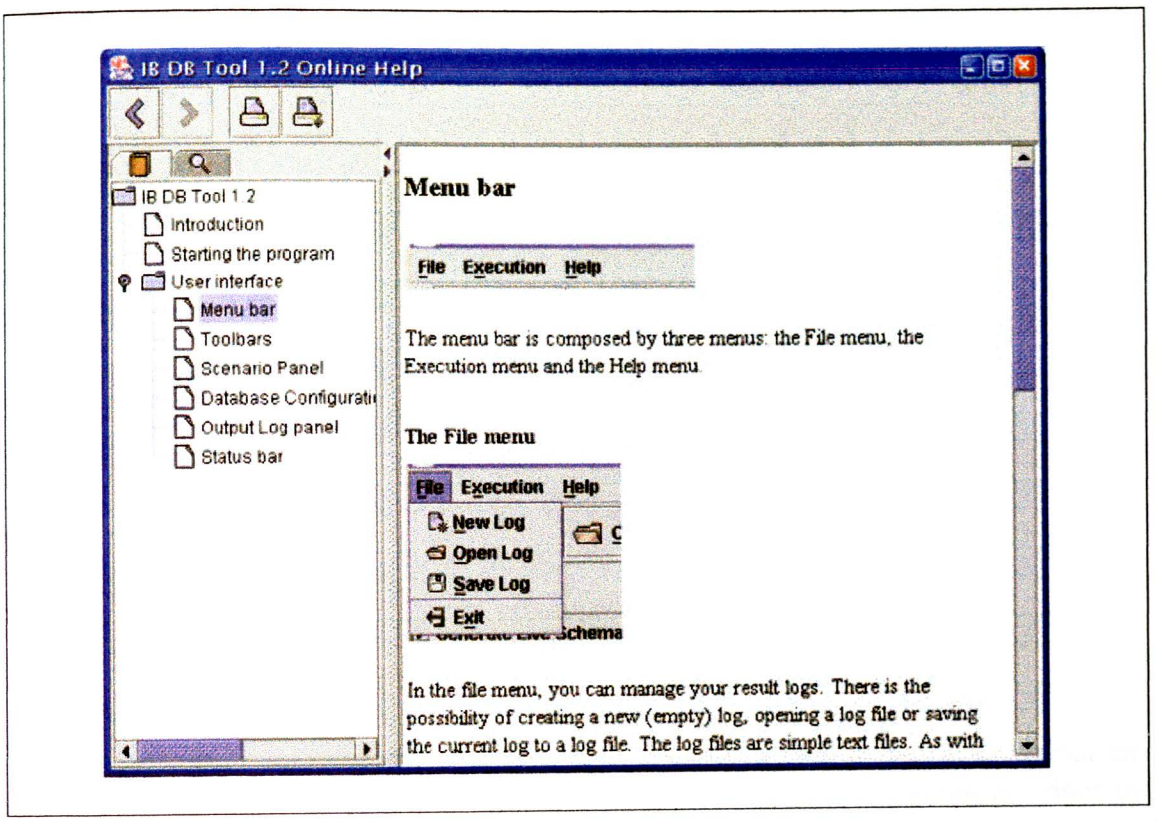


Figure 3-15 – Online Help Window

The help documents may also be searched by keywords and printed.

The whole help document, written by the intern, is available in Appendix B.

3.2 Maintaining the Lease Server

The Lease Server isolates DHCP specific details from client programs that need to query a Cisco Network Registrar (CNR) server. The process is the following:

1. The client application sends a query message to the Lease Server;
2. The Lease Server builds the corresponding DHCP packet, associates a key with it and sends it to the CNR;
3. When the Lease Server receives the answer packet from the DHCP with the key associated in step 2, it parses the packet information;
4. The Lease Server then sends the response message back to the querying application.

The following figure shows this process.

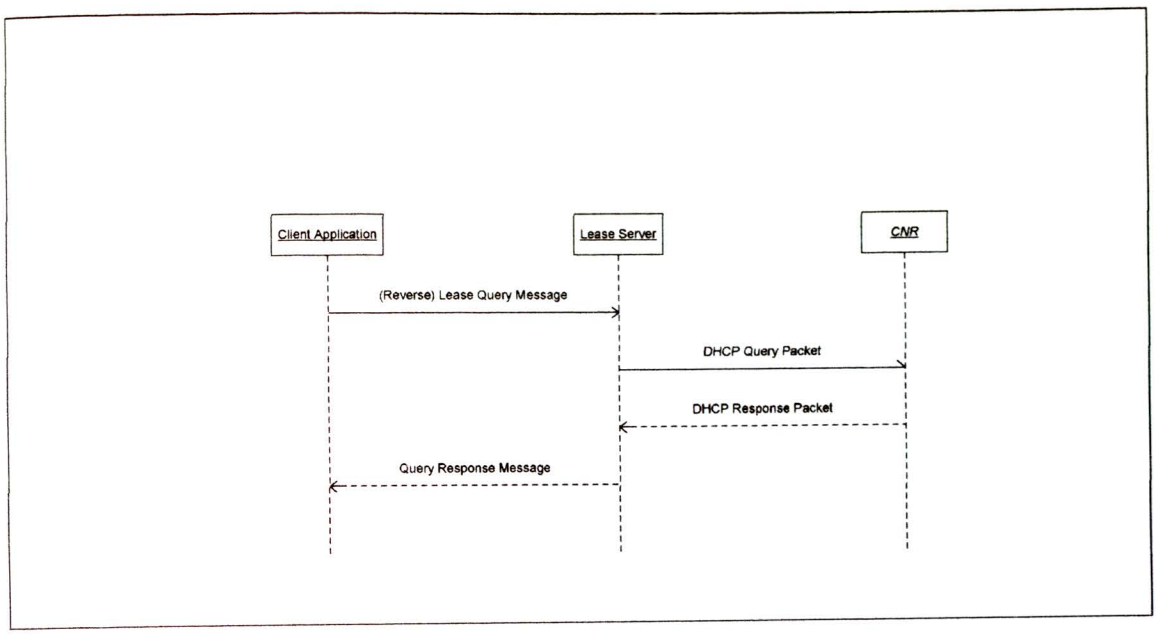


Figure 3-16 – Use of Lease Server Sequence Diagram

For more details on the created and parsed DHCP packets as well as the messages that are exchanged with client applications, please refer to the Appendix C, small document written by the intern following a client's urgent request.

The queries might be of two types: if the client application wants to know the MAC address using some given IP address, it issues a "Lease Query"; if it wants to know the IP address associated to a given MAC address, it issues a "Reverse Lease Query".

Correctly Parsing the Response DHCP Packet

The response DHCP packet that the CNR sends back to the Lease Server contains several fields holding different information. That information is parsed by the Lease Server and sent back to the querying client application in a more convenient data structure.

The way the Lease Server was first parsing was incorrect. It assumed that the order and size of the fields was always the same and it simply relied on that theoretical order to identify the values (ex. the first field was the IP address, the second was the MAC address, etc.). Although this technique worked well for some clients, it did not correctly retrieve the information from the DHCP packet for a particular client where the equipment structure was a bit different. In fact, the differences in the equipment structure configuration reflected in the configuration of the DHCP packet data itself.

The solution was clear. The optional DHCP packet fields have a code area identifying their meaning. It was just a simple matter of searching for the field with the code of the information we are looking for. The method of getting some data optional field from a DHCP packet is described in the following activity diagram.

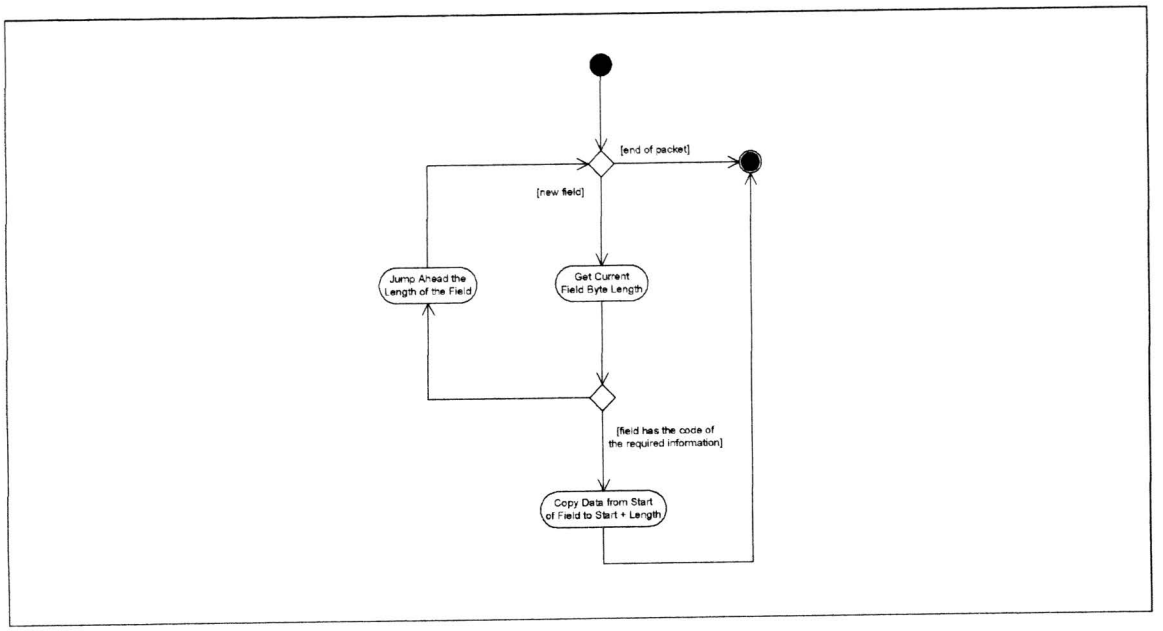


Figure 3-17 – DHCP Parsing Activity Diagram

Starting in the first data field of the packet, the program gets its byte length, information that is also present along with the field code. It then checks if that field code is the one it is looking for. If it is, it copies the information between the start of the field until that start plus the byte length of the field. The process ends. If it is not, it jumps to the next field, by advancing the byte length of the current field. If there is a new field at that location, the process is repeated. If the packet ends at that location, the optional information was not present in the packet.

3.3 Maintenance of the "Infrastructure" Applications

As it was referred in the first paragraphs of this chapter, the Infrastructure applications update a set of tables that hold information about a client's hardware and software infrastructure. The "client" is a large scale network operator that uses the Imagine Broadband Instant Provisioning and / or Imagine Broadband Service Assurance products. The Infrastructure applications are run periodically to keep the database updated with the client's current hardware and software infrastructure configuration and state.

Although each of the Infrastructure application has a specific and distinct purpose, almost all of them share the same process overview, which is depicted in the next figure.

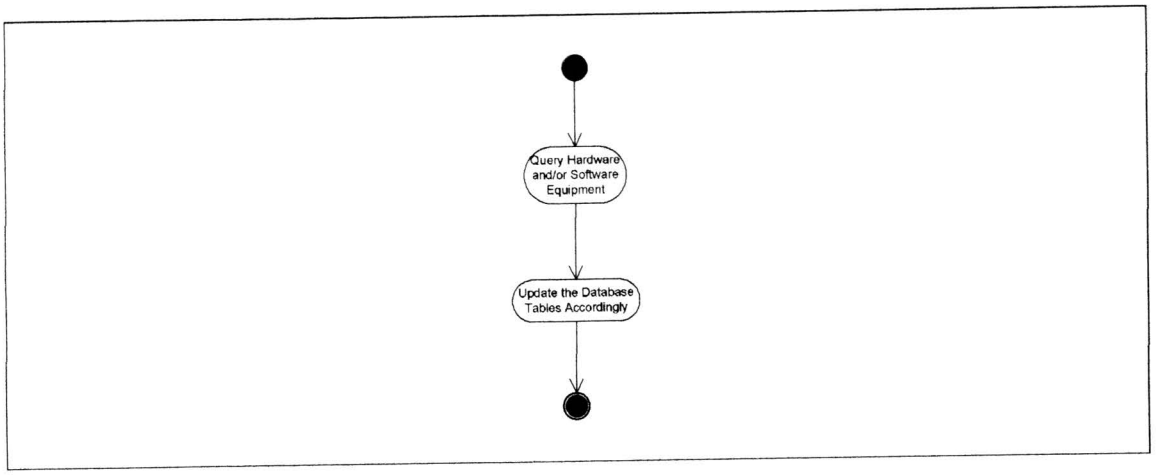


Figure 3-18 – Overview of the Infrastructure Applications Process

These applications work with specific networking equipment and software. They update tables in a very large and complex database schema. It is not in the scope of this document to explain the equipment and software involved or the database schema of the Imagine Broadband products. Therefore, the reader not familiar with networking equipment terminology and concepts may find the rest of this chapter somewhat obscure, but will get the idea of what the programs do.

In the next subsections, the applications that were maintained are described.

updateHFCIPHist

The "Cisco Network Registrar" (CNR) server acts as a DHCP server. Cable modems contact a CNR to get an IP address to use to connect to the internet. The CNR's that run in the client environment have an extension, developed by Imagine Broadband, which outputs to a text file all the IP addresses that the CNR has leased to cable modem MAC addresses. The log text file also contains the date of the lease, the MAC address of the CPE[6] to which the cable modem is attached as well as the subscriber that owns the cable modem in question.

The information in the log file is to be inserted in the database, to be stored and accessed conveniently by any IB product requiring historical information. This is the task of the updateHFCIPHist program – to access the CNR servers, get their log files, parse them and insert the information in the database. The data must be processed first. It is impossible to continuously insert all the leases into the database, as their number easily gets to the millions. Redundancy checks must be performed. No duplicate subsequent leases of the same IP address to a certain cable modem MAC address for the same CPE MAC address and subscriber should exist in the database – just an historical of IP addresses leased.

Getting the Log File by Secure FTP

The program used the File Transfer Protocol to get the log text file from the CNR server. In the production environment of the clients, all the processes must be performed in the most secure way. The program should then be adapted to use the Secure File Transfer Protocol.

[6] Short for customer premises equipment - communications equipment that resides on the customer's premises.

The program logical structure had a class that encapsulated an FTP connection and that was used to get the file from the machines where CNR's were running. On the other hand, other applications of the package used Expect scripts to use local Telnet or FTP clients to perform the desired operations. It was then decided that the best and quicker way to execute this alteration would be creating a new Expect script that controlled an SFTP client to connect to the desired computer and get the desired file. This way, the approach would be the same in all the Infrastructure applications.

The new Expect file has been created and the program now supports the two methods for the file transfer: FTP or SFTP. The script to be used is specified in the configuration file of the application. Needless to say, the new method required that an SFTP server would be installed in all the computers that run CNR's.

New Lease Redundancy Check Algorithm

As referred before, the leases in the log file must be submitted to a redundancy check against the data already in the database before deciding they should be inserted or not. It is not feasible to insert all the data into the database, as its quantity grows to unbearable values.

The first algorithm that was being used consisted in, for each lease of the log file, getting the latest lease from the database for the cable modem and CPE MAC addresses of the logged lease and checking if the IP address and the subscriber were the same. If they were, the lease from the log file would be discarded as it was a duplicate, otherwise, the new lease entry would be inserted in the database. This process of "getting the latest lease" implied that the leases from the log file are treated in ascendant date and that they are all more recent than the ones in the database, for the program to work. In fact, this condition isn't always true. For example, when processing two CNR's logs, after the first has been processed and its data inserted, when processing the log file of the second, the leases are most likely to be older than the recent ones from the first CNR already present in the database. If a cable modem got IP addresses from the two CNR's during that period, undesired results may occur.

The first algorithm was incorrect. It could lead to undesired results under special circumstances. As these circumstances were rare, it stayed in action during a considerable amount of time. Nevertheless, a new one has been thought of and decided to be implemented. Instead of getting the latest lease for the given MAC addresses, the new redundancy check would get, from the database, the lease immediately before and the lease immediately after the date of the current lease being processed, for its MAC addresses. If the leased IP address was the same for the immediately before lease, then we drop the current one. If it is also the same for the immediately next lease, then we have already an error in the database, as no subsequent duplicate IP addresses for the same MAC addresses and subscriber should exist. The error should be reported. If we can't find any previous lease for the given MAC addresses or the previous IP address is different from the current one, then we either insert the current one, if no immediately after is found or, if an immediately after lease with the same IP address is found, we just change its lease date to be the one of the current lease from the log file. The lease entries in the database should always represent the first lease of a group of subsequent ones.

The activity diagram of this program is shown next.

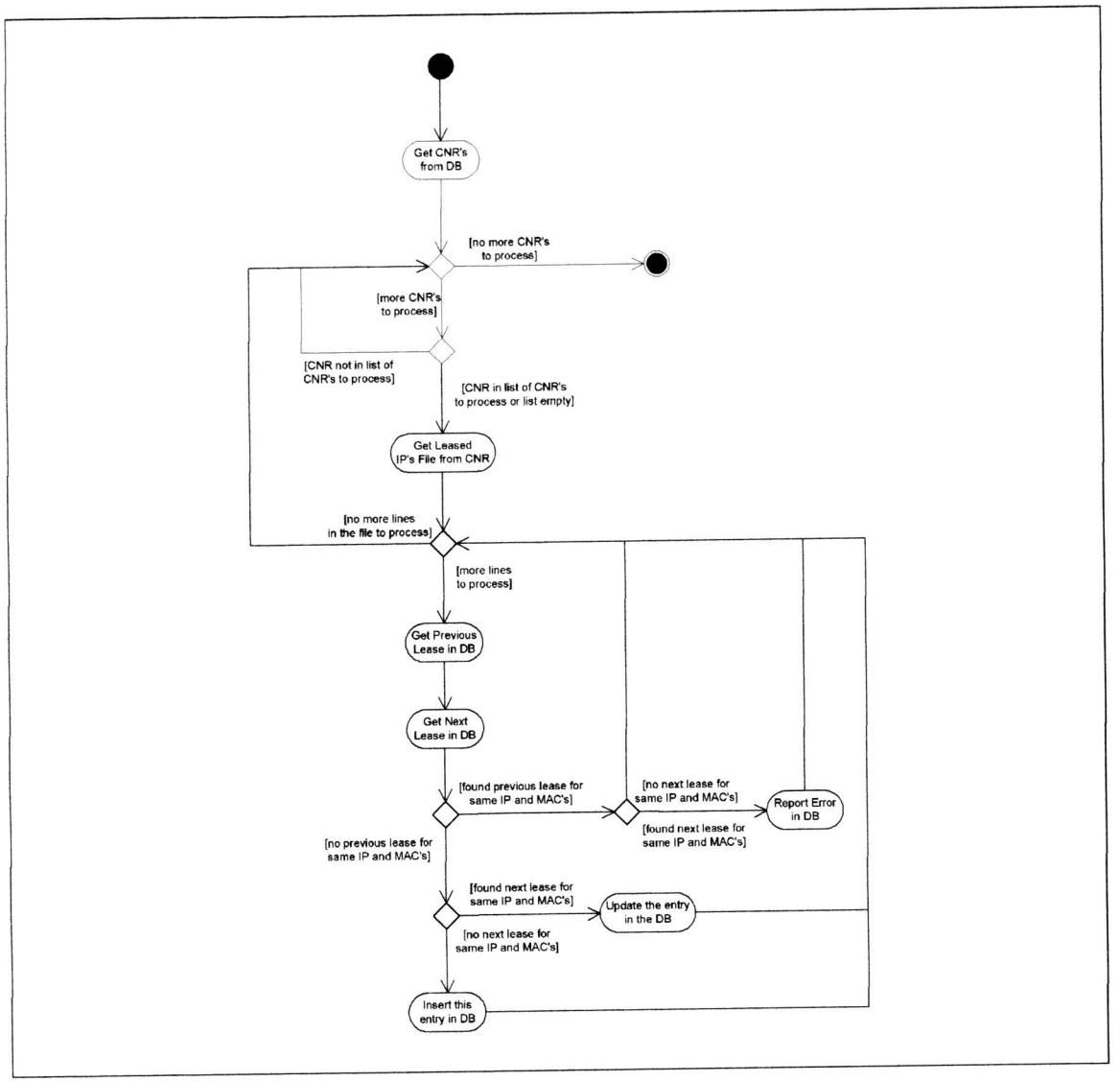


Figure 3-19 – Activity Diagram of updateHFCIPHist

The information about the client CNR's and the machines in which they are running is present in the database. The program queries the database to know the computers to connect to and get the log files. Then it parses the log files applying the new described algorithm.

updateHFCScope

This program also deals with the "Cisco Network Registrar" (CNR) server. Each CNR has a set of associated "scopes" of IP addresses. Each "scope" is a range of IP addresses; it stores the start and end IP address of that range.

Information about the scopes of each CNR should be kept updated in the database. The updateHFCScope program connects to the CNR's that are present in the database and executes a command that reports the scope information for that CNR. That information is then parsed and the database is updated with it.

Using a Secure Connection: SSH

The connection was made using the Telnet protocol and controlled by Expect scripts. This requirement is analogous to the Secure FTP connection for updateHFCIPHist. To improve security, the program should support SSH connections to the computers where the CNR's are running. As this program already used Expect and client software (Telnet) to connect to the CNR's, this requirement was met by creating a new Expect script that used an SSH client instead of the regular Telnet. Again, the machines where the CNR's are running need to have an SSH server installed.

Improving the Updating Procedure

The program started its execution by deleting all the scope information in the database. The new scope information would then be inserted after contacting the CNR's. However, this approach didn't take into account the possibility of a connection failure with some or all the CNR's. In this case, the scope information respecting the unreached CNR's would be lost.

To avoid this information loss, the procedure had to be changed. And the solution was clear: the program should only delete from the database the scope information regarding a successfully connected CNR that returned valid data. This way we ensure that we will only delete information if we really have new information to insert.

The new procedure has been implemented and the complete program execution process is depicted in the following figure.

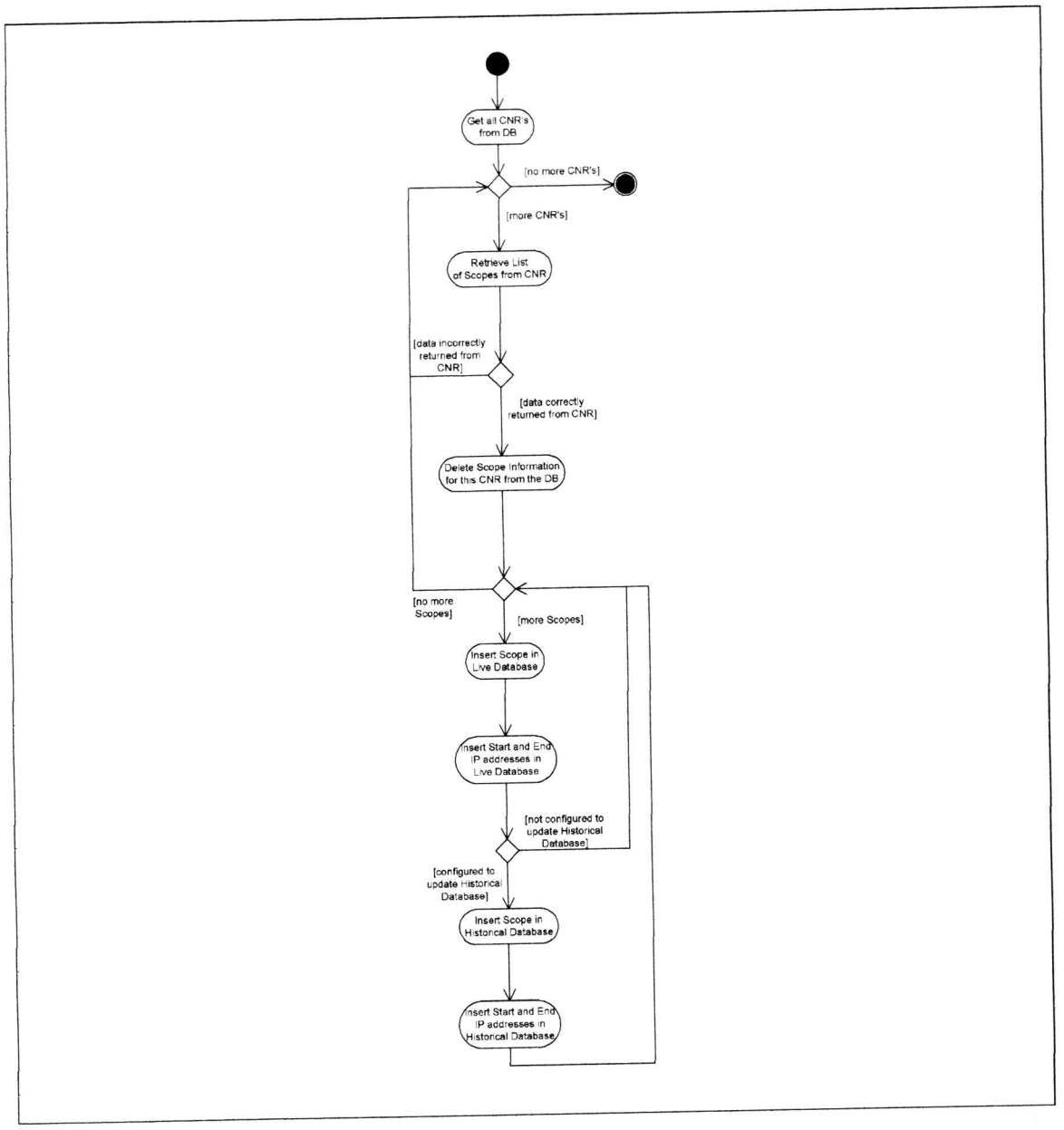


Figure 3-20 – updateHFCScope Activity Diagram

updateHFCTableLocationHSDQos

A long name for a small program. Its task is to fill a table called "LocationHSDQos" in the database. This table connects "segments" in the "Location" table to "quality of service" entries in the table "HSDQos". It starts from the "segments" and gets to their "QoS" following a number of links between tables.

Following a New Path to "QoS"

New versions of the Imagine Broadband products started using new tables of the database schema. It was necessary to update the path of tables that updateHFCTableLocationHSDQos runs through to get to the QoS of each "segment". For backward compatibility, the old path should also be maintained. A configuration parameter would define the path to use.

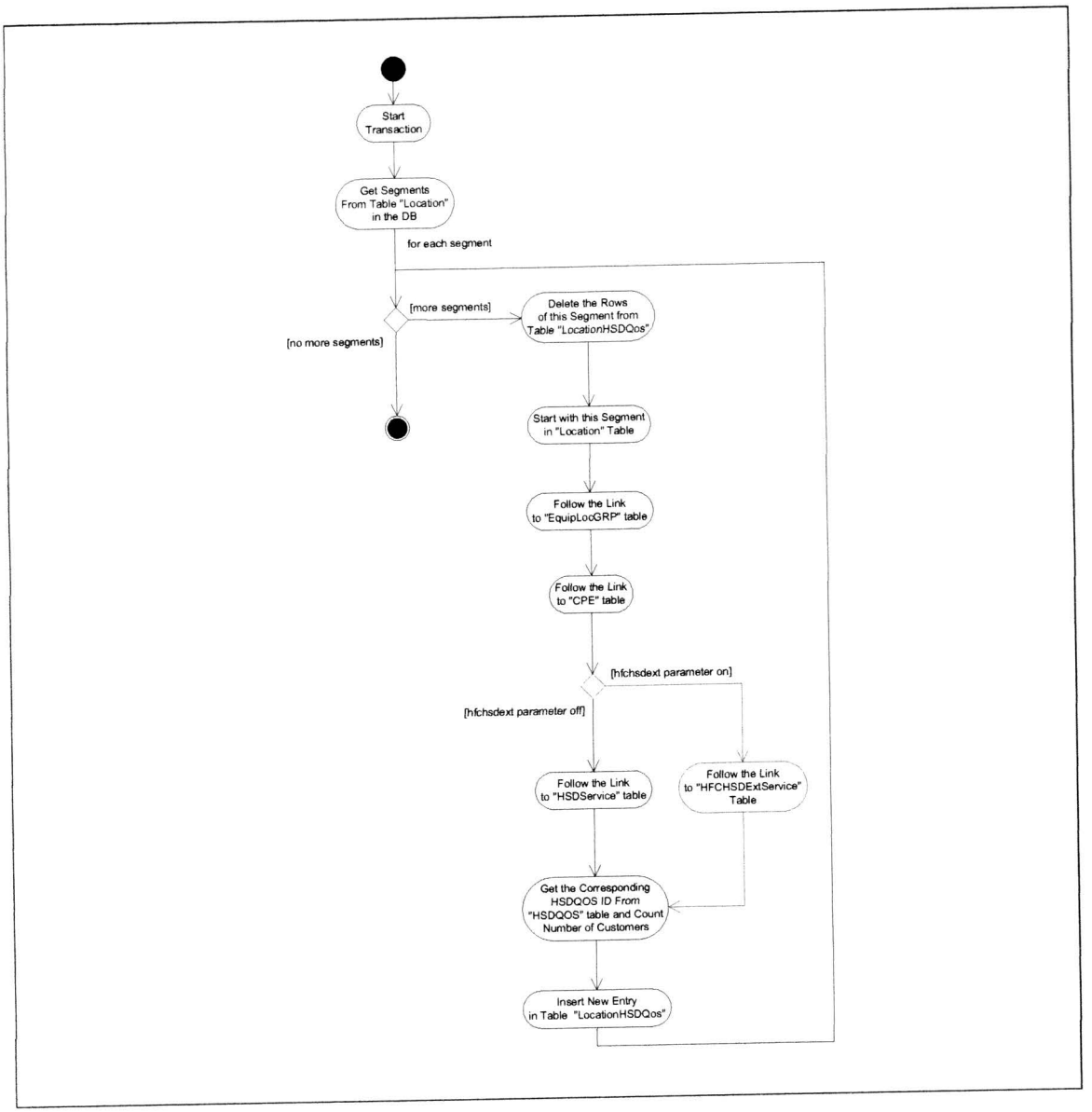


Figure 3-21 – updateHFCTableLocationHSDQos Activity Diagram

The activity diagram above shows the program execution. The new path difference is marked in blue.

updateScopeUBRLink

The purpose of this program is to associate each scope in the database with the UBR[7] that belongs to its IP address range. Unlike the maintenance work in the other Infrastructure applications, this is a new program and was actually developed from scratch.

To achieve its purpose, the program queries all the scopes in the database and, for each one of them, gets the CMTS[8] cards whose IP address fall within the IP address range of the scope.

[7] A hybrid fiber coaxial (HFC) network is a telecommunication technology in which optical fiber cable and coaxial cable are used in different portions of a network to carry broadband content (such as video, data, and voice).

Then, it obtains the internal identifier of the UBR that contains the selected CMTS cards. Finally, it inserts the obtained UBR identifier in the proper column of the current scope entry. This process is depicted in the following activity diagram.

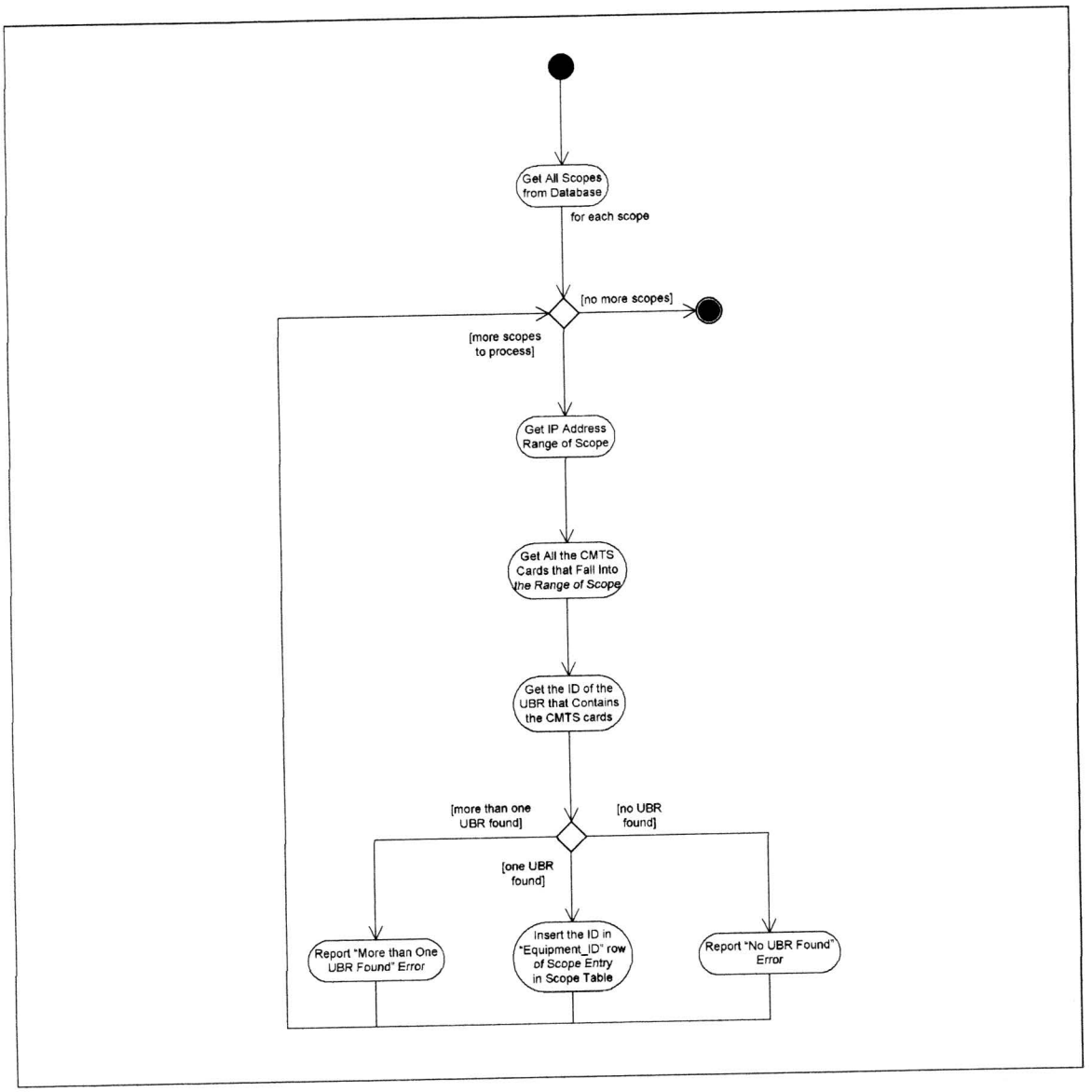


Figure 3-22 – updateScopeUBRLink Activity Diagram

If the program finds CMTS cards that belong to more than one UBR, then it should report and log the inconsistency error of the database. It should also report and log if it did not find any UBR at all for some scope.

[8] A cable modem termination system (CMTS) is a component that exchanges digital signals with cable modems on a cable network. A cable modem termination system is located at the local office of a cable television company or ISP.

3.4 Aspects to Improve

Apart from the continuous maintenance work, there are details that should be improved in the Infrastructure applications. They are badly documented. Although some diagram work has already been done by the intern, there is still a lot to improve. The existing documentation is extremely succinct and obscure, to the point that in some cases it helps nothing in understanding the programs. Also, some applications suffer from a badly presented and cryptic code. Most of it is uncommented and the variable names are confusing or barely explicit. Work to rewrite the code in a presentable, understandable and documented way has already been planned. It should occur as soon as possible.

3.5 Technologies, Methodologies and Tools Involved

In this section, we list the key points of the technologies, methodologies and tools involved in this project. The purpose is simply to describe their aspects or characteristics that were fundamental to the project.

Object Oriented Programming

All the programs in this project were developed using the Object Oriented Programming (OOP) paradigm. Describing OOP and its advantages would take one or more books. A well-designed OO system is entirely composed by "objects" that interact with each other. These "objects" may be imitations of "real life" objects or any kind of computational data entity.

Using an object-oriented approach to design systems improves system modularity and induces code reuse. Systems are easier to understand, improve and debug. As the system is composed of modular parts, if they are correctly planned, they can be reused in other systems where the same functionality is necessary.

Java

All the applications of the second project were programmed using the Java programming language. Java is one of the most popular object-oriented programming languages of the moment. Among its many advantages, that raised its popularity and use, we have its unmatched portability, its memory control mechanisms and its robust and complete API.

The Java code is compiled to byte-code, a kind of intermediary code that is then effectively run on the Java Virtual Machine. As Sun, the creator of the Java language, developed Virtual Machines for all the major hardware platforms and operative systems, the compiled byte-code (and therefore, the Java code) is portable to any of the supported platforms, theoretically without a single alteration. If the developer has some precautions in mind, this theory is effectively proven in practice, as it was with this project – the applications have been tested and run in the Windows and Solaris platforms. Also, the Java Virtual Machine provides mechanisms that make memory and object control manipulation as easy as ever. The famous Garbage Collector is responsible for periodically removing unreferenced data from the system's memory, thus preventing memory leaks. Of course, these mechanisms reflect negatively on the program's performance.

The most important advantage in our case is the portability. Large scale network operators have the most diverse equipment with different operative systems. It is necessary that one

particular product is able to run in any platform without additional work. Java plays a fundamental role in this aspect.

The easy integration of Java programs with the World Wide Web was also a key advantage that brought much success to the Java language. However, it had no relevance to our project.

Official website: http://java.sun.com

JDBC

The Java DataBase Connectivity (JDBC) API provides database access from Java applications to any database server. Taking advantage of the philosophy of portability behind Java, it defines sets of interfaces that JDBC driver developers must follow. The result is that the Java code written to perform some operations in a particular database server will work the same results, without any alteration, with any other database server to which there is a JDBC driver.

Besides the fundamental aspect of portability, the JDBC API makes remote database connection and manipulation inherently simple and flexible.

In the IB OSS DB Populator and the Infrastructure projects, the Oracle JDBC driver has been used for all the database operations with Oracle databases, which are used by the Imagine Broadband products.

Official website: http://java.sun.com/products/jdbc

JavaHelp

When facing the problem of attaching an online help document to the IB OSS DB Populator program, there were the following requirements and restrictions: there was the need for a functional and portable system, with searching and printing facilities, and the time for its development was very short (it wasn't even a priority at the moment).

Once again, the solution came from Sun, with its JavaHelp technology. JavaHelp permits developers to include online help documents in their applications or any kind of systems. It offers base functionalities, like a searching engine, that can be extended. The graphical interface of the system is based completely on the "Java Foundation Classes", which allows it also to be changed and extended.

The help documents for the JavaHelp system should be developed in HTML v.3 (Hyper Text Markup Language). The table of contents and other control information are defined in specific JavaHelp files, which can be created using specific third-party applications, like Software 7 Helen.

Calling the online help system from a Java application is then as simple as creating a few JavaHelp objects and calling simple methods. Context-sensitive help is also easy to implement – the GUI components are associated with help documents and a special event listener is registered with them.

Official website: http://java.sun.com/products/javahelp

Expect

Expect is a relatively unknown but very useful tool. Expect automates interaction with programs. For the reader unfamiliar with Expect, it is easier to understand its purpose with an example. Suppose you are tired of having to type your login and password every time you

login to any server using Telnet. You can then write an Expect script that starts the Telnet client and reads its output. The script you write should detect the line of text that the Telnet server sends to the client when asking for the login. At that phase, your script should direct Expect to automatically specify your login. Then, the same process is used for the password.

Expect is much used for testing purposes. Testers build test Expect scripts that interact with programs "expecting" them to behave as planned and reporting when and wherever they fail.

In our case, Expect is used to interact with Telnet, FTP and their secure counterparts, SSH and SFTP to establish connections with remote servers and get textual information or files from them.

Official website: http://expect.nist.gov

Sybase PowerDesigner

PowerDesigner is an system modeling tool. It helps in several aspects of project design and modeling. At Imagine Broadband, it is used to model the conceptual database schema of the different products and generate the physical schema SQL scripts that are used to effectively create the databases in Oracle database servers.

Official website: http://www.sybase.com/products/enterprisemodeling/powerdesigner

Microsoft Visual Source Safe

The Version Control System from Microsoft is used to keep and manage the versions of all the Imagine Broadband products. It stores and backups all the versions of a project and its files. It also controls concurrent accesses to a file, not allowing more than one user to edit one same file simultaneously. Any older version may be recovered at any time.

Visual Source Safe also provides good version comparison functionalities – it offers various methods that help the user to find what has changed from one version to another.

Official website: http://msdn.microsoft.com/ssafe

Adobe Photoshop

This most famous photo-editing software was just used to put together the Imagine Broadband OSS Database Populator program logo, which uses a pre-existing company logo.

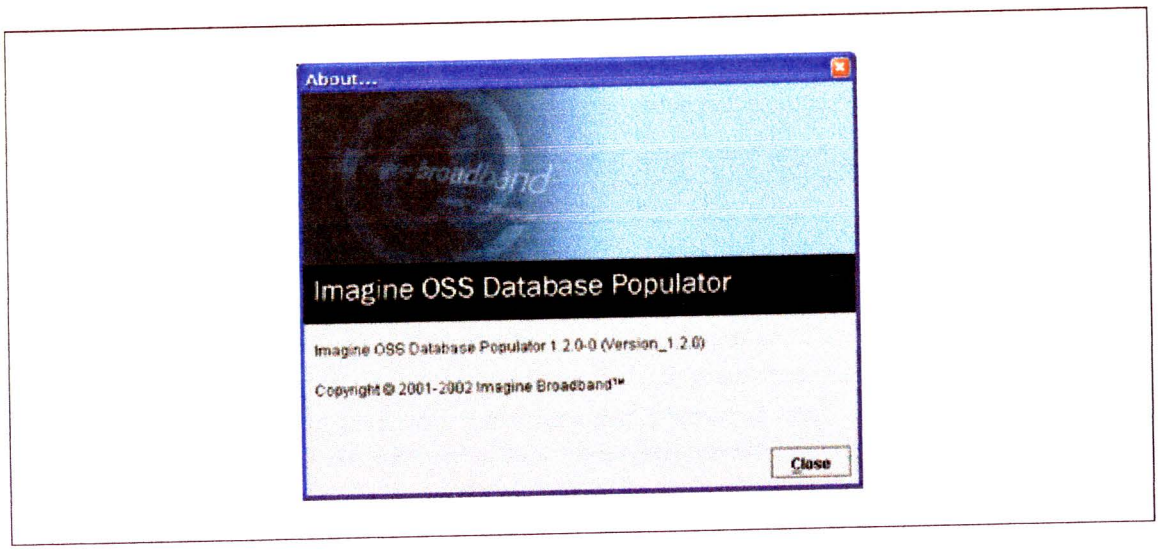


Figure 3-23 – IB OSS DB Populator Logo

Official website: http://www.adobe.com/products/photoshop

4 Conclusions

Here, we refer the conclusions that can be drawn from each internship project and from the internship experience as a whole.

4.1 The Helpdesk Project

The first project of the internship wasn't much of a challenge. The intern already had some experience in the area of web development, including professional work. Nevertheless, it wasn't easy. The project was very ambitious and presented itself as being of considerable dimensions and a certain complexity. Initial time restrictions prevented a careful planning and structuring of the system before effective development would take place. The hunger for working functionalities as soon as possible resulted in a badly structured system, difficult to improve. When time restrictions loosened up, the intern dedicated time just to fix these design issues. It made the system more modular and reusable in its key parts, but there is still work to be done in this area.

The development was, despite all, a success. All the requirements were met and the initial expectations were even surpassed at the end of the first phase. The project enabled the intern to consolidate his knowledge and experience about web development and related activities.

4.2 The Common Package Project

The second project was a totally different experience. Apart from the initial work of improving an application, the rest of the internship was spent on maintenance work – new requirements / problems would arrive from clients at any moment and they had to be met / corrected usually in a few days. It was a job more hard than interesting. One big difficulty was the lack of documentation about the applications that were being maintained. In many cases, to effectively understand what the applications did, code analysis had to be done which only delayed the process.

Nevertheless, the results are again positive. The improved application is recognizably better, more functional and more stable, as all the discussed improvements and more were executed. The maintenance tasks on the other applications from the Common Package have always been performed successfully, with more or less difficulty.

This project made the intern experience the difficulties of client oriented work. Short deadlines and days working late were not rare. Technically, the intern was able to regain experience with Java, database (SQL) and network programming, after some college projects in the area.

4.3 Global Appreciation

This has been a peculiar internship. Unlike the vast majority of the internships, which comprise only one project that is planned from the first to the last day, this one consisted of two projects, being the tasks of the second based on client and company needs rather than a plan. The intern was a company human resource, like any other regular employee of the company at the same level and treated accordingly, being allocated to the projects and tasks that the company needed to be performed or advanced at each moment. Although this has

made the internship perhaps a bit more difficult and amorphous, it has surely provided an experience as close as possible to the real corporate work environment.

The experience was very positive. The intern integrated successfully in the company and in the way the company works. Proving that is the satisfactory work developed by the intern, having answered successfully to all the proposed tasks.

5 Bibliography

1. Gregory Collins - Help Desk Specification v. 0.5
   Imagine Broadband, 2002/03/18.
2. Gregory Collins - Phase 1 Tech Support Details
   Imagine Broadband, 2002/03/19.
3. Aldric van Gaver, Thomas Parle, Pedro Pontes – Populator Detailed Design v. 1.3
   Imagine Broadband, 2002/07/23.
4. Aldric van Gaver, Thomas Parle, Pedro Pontes – Populator User's Guide
   Imagine Broadband, 2002/07/19.
5. Joseph Gedeon – ScriptRunner User's Guide
   Imagine Broadband, 2002/04/08.
6. Aldric van Gaver, Edouard Sabourin, Serge Petit, Pedro Pontes – Insfrastructure Project Detailed Design v. 1.4
   Imagine Broadband, 2002/09/10.
7. Pierrick Hascoet – Lease Server Detailed Design v. 1.1
   Imagine Broadband, 2002/02/27.
8. João Pascoal Faria – Acetatos sobre UML
   Faculdade de Engenharia da Universidade do Porto, 2001.

Appendix A - Imagine Broadband Support Plans

A.1 Gold Service

Description	Offering
Service hours On-site support Technical Telephone Support Email/Fax/Web	 Separate contract 24 hrs x 7 days 24 hrs x 7 days
Response Times Service affecting (critical) Non-Service affecting (major and minor)	 Customer will operate Level 1 diagnosis. Customer will telephone hotline (Level 2) after initial diagnosis and completion of a service investigation request (SIR) form. Imagine broadband will respond within 20 minutes and will provide regular updates on the investigation every two hours until a solution or work around has been implemented for the product. Customer will operate Level 1 diagnosis. Customer will telephone hotline (Level 2) initial diagnosis and completion of a service investigation request form. Imagine will respond within 2 days of receiving the SIR. Imagine will provide regular updates on the investigation every business day (for major requests) and three business (for minor requests) days until a solution has been provided (or made available).
Reviews and reports Reviews Reports	 Operational review every 3 months (1 day each) Activity report for each month
Training	Token for 10 man-days of training provided for the maintenance year.

A.2 Silver Service

Description	Offering
Service hours On-site support Technical Telephone Support Email/Fax/Web	 Separate contract 0900 to 1800 hrs each day. This will apply to local Territory time only.
Response Times Service affecting (critical)	 Customer will operate Level 1 diagnosis. Customer will telephone hotline (Level 2) after initial diagnosis and completion of a service investigation request (SIR) form. Imagine broadband will respond within 20 minutes during

Description	Offering
Non-Service affecting (major and minor)	service hour's support. Will provide regular updates on the investigation every four hours until a solution or work around has been implemented for the product. *Customer will operate Level 1 diagnosis.* Customer will telephone hotline (Level 2) initial diagnosis and completion of a service investigation request form. Imagine will respond within 2 days of receiving the SIR. Imagine will provide regular updates on the investigation every second day (for major requests) and each week (for minor requests) until a solution has been provided (or made available).
Reviews and reports Reviews Reports	Operational review every 6 months (1 day each) Quarterly activity report provided
Training	Token for 5 man-days of training provided for the maintenance year.

A.3 Bronze Service

Description	Offering
Service hours On-site support Technical Telephone Support Email/Fax/Web	Separate contract Separate contract 0900 to 1800 hrs. Excludes public holidays and weekends. This will apply to local Territory time only.
Response Times *Service affecting (critical)*	*Customer will operate Level 1 diagnosis.* Customer will email/fax Imagine within the business day after initial diagnosis and completion of a service investigation request (SIR) form. Imagine broadband will respond within the next business day and will provide regular updates on the investigation every four hours during the business day until a solution or work around has been implemented for the product.
Non-Service affecting (major and minor)	Customer will operate Level 1 diagnosis. Customer will email/fax Imagine after initial diagnosis and completion of a service investigation request form. Imagine will respond within five business days of receiving the SIR. Imagine will provide regular updates on the investigation every third business day (for major requests) and each week (for minor requests) until a solution has been provided (or made available).

Description	Offering
Reviews and reports Reviews Reports	 An operational review will be carried out. An activity report will be produced every six months

Appendix B - IB OSS DB Populator GUI User's Guide

B.1 Introduction to the Imagine Broadband DB Tool 1.2

The goal of this program is the creation and population of DB instances. It consists of a set of applications that read data from text files, analyze it and perform operations on tables, like creation, data insertion and deletion.

Version 1.2 now includes a user-friendly graphical interface, which allows you to control the execution process and manage the output logs in a very easy and intuitive manner.

B.2 Starting the Program

The PopulateDB executable offers a very easy way to control the execution of a series of scripts that perform actions in a database instance. The set of these actions may consist in a creation of a DB schema or its population, for example. Each series of scripts to be run is called a "scenery". The program syntax is the following:

```
java PopulateDB <data_dir> <config_file> <scenario> <scripts_dir>
```

where:

data_dir is the directory where the files holding the data to be inserted in the database are;

config_file is the path to the XML configuration file;

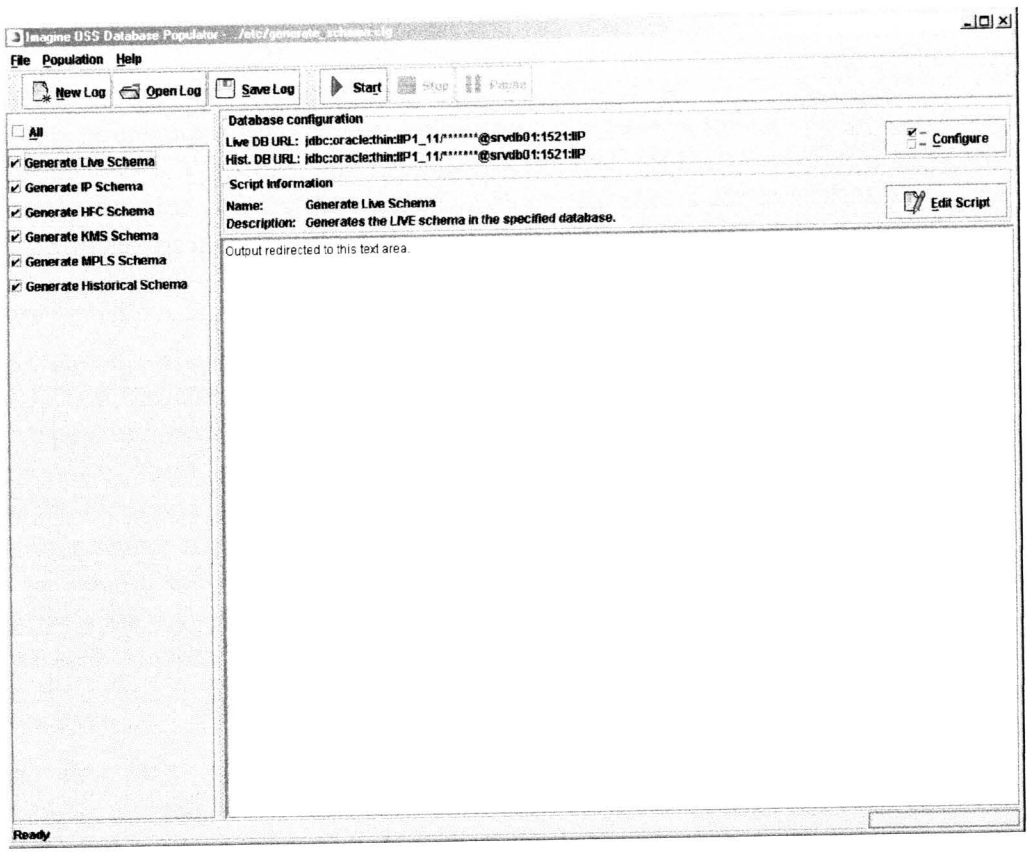
scenario is a file that contains the name of the script files to be included in the execution process, one per line;

scripts_dir is the directory where the scripts specified in <scenario> are.

In the "bin" directory you can find batch files or shell scripts that call PopulateDB with different scenarios depending on the goal of the process.

B.3 Main Window

The following is a sample of the main window of the application:



The main window is composed of several components:

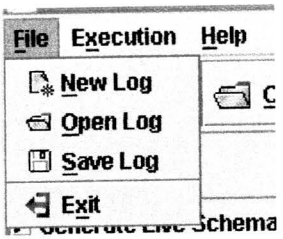
- The menu bar, providing access to all the functionalities of the application;
- The toolbars, for quickly controlling logs and the execution of the scripts;
- The scenario panel, in the left, displaying the names of all the scripts that compose the specified scenario and enabling the user to check if they should be run in the execution or not, through the checkboxes;
- The database configuration and script information/edition panel, in the upper right area of the window. It displays the current database configuration parameters and provides a way to change them. It also displays the selected script name and description and provides a way to view and edit the script (only if the program is running in advanced mode);
- The large output log window, where the results of the execution of the scripts are presented;
- The status bar, displaying information about the status of the scenario execution.

These components will be detailed in the following sections.

Menu Bar

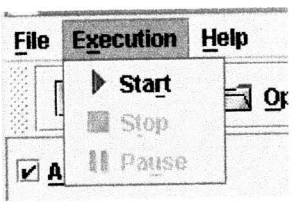
The menu bar is composed by three menus: the File menu, the Execution menu and the Help menu.

The File Menu



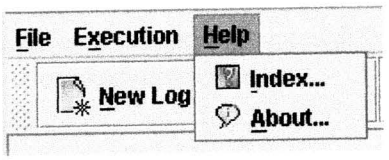
In the file menu, you can manage your result logs. There is the possibility of creating a new (empty) log, opening a log file or saving the current log to a log file. The log files are simple text files. As with all the standard user interfaces, the File menu also allows you to quit the program by choosing "Exit".

The Execution Menu



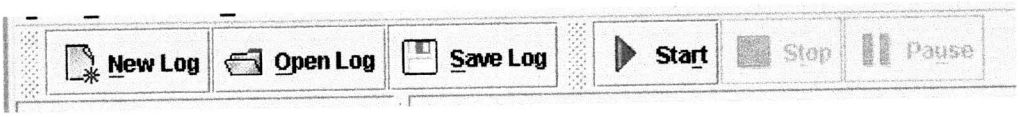
The execution menu offers the commands to control the execution of the scenario (set of scripts). You can start, stop or pause the execution by choosing their respective menu items. Note that when pausing or stopping the execution, there may be a considerable delay between your action and the actual pause. This is due to the fact that the program waits for a secure zone to pause or stop the execution. By secure zone we mean a script execution zone where the database is in a consistent state and no resource locks are acquired. The "Stop" and "Pause" menu items become enabled when the execution is started.

The Help Menu



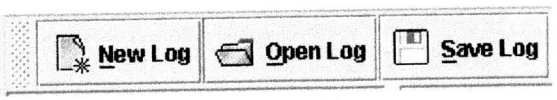
Again, the Help menu follows the standard GUI conventions, providing access to online help, by choosing the "Index..." menu item and information about the product, when choosing the "About..." menu item.

Toolbars



The menu bar is composed by three menus: the File menu, the Population menu and the Help menu.

The Log Toolbar



In the Log toolbar, you can rapidly access functionalities that manage the execution output logs. You can create a new (empty) log, open an existing log file or save the current log to a log file. The log file is a simple text file.

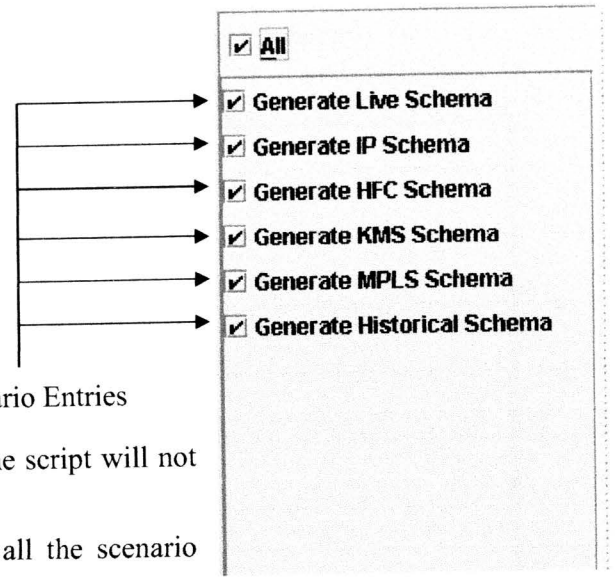
The Execution Toolbar

The Execution toolbar provides quick access to the execution control. There is a button for starting the execution, one for stopping it and an extra one to pause or resume the execution of the set of scripts of the scenario. Note that when pausing or stopping the execution, there may be a considerable delay between your action and the actual pause. This is due to the fact that the program waits for a secure zone to pause or stop the execution. By secure zone we mean a script execution zone where the database is in a

consistent state and no resource locks are acquired. The "Stop" and "Pause" buttons become enabled when the execution is started.

Scenario Panel

The scenario panel displays the scripts that compose the scenario specified when calling the program. Each scenario entry corresponds to a script. Each entry has a checkbox that controls whether the respective script should be executed or not. When it is checked, the script will be executed, when it is not checked, the script will not be run.

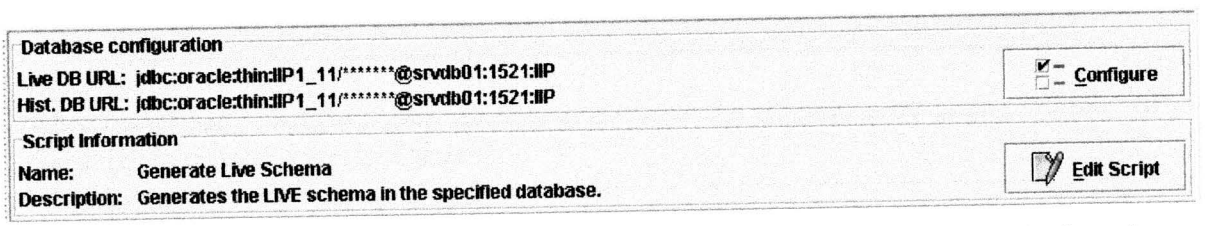


Scenario Entries

The "All" checkbox controls the state of all the scenario entries checkboxes.

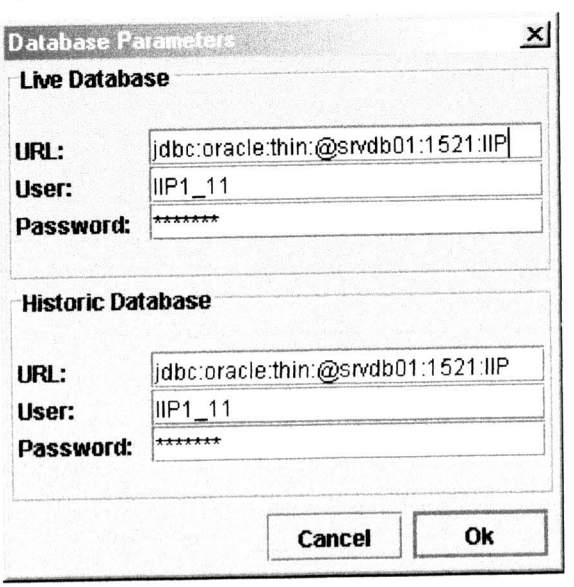
When the execution of the scenario takes place, all the scenario entries that are successfully executed turn to green, as the other that were not successfully populated turn to red.

Database Configuration/Script Information / Edition Panel



This panel displays the current database configuration of the program. It also displays the name and the description of the currently selected script.

By choosing the "Configure" button, the user is presented with the presented configuration dialog, in which he/she can alter the configuration parameters, namely the URL, User and Password for the Live and Historical databases.



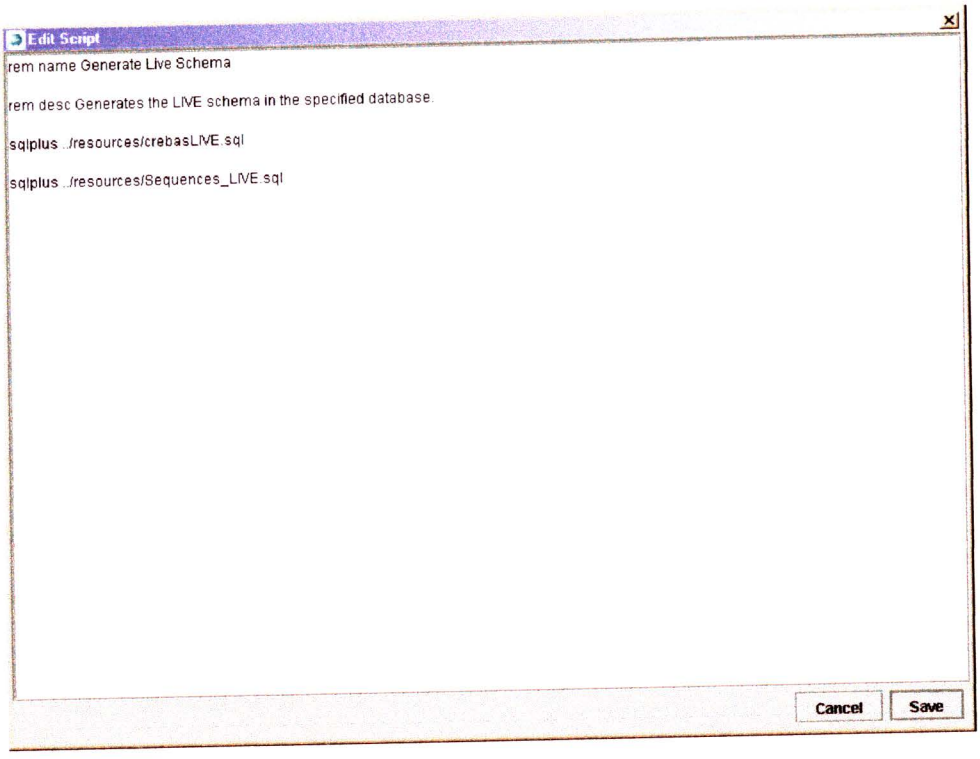
Note that the URL should **NOT** feature the login and password. These values should be inserted in their respective text boxes.

Also, have in mind that the alterations take effect for the current program execution only. The next time the program is executed, the configuration parameters are read from the XML configuration file.

By hitting the "Edit Script" button, you can view and edit the selected script file. To have access to this functionality, you must be

running the program in "Advanced" mode. See "Chapter 4 - The XML Configuration file" for more details on how to run the program in "Advanced" or "Normal" mode.

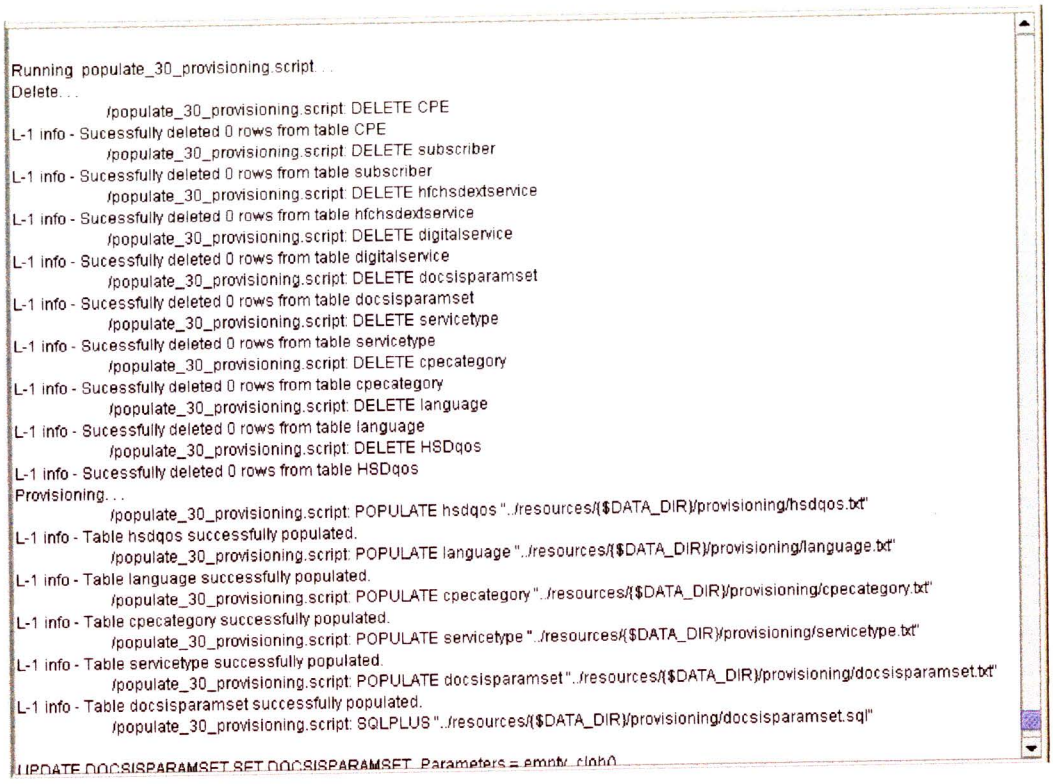
The following is an image of the script editor displaying a sample script:



After making any changes to the script, you can save the alterations by hitting "Save". To discard your updates, choose "Cancel".

Output Log Panel

This panel displays the outcome of the execution process. The text displayed is the one created by the applications that are called by the scripts. It is, thus, the same as you would get if you run one of these applications separately.



The text in the output text area may be cleared or saved using the "File" menu or the Log toolbar. Please refer to sections 0 and 0.

Status Bar

The status bar displays information about the status of the execution. It also shows a progress bar that provides a visual indicator to the progress of the execution of the scenario.

When errors occur, so that the user does not miss them, the status bar displays a warning, like depicted in the following image:

Appendix C - Lease Server Packet Manipulation Specification

C.1 Introduction

This document intends to describe the functioning of the Imagine Broadband LeaseServer product. It specifies how LeaseServer builds the packets that are sent to query the CNR and how it processes and parses the information in the respective response packets, to be sent back to the client application. For a presentation and more information on LeaseServer, please refer to "LeaseServer Detailed Design".

C.2 Sending a Query to the CNR

When LeaseServer receives a lease/reverse lease query from a client application, it creates a new "DHCP Message" (packet) to be sent to the CNR. The following sections detail the packet that is created and sent.

Sending a Lease Query

In the case of a lease query, when we want to know the MAC address associated to some IP address, the DHCP packet to be sent to the CNR is created with the following data:

Field	Length (octets)	Value
op (message op code)	1	1 (BOOTREQUEST)
htype	1	0
hlen	1	0
hops	1	0
xid (transaction ID)	4	<transaction number generated by application>
secs	2	0
flags	2	0
ciadr (client IP address)	4	<specified client IP address>
yiaddr	4	-
siaddr	4	-
giaddr (relay agent IP address, to which the response will be sent)	4	<IP address of the machine where the LeaseServer is running>
chaddr (client MAC address)	16	0 (this field is set to 0 – it's the information we're trying to get from the CNR)
sname	64	-
file	128	-
option 53	1	13
option 55	3	51, 82, 3 (values of each byte)

After the creation of this packet, the LeaseServer sends it to the CNR and stores the used transaction ID. It then waits for a response from the CNR respecting that transaction ID (and other eventual pending responses).

Sending a Reverse Lease Query

As for a reverse lease query, when we want to know the IP address leased to a specific MAC address, the DHCP packet to be sent to the CNR is created with the following data:

Field	Length (octets)	Value
op (message op code)	1	1 (BOOTREQUEST)
htype	1	1
hlen	1	6
hops	1	0
xid (transaction ID)	4	<transaction number generated by application>
secs	2	0
flags	2	0
ciadr (client IP address)	4	0.0.0.0 (this field is set to 0 – it's the information we're trying to get from the CNR)
yiaddr	4	-
siaddr	4	-
giaddr (relay agent IP address, to which the response will be sent)	4	<IP address of the machine where the LeaseServer is running>
chaddr (client MAC address)	16	<specified client MAC address>
sname	64	-
file	128	-
option 53	1	13
option 55	3	51, 82, 3 (values of each byte)

After the creation of this packet, the LeaseServer sends it to the CNR and stores the used transaction ID. It then waits for a response from the CNR respecting that transaction ID (and other eventual pending responses).

C.3 Receiving the Response from the CNR

The LeaseServer receives incoming answer packets from the previously queried CNR's. For each one of them, the program gets its transaction ID that identifies the client application to which the LeaseServer must, in its turn, respond with the parsed information from the received packet. The following sections detail the procedure in the case of a response to a lease query and a reverse lease query.

Response to a Lease Query

After retrieving the transaction ID from the received packet, the program gets information about the query itself, namely, if it was a lease query or a reverse lease query. In the first case, LeaseServer parses the MAC address from the answer packet, by converting the binary values to hexadecimal values and separating each pair of MAC address components with the character ':', so that the address is represented in its standard human readable form (XX:XX:XX:XX:XX:XX) where XX is an hexadecimal value.

If the returned MAC address is 00:00:00:00:00:00, i.e. invalid, then the program marks the result as having received an invalid packet, otherwise additional parsing follows as described next. In the case of a correct packet, the MAC address is stored in the result entity to be sent to the querying client application.

Then it's time to parse the option value 82 eventually present in the received response packet. This option stores information about a relay agent that may be present or not. Suboption 02 contains the MAC address of that relay agent. LeaseServer runs through the option 82 looking

for suboption 82. When and if it finds it, it is parsed, using the same process as referred above for the MAC address of the client, and stored in the result entry.

Finally, LeaseServer searches for the option 3, which contains the IP address of the CMTS. If it is present, it is converted from binary representation to the common human representation form (xxx.xxx.xxx.xxx), where each xxx is a decimal value between 0 and 255. It is then also stored in the result entity.

To summarize, the following table represents the response entity, the values it holds and their origin:

Field	Value	Origin
0 (received packet is valid or invalid)	"false" -> invalid; "true" -> valid.	chaddr field in the received packet. If the MAC address is 00:00:00:00:00:00, then the received packet is invalid otherwise it is valid.
1 (client MAC address)	The client MAC address, represented in hexadecimal form (XX:XX:XX:XX:XX:XX).	chaddr field in the received packet.
2 (Relay agent MAC address)	The relay agent MAC address, represented in hexadecimal form (XX:XX:XX:XX:XX:XX).	Suboption 02 of option field nr 82 in the received packet.
3 (CMTS IP address)	The IP address of the CMTS, in decimal representation (xxx.xxx.xxx.xxx).	Suboption number 3 in the received response packet.

This entity is then sent to the client application that asked for the query and is associated with the current transaction ID.

Response to a Reverse Lease Query

If this response regards a reverse lease query, LeaseServer proceeds by parsing the client IP address from the response packet. It fetches the value of the ciaddress from the packet and converts it into decimal and dot representation (xxx.xxx.xxx.xxx), where each xxx is a decimal value between 0 and 255. It then associates this value with the result entity to be sent to the querying client application.

Then the same process is used to fetch and convert the IP address of the eventually present CMTS, in this case getting this value from the option number 3 of the received response packet. After parsing the eventually present data, it is associated to the result entity.

To summarize, the following table represents the response entity, the values it holds and their origin:

Field	Value	Origin
0 (received packet is valid or invalid)	"true" -> valid.	In this case, it is always valid, if a packet has been received and there was a corresponding application waiting (associated with the transaction ID present in the packet).
1 (client IP address)	The client IP address, represented in decimal dot form (xxx.xxx.xxx.xxx).	ciaddr field in the received packet.
2 (Relay agent MAC address)	Not set.	N/A
3 (CMTS IP address)	The IP address of the CMTS, in decimal representation (xxx.xxx.xxx.xxx).	Suboption number 3 in the received response packet.

This entity is then sent to the client application that asked for the query and is associated with the current transaction ID.

Appendix D - What is the "Imagine Instant Provisioning" Product?

This is a transcription of the "Imagine Instant Provisioning" product presentation from the Imagine Broadband website – http://www.imaginebroadband.com.

D.1 Fundamentally Changing Marketing for Large Scale Network Operators

Most provisioning software is extremely complex and requires significant modification to enable successful integration of a new service. Marketing is therefore invariably tied to lengthy schedules and high implementation costs. Imagine Instant Provisioning™ overcomes this and reduces both the time and cost of deployment and operations management.

Central to Imagine Instant Provisioning™'s effectiveness is its generic, XML configurable, back-end. This means that all the actions for activating and provisioning a service can be encapsulated by script instructions. Consequently, development times are measured in hours and days rather than weeks and months.

Imagine Instant Provisioning™ also simplifies the system interface with easy to customize web based front-ends for customer service representatives, service management engineers and customers. It enables secure service activation and customer self provisioning, and it has the inherent scalability to handle an ever increasing customer base.

Speeds the Introduction of New Services

With its XML scripting tool, Imagine Instant Provisioning™ dramatically reduces the time to create and provision new or enhanced services. Even an entirely different service requiring new business logic and network elements can be rolled out in less than a month.

Enables Cost Effective Trial Marketing

Imagine Instant Provisioning™ allows you to cost effectively trial a new service or package within selected geographic regions. Your trial can be as small as one segment of HFC or one DSLAM line card.

Supports Multiple Provisioning

Imagine Instant Provisioning™ supports provisioning of multi-tiered services, ISP services such as e-mail and chat, and different technologies such as interactive TV including all industry middlewares. It also supports open access models including DSL, cable, ETTH, FTTH and wireless from the outset, and allows plug and play provisioning of multiple CPEs.

Drives Business Expansion

Business expansion, whether by organic growth, merger or acquisition, is simplified by Imagine Instant Provisioning™'s vendor agnostic, robust and highly scalable architecture. Scalability has been fully tested for both front-end and back-end transactions.

Cuts Costs Through Customer Self Provisioning

Imagine Instant Provisioning™ significantly reduces provisioning and operations costs through integration of a web based customer plug and play self provisioning system for all broadband and large scale network services including high speed data, TV mail, broadcast TV, video, IP Virtual Private Networks and voice. It does not require MAC addresses from the modem or the PC at home and allows both tiered and packaged services to be self provisioned.

Eliminates Fraudulent Use

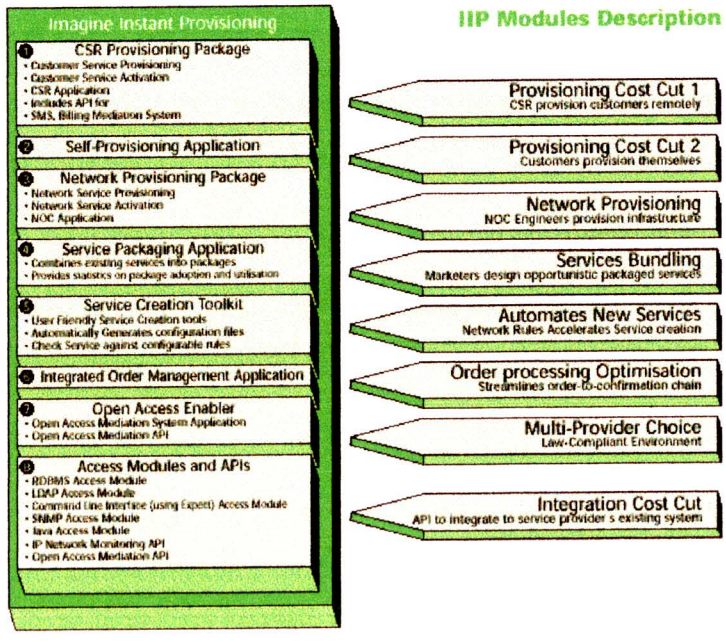
Advanced security features eliminate revenue loss through fraudulent use. Customers are automatically directed to the self provisioning system to provision a new modem, to move the modem from one area to another and use it, to change the service or to subscribe to new services.

Supports Real Time Viewing of Service Standards

Imagine Instant Provisioning™ will integrate with Imagine Service Assurance™ to give a real time customer view of service delivery quality standards.

D.2 Comprehensive Core Functionality

Imagine Instant Provisioning™ is a modular, out of the box, software solution. The core product consists of the following modules and capabilities:



IIP Kernel

Heart of the provisioning system, providing transactional monitoring, load balancing / redundancy, queuing and recovery. Open API SMS IP mediation and billing systems enabling fast integration in existing business critical applications.

Customer Service Provisioning

Assesses whether a particular service can be supplied to the subscriber and, if so, reserves and assigns the appropriate network resources prior to subscriber provisioning. There is one module per type of service, eg. High speed data over HFC, High Speed Data over DSL, etc. There are a large number of modules developed for different types of services. The system allows for operators to create their own modules in an extremely short period of time.

Customer Service Activation

Enables a subscriber service by configuring the appropriate network and infrastructure elements. There is one module per type of service, eg. High speed data over HFC, High Speed Data over DSL, etc. There are a large number of modules developed for different types of services. The system allows for operators to create their own modules in an extremely short period of time.

Customer Service Representative Application

Web based application enabling CSR to perform end to end subscriber provisioning and service activation in a secure, fraud resistant manner.

D.3 Wide Range Of Tools

In addition to the core product, Imagine Instant Provisioning™ recognises and responds to the varying needs of broadband and large scale network service businesses and operators by offering a comprehensive range of easy to deploy specialist modules:

Service Creation Toolkit

Simplifies the creation of new business rules through the use of XML scripts / templates and a rule editor. Rules can include subscriber provisioning, operator platform provisioning, subscriber activation, operator platform activation and service quality assurance.

Network Provisioning Package

Allows quick and easy deployment of new services on the existing platform. The Platform Service Provisioning Back-end checks whether the network and infrastructure can support the new service, while the Platform Service Activation Back-end configures the resources to enable the new service.

Self Provisioning Application

Web based application enabling subscriber end to end self provisioning and activation in a secure, fraud resistant manner. Offers significant savings in operational expenses and truck roll costs.

Service Packaging Application

Web based application allowing service managers to create service packages, enable/disable service by location and change service parameters. Greatly simplifies the creation of promotional packages and flexible service pricing.

Open Access Mediation Module

Gives you the potential, through your own access network, to offer subscribers a direct route to third party service providers offering products such as information, entertainment and connectivity.

IP Virtual Private Networks (IP VPN) Module

Providing for business users the most powerful IP VPN provisioning and self provisioning functionality and flexibility over varying IP infrastructure.

Appendix E - What is the "Imagine Service Assurance" Product?

This is a transcription of the "Imagine Service Assurance" product presentation from the Imagine Broadband website – http://www.imaginebroadband.com.

E.1 Real Time Monitoring and Control

Large scale network infrastructures are typically managed using a component orientated view of the environment. Within this, if a component fault occurs, operations staff can trace the problem to that individual component and instigate a remedy. The trouble is that such an approach gives no indication of the impact of the fault on the quality of customer service. As a result, there's no way of telling in real time how your business is performing. Imagine Service Assurance™ eliminates that weakness.

Imagine Service Assurance™ software provides a complete view of the infrastructure and has all the tools necessary to actively manage components. But, in addition, it utilises innovative Imagine Service Emulator Agent (I-SEA™) technology to emulate a customer view of how the system is actually performing. If a component develops a problem, you can immediately see the impact on service quality.

Enhances Infrastructure Management

Imagine Service Assurance™ enables you to monitor in real time infrastructure performance, security and service quality.

By co-ordinating information from the Imagine Service Emulator Agents, component alarms and correlation rules it identifies both the source and service impact of any problem. When integrated with Imagine Instant Provisioning™, it also identifies individual customers affected and so helps set priorities when actioning the remedy.

Streamlines Component Management

Imagine Service Assurance™ is modular and has all the tools necessary to manage and monitor each system component. It also allows other tools of choice that may already be in use to be integrated through open APIs or middleware products.

Simplifies Management and Monitoring

Infrastructure management is simplified by graphical user interfaces that are customisable, multi-language, and can be easily re-branded. The network operator's interface is Java based and there are web based interfaces for customer service representative, service managers and company executives with easily tailored information according to the audience and business processes.

Reduces Operating Costs

A context sensitive knowledge management system gives instant access to problem resolution procedures. This, along with the ability to prioritise activities according to customer importance, greatly simplifies fault remediation and so speeds time to resolution.

Ensures Performance Quality

With its innovative Imagine Service Emulator Agents, Imagine Service Assurance™ allows you to continually monitor and adjust components to maintain consistent delivery of your optimum service standards.

Maximizes Service Quality Achievement

Imagine Service Assurance™ enables comprehensive monitoring of performance relative to service level agreements. It does this through an advanced Service Level Agreement (SLA) Management tool that utilises an exact view of service quality provided by the Imagine Service Emulator Agents.

Improves Customer Satisfaction Levels

Imagine Service Assurance™ integrates its web based CSR user interface with the Imagine Service Emulator Agents. This allows precise monitoring and control of service standards by customer service representative (CSR) which, in turn, leads to consistently higher customer satisfaction levels and reduced customer churn.

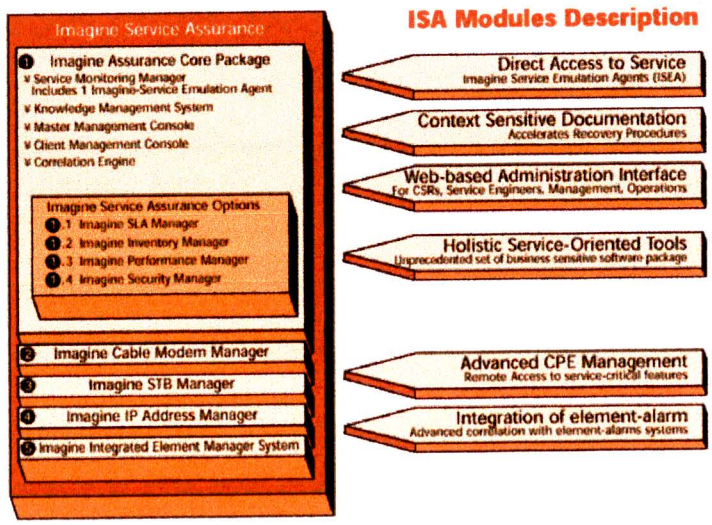
Supports Business Development

Imagine Service Assurance™ is vendor agnostic. It provides a stable platform allowing for newly developed services or heritage services gained through merger or acquisition to be more easily deployed and effectively managed.

E.2 Comprehensive Core Functionality

Imagine Service Assurance™ is a modular, out of the box, client server application. The core product enables the modules to interact with each other and provides memory, transaction and database management.

It consists of the following modules:



Service Monitoring Manager

Web server that presents a unified view of the status of your network by compiling results from all the agent engines.

Imagine Service Emulator Agent

Runs sophisticated service specific scenarios emulating end user activity. Because services are different by nature (for instance video is different from high speed data) the I-SEA™s are also different.

Knowledge Management System

Provides context sensitive access to the exact page of vendor/supplier documentation, manuals, component and service alarm class definitions as well as recovery procedures.

Master Management Console

Provides a graphical view of the operator platform with detailed views of the service and component alarms.

Client Management Console

A simplified version of the Master Management Console GUI that presents only the service related alarms.

Correlation Engine

Correlates service alarms with component alarms to identify components responsible for failure and also determine the subscribers impacted.

E.3 Wide Range of Tools

In addition to the core product, Imagine Service Assurance™ recognises and responds to the varying needs of large scale network service businesses by offering a comprehensive range of modular tools that are fully integrated, out of the box, and require no system integration efforts.

Imagine SLA Management Product

Verifies that subscriber usage is in accordance with the service contract. It measures Key Performance Indicators (KPIs), compares measured KPIs with targeted KPIs and generates service level assurance reports.

Imagine Inventory Management Product

Maintains the inventory of all the infrastructure equipment including hardware and software. Can be used with the Imagine Element Manager System to support configuration management.

Imagine Performance Management Product

Monitors performance from both the subscriber and platform view.

Imagine Security Manager

Monitors security from both the subscriber and platform view.

Imagine Cable Modem Manager

Enables management of the subscriber cable modem using the MAC address, IP address of PC or the subscriber name. The modem is monitored in terms of activation/deactivation, remote operation/maintenance (Cable Modem reboot, SNR measurement, Transmit power

level, etc), performance and bandwidth (downstream and upstream utilisation, power ranging rate, etc), and inbound and outbound data.

Imagine DSL Modem Manager

Enables management of the subscriber DSL modem using subscriber phone number, IP address of PC, or the subscriber name. The modem is monitored in terms of activation/deactivation, remote operation/maintenance (admin and operational status, upstream and downstream speed, the SNR margin (DS and US), Attenuation, Interleave delay, transmit power), performance and bandwidth, and inbound and outbound data.

Imagine Set Top Box Manager

Enables management of the subscriber STB using the MAC address, IP address or the subscriber name. The STB is monitored in terms of:

- loss of forward path signal including loss of video, audio and teletext
- loss of return path signal
- error in PSI/SI
- loss of ECM and EMM processing
- loss of IP address
- application errors
- code downloads
- freezing pictures
- performance of processor, video chip and memory
- changes to configuration settings
- performance of Ethernet port
- quality of service available in the DOCSIS (bandwidth)
- software version present on the STB
- STB hardware version

Imagine IP Address Management Product

Manages IP addresses from both the subscriber and platform view. Imagine IP Address Management can provide historical usage, RIPE reporting and sophisticated DNS management functions.

Imagine Integrated Element Manager System

Collates and manages all platform component alarms. Imagine Service Assurance™ interfaces with all the industry's main Manager of Managers.

Imagine Service Emulator Agents

This exclusive I-SEA™ technology executes the advanced proactive monitoring on which Imagine Service Assurance™ is based.

These agents emulate end-user activity and measure in return the specific parameters related to the actual delivery of the service at the end-user side.

Because all services are different, there is a different I-SEA™ agent for each service.

- Imagine Service Emulator Agent for High Speed Data over HFC
- Imagine Service Emulator Agent for High Speed Data over DSL
- Imagine Service Emulator Agent for High Speed Data over Ethernet
- Imagine Service Emulator Agent for Digital Television
- Imagine Service Emulator Agent for Interactive Television Monitoring
- Imagine Service Emulator Agent for IP Telephony
- Imagine Service Emulator Agent for consumer or business customer applications such as IP VPN

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000065320