Universidade do Porto
Faculdade de Engenharia
FEUP

Luís Albino Nogueira Ramos

# Productization of Packaging Procedures and Tools
## at Imagine Broadband

LEIC
1/RAMI

LEIC
Porto, 2002

LEIC Internship Report 2001/02

# CONFIDENTIAL REPORT

## Productization of Packaging Procedures and Tools At Imagine Broadband SARL

*Luís Albino Nogueira Ramos*

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA

FEUP coordinator: Prof. André Puga

Imagine Broadband coordinator: Pierre Walter

**Faculdade de Engenharia da Universidade do Porto**

**Licenciatura em Engenharia Informática e Computação**

2002-10-08

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 65326
CDU 004(047.3)
Data 14 / 3 / 2003

*To those who will never dedicate a work to themselves [35]*

**Abstract**

This document reports the work done during the internship of Luis Ramos at Imagine Broadband since March 2002 during six months.

In order to maintain the required high level standards of quality, Imagine Broadband need a quick and effective cycle of development. To achieve this, the company needs to continuously update and improve its internal methods of production (design, development, packaging and integration/testing).

The main theme of the internship was software packaging process automation. The productization of the packaging procedures and tools consists in the automation of the process of creating deployable packages. These packages are constructed by the transformation of development projects (source code and additional files) into ready to delivery packages.

The introduction makes a quick overview of all the context of the internship as of its content.

After describing the structure of a package and the procedure to construct it this document detail the tools available in the beginning of the internship to enhance the packaging process. The most important part of the document consists in the description of the new tool: "Imagine Broadband Packager" and the result analysis that makes a comparison between the former packaging system and the new packaging system represented by the new tool.

This new tool is a Java application that fully automated the packaging process.. Firstly, the work consisted in the translation of the available tools from Shell scripts to Java. The implementation of a Graphical User Interface was done and soon the new tool encapsulated the totality of the packaging system.

Another important topic in this document is the description of the Configuration Management Tools developed during the internship. The complete development process was followed and documented. The brief work done during the internship in the Integration/Testing team is also described in this document. Moreover, the presentation of used technologies may contain interesting information regarding the technologies involved.

The major virtue of this internship resides in the developed packaging tool. This tool has enhanced the packaging process enormously. The major advantages brought by the new tool are related to process control, speed and automation that led to a high increase of productivity. With it, the time consumed with packaging duties was significantly reduced.

**Acknowledgements**

Many thanks to all the people at Imagine Broadband for the support and teaching; among others, Olivier Brie, Eric Duchet and Gilles Combe. Special thanks to Pierre Walter for the transmission of his expertise and to Loic Rioual for his, beyond professional, support.

Thank you very much to professor André Puga for his help in performing this final step of my degree and professor Raul Vidal representing *Faculdade de Engenharia da Universidade do Porto* to which I owe a lot.

To my family and friends that never complained about their ever increasing phone bills.

Sara.

## Glossary

| Term or Abbreviation | Definition |
|---|---|
| CIFS | Common Internet File System |
| CMTS | Cable Modem Termination System |
| CNR | Cisco Network Registrar |
| CRM | Costumer Relationship Management |
| DHCP | Dynamic Host Configuration Protocol |
| DOM | Document Object Model |
| DSL | Digital Subscriber Line |
| FTP | File Tranfer Protocol |
| GUI | Graphical User Interface |
| HFC | Hybrid Fibber Coaxial |
| HTML | Hyper Text Markup Language |
| IIP | Imagine Instant Provisioning |
| IP | Internet Protocol |
| J2SE | Java 2 Standard Edition |
| JAR | Java Archive |
| JFC | Java Foundation Classes |
| JRE | Java Runtime Environment |
| LDAP | Lightweight Directory Access Protocol |
| RMI | Remote Method Invocation |
| RPM | Redhat Package Manager |
| SAX | Simple API for XML |
| SMB | Server Message Block |
| UBR | Universal Broadband Router |
| VSS | Microsoft Visual Source Safe |
| XML | eXtended Markup Language |
| | |
| Packaging | The process consisting in every action done to create a deployable package. In the context of this document should be read: Software Packaging |
| Productization | To transform in a product, in something worth selling |

## Contents

1 Introduction ..... 1
1.1 Imagine Broadband ..... 1
1.2 Imagine Broadband Packaging ..... 2
1.3 Imagine Broadband Configuration Management Tools ..... 3
1.4 Imagine Instant Provisioning Pre-Integration ..... 3
1.5 Work Plan ..... 4
2 Imagine Broadband Products and Processes ..... 5
2.1 Imagine Broadband Production Process ..... 5
2.2 Imagine Instant Provisioning ..... 6
2.3 Imagine Instant Provisioning Pre-Integration ..... 7
3 Used Technologies ..... 9
3.1 Operating Systems ..... 9
3.2 Shells ..... 9
3.3 Markup Languages ..... 10
3.4 Java 11
3.5 Microsoft Visual Source Safe ..... 12
3.6 Packaging Technologies ..... 13
3.7 Other ..... 13
4 Imagine Broadband Packaging ..... 15
4.1 Introduction ..... 15
4.2 Packages Description ..... 15
4.3 Packaging Procedures ..... 19
4.3.1 Introduction ..... 19
4.3.2 Microsoft Visual Source Safe Extraction ..... 20
4.3.3 Source Code Compilation ..... 21
4.3.4 RPM packages build ..... 22
4.3.5 Install Anywhere project ..... 24
4.3.6 Package Publication ..... 25
4.3.7 Summary ..... 25
4.4 Former Packaging System ..... 26
4.4.1 Introduction ..... 26
4.4.2 Microsoft Visual Source Safe Extraction ..... 26
4.4.3 Source Code Compilation ..... 27
4.4.4 RPM packages build ..... 27
4.4.5 Install Anywhere project ..... 28
4.4.6 Package Publication ..... 28
4.4.7 Summary ..... 30
4.5 New Packaging System: Imagine Broadband Packager ..... 31
4.5.1 Introduction ..... 31
4.5.2 Requirements ..... 31
4.5.3 Logical Architecture ..... 35
4.5.4 Object Data Model ..... 37
4.5.5 Physical Architecture ..... 44

4.5.6 Development Details ..... 45
4.5.7 Summary ..... 51
4.6 Results Analysis ..... 52
4.7 Conclusion and further work prespectives ..... 54
5 Imagine Broadband Configuration Management Tools ..... 55
5.1 Rpm Browser ..... 56
5.1.1 Introduction ..... 56
5.1.2 Requirements ..... 56
5.1.3 Object Data Model ..... 57
5.1.4 Development Details ..... 58
5.1.5 Conclusion and further work prespectives ..... 58
5.2 Rollback Tool ..... 59
5.2.1 Introduction ..... 59
5.2.2 Requirements ..... 59
5.2.3 Object Data Model ..... 61
5.2.4 Development Details ..... 62
5.2.5 Conclusion and further work prespectives ..... 64
6 Conclusion ..... 65
Links and References ..... 67
Appendix A: ReleaseNotes.xml structure ..... 69
Appendix B: Ant's build.xml automated generation ..... 71
Appendix C: RPM Specification File ..... 73
Appendix D: IIP Packaging ..... 74
Appendix E: IBL Rollback Tool Specification ..... 82

**Pictures List**

1. Imagine Broadband products life cycle ..... 5
2. Install Anywhere Commercial Package structure ..... 17
3. Screenshot: Install Anywhere installer ..... 18
4. Imagine Broadband Packaging Process ..... 19
5. Components diagram: VSS extraction schema ..... 20
6. Components diagram: Ant Build Process ..... 21
7. Screenshot: Install Anywhere project editor ..... 24
8. Screenshot: VSS Extraction Shell script ..... 26
9. Screenshot: Imagine Broadband Delivery Center ..... 29
10. Use Cases diagram: Imagine Broadband Packager ..... 32
11. Logical Packages diagram: Imagine Broadband Packager ..... 35
12. Classes diagram: Imagine Broadband Packager Logical Package GUIUtil ..... 37
13. Classes diagram: Imagine Broadband Packager Logical Package Util ..... 38
14. Classes diagram: Imagine Broadband Packager Logical Package Shell ..... 39
15. Classes diagram: Imagine Broadband Packager Logical Packages GUI and Core ..... 41
16. Components diagram: Imagine Broadband Packager Physical Architecture ..... 44
17. Screenshot: Imagine Broadband Packager Extraction Tab ..... 46
18. Sequence diagram: Executing a Shell script while building a project ..... 48
19. Screenshot: Imagine Broadband Packager RPM packages comparison ..... 50
20. Use Cases diagram: RPMBrowser ..... 56
21. Classes diagram: RPMBrowser ..... 57
22. Screenshot: RpmBrowser main window ..... 58
23. Use Cases diagram: Rollback Tool ..... 59
24. Classes diagram: Rollback Tool ..... 61
25. Screenshot: Rollback Tool main window ..... 62
26. Screenshot: Rollback Tool DiffTable ..... 63
27. File example: ReleaseNotes.xml ..... 69
28. File example: pkgNames.xml ..... 71
29. File example: RPM Specification File ..... 73

# 1 Introduction

This document is meant to describe the internship made by Luis Ramos at Imagine Broadband since the 18th of March of 2002 during 6 months. This internship consisted in an agreement between Imagine Broadband, FEUP and Luis Ramos.

This internship had three distinct themes: Packaging, Configuration Management and Integration Testing. The main themes were Packaging and Packaging Process Automation.

The presentation of the company, Imagine Broadband, will start this report. An introduction to the Packaging scenario and to the developed tool, "Imagine Broadband Packager", will appear soon as main subjects of the internship.

The two other themes will be presented briefly in section 1.3 Imagine Broadband Configuration Management Tools that will introduce the environment in which the Configuration Management tools were developed and section 1.4 Imagine Instant Provisioning Pre-Integration that will present the work done by the intern in the Integration/Testing team at Imagine Broadband.

## 1.1 Imagine Broadband

Imagine Broadband™ is the leader in the development and delivery of operational solutions for organisations utilising complex communications infrastructures such as broadband networks, IP networks, broadcast systems and related services and applications.

The company was established to address the technology challenges, particularly those of service execution that the communications industry consistently encounters. As such, Imagine Broadband has established unrivalled expertise in many key areas including service quality delivery, operational and **deployment efficiency**, and the implementation of multi-faceted new services.

The unique services and technical capabilities of Imagine Broadband are applicable to businesses that rely on large-scale networks to achieve their business objectives such as cable operators, telecommunications operators, banks and financial institutions, utilities suppliers, energy business, home entertainment providers and many others.

Imagine Broadband enables its clients to deliver world first technology straight to the consumer by integrating and developing delivery systems for the communications and broadband market.

In today's highly competitive marketplace, network and network service operators are driven to continually enhance their product range. Once new products are announced, achievement of the stated launch date can have a significant impact on share price. **With its proven record of always achieving technical goals according to agreed time scales** Imagine Broadband has become one of the leading contributors to the rapid development of communications infrastructures for both business and domestic markets.

Imagine Broadband is a world wide company with half a dozen offices spread all over the world. All the development is done in the Product Development Centre at Sophia Antipolis, France, where the internship took place.

The importance of the kind of company in which an internship takes place is obviously large. As such, the described context can show in advance, that high levels of expertise, competitiveness and challenging are directly connected with every one working in Imagine Broadband and, as so, to the intern. The achieving of quick deliveries and the existing fast cycles of software life since requirements, development, packaging, integration/testing, installation until the product goes in production lead to a competitive environment which represents a perfect place for a full integration in a professional environment.

### 1.2 Imagine Broadband Packaging

In order to maintain these high level standards of quality, quickness and effectiveness Imagine Broadband needs high organized procedures to control all the software life steps. A high level of expertise is also required. To achieve this, the company needs to continuously update and improve its internal procedures of software production (requirements analysis, design, development, packaging and integration/testing).

The largest part of this internship is the work done in the packaging section. Packaging is the part of software life in which *Source Code* is transformed into "ready to delivery" packages.

The objective of this internship, regarding Packaging, was to maintain and gradually improve basic packaging procedures that have been put in place by Imagine Broadband in the past, this including the production of packages in order to grant time effective deliveries. Packaging tasks were the core of the internship as a high frequency of deliveries to clients is always demanded in Imagine Broadband.

The former packaging system was mainly based in UNIX Shell scripts that were repeatedly run to build packages. The maintenance of these scripts was a time consuming task as Imagine Broadband needed two employees working full time to manage packaging and the maintenance of the packaging system.

The high frequency of deliveries made Imagine Broadband need a more automated packaging system in order to follow the growth and evolution of the company, and generally, of the market.

The replacement of UNIX Shell scripts by Java developed tools was the planned procedure to accomplish the goal of this internship. No exact planning was made since it was impossible to predict whether or not the available time (beyond packaging tasks) would permit the intern to efficiently develop the required tools.

This inexact planning lead to the development of an independent Java tool called **"Imagine Broadband Packager"** that basically transformed packaging duties in a "half a dozen mouse-clicks" and a stable Java application to maintain. This tool will be presented in section 4.5 New Packaging System: Imagine Broadband Packager as the main detailed part of this report. Chapter 4 Imagine Broadband Packaging will detail the packaging process as the packages structure. After the presentation of the Former Packaging System, a full description of the developed tool "Imagine Broadband Packager" will be done. It is found of great importance the comparison between past and present systems in order to have a **metric of success** to the most important component of the internship. For that reason section 4.6 Results Analysis is fully devoted to that comparison.

### 1.3 Imagine Broadband Configuration Management Tools

A second part of the internship consisted in the development of a pair of tools that could make Configuration Management on Imagine Broadband product's target platforms.

Configuration Management consists, basically, in every task done to manage the configuration state of a given system.

To understand the kind of configuration that had to be done in the target platforms of Imagine Broadband products one should first deeply understand how the installation and deployment was done, and so, the person doing packaging was the best fit for that work.

There was much to do about System Configuration on Imagine Broadband products. The tools to develop should handle package de-installation and other system configuration functionalities such as, Installation Reports, Installation Backup and Rollback functionalities, among others.

The most important requirements for Imagine Broadband products System Configuration were gathered and the specification was done. As the development of the tool was being done the requirements were being reviewed, which is to say, that a lot of functionalities and practical details about the tool were set when the development was already taking place.

The application finished being fragmented in two tools: the Rollback Tool and the RpmBrowser. These tools will be described in detail in chapter 5 Imagine Broadband Configuration Management Tools.

### 1.4 Imagine Instant Provisioning Pre-Integration

The product support part of the internship consisted in a participation in a completely different section of the company: Integration/Testing.

This work was brief but profoundly rich as the acquisition of vast and completely new knowledge about the product in question and, generally, about the company processes and culture.

The aim of this planned share of the internship was to gradually permit the integration of the intern in the Testing of Imagine Instant Provisioning: Imagine Broadband main product that will be described in section 2.2 Imagine Instant Provisioning. This would allow the intern to Test/Integrate the packages of its own authorship, in other words, the installation and configuration of new releases of product modules would be done by the intern instead of being done by the Testing/Integration team members.

It is very important to mention, from the start, that all the planning was extremely flexible and, as such, there was plenty of space for adaptation as the company insists in the gradually and constant renewing of its resources and internal structure.

These tasks related to Integration/Testing were not fully executed due to the lack of time for the intern to properly execute them. A complete description of the plan and what was, in fact, done will be given in section 2.3 Imagine Instant Provisioning Pre-Integration.

### 1.5 Work Plan

As explained before, it is very important to mention that all the planning was extremely flexible and all planned tasks were not meant to be done exactly in a scheduled period. This fact, lead to the dynamic reallocation of the intern to different tasks, depending on available time.

In the first month of work the intern spent his time essential in expertise in the packaging process. The intern was responsible for a segment of the process since the second week of work. With the expertise in the process and its procedures, the intern started to work on the improvement of the tools, still working only in Shell scripts.

The packaging tasks were performed during all the internship. These tasks consisted in the practical application of the available tools to build packages according to the currently settled deployment methodology and architecture.

The Configuration Management tools were developed in April and May, after which, the study, design and development of the new packaging tool started. The packaging tool started to be used in the end of June for testing and was totally ready in end of July.

The brief but rich in knowledge acquisition work in the Integration/Testing section was done in the end of June. The fact that this work was done in parallel with the development and testing of the packaging tool is significant. Further discussion will be taken in section 2.3 Imagine Instant Provisioning Pre-Integration.

The final stage of the internship, the last month, was completely filled with further testing and improvement of the new Packaging Tool. These improvements consisted not only in the refinement of the implemented functionalities but also in the addition of new ones.

# 2 Imagine Broadband Products and Processes

This chapter aims to describe more detailed the company where the internship took place by presenting an overview on some topics of extreme importance to the internship.

First, a description of the production process of Imagine Broadband will be done. The understanding of this process is essential to the comprehension of where the work done during the internship fits in the production process of the company.

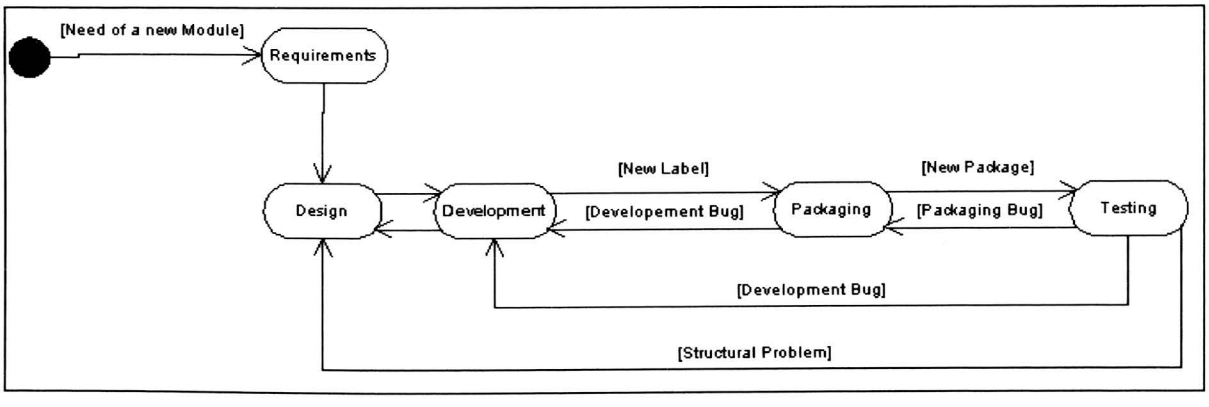
One of the main products of Imagine Broadband will be presented as packaging was done based on it (although the "Imagine Broadband Packager" was done for a more generic group of products).

All the information in this chapter is peripheral to the internship. Its aim is to frame the work done during the internship in the company and in its products.

## 2.1 Imagine Broadband Production Process

Imagine Broadband is involved in a high competitive market and used to a very high quality of delivery standards and, as such, it needs great effectiveness in all software production steps.

The following picture shows the software life cycle of any product in Imagine Broadband but especially of Imagine Instant Provisioning modules (see section 2.2 Imagine Instant Provisioning).




1. Imagine Broadband products life cycle

This picture enables one to understand where packaging fits in the life cycle of a product of Imagine Broadband. It describe a rather usual software production process that can be seen in a vast bibliography. [1]

The aim of this picture is to describe briefly the software production process at Imagine Broadband and, as such, inner details of each step will not be brought into focus but only the interactions between the sections. This description is intended to give a global overview of the production process of the company and of the framing of Packaging in it.

As it can be seen in the picture, the process starts with a [Need of a new Module] (or product). Requirements and Design are the first two steps of the process. As they are outside the scope of this internship they will not be described and can be regarded as the standard processes of requirements analysis and design.

After the development of the new module, the developer makes the code available for packaging using a [New Label] (see section 4.3.2 Microsoft Visual Source Safe Extraction for more details on the Source Control System being used).

The module is packaged and a new package (ready for delivery) is passed to the Testing Team. Since it is in the packaging step that the final code compilation is done (compatibility checks), some problems may arise and a [Development Bug] may occur. This leads to the repetition of the process described in the previous paragraph, namely the [New Label] step.

The Testing procedure may detect a [Structural Problem] in the module developed and some corrections in the design of the module may be needed. Both, [Development bugs] and [Packaging Bugs] report to problems related to the named tasks, respectively, Development and Packaging. In all these three cases the process repeats until the Testing Process is passed successfully and the package is ready for delivery to the client. After the acceptance of the delivery by the testing team, the products are sent to the client to be installed on target platforms.

All bug reporting at Imagine Broadband is done using Bugzilla (see chapter 3 Used Technologies for details).

It is important to notice that the scale of time in which these steps iterate are of the order of days and weeks for **each module** and not rarely of hours if taking all modules in count.

These steps are present in all products or modules of Imagine Broadband and are usually preformed by different individuals since normally the objects of this process are modules that are part of a big product and have to fit in a major architecture, deployment infrastructure and relationships. This multi-resources and teams process enables a better validation system regarding general correction and effectiveness of deliveries.

## 2.2 Imagine Instant Provisioning

Imagine Instant Provisioning is one of the main products of Imagine Broadband. As the understanding of this product is not essential to the full understanding of the work done during the internship a brief description will be presented.

Imagine Instant Provisioning is a provisioning system that enables large-scale communications Service Providers with corporate or domestic customers such as cable operators, to make the provisioning for broadband and large scale network services to their clients including High Speed Data, TV mail, broadcast TV, video, IP Virtual Private Networks and voice.

Imagine Instant Provisioning also involves functionalities related to Costumers Relationship Management (CRM), service activation, web based customer plug and play self provisioning system, among others.

To achieve all these functionalities Imagine Instant Provisioning deals with technologies such as Hybrid Fibber Coaxial (HFC) Networks, Digital Subscriber Line (DSL) Networks, Lightweight Directory Access Protocol (LDAP) and works with network elements such as Cable Modems, Cable Modem Terminations Systems (CMTSs), Universal Broadband Routers (UBRs), Dynamic Host Configuration Protocol (DHCP) servers, Cisco Network Registrar (CNRs) among others. Oracle databases are also used to maintain all system information.

Imagine Instant Provisioning is a Java application. It is based on its Backend, which enables access, among others, to databases and LDAP servers. All data management, in Imagine Instant Provisioning, is done through its Backend and using specialized Remote Method Invocation (RMI) interfaces. This Backend can have a diverse collection of plug-ins adjusted to the needs of each client.

The functionalities themselves are offered through Front-ends that are Java applications or Java Server Pages, usually running on different machines that access the Backend to do some kind of manipulation on the network elements in order to achieve some task, for example, provisioning some Cable Modems with an available Network Service.

"Imagine Instant Provisioning reduces both the time and cost of deployment and operations management. Deployment times are measured in hours and days rather than weeks and months." This statement about Imagine Instant Provisioning is a good pointer to what kind of production/delivery standards the company needs to establish.

For such propose, Imagine Instant Provisioning has an architecture that permits a fragmented delivery process, in other words, the product is structured in modules which permit a more quick and effective way of delivery/upgrade. This subject will reappear later at the time of the discussion on package types and packaging/installation structure.

The technologies related to this product will not be detailed as they are not included in the scope of the internship not even in its objectives.

This brief description of Imagine Instant Provisioning is very important to the understanding of the work done during the internship as it presents the context in which the work was done.

### 2.3 Imagine Instant Provisioning Pre-Integration

The participation of the intern in the Integration/Testing section may be catalogued as Pre-Integration.

After the packaging process, resultant packages are supposed to be integrated, which means, to be installed, configured and tested (regressed). The first two items in the previous enumeration are included in the Pre-Integration: installation and configuration. These two tasks might be found quick and simple, but, as Imagine Instant Provisioning embody about 30 different modules, these tasks require a lot more knowledge than one might predict at a first glace.

The integration in the Testing Team started with formation that was given by members of that team. First, it consisted of two sessions where the product architecture was presented,

and after, in the training of the configuration and usage of some of the product modules, more precisely, the Imagine Instant Provisioning Backend.

The formation was followed by a week in which the intern practiced what he had learned. It is important to mention that this procedure was taken in parallel with packaging duties that never stopped during all the internship.

The first packages arrived and the Pre-Integration was done by the intern. More specifically, this task consisted in the installation and configuration of the new modules in a specific platform for testing. At that time, there was one platform per client that mapped, in the possible way, the system configuration of the product in the platform of each client.

As time passed through, the intern started to be overloaded with work. After a week doing Pre-Integration tasks it was decided that a correction to the process had to be done, and so, this schema was abandoned.

The main reason of this decision was the lack of time of the intern to do all this work. The fact that Packaging and Pre-Integration are consecutive tasks is also very significant. These steps were done when a delivery was ready for packaging/integration/testing. Deliveries were usually not of one package/module, but of more than one.

Having more than one person doing these tasks was found more productive. Packages could be done and immediately buffered/piped to another person who was in charge of the installation, configuration of those same packages. This led to an increase on the speed of the process. Moreover, the **validation of the packaging process** itself could be done more accurately (the testing of any work by another person besides the one that did it is always much more reliable than the testing done by the same person that did the work).

Besides this, there was another major reason for this decision which was the fact that the development of the "Imagine Broadband Packager" was on going and the overload of work was evident.

In the previous paragraphs it was left clear why this procedure was establish this way rather than in the planned form and it should be clear that it was taken not by incapability of the intern but rather for process improvement reasons.

This issue is also related to the lack of resources in the Testing Team which lead to the planned idea of having the intern in it. After this decision new resources joined full time the Testing Team to fulfil the preceding needs.

# 3 Used Technologies

This chapter will make a brief summary of the technologies used during the internship. Along with the brief descriptions, links to related documentation are provided.

## 3.1 Operating Systems

### Solaris

Solaris is the operating system that Sun Microsystems provides for its family of Scalable Processor Architecture-based processors as well as for Intel-based processors. Sun has historically dominated the large UNIX workstation market. The system's availability (meaning it seldom crashes), its large number of features, and its Internet-oriented design are the most important features of this operating system. It is considered the leading UNIX environment today. [2]

### Tru64

Tru64 is a UNIX Operating System of Compaq Corporation. Tru64 is a 64-bit advanced kernel architecture. It provides symmetric multiprocessing (SMP), real-time support, and numerous other features. Tru64 UNIX complies with numerous standards and industry specifications. [3]

### Linux

Linux is a UNIX-like operating system that was designed to provide personal computer users a free or very low-cost operating system comparable to traditional and usually more expensive UNIX systems. Linux is very efficient and fast-performing system. [4, 5]

### Microsoft Windows 2000

Microsoft Windows 2000 is a commercial version of the world wide known Microsoft's evolving Windows operating system. It is designed to appeal to small business and professional users as well as to the more technical and larger business market. [6, 7]

## 3.2 Shells

A shell is essentially the command prompt one uses. It is the "look and feel" of a text based UNIX environment. Each shell offers, besides the look and feel, additional features that enable the user to manipulate its system.

There are two types of shells known as Bourne shells (after their inventor) and C shells, which are a variant of Bourne shells. The original Bourne shell is known simply as 'sh'. It was the first UNIX shell and it is available in all UNIX systems. The C shell is an improved

and slightly newer shell, it adds a C type syntax to the shell, and more importantly adds job history and job control. [8]

**Bash (Bourne Again SHell)**

Bash is a modern shell that combines many of the advantages of the C shell with much of the standard features of the basic shell; it also adds a command line editing and improved flow control. It is intended to be a full implementation of the IEEE Posix Shell and Tools specification. Its programming language is compatible with the Bourne shell (sh).

**Ksh (Korn Shell)**

Korn Shell is an interactive command shell that provides access to the UNIX system and to many other systems, on the many different computers and workstations on which it is implemented. KornShell is also a programming language for writing applications. It has the best features of C shell and Bourne shell, plus many new features of its own. Thus ksh can do much to enhance productivity and work quality, both in interacting with the system, and in programming. Ksh programs are easier to write, and are more concise and readable than programs written in a lower level language such as C. [9]

**Rsh**

RSH is a standard UNIX command that allows one to execute shell commands in a specified hostname. To do so, it must connect to an rshd service (or daemon) on the hostname machine.

**RSHD/NT**

Rshd/nt is the rsh daemon for windows that handles rsh command calls. It is a daemon that is responsible for processing the commands in the host and to send the results back to the machine where the rsh commands were executed. [10]

### 3.3 Markup Languages

**XML (Extended Markup Language)**

Extensible Markup Language (XML) is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. XML, a formal recommendation from the World Wide Web Consortium (W3C), is similar to Hypertext Markup Language (HTML). Both XML and HTML contain markup symbols to describe the contents of a page or file. HTML, however, describes essential text and graphics content only in terms of how it is to be displayed and interacted with. XML describes the content in terms of what data is being described. This means that an XML file can be processed purely as data by a program or it can be stored with similar data on another computer or, like an HTML file, that it can be displayed. [11]

### 3.4 Java

Java is a programming language expressly designed for use in distributed environments. It is an object-oriented programming language. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build a small application module. [12, 13, 14, 15]

#### Java Virtual Machine

The Java Virtual Machine is the cornerstone of any Java platform. It is the component of the technology responsible for its hardware, and operating system, independence, the small size of its compiled code, and its ability to protect users from malicious programs.

The Java Virtual Machine is an abstract computing machine. Like a real computing machine, it has an instruction set and manipulates various memory areas at run time.

The Java Virtual Machine knows nothing of the Java programming language, only of a particular binary format, the class file format. A class file contains Java Virtual Machine instructions (or bytecodes) and a symbol table, as well as other ancillary information. [16]

#### Java Foundation Classes

Java Foundation Classes (JFC) are pre-written code in the form of class libraries (coded routines) that give the programmer a comprehensive set of graphical user interface routines to use. JFC is an extension of the original Java Abstract Windowing Toolkit (AWT). JFC and Swing, an additional set of program components, are written in Java, without window-system-specific code. This facilitates a customizable look and feel without relying on the native windowing system, and simplifies the deployment of applications. JFC are integrated into Java 2 platform, Standard Edition (J2SE). [17]

#### Remote Method Invocation

Java Remote Method Invocation (RMI) is a Java system that enables the programmer to create distributed Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java Virtual Machines, possibly on different hosts. A Java technology-based program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism. In terms of compilation RMI needs a special compiler to generate special file like stubs and skeletons. [18]

#### Ant

Apache Ant is a Java-based build tool. It is known to be like Make, but without Make's wrinkles. Make is a UNIX tool that controls the compilation of source code in an organized way. [19] Make-like tools are inherently shell-based which limits its utility. Instead of a

model where it is extended with shell-based commands, Ant is extended using Java classes. It was developed to overcome the limitations of Make-like tools in terms of software development across multiple platforms.

Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface. Its biggest advantage is the ability to be cross platform, i.e. to work anywhere and everywhere, like Java itself. [20]

#### JAR Files

The Java Archive (JAR) file format enables anyone to bundle multiple files into a single archive file. Typically, a JAR file contains the class files and auxiliary resources associated with applets and applications.

The JAR file format provides many benefits, such as, security, compression, extensions management (plug-ins), portability (same as Java), versioning and control (extra information can be added to any JAR file). [21]

#### Xerces

Xerces provides world-class XML parsing and generation. Fully-validating parsers are offered for both Java and C++, implementing the W3C XML and DOM standards, as well as SAX standard. The parsers are highly modular and configurable. [22, 23, 24]

### 3.5 Microsoft Visual Source Safe

Version control is a critical facet of team software development, preventing users from accidental file loss, allowing back-tracking to previous versions, branching, merging and the managing of releases. Microsoft Visual Source Safe (VSS) is a Source Control System that provides this robust versioning and configuration management functionality along with security and audit trail capabilities.

Versioning features, such as labels, provide snapshots of a project for the quick retrieval of any previous version in the software life cycle. Difference reporting provides quick access to changes across separate versions of the same file, enabling developers to know immediately what lines of code have changed.

When using VSS, files and projects are stored in a **database**. This safely stores the information and provides important historical information and version tracking.

A **project** is a collection of files (any type) stored in a VSS database. Files can be added, deleted, edited, and shared within and among projects. A project is like an operating system folder, but has better support for file merging, history, and version control.

VSS can maintain multiple versions of a file or project, including a record of the changes from version to version. Old versions track is kept in three ways - by internal version number, by date, and by user-defined **labels**. The internal version number is assigned and maintained by VSS. Labels can be associated with any version of any file or project. After creating a label, files associated with it can be retrieved with a particular state of your project.

To setup VSS one or more databases need to be available and to use them a client application is provided. The interactions with the information in the database are done with this client application. [25]

## 3.6 Packaging Technologies

### Redhat Package Manager

The Redhat Package Manager (RPM) is a powerful command line driven package management system capable of installing, uninstalling, verifying, querying, and updating software packages. Each software package consists of an archive of files along with information about the package like its version, a description, etc.

After installing RPM in a system, the RPM database is set and one may manipulate RPM packages by an option driven command line interface. [26, 27]

### Install Anywhere

Install Anywhere is a multi-platform software deployment solution that enables easy installation and customization of deployed software through its GUI installers.

With Install Anywhere multiple features can be easily integrated in a installer like the acceptance of a License Agreement, selection of groups of modules to install, presentation of a Summary before installation or even post-install configuration through input read with the GUI and posterior file manipulation. Besides these features, Install Anywhere offers the possibility of integrating specific Java Code in its installers. With this feature one can enhance the installer with Graphical Components, run installation scripts (RPM packages installation), compose RPM packages testing panels showing if the installation can proceed, etc. [28]

## 3.7 Other

### Samba

Samba is a popular freeware program that allows end users to access and use files, printers, and other commonly shared resources on a network. Samba is often referred to as a Network File System and can be installed on a variety of operating system platforms.

Samba is based on the common client/server protocol of Server Message Block (SMB) and Common Internet File System (CIFS). Using client software that also supports SMB/CIFS (for example, most Microsoft Windows products), an end user sends a series of client requests to the Samba server on another computer in order to open that computer's files, access a shared printer, or access other resources. The Samba server on the other computer responds to each client request, either granting or denying access to its shared files and resources. [29, 30]

### Bugzilla

Bugzilla is a database for bugs. It enables its users to report bugs and assign these bugs to the appropriate developers. Developers can use Bugzilla to keep a to-do list as well as to prioritize, schedule and track dependencies.

Not all 'bugs' are bugs. Some items in the database are known as Enhancement Requests or Requests For Enhancement. A Request For Enhancement is a bug whose severity field is set to 'enhancement'. It is used to have a list of improvements to implement. This enables anyone to know what is happening with the developing process of a given project. [31]

### Microsoft Batch Files

Microsoft Batch Files can be thought as Microsoft DOS or Windows shell scripts. They have a specific language. It is not as versatile as general shell languages but offers Microsoft operating system users some power to perform useful actions.

# 4 Imagine Broadband Packaging

## 4.1 Introduction

This chapter will describe the core of the internship. The themes of the work done are Software Packaging and Packaging Process Automation. The result of this work is the developed tool called "Imagine Broadband Packager" that substituted a Former Packaging System. This tool enhanced the automation capabilities of Imagine Broadband Packaging System and turned it into a framework easy to use and maintain.

Packaging is the part of software's life in which development projects (*Source Code* plus additional files) are transformed into "ready for delivery" packages (usually installers).

The most important part of the internship included packaging tasks. These tasks involved the practical application of the available tools to build packages according to the currently settled deployment methodology and architecture.

Picture 1 showed that Packaging is a critical step in the software life cycle as it has the responsibility of build packages that should be effective deployed on target platforms. This idea should be kept in mind as it is very important to understand that all the production and evolution that took place on the Packaging section, was executed under strict time conditions imposed by the practises of the company and by the life cycle of its products.

This chapter will start with the description of the composition of packages that are actually delivered as it is fundamental to understand of what they consist before going into details regarding their creation. The internal procedures of Imagine Broadband packaging section will be detailed, reporting the general schema that it is used to build packages (section 4.3 Packaging Procedures). Section 4.4 Former Packaging System will explain how packages were actually done in the beginning of the internship. After this, the new packaging tool (developed during the internship) will be detailed in section 4.5 New Packaging System: Imagine Broadband Packager. In this section the tool will be presented as an independent software module, with UML diagrams explicating the conceptual basis of the tool. In section 4.6 Results Analysis the obtained results will be object of a pragmatically analysis and finally, within the conclusion of the chapter, a list of further work perspectives will be presented.

## 4.2 Packages Description

Imagine Broadband as two main products that are the only ones in which this internship and the Packaging System are framed. For these two products, the deployment philosophy is based on the delivery of packages easy to install on target platforms that should offer a GUI.

At Imagine Broadband the Packaging System constantly evolutes, but since the beginning of deliveries of Imagine Broadband products the main structure of their packages was not changed. This global structure will be detailed in the next paragraphs.

At Imagine Broadband, products are deployed using modules. These modules usually consist in basic features of the main product or plug-ins to the main modules. As an example, Imagine Instant Provisioning has its Backend composed of a couple of core modules and then

a collection of several plug-ins that enable the Backend to respond to different kind of requests.

All the development for these modules is done using Sun's Operating System: Solaris. The software, as it is mainly written in Java, can be run in several platforms. In client platforms the Operating system is preferentially Sun Solaris but not obligatory as products are running over Compaq's True64 in one of the clients (see chapter 3 Used Technologies for further information about these Operating Systems). The main part of the packaging is done on Sun Solaris. These facts will have further consequences in the software used for packaging.

As the major part of the modules are written in Java, the best solution for the delivery are Java Archives (Jars) (see chapter 3 Used Technologies for further information on Java Archives).

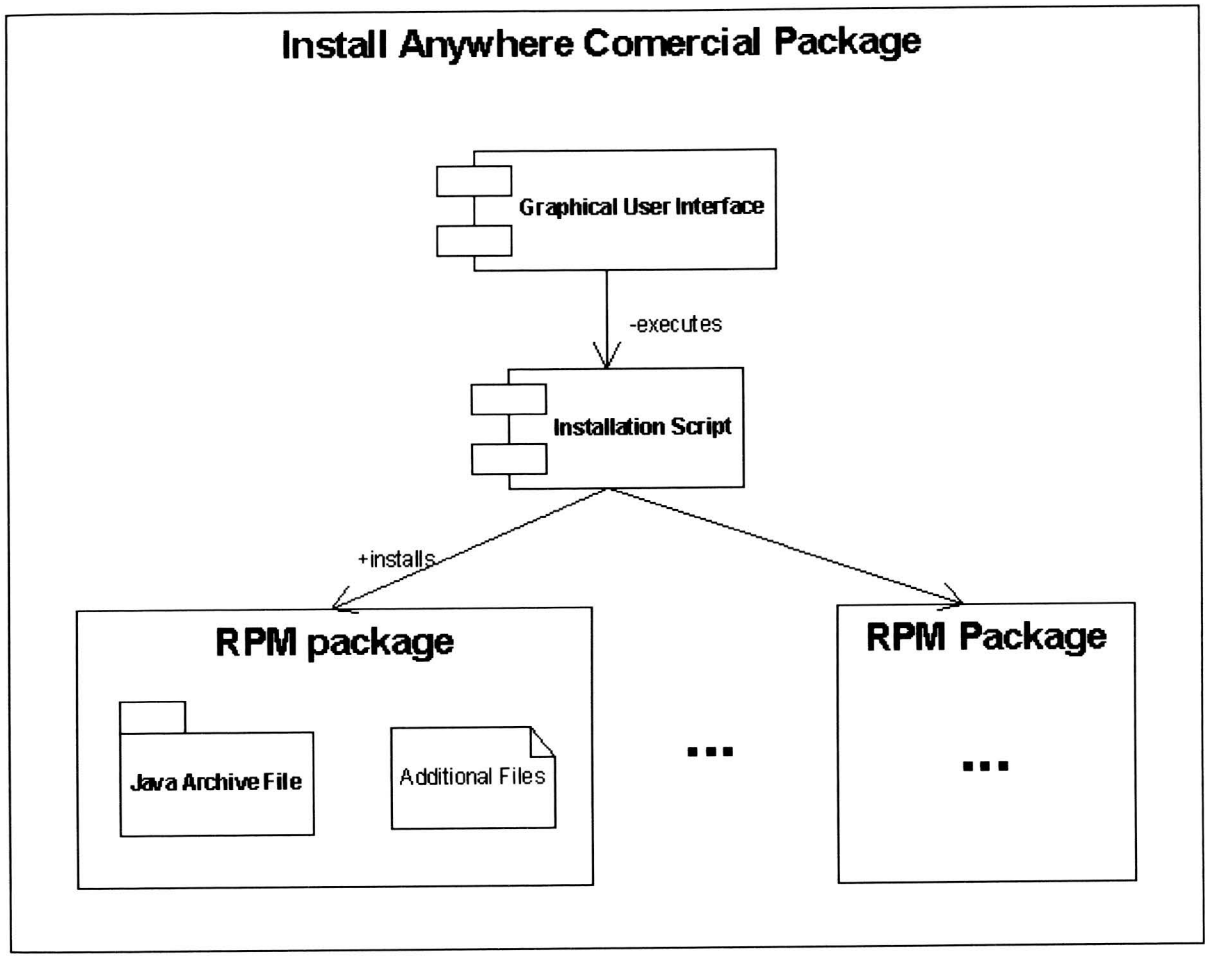
Together with these JAR files all modules have some additional files to be deployed (executables, configuration files, dependency libraries, Third Party libraries, among others). These files, that constitute a package, are organized in a directory structure convenient for the execution of modules as for their configuration.

The mainstream solution for deployment in UNIX environments is Redhat Package Manager (RPM). RPM was original conceived for Linux (see chapter 3 Used Technologies) but as it grown, releases for other UNIX systems became available.

With RPM, multiple files can be easily deployed as a single package. RPM offers functionalities for package installation, upgrade and de-installation (see chapter 3 Used Technologies for further discussion about RPM).

Despite RPM offers various options for packaging management there was a need of a GUI during the package installation for several tasks achievement. For this purpose Install Anywhere was chosen. Install Anywhere is a multi-platform software deployment solution that enables easy installation and customization of deployed software through its GUI installers (see chapter 3 Used Technologies for further information about Install Anywhere).

An Install Anywhere project represents a Commercial Package, i.e. an installer that is the final form of the package to deliver. The following picture shows the composition of an Install Anywhere package.




2. Install Anywhere Commercial Package structure

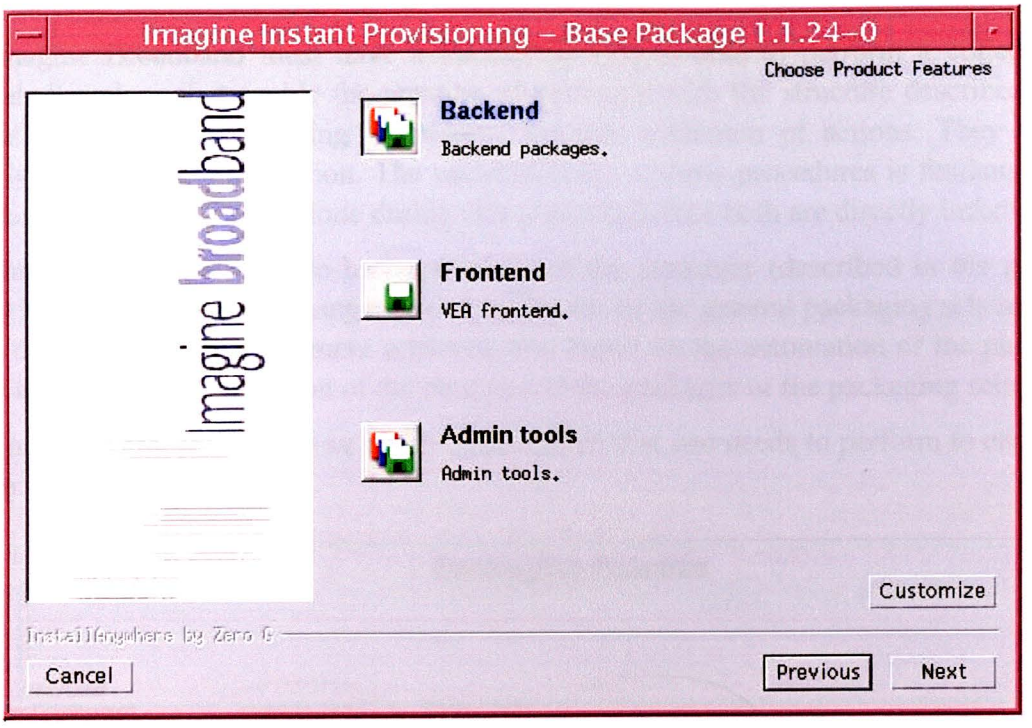
As shown in the picture above a Commercial Package (installer) may contain several RPM packages that contain themselves the modules.

The installer consists in a sequence of panels that lead the user through the installation of the package. The panels of a typical installer are:

- Installer Introduction;
- Module Description;
- License Agreement;
- Feature Selection (when the module has more than one feature);
- RPM Test Panel (with results of the installation tests of RPM packages);
- Module Configuration (configuration fields are read from the interface);
- Installation Summary (showing a summary of features, RPM packages and configurations before installing the package);
- Installation Progress;

- Installation (Un)Success Message.

The following picture shows an example of a Panel during an installation:




3. Screenshot: Install Anywhere installer

Install Anywhere packages can include an embedded Java Virtual Machine specific for each target platform (see chapter 3 Used Technologies for further information on Java Virtual Machines). This feature reduces the specific system requirements for the target platform, which are, basically, a standard shell UNIX terminal and the RedHat Package Manager.

After installing a module of an Imagine Broadband product, the user usually needs to configure it in order to run it. Packages are deployed in a way that no configuration related to package structure is needed. The post install configuration is related to module specific needs (in those cases where it is not possible to include configuration features in the installer). There are some details about the deployment structure of packages that was changed during the internship but that are not worth detailing in this document.

This section described the composition of the packages deployed by Imagine Broadband. The following sections will detail how these packages are done, what are the tools used to perform that task and where does the internship work fits in this schema.

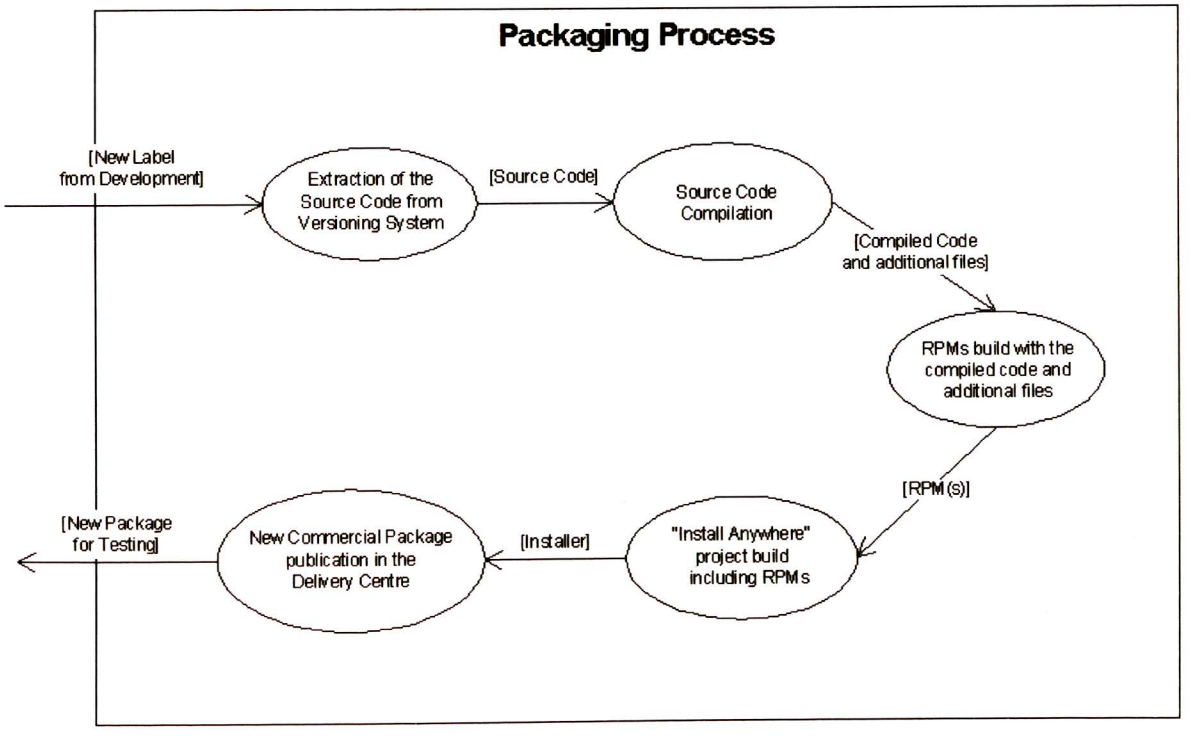
## 4.3 Packaging Procedures

### 4.3.1 Introduction

Imagine Broadband must have a Packaging System able to perform a collection of (automated) actions that enable the creation of packages with the structure described in the previous section. The Packaging Procedures are this collection of actions. They will be generally described in this section. The understanding of these procedures is fundamental to the understanding of the work done during this internship since both are directly linked.

During the internship, the basic structure of the packages (described in the previous section) has not suffered any change. The same stands for the general packaging schema. As it will be described, the improvement achieved was based on the automation of the packaging process and not in the innovation of the structure of the packages or the packaging schema.

The following picture shows the five basic steps that one needs to perform to create one package:




4. Imagine Broadband Packaging Process

To set up a full Packaging System to construct packages with the structure described in section 4.2 Packages Description and using the process shown in the previous diagram, some specific platform requirements are needed. These requirements are related to the availability of all used technologies. Further technological details will be given in following sections.

In the next sections each of the steps shown in picture will be fully detailed.

### 4.3.2 Microsoft Visual Source Safe Extraction

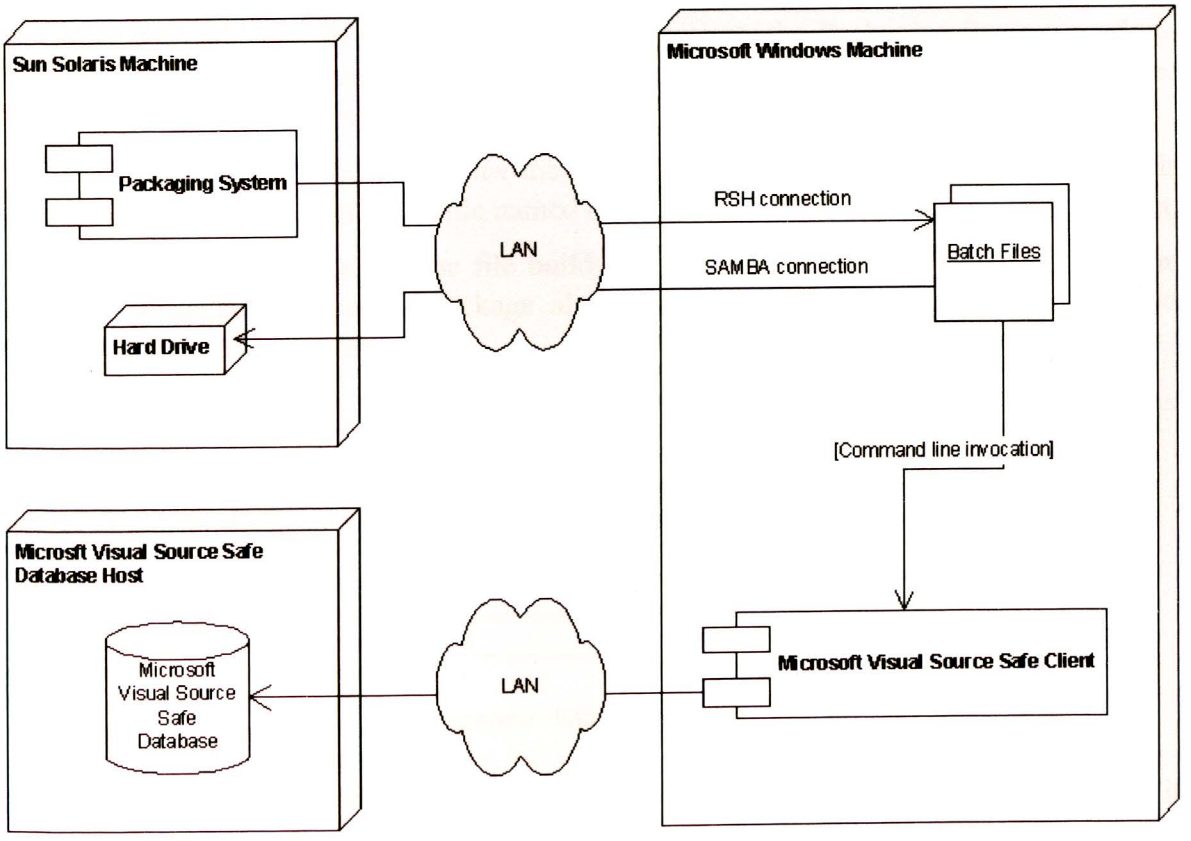
Microsoft Visual Source Safe (VSS) is the Version Control System used in Imagine Broadband for all software development projects (see chapter 3 Used Technologies for further information on VSS). This section will describe the interaction between the Packaging System and VSS.

VSS is used to keep all the versions of all development projects. Development projects usually consist in a collection of Source Code files and another set of additional files (executables, configuration files, etc.). These projects are the source of all packages. Usually, each development project (in VSS) gives origin to one package.

The extraction of a development project (future package) from a VSS database consists just in a small action on a VSS client, but, as Packaging Procedures should be automated, this option did not fit in the needs of the packaging system because it would depend on a manual action. Nevertheless, this client can be used through the command line with the correct arguments. As this second option can be easily used in an automated process, it was chosen to extract development projects from VSS.

Since all packaging is done with the Operating System Sun Solaris, the Source Code should be extracted to a machine running Sun Solaris (UNIX machine). Since VSS needs a Microsoft Windows machine (Windows machine) to run, a bidirectional connection between the two machines was needed.

The following picture shows how this task is achieved:




5. Components diagram: VSS extraction schema

In the Windows machine there are appropriated Batch Files that deal with the correct call to the VSS client. Other requirements for the success of this schema are the Remote Shell (RSH) daemon running on the Windows machine (so that the Batch Files can be called from the UNIX machine) and Samba running on the UNIX machine (so that the Windows machine can write to the UNIX machine's hard drive). See chapter 3 Used Technologies for details on these technologies.

The usage of the schema is the following: a RSH call is made from the UNIX machine to the Windows machine. In this call, the Batch Files are invoked with the proper arguments. The Batch Files run and pass the extraction directory (passed as RSH command argument) to the VSS client that will write the files in that directory using Samba.

All the features must be configured correctly. The VSS client must know the user that is running the commands, the host of the Database and directory to which it should write the extracted files. The UNIX machine's hard drive should be mapped (with Samba) in the Windows machine in order to enable the extracted files to be written in the UNIX machine.

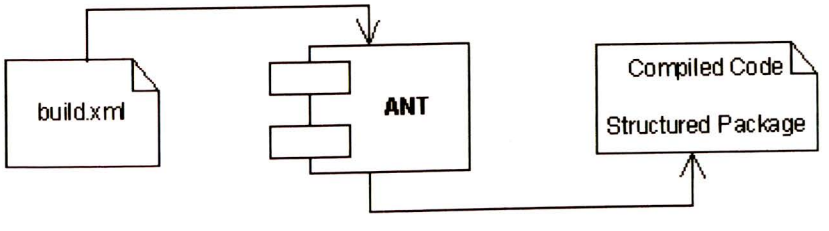
In order to use this schema to extract the files of a project from a VSS Database the Packaging System needs to know what database, project and label to use (see chapter 3 Used Technologies for further information on these concepts). These details are involved in the improvement of the procedures so they will be described in the section 4.4 Former Packaging System and then in section 4.5 New Packaging System: Imagine Broadband Packager.

### 4.3.3 Source Code Compilation

After the extraction of the Source Code of a project, the Packaging System needs to compile the project. The compilation of the Source Code is done using Apache Ant (see chapter 3 Used Technologies for further details).

The compilation with Ant enables the process to be totally automated. To have this automation one needs to construct a file named build.xml that serves as the only input for Ant.

After Ant execution (using the file build.xml as its input) the Packaging System can obtain the compiled code and a package already structured to be processed in the next automated step (see picture 6).




6. Components diagram: Ant Build Process

This build.xml file consists in a list of instructions to compile a collection of files and it may also enclose instructions related to pre and post-compilation tasks.

The build.xml file is a XML file and, as such, it needs to obey to a collection of syntax rules. Moreover, Ant defines a dictionary of terms that can be used to take action on Source

Code and related files (there are other types of manipulation that can be done with Ant but not fundamental to the understanding of this schema).

The main operations allowed in a build.xml file are:

- Create, copy, move, rename and delete files;

- Java Source Code compilation (java Compiler);

- Java RMI compilation (RMI compiler, see chapter 3 Used Technologies for a brief description);

- JAR files creation.

With these four basic types of actions one can compile the code (with the Java compiler and RMI compiler), build the JAR files and construct the deployment structure of the package.

To construct the exact list of instructions the Packaging System needs to know exactly what it is supposed to do to the package. For this purpose, there are in VSS, in every project, two files that the Packaging System uses to perform its procedures. These files are the Release Notes and the Package Information File.

Release Notes are enclosed in a XML file (ReleaseNotes.xml) that encloses all the information about the package: Name, Label, Description, Dependencies, Bugs, etc. (see appendix A: ReleaseNotes.xml structure for further details about the structure of these files). This file is maintained by the developer of each package and represents the most important source of information for the Packaging System.

The other file, the Package Information File (pinfo.cfg), contains only information regarding the versioning of the package and is the only one that is written in VSS by the Packaging System in order to maintain a trace of the evolution of package's versions.

Even with the information contained in these two files there are still several kinds of packages that differ either in the form of the compilation or in the needed structure after compilation, i.e. information not contained in these files. The first way to workaround this problem is the maintenance of one static build.xml file per package that deals with all the compilation details for each package (excluding details focused in ReleaseNotes.xml and pinfo.cfg). This schema leads to a big effort to have these files up to date. As so, an automated process of building these build.xml files was needed. As the automatic generation of build.xml files changed with the improvements of the Packaging System the details will be given in the following sections.

Summarizing, after this step, the Source Code is compiled and enclosed in a package that is structured in a different directory from the one it was extracted to.

#### 4.3.4 RPM packages build

As described in chapter 3 Used Technologies, RPM enable efficiency in package delivery and in version control (installation, upgrade and removal).

As for compilation, in the build of RPM packages, a Specification File serves as the only input. A RPM Specification File contains all the information required to build a RPM

package: package information, package upgrade type and files list/types (see appendix C: RPM Specification File for further information about RPM Specification Files).

The RPM Specification file is constructed based on the Package Information File. In this stage, the Package Information File must be updated in order to include more information about the package than the original Package Information File extracted from VSS. There must be a procedure that parses the Release Notes and write some selected information to the Package Information File. This update is done when building the package. This file is used to ease the process of pass the data from the ReleaseNotes to the RPM Specification File. It is used as an intermediate data holder. After transferring the information contained in the Package Information File to the RPM Specification File in the needed way (parsing and formatting of tags), the files to include in the RPM package must be listed. To do this list, the Packaging System uses the directory structure constructed by the previous building process (which used Ant).

With an RPM Specification File correctly created, the build of a RPM package can be started with the RPM command. This command needs a RPM building environment to be configured before being invoked. The environment configuration needed consists in the placement of the RPM Specification file and the package files in a proper directory defined in the configuration of RPM. After configuring the RPM building environment one just needs to call the rpm command in order to get the new RPM package (with the files of the project included and with the given header information).

As said in the previous section, the version control of the package is done using the Package Information File (pinfo.cfg) in which the version of the RPM package is written. After building a RPM package the Packaging System must commit the original Package Information file to VSS. This file should be updated with the new RPM package version (further details will be given in next sections).

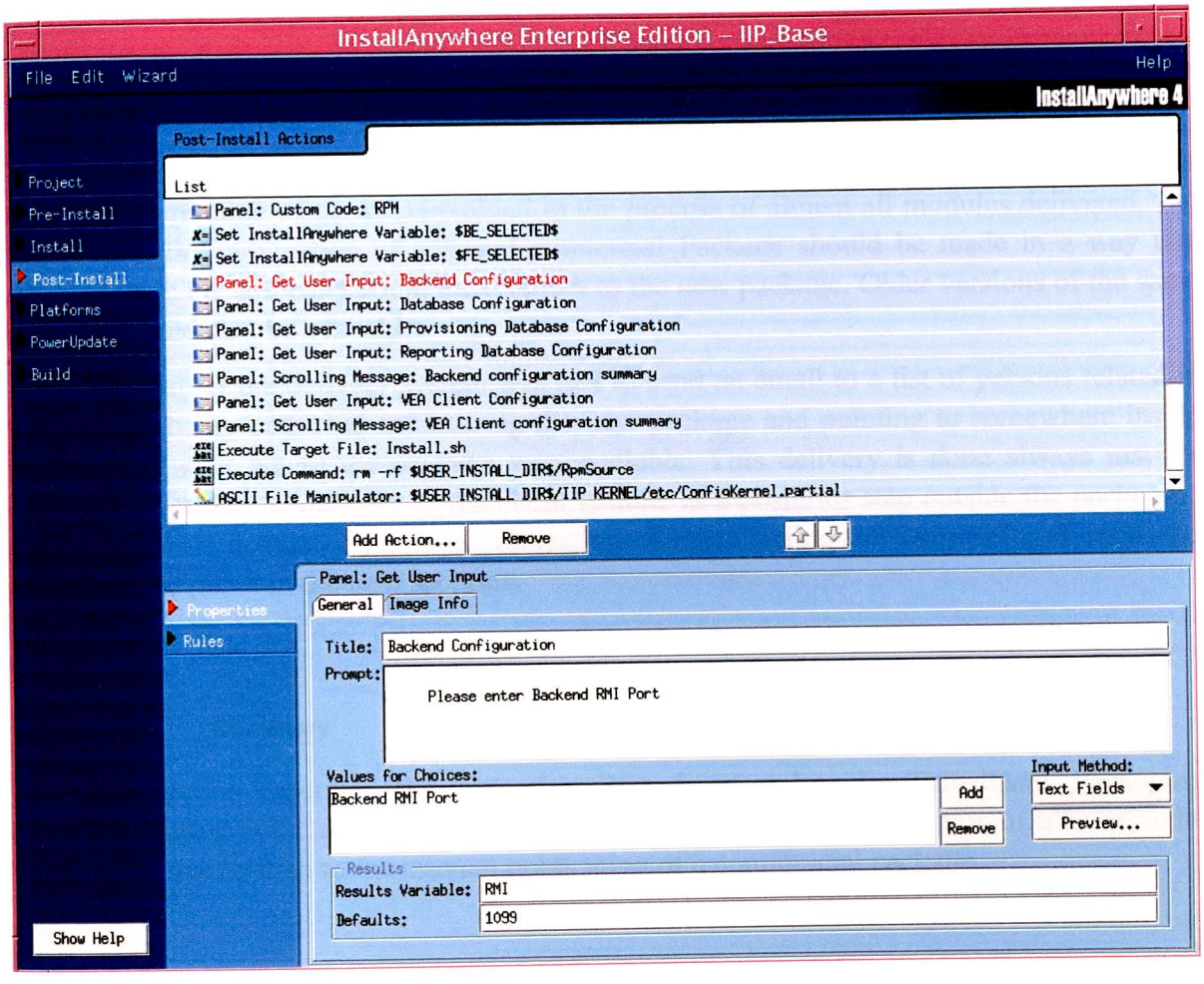
The format of the version number used in Imagine Broadband is "X.Y.Z-W". The package upgrade type can be "Full Build", "Software Upgrade", Patch Upgrade" or "Internal Build". Each of these terms refers to the increment of the digit X, Y, Z, W, respectively, in the version number.

There are various types of versions used. The first version is controlled by the developer and is the Label version. It is used for development reference and for extraction. The second version is the RPM package version which represents the compilation version. The third one is the commercial package version that is used in the commercial packages built in the next packaging step (see section 4.3.5 Install Anywhere project). Obviously, none of these versions are synchronized. For one package, they can evolve independently.

This step of the creation of a delivery consists in the creation of RPM packages that enclose a complete package (Source Code and additional files).

### 4.3.5 Install Anywhere project

This section describes how Install Anywhere packages are created. To do that, there is an Install Anywhere project editor which is a GUI that enables one to create the installers. All the steps of the installer are created and configured using this interface. The next picture shows a screen shot of that interface.




7. Screenshot: Install Anywhere project editor

If there is a new package to be delivered, a new Install Anywhere project must be created, configured and published. But, in most cases the packaging demands are related to existing packages/projects, as such, the usual tasks on an Install Anywhere project are the update of RPM packages to include in the installer, the update of version numbers (Installer and License Agreement) and the building of the installer.

This stage of the packaging process (the achievement of mentioned tasks) was profoundly improved. Before the internship everything was done manually with the help of the project editor and, after the internship, Install Anywhere automated build features were used (see next sections for further details).

At this point, it is important to recall from section 4.2 Packages Description, that despite in the two previous steps (Source Code compilation and RPM packages building) there was always one compilation and one RPM package per project (despite some exceptions), now,

one Install Anywhere project can encapsulate more than one RPM package as a Commercial package can include more than one "development package" (see picture 2).

After the creation of a new installer it must be functionally tested by its author as problems may occur. This first test was manually done by the person doing the packaging. It consists just in the verification that the installer works and that it correctly installs the package.

#### 4.3.6 Package Publication

A new Commercial package should be delivered after its creation. There are always a considerable number of persons involved in the process of almost all modules delivered. For this reason, the publication of a new Commercial Package should be made in a way that anyone inside the company could have access to the new package. Older versions of the same package should also be available.

The base of the delivery system was just to send an email to a list of persons related to the package being delivered announcing the new package and pointing to somewhere inside the local network where the package was available. This delivery is done always just for internal purposes. The delivery to the client is done in a posterior step outside the packaging scope. All other details of package publication changed during the internship, as such, the details will be given in the next sections.

#### 4.3.7 Summary

This section described the Packaging Procedures at Imagine Broadband. Packaging Procedures mean all the actions taken to create packages, since the extraction of files from the Version Control System to the internal publication of a commercial package.

## 4.4 Former Packaging System

### 4.4.1 Introduction

In the previous section the general Packaging Procedures were described. This section will describe the Packaging System available at Imagine Broadband in the beginning of the internship.

The former packaging system was mainly based in UNIX Shell scripts that were repeatedly run to build packages. Almost all packaging steps described in the previous section were done with the help of UNIX Shell scripts and some tasks had to be done manually. The maintenance of these scripts was a time consuming task (see appendix D: IIP Packaging for a detailed description of the former packaging process).

In the next sections all the tasks will be detailed so that, later, the new Packaging System can be easily introduced.

### 4.4.2 Microsoft Visual Source Safe Extraction

For the extraction of the Source Code from VSS the Former Packaging System used a single UNIX Shell script that made all the work: extract.ksh. The selection of the VSS Database, Project and Label to extract (see chapter 3 Used Technologies for further detail of these concepts) was done with Shell menus that enabled the user to select the desired values for each. In the next picture one of those menus is shown, in this case, the VSS database selection (in this example product means Database):

```
# ./extract.ksh
++++ VSS Extract ++++


++ User Login ++

1) Eric.Duchet
2) Olivier.Depaye
3) Pierre.Walter
4) Luis.Ramos
Select username: 4

User name     : Luis.Ramos
Hostname      : ws-pwalter
Working folder: /export/home/hbo/lra

Should I clean extraction directory /export/home/hbo/lra/vss (y/n): n
+++++ VSS Extract ++++


++ Product selection  ++

1) IIP
2) ISA
3) CMN
4) NEXT
5) EXIT
Select a product for extraction: 1
Getting VSS projects for IIP...
```

8. Screenshot: VSS Extraction Shell script

These menus were often slow because every time the user iterates between them the script needed to do a call to the machine where the VSS client is (recall section 4.3.2 Microsoft Visual Source Safe Extraction where these calls to VSS were described).

After the selection of a Database the script shows a list of Projects available in the selected Database. After that second selection a new menu is shown, this time with the list of Labels available for the selected Project. With the selection of a Label the user could get the files included in that Label to his home directory.

After the extraction of the Label the script parses the extraction directory in the home directory of the present user and searches for Package Information Files to update using the respective Release Notes file (recall section 4.3.3 Source Code Compilation and 4.3.4 RPM packages build). Every time a single Label was extracted from VSS, this script tried to update all Package Information Files available in the home directory of the user. This was not a good measure but there was no other way to update Package Information Files as selections of previous menus were not gathered.

### 4.4.3 Source Code Compilation

After the description of how the Source Code was extracted from VSS in the Former Packaging System, one should recall the section 4.3.3 Source Code Compilation. In order to compile the Source Code extracted from VSS one needs to create a build.xml file that is used as the only input to Ant. In this file all details about the compilation and relocation of the extracted package are taken in account.

In the Former Packaging System, there was a script called genXml.ksh that generated the build.xml file for each package using the Release Notes of the package.

This script had to be executed from the directory where extracted files were. After the generation of the build.xml file, Ant should be executed in order to compile the Source Code.

There were exceptions to this case. A lot of packages needed a special build.xml file (that could not be generated by the script). For these cases, there was a repository of package specific build.xml files.

### 4.4.4 RPM packages build

The creation of RPM packages was also handled by a single script called buildrpm.ksh.

The interface that this script offered was close to the one of the extraction script (see section 4.3.2 Source Code extraction for further details) but this time in a rather complex way.

During the execution of this script several options had to be selected:

- Type of upgrade;
- Product of the package;
- Target platform of the package;
- Project of the package;

After the selection of all the options, several confirmations were asked between the steps of the creation of the RPM package.

This script was written to deal with the creation of several RPM packages at the same time, and this made it too complex for building just one RPM package as it was needed most of the times. After the execution, the location of the new RPM package was reported as well as the location of a log file.

### 4.4.5 Install Anywhere project

The first task to perform in order to build a Commercial Package is to copy the recently created RPM package(s) to the proper location, to be integrated with Install Anywhere. Other obligatory task is to update the License Agreement which contains a version number. Both these tasks were performed manually.

After updating the RPM package(s) and the License Agreement, the project is opened with the Install Anywhere project editor. With the editor, the version of the Commercial Package was updated and further configuration of the project could be made as it was sometimes needed.

The final step in the generation of the package was to build the updated Install Anywhere project that gives origin the installer to be delivered. This task was achieved easily through the Install Anywhere project editor.

### 4.4.6 Package Publication

In the beginning of the internship, the package publication was done in a rather rudimentary way. But soon after it, this step was improved (not by the intern). Both schemas were based in the sending of an email announcing the new package to a selected group of persons.

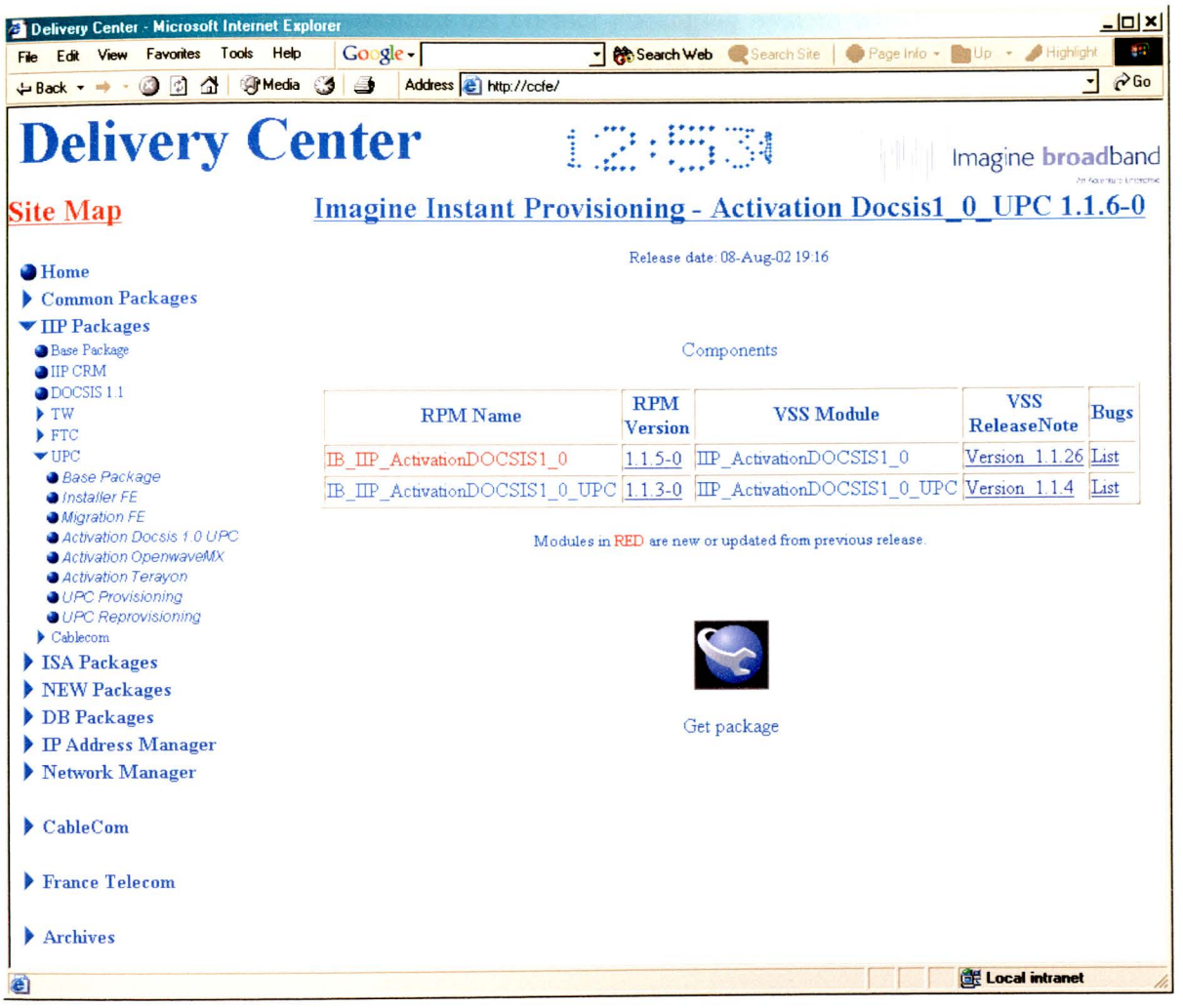
The former schema used a shared local server to place packages there and to make them available to every one inside the company. The packages were copied to that location manually either using FTP or Samba (see chapter 3 Used Technologies).

The improvement of this step took place in the beginning of the internship. An internal Delivery Center was developed. It consists in a Web site (part of the company's intranet) in which all packages are available in an organized way.

The Delivery Center presents packages using a Table of Contents file that has to be generated after the building of a package. This file contains information about the package content, i.e. the versions and names of the installer, the RPM packages enclosed in it and the VSS project names and version. Besides this information there are links for information files about each RPM package and each label of a VSS project. These files are called Reports.

For publishing the generated packages the Former Packaging System had a script called UpdateDeliveries.sh. This script offered a Shell menu to select the package to publish. It used a configuration file to copy the installer to the Web site directory and to generate the Table of Contents of the package.

The following picture shows a screenshot of a package page in the Delivery Center.




9. Screenshot: Imagine Broadband Delivery Center

In this picture, the tree shown in the left side of the page contains links to all modules. It represents the organization of almost all Imagine Broadband products and clients.

Install Anywhere always generates an installation Web page for all packages. This page offers an easy way to download the various installers by presenting a list of installers built for specific Operating Systems and including specific Java Virtual Machines. This download page is used in the Delivery Center. The image "Get Package" in the previous picture is a link that leads to that download page.

The main pages of each product contain a list of news which is a historical of the modules delivered. The node "Archives" in the tree encloses all the versions of delivered packages.

For the delivery itself, an email was sent with the Table of Contents of the new package and with a link to the Delivery Center.

All this details regarding the Delivery Center were developed during the internship by the Packaging Manager.

### 4.4.7 Summary

This section briefly described the Former Packaging System which was the Packaging System available in the beginning of this internship at Imagine Broadband. This system was based on Shell scripts and in manual actions. This description was done to give a base for comparison with the new system that will be described in the next section.

## 4.5 New Packaging System: Imagine Broadband Packager

### 4.5.1 Introduction

The high frequency of deliveries made Imagine Broadband need a more automated packaging system in order to go with the growth and evolution of the company and in general of the market.

To accomplish the aimed goals a plan should be elaborated to define what would be done. The result of this planning, was, that there was no need to change packaging procedures as they were. Instead, it was plain agreement that the Former Packaging System (described in the previous section) should be replaced. The new system should reduce the Shell scripts but especially their interface and generalization inabilities. The advantages and disadvantages of this main option will be discussed in section 4.6 Results Analysis.

These ideas lead to the development of an independent java tool called "**Imagine Broadband Packager**" which encapsulates a complete Packaging System. This new application transformed the Packaging System in a fully automated system that consists in a stable Java application easily configurable.

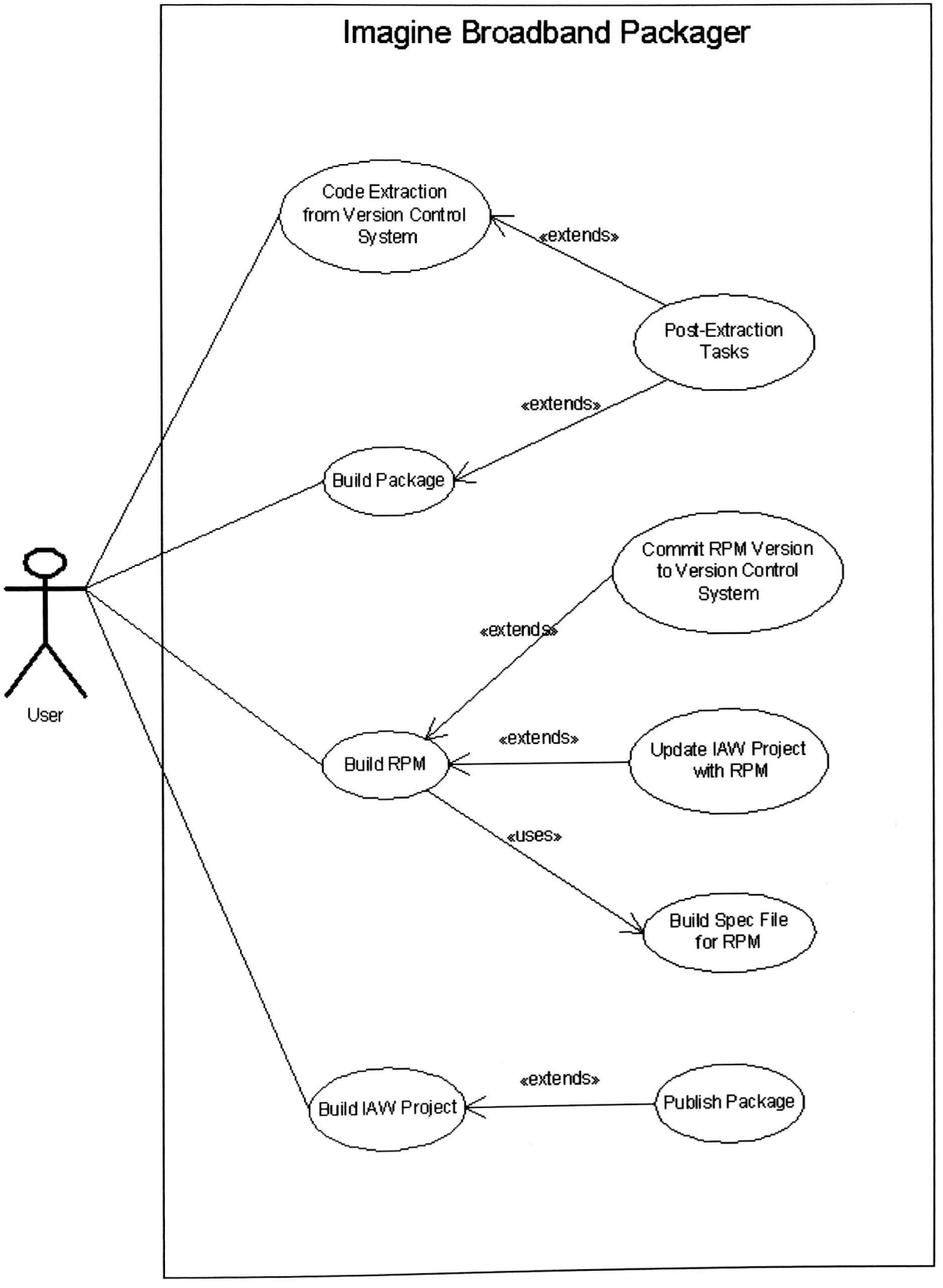
The following sections contain a collection of UML (Unified Modelling Language) diagrams that were used to design the tool. The diagrams will conceptualize only the new ideas related to the new tool. The conceptualization of ideas explained in previous sections will be avoided.

### 4.5.2 Requirements

This section will describe the requirements of the new Imagine Broadband Packager.

Imagine Broadband Packager should be a tool created to run in a specific environment. It is not supposed to be installed on a target platform as its target platform is the same kind of platform as the development platform, in other words, the tool should be able to run in a platform with some specific pre-requirements (as it can understood from section 4.3 Packaging Procedures).

The functional requirements analysis of the new tool was intrinsically quiet well described in the previous sections. Due to this, all the functional requirements of the tool can be expressed with a simple Use Cases diagram as follows.




10. Use Cases diagram: Imagine Broadband Packager

This diagram is a translation of the picture 4. In this diagram, there is only one actor/roll that will interact with the tool. This roll is usually performed by the person is charge of the Packaging System, i.e., the person that use the Packaging System to create and deliver packages. There are nine use cases in the diagram:

**Development Project extraction of Version Control System**

This Use Case consists in the actions described in section 4.3.2 Microsoft Visual Source Safe Extraction. The user of the tool should be able to extract the code from a Version Control System and have it available in his home directory for further processing.

**Build Package**

As before, this Use Case consists in the actions described in section 4.3.3 Source Code Compilation. The user of the tool should be able to compile the Source Code available in his home directory and to structure the compiled Source Code and additional files in a different directory.

**Post Extraction Tasks**

This Use Case encapsulates extra functionalities related to actions to be taken on extracted files. These tasks are: Package Information File update, AutoConfig File update and generation of the Package Report. The Package Information file is update to serve as base for the RPM Specification File (this was described in section 4.3.4 RPM packages build). The AutoConfig File update is a new feature not regarded in the Former Packaging System and it consists in the update of the package version in a specific file of some modules. The Report file is used when the delivery of a new package is made. It is an easy access description of the package and encapsulates a simplified version of the development Release Notes file.

This Use Case "extends" both "Package build" and "Development Project extraction from Version Control System" Use Cases mainly because these tasks can be executed automatically after the Extraction of files or (if some error happens in some file) with a mouse action before the Building Process.

**Build RPM**

This Use Case consists in the creation of a new RPM package (further details were given in section 4.3.4 RPM packages build). The user must be able to create a new RPM package from a Package Information File associated with a collection of files.

**Commit RPM package version to Version Control System**

This Use Case consists in the commission of the Package Information File to the Version Control System. As explained in the end of the section 4.3.4 RPM packages build, the Package Information File encloses the latest version of a RPM package and is used to control its version.

This Use Case "extends" the "Build RPM" Use Case because the commission of the Package Information File to the Version Control System is part of the RPM Package Build process.

**Update Install Anywhere project with RPM packages**

The update of the Install Anywhere project with recently created RPM packages consists simply in the copy of the new RPM packages to a proper place within the Install Anywhere project directory.

This Use Case "extends" the "Build RPM" Use Case for the same reasons of the previous Use Case.

**Build Spec File for RPM Package**

The aim of this Use Case is to give self existence to the creation of the RPM Specification File as a separated task (as it can be performed separately). The Use Case "Build RPM" "uses" this Use Case. This means that the creation of a RPM Specification File is enclosed in the creation a RPM package.

**Build Install Anywhere Project**

Another Use Case of the tool is the Build of an Install Anywhere project. This Use Case encapsulates the tasks described in section 4.3.5 Install Anywhere project, and, as such, it includes the update of the License Agreement, of the package's version and the mentioned creation of the new Commercial Package.

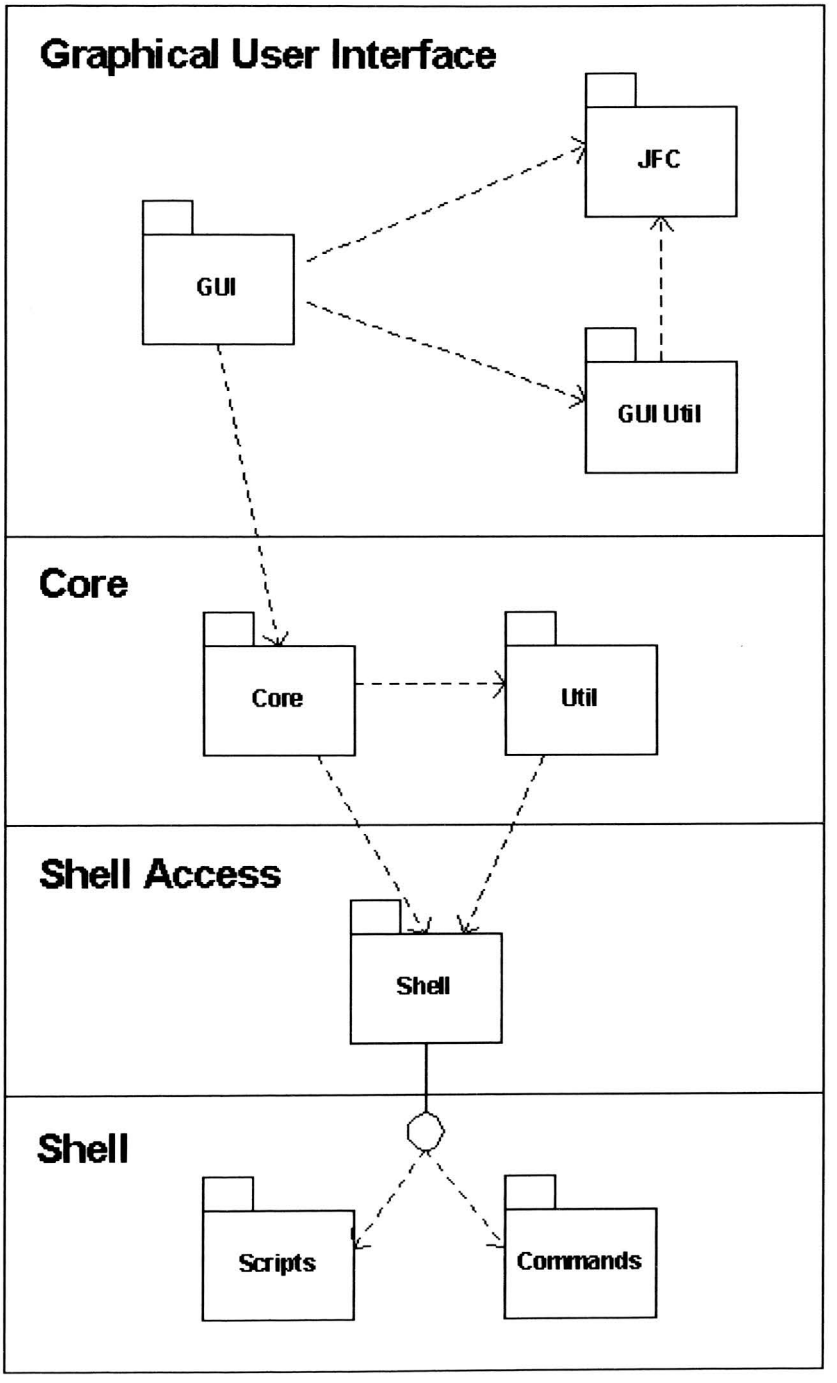
**Publish Package**

The last Use Case in the diagram is the Package Publication. This Use Case consists in the delivery of the package to other sections of the company (see section 4.3.6 Package Publication for details). This Use Case "extends" the previous Use Case because in this way this task can be seen as a simple part of the Install Anywhere build process.

In order to implement an application that offers the user the functionalities described in this section the standard software development procedures were followed. As such, the design of the tool was done and the following design components emerged.

### 4.5.3 Logical Architecture

This section will describe the logical architecture of the application. The following Logical Packages diagram enumerates the logical packages and its relations:




11. Logical Packages diagram: Imagine Broadband Packager

The logical architecture of the application has four main layers, which are, the Graphical User Interface, the Core, the Shell Access layer and the Shell itself.

In the first layer, three packages are defined. The GUI which implements all the specific GUI features of the tool, the package GUIUtil that implements some extra graphical features

and the JFC package which is used by the other two packages (see chapter 3 Used Technologies for further details on Java Foundation Classes).

Layer two, the Core layer, is responsible for all main task achievement and coordination. This layer contains the package Core that implements all the basic functionalities of the tool and, as such, encloses all the central classes of the application. It is used by the GUI package of the first layer and uses the package Util that encloses utility classes.

The Shell Access layer is a layer constituted by only one package: Shell. This package implements a complete interface between the Java application and the Shell scripts.

The fourth layer represents the Shell itself and encloses one package where the scripts are enclosed and another with the commands used by the application.

For conciseness, all classes enclosed in these packages will be listed only in the following section where classes diagrams will be used to detail each package.

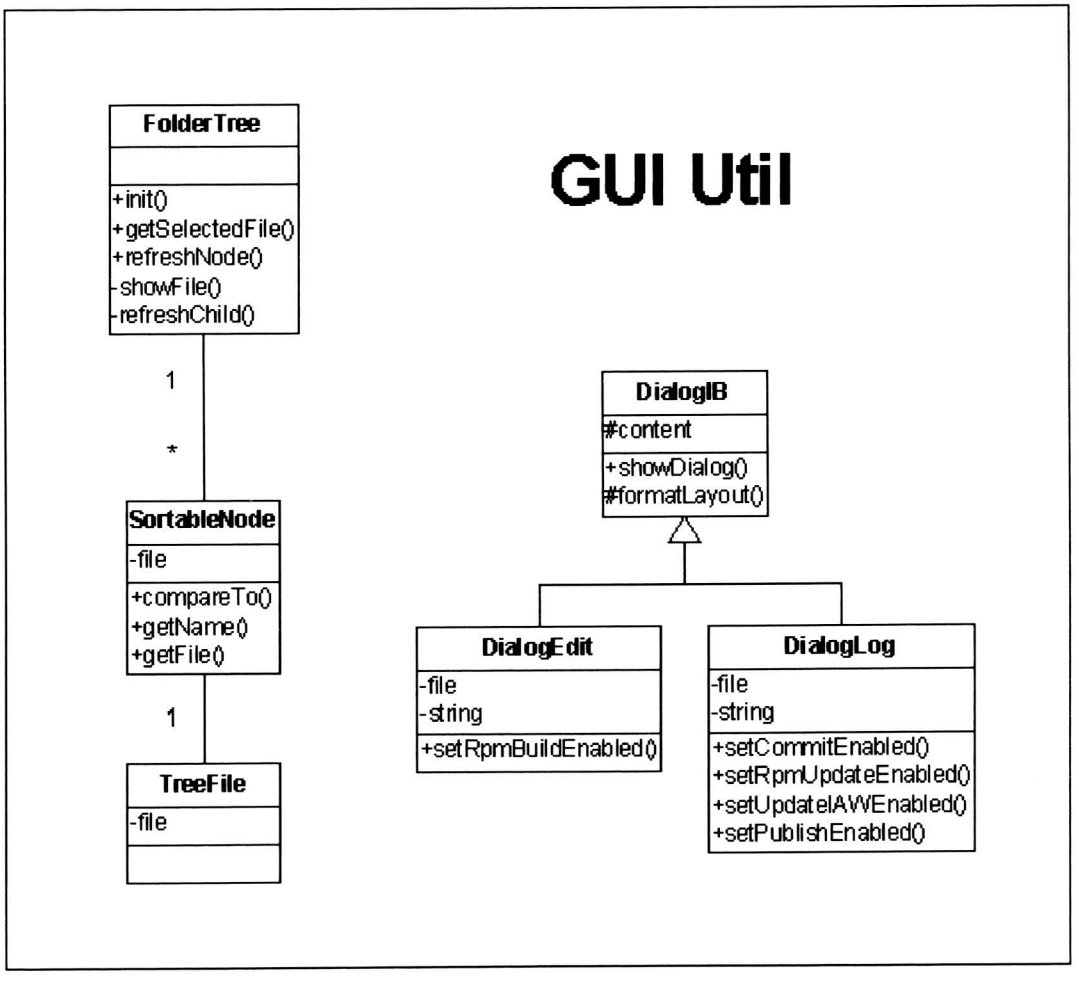
### 4.5.4 Object Data Model

In this section the logical architecture of the application being described will be fully detailed. A diagram will detail each of the logical packages presented in the previous section.

The first packages to be described will be the secondary ones, so that, when presenting the main packages a global idea of the tool can be formed.

#### GUI Util

The following diagram shows all the classes of the GUIUtil package of the first layer (see previous section) and the relations between them.




12. Classes diagram: Imagine Broadband Packager Logical Package GUIUtil

In this package there are two functionalities implemented: Dialogs and FolderTrees. The description of this diagram will be oriented to functionalities as this is a Utilities package.

The FolderTree extends a JTree which is a JFC class. This FolderTree is used to show a directory structure in an interface component. The method init of the class FolderTree is used to create a FolderTree and the argument is a file path that will be used as the root node of the tree. From a FolderTree one can get the selected file in the tree with the method getSelectedFile. An additional method for refreshing nodes is implemented: refreshNode. This method makes the data in the tree coherent with the actual file system being mapped.

A FolderTree list its components (nodes) in an ordered way and to achieve that a SortableNode class was implemented. This class, besides making the nodes naturally ordered, encloses a TreeFile that is a class that has the information of the system files being shown in the tree.

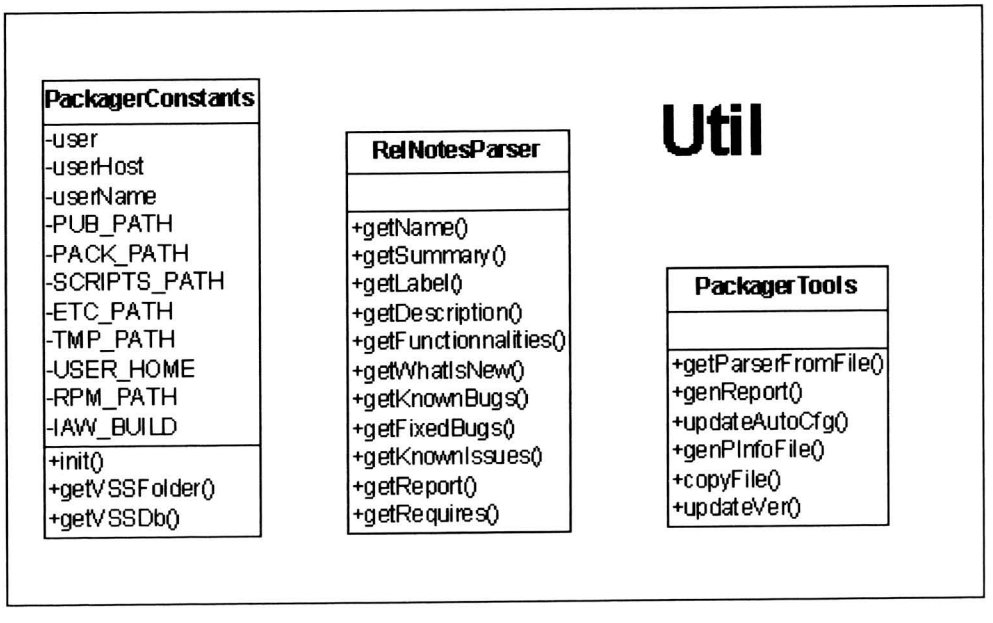
The second functionality offered by this package is Dialog creation. The abstract class DialogIB deals with basic functionalities of dialogs, such as formatting the layout and controlling the layout type. Each of the sub classes, namely, DialogLog and DialogEdit, implements a Dialog with specific characteristics as follows. DialogLog is a Dialog that just shows the content of a file or string. The DialogEdit, besides showing the content of a file or a string, enables the user to edit and save its content. The methods shown in the subclasses (DialogLog and DialogEdit) are methods implemented to enable Dialogs to show other buttons with specific functionalities than the basic ones.

The complexity involved with this package is related to the implementation of the tree and to the maintenance of the Dialogs in terms of compatibility with the extra functionalities offered. This package is the base for the easy implementation of the interface. With it any method of the application can easily create and customize a specific Dialog to show the content or to edit a string or a file. Moreover, the FolderTree implements all the needed interactions with the file system in a rather simple way.

The advantage of a utilities package is its reusability. These classes can be used in different and unrelated applications (Dialog classes were used in the Configuration Management tools (see chapter 5 Imagine Broadband Configuration Management Tools).

**Util**

The following diagram presents the architecture of the package Util.




13. Classes diagram: Imagine Broadband Packager Logical Package Util

This package consists in independent classes that offer diverse utilities to other packages of the application.

The first class, called PackagerConstants, just parses the configuration file (config.xml) and writes its content as constants accessible by any other class in the application. These constants can be accessed statically (without instantiation). In this class it is used Java introspection which enables a static method to read values from a file and to write them in the class dynamically.

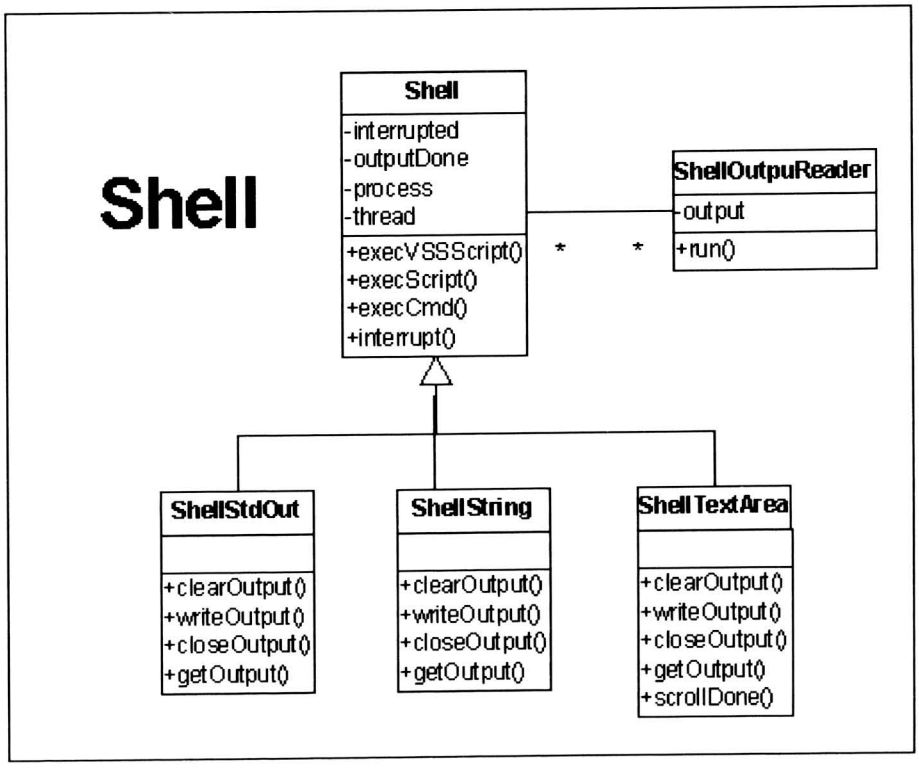
The RelNotesParser implements methods that enable the easy retrieval of the content of specific XML tags from a Release Notes file (see appendix A: ReleaseNotes.xml structure for further information about this file). The method getRequired and getReport are exceptions as they not only extract the content of more than one tag as they also parse it in order to obtain a different layout. This class uses Xerces to easily parse the content of the XML files (see chapter 3 Used Technologies for further details on Xerces).

The PackagerTools class implements some end-user functionalities that include the full implementation of the Use Case "Post Extraction Tasks" (see section 4.5.2 Requirements for further detail). The three post extract tasks defined in the requirements section are achieved by three methods in this class: genPInfoFile, updateAutoCfg and genReport.

The method copyFile of the class PackagerTools is a utility method that copies a file from one origin to a destination. The method updateVer is the generic method in the application to update all kind of version (from a version and an increment type).

**Shell**

The next diagram shows the architecture of the package Shell.




14. Classes diagram: Imagine Broadband Packager Logical Package Shell

The first class, called PackagerConstants, just parses the configuration file (config.xml) and writes its content as constants accessible by any other class in the application. These constants can be accessed statically (without instantiation). In this class it is used Java introspection which enables a static method to read values from a file and to write them in the class dynamically.

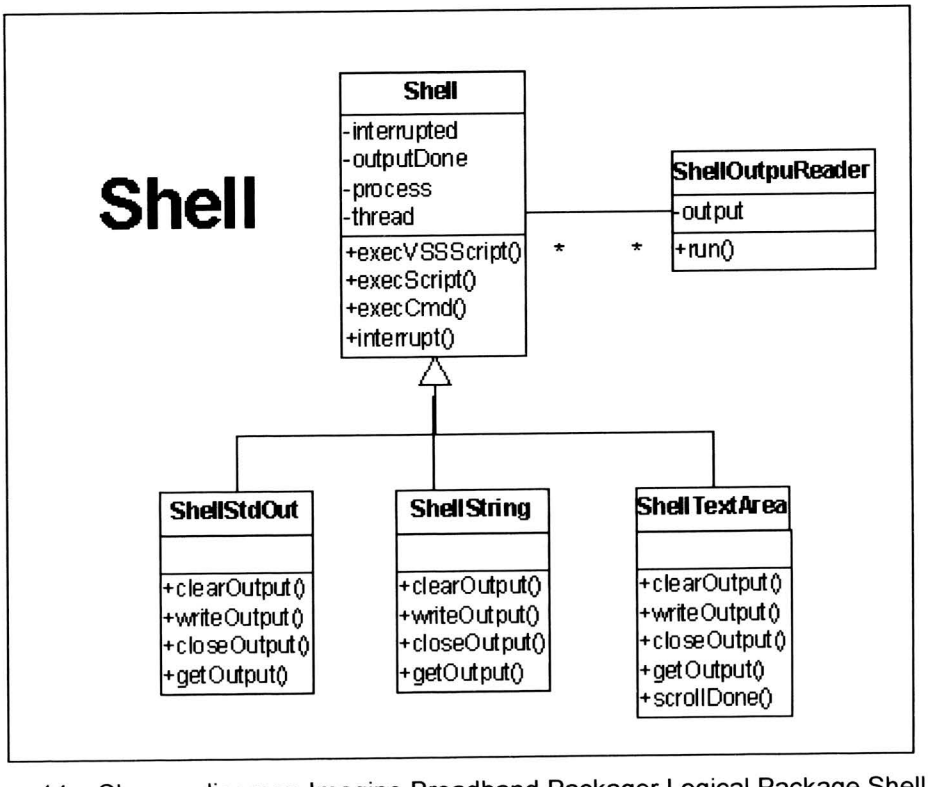
The RelNotesParser implements methods that enable the easy retrieval of the content of specific XML tags from a Release Notes file (see appendix A: ReleaseNotes.xml structure for further information about this file). The method getRequired and getReport are exceptions as they not only extract the content of more than one tag as they also parse it in order to obtain a different layout. This class uses Xerces to easily parse the content of the XML files (see chapter 3 Used Technologies for further details on Xerces).

The PackagerTools class implements some end-user functionalities that include the full implementation of the Use Case "Post Extraction Tasks" (see section 4.5.2 Requirements for further detail). The three post extract tasks defined in the requirements section are achieved by three methods in this class: genPInfoFile, updateAutoCfg and genReport.

The method copyFile of the class PackagerTools is a utility method that copies a file from one origin to a destination. The method updateVer is the generic method in the application to update all kind of version (from a version and an increment type).

**Shell**

The next diagram shows the architecture of the package Shell.




14. Classes diagram: Imagine Broadband Packager Logical Package Shell

This package represents the implementation of the interface between the Java application and the Shell (see chapter 3 Used Technologies for a definition and type enumeration).

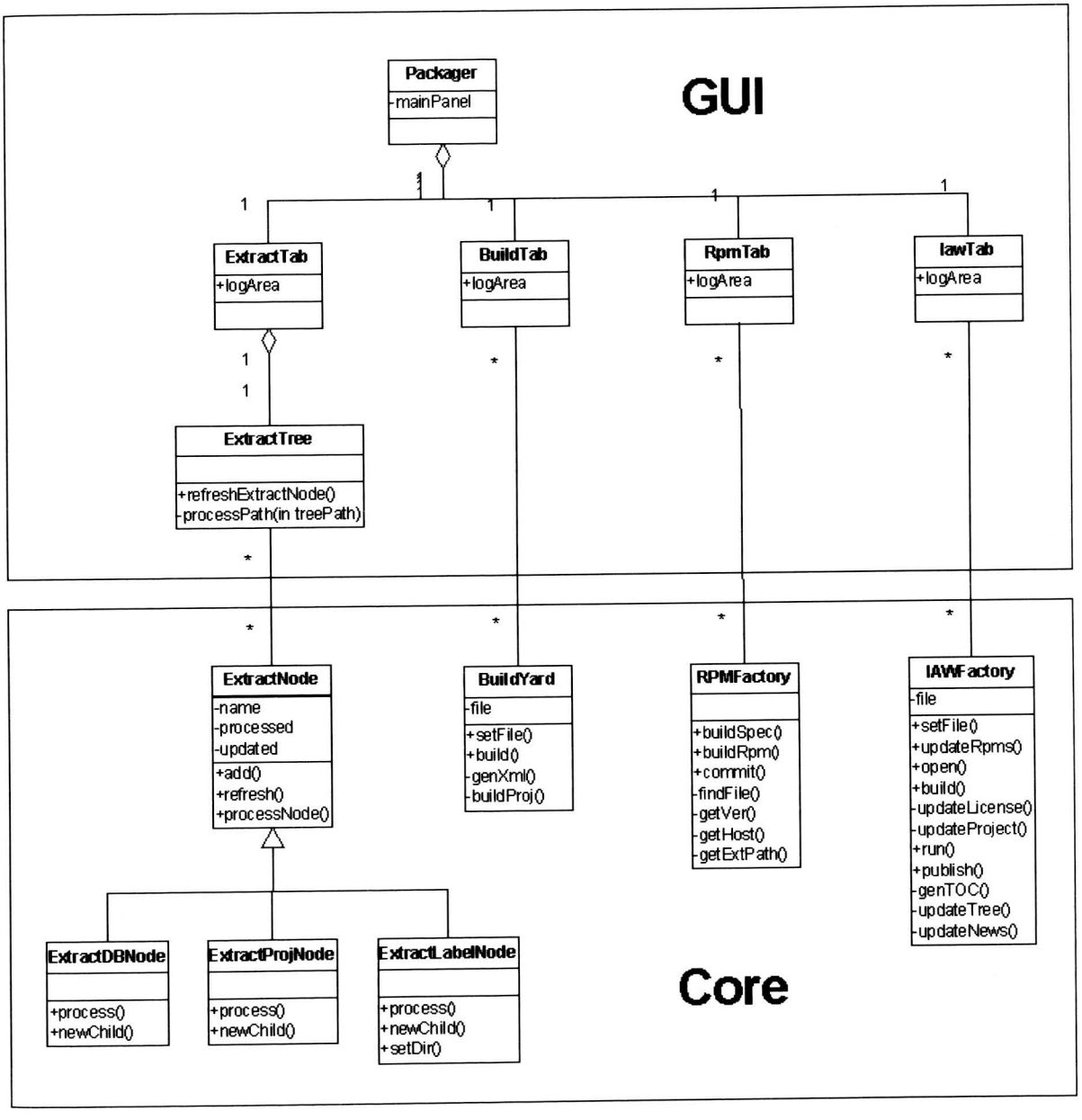
The main implementation of the interface is in the abstract class Shell. It implements methods that execute commands or shell scripts and write the output to a specific output. This specific treatment of the output is made in each subclass that maps various types of output, namely, ShellStdOut, ShellStr and ShellTextArea that map, respectively, the standard output, a String and a TextArea (JFC component). The Shell interface also enables one to interrupt the execution of commands or scripts as to extract the complete output of a finished execution.

The class ShellOutputReader is a thread that runs as a daemon and reads the output of a command (directly from the process) and writes it into the Shell using specific methods offered by the subclasses of the class Shell.

Each subclass extends the class Shell and implements the abstract methods in it. These methods are: clearOutput, writeOutput, closeOutput and getOutput. The first three methods are used by the ShellOutputReader to interact with the Shell and the fourth is part of the implementation of the Shell interface, it returns the content of the output when the current execution is finished.

### GUI and Core

The next diagram presents the main architecture of "Imagine Broadband Packager". In this diagram the two main packages are described: the GUI and the Core. They are shown together to ease the understanding of the connections between them.




15. Classes diagram: Imagine Broadband Packager Logical Packages GUI and Core

This diagram encloses all the main logical architecture of the application and is with its components that all the previous packages interact.

As it was expected, from the structure of the Packaging Process, a vertical division emerges in this diagram as it conceptualise also the five basic steps of the Packaging Process (see picture 4). In the tool, the five basic steps are reduced to four sections. This happens because the publication of a package is done with a simple action in the Install Anywhere section (see picture 10).

### GUI

The main class of the application, the class Packager, is a rather simple class as it extends a JFrame (JFC component which represents a Window) and places in it four JTabbedPanes (JFC components, see the upper left corner of the window shown in picture 17).

Each of these four tabs contains one of the four main interface classes, namely, ExtractTab, BuildTab, RpmTab, and IawTab. These classes implement the interface of each of the four main steps of the Packaging Process.

All the four main panels have a similar format based on three components: a toolbar with buttons (from where the actions are taken), a tree (where the project/file selection can be easily done) and a log area (where the output of scripts can be written).

The first important interaction between this logical package GUI and another one is the use of the FolderTree utility. The ExtractTab has a special kind of tree that will be described later but the other tabs have standard file trees. These trees are totally implemented by the class FolderTree of the package GUIUtil. So, each of the tabs has a FolderTree directly associated that is used to interact with the file system.

As said before, the ExtractTab has a special kind of tree. This tree is implemented by the class ExtractTree. Instead of mapping a directory structure, it maps a VSS server (list of databases). This tree has always a static number of levels which is three. One to map available databases, the second to map projects inside each database and the last one to map labels inside each project (see chapter 3 Used Technologies for detail on these concepts). This class has the method refreshExtractNode that is used to update the information shown in the tree and other internal method processPath that is the most important as it calls the appropriate methods in the tree components to make extractions from VSS.

### Core

Descending in the diagram to the Core package one may see that each of the interface panels has access to its factory (class in which tasks are executed).

One ExtractTree contain several ExtractNodes that are responsible for the processing of data and execution of extraction tasks. As explained before, there are always three levels in an ExtractTree. For each level there is a specific type of Node (ExtractDBNode, ExtractProjNode and ExtractLabelNode). These Nodes are responsible for the implementation of the specific extraction methods for each type of node (see section 4.5.6 Development Details for further details on these classes).

The first factory is called BuildYard. This class implements all the methods related to the building of a project. It is the implementation of the packaging step "Source Code compilation" (see picture 4) and the Use Case "Build Package". The most important method implemented is called build and is the one that is used to build a package (see section 4.5.6 Development details for further detail on this class implementation).

The second factory, RpmFactory, is the exact mapping of the packaging step "RPM Package Build" and the implementation of the Use Case with the same name and its derivations. The methods of this class will be detail in section 4.5.6 Development Details.

The final factory is called IAWFactory and implements the two last packaging steps, "Install Anywhere project build" and "Package Publishing". This class implements the Use Cases related to the two mentioned packaging steps.

These two packages, GUI and Core, enclose the major part of the implementation of the tool and, as such, they were the most time consuming task in the development of the Packaging Tool.

All this final classes (factories) and the extracted nodes have to use the shell to run some specific shell scripts that do the most basic work as running shell commands or even managing files. The logical package "Shell" offers an "easy to use" interface with the Shell. This interface consists in a list of methods that enables any class using the interface to run scripts or commands with a single line of code and to redirect the output of those scripts or commands to wherever they need. This interface was used in two different ways.

The main use of the Shell interface was done integrating an instance of the Shell interface (ShellTextArea) in each factory and having the output of the scripts redirected to the respective interface log area (which is an attribute of the tabs in the GUI logical package). This schema will be more detailed in section 4.5.6 Development details.

The other standard way of using the Shell interface is to create an instance of a Shell interface (ShellString) and run the scripts or commands in it and getting the output with the method getOutput of the Shell.

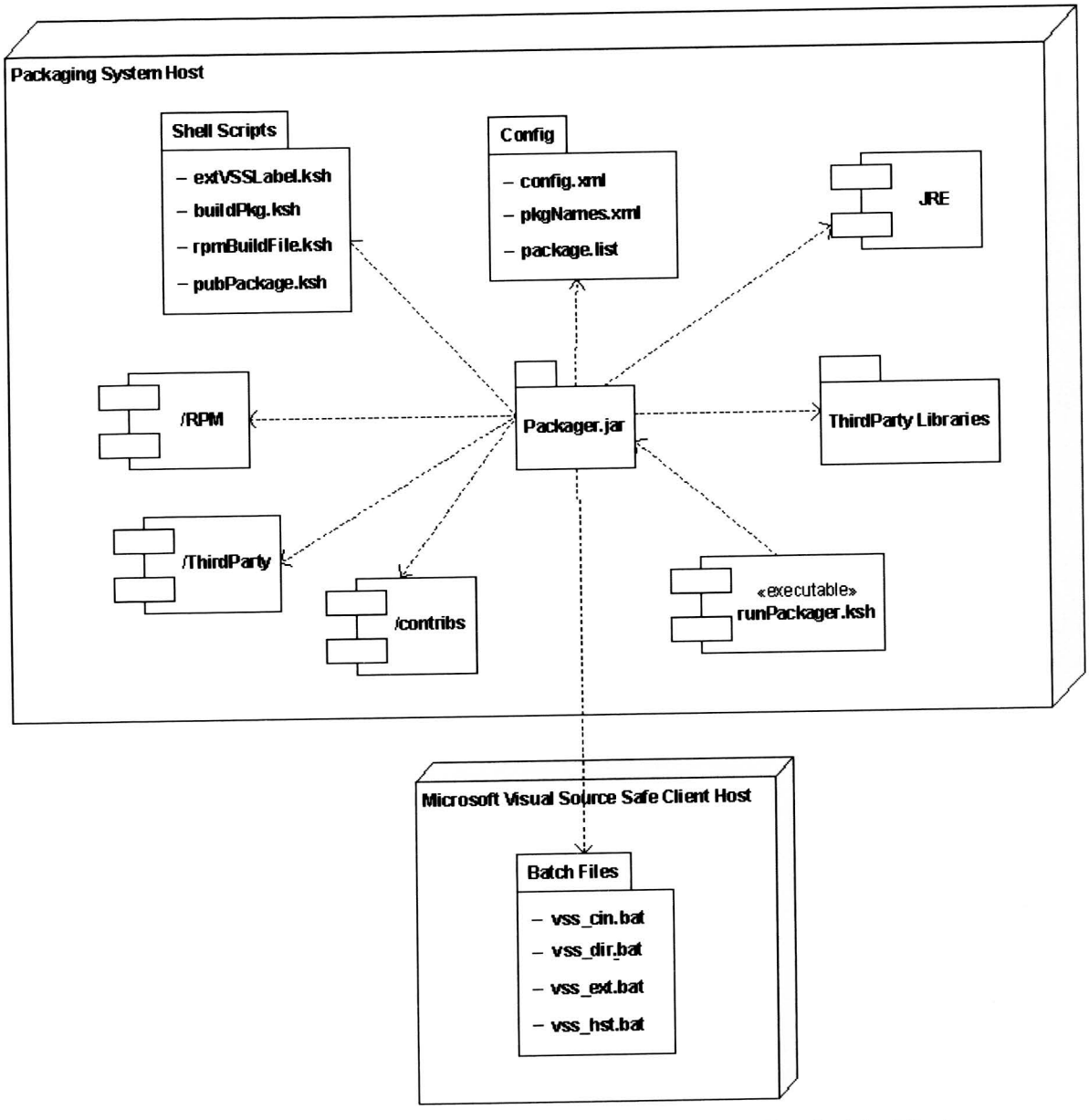
All the classes in these two main logical packages use the Dialogs implemented in the GUIUtil logical package. To use those dialogs one just need to create an instance of one of the dialog classes (DialogLog or DialogEdit) and customize it as needed.

The logical package Util of the same layer as the Core is fully used by the factories.

This section should leave clear all the design of the application as the main way the diverse parts of it interact with each other.

### 4.5.5 Physical Architecture

The following diagram shows the Physical Architecture of the Packaging System.




16. Components diagram: Imagine Broadband Packager Physical Architecture

This diagram can be completed with the picture 5. The tool should be executed via a startup script (runPackager.ksh in the diagram). This script should only invoke the proper Java class enclosed in the archive Packager.jar. This archive represents all the Java files that implement the tool. The package "Third Party Libraries" contains Third Party Java Archives with classes that implement several utilities. The component "JRE" represents the Java Runtime Environment which represents the Virtual Machine running inn the system (see chapter 3 Used Technologies for further details on Java technology).

The package "Config" encloses three configuration files, one for the application itself and the other two for the packaging system (specific tasks configuration).

The package "Shell Scripts" contains all the scripts that are used by the tool for packaging purposes, i.e. scripts that in one way or another perform a specific packaging task. This physical package is exclusively used by the logical package "Shell".

The three components, "/RPM", "/ThirdParty" and "/contribs" are in fact three auxiliary repositories that are used in the packaging process for specific purposes. The "/RPM" component is a repository of built RPM packages, "/ThirdParty" is a repository of Third Party libraries that is used to compile the Source Code and "/contribs" is the biggest repository with all the versions of all the compiled modules and is used as dependencies supplier for compilation.

In the node with the VSS client there is only one package that contains the VSS access scripts (see section 4.3.2 Microsoft Visual Source Safe Extraction for details on the use of these scripts and section 3 Used Technologies for further information on Batch Files).

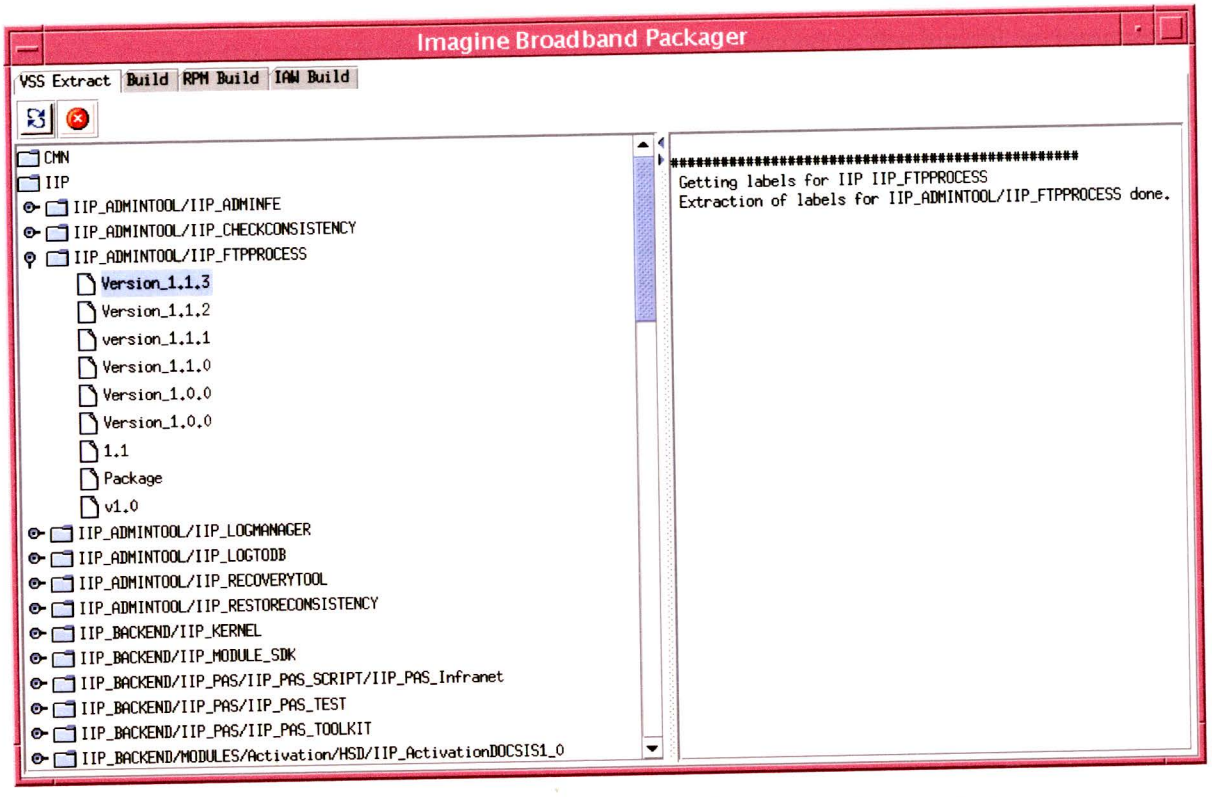
This basic physical architecture enables one to understand the not so simple structure of a framework like the one being described.

### 4.5.6 Development Details

In this section some development details will be selected to give a more deep insight over the new Packaging System. This section will also make a brief overview of the Graphical User Interface and the way it should be used, connecting this information with the development details.

### Extraction

The next picture shows a screenshot of the "Imagine Broadband Packager" in the first tab called Extraction Tab.




17. Screenshot: Imagine Broadband Packager Extraction Tab

In this picture the basic application interface structure can be understood. The window contains four tabs (top left corner of the picture): "VSS Extract", "Build", "RPM Build" and "IAW Build". These four tabs were already described in previous sections. For each of these four tabs there are three main interface components: the toolbar, the tree and the log area.

In this tab, the toolbar only contains two buttons, the refresh button and the stop button. Both these buttons are present in all the tabs. The former is used to update the information contained in the tree (whatever is the type of tree) and the latest is used to stop the execution of commands.

The log area (shown in the right side of the window) is used to show the output of the commands being executed.

The tree (shown in the left side of the window) used in the ExtractTab is different from a standard folder tree as it shows not files and a directories structure but VSS databases, projects and labels. For example, in the previous picture, the databases are "CMN" and "IIP" (tree roots). The database "IIP" is expanded and its projects are shown. In this particular example, the project named "IIP_FTPPROCESS" (which is inside a organizational folder called "IIP_ADMINTOOL") is opened and the tree shows the labels of that project (see chapter 3 Used Technologies for further details on concepts like VSS databases, projects and labels). The interaction with the user is done via mouse actions that can open databases and projects with clicks or update them with the "refresh" button in the toolbar. To extract a project from VSS the user needs to double click in a label to execute the command

("Version_1.1.3" in the picture, for example). After the extraction of files from VSS, some post extraction tasks are performed. The result of these actions are showed with popup windows that appear with the content of updated files like the Package Information File, the AutoConfig file or the generated Reports.

These actions are implemented via the ExtractNodes (see section 4.5.4 Object Data Model). Each of the nodes in the tree has a type that is mapped in the logical architecture of the application with the classes ExtractDBNode, ExtractProjNode and ExtractLabelNode. These classes implement a method called process that extracts the information from VSS according to the type of node. The classes ExtractDBNode and ExtractProjNode use temporary files that enclose information from VSS and can be updated, as already said, with the use of the button refresh. The method process in the ExtractLabelNode class is the method that does the actual extraction of files from VSS using polymorphism. These methods are called using the ExtractTree class and the extraction commands are executed within them.

**Build**

In the "Build" tab the interface is very similar to the "VSS Extract" tab. The biggest difference is the type of the tree. In the "Build" tab the tree is a "FolderTree" that shows directories and files in the home directory of the user. With this tree, the user can select the project to build.

Besides file manipulation actions like adding or deleting files or folders, the toolbar in the "Build Tab" offers the most important feature in this tab: the "build" button. After selecting a folder with a project to build, the user can use the button "Build" to build the project.

The new Packaging System offers a rather simple and quick way to build a package. One single method is used to do all the work, the method build which is the main interface of the "BuildYard".

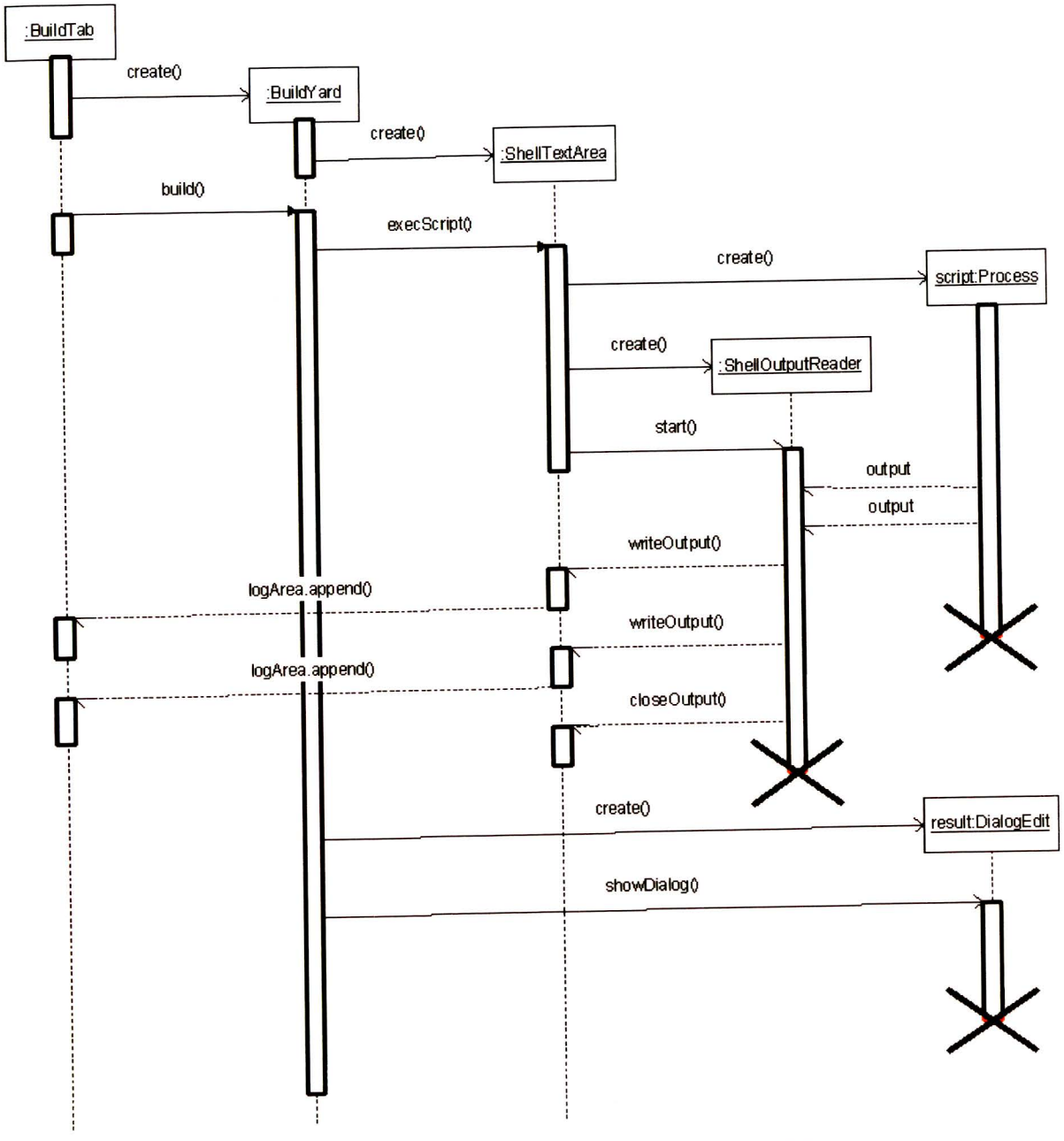
Four basic steps are now used to build a package: setup build environment, generation of the build.xml file, compilation and JAR file presentation.

The first step consists in the inclusion of dependencies for the build. This step is now totally automated. It is performed by parsing the release notes and copying all dependencies to a directory that included in the classpath for compilation.

The second step, the generation of the build.xml file, suffered some modifications worth describing. To build a project, the user needs a build.xml file (see section 4.3.3 Source Code Compilation for further details). This file can be automatically generated with the use of a XML configuration file (see appendix B: Ant's build.xml automated generation) or it can be simply copied from a static repository where specific build.xml files are kept. If there is a specific build.xml file in the repository, it is used; otherwise, the configuration file is used to generate the build.xml file. This feature conjunctly with the enhanced automated generation of the build.xml file transformed the build of a package in a rather easy task. Moreover, after the build of a project, the user is now able to see the content of the JAR file created by the third step: compilation.

The method build in the "BuildYard" should now be easy to understand since it just proceeds using auxiliary methods that implement the four steps described above.

The following sequence diagram represents the interaction between the Shell, the ShellOutputReader, the Factory and the GUI in a usual use case: the build of a project. This sequence happens when the user presses the "Build" button on the toolbar of the "Build" tab and activates a mouse action handler implemented in the BuildTab.




18. Sequence diagram: Executing a Shell script while building a project

A lot of information can be extracted from this diagram. The BuildYard and the ShellTextArea are created in the beginning of the application when the interface is setup.

The mouse action handler, in the BuildTab, is invoked, and in its turn, it invokes the method build in the BuildYard. This method does all the work related to building a project (environment setup, build.xml generation, compilation, and post compilation tasks). The

diagram focus not on the build of a project but on the utilization of the Shell interface, i.e. the interactions occurring after a "Factory" executes a Shell Script. In this case the "Factory" is the "BuildYard".

To execute a script, the "Factory" just needs to invoke the method execScript in the Shell with the proper arguments. The Shell, to execute the script, takes the following actions: launches a Process to run the Shell Script, links the Process with a ShellOutputReader and starts it. The ShellOutputReader can (using a stream) directly read the output of the Process (Shell Script executing). It uses the method writeOutput of the Shell so that the output can be appended to the log area in the interface. After the invocation of the method closeOutput in the Shell, usually, a Dialog box is created to show the content of some file created by the Shell Script. In this particular case, the Dialog is used to show the content of the JAR file created after the compilation.

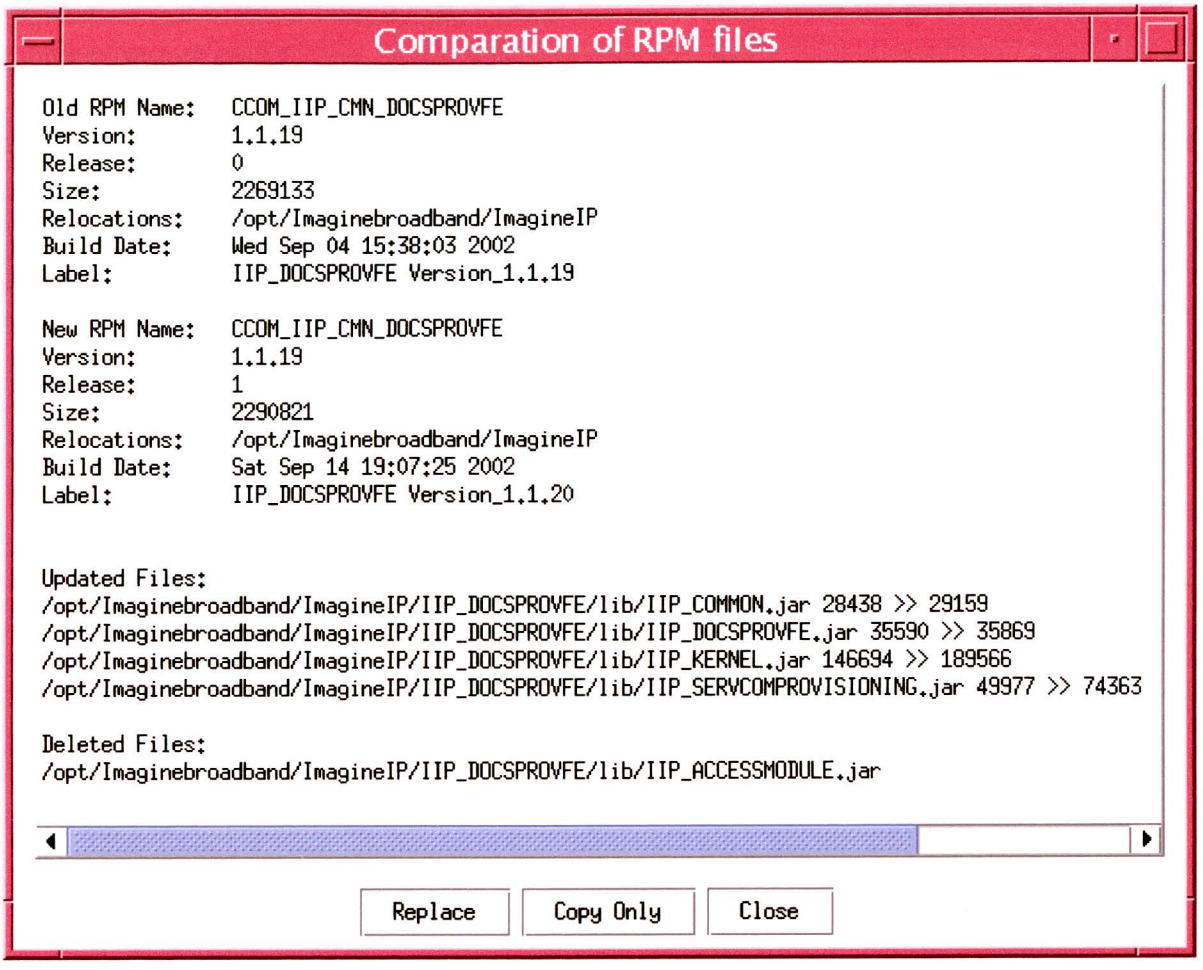
This process encloses the biggest complexity of the tool. The interaction between the application and the Shell had to be easy to use and effective, as such, the implementation of this package was relatively time consuming and it was done gradually with consecutives improvements.

**RPM Package Build**

The RPM package build details are not worth describing in this document. The class "RPMFactory" encloses all the methods required to make the build of RPM packages in a simple manner. Analysing the results of the work done in this part is enough as no special complexity was involved in its improvement ("RPMFactory" implementation). Nevertheless, one detail in this step of the Packaging Procedures should be described: the update of the RPM packages in Install Anywhere project.

This feature simply did not exist in the Former Packaging System. The new Packaging System, if thought from the functionality point of view, makes this task in a single click and moreover it enhances the process by increasing the level of control over it.

After creating a RPM package the user can see the content of it. Meanwhile, other functionalities are offered: the commission to VSS of the version of the package (see section 4.3.4 RPM Packages Build) and the update of Install Anywhere project. Before doing this update (which is done using a configuration file to get the location of the respective Install Anywhere project) the application offers the user the possibility of checking the differences between the new RPM package and the former. The next picture shows an example of that comparison.




19. Screenshot: Imagine Broadband Packager RPM packages comparison

### Install Anywhere Build

The reasoning applied to RPM packages generation is also applied to the build of an Install Anywhere project. The analysis of the results over the improvement is considered enough. Again as the RPM package build, the specific "Factory" implements all the methods required to fully automate this task: "IAW Factory".

The publication of a new package in the Delivery Center was totally renewed and automated with the new Packaging System. This task is performed in the application with a simple action where a Dialog box is shown with the content of the Table of Contents of the package and a button offer the user the possibility to publish the package.

### 4.5.7 Summary

This section detail the New Packaging System: Imagine Broadband Packager. This new tool enhanced drastically the execution of the Packaging Procedures. It consists in a Java application that offers a friendly GUI where the user can perform the packaging steps with a couple of mouse clicks.

To implement an application like this, a deep knowledge of the Packaging System was required as a good prediction of what would happen in the Packaging scenario of the company.

In this section the application was fully conceptualized and some development details were described as well as the GUI.

### 4.6 Results Analysis

As it is important to compare past and present systems to get a metric of success and as this is the most important fraction of the internship, this section, will present an analysis of the results obtained with the new Packaging System.

As seen previously, almost none of the Packaging Procedures have changed. The same happened to the main structure of the packages. By the contrary, the Packaging System itself suffered great improvement with the development of the new application. Nevertheless, all the system was changed. During the internship, all the details in the scripts, processes and structures were changed at least for adaptation to the new Packaging System. Even the scripts described in section 4.4 Former Packaging System were improved in the beginning of the internship and then adapted to fit in the new system.

The structure of the packages and the Packaging Procedures were maintained because there was no need to improve or adapt them in a considerably scale besides the small adaptations that were done.

The first major option of this internship was taken when it was decided that Java would be used for all the development of the new tool and that Shell scripts should be reduced. This option has advantages and disadvantages as follows.

The major advantage of writing new packaging tools with Java is related with the experience of the intern with it. Besides this, other points should be mentioned. Java can offer the needed abstraction to a process that is always evolving. Java is a high level language that enables the implementation of features in a more lasting way than the lower level Shell script language. For example, the use of configuration files is clearly easier with Java than with Shell script language; not only because of its XML parser but, for example, because of the easiness of attributes manipulations. The third party libraries that can be used with Java for the execution of various tasks are another important advantage of using Java. With Java the code is much easier to maintain. With the reformulation of the packaging tools, the restructuring and update of the packaging system is also done at the same time.

The disadvantages of completely restructure the packaging tools are related to the need of a complete new development and to the translation that had to be done of already functional code in order to improve it.

The following paragraphs will make a brief comparison between the performances of the two Packaging Systems in terms of each of the packaging steps.

All the steps were improved in terms of easiness of use as the usability of the new GUI enhancement is enormous compared to the difficulty and slowness of use of the Shell script menus. Other point is related to the automation of the process as before there were a lot of actions to be performed manually and now all the process is completely automated.

The extraction of development projects from the Version Control System (the first packaging step) was deeply improved. Now, the extraction of the development projects (Source Code) is done with some simple actions in the Graphical User Interface. Before, it

was done with the help of a script. Besides, now the post extraction tasks are performed totally automatically as before much of the work was done manually.

The build of a project was improved in terms of control over the process. This improvement is related to the control over the build.xml file content and its generation but also to the setup of the build environment. Before, this setup was not controlled and now, only the dependencies of a given project are automatically included for compilation.

In the Former Packaging System there was no control over the versions of the dependencies of a project and the trace of a compilation problem can be stored in a log file that is used to report the problem to the development team. This process was not done automatically before.

As a final improvement detail on the build process, the presentation of the built JAR file at the end of the build process can be mentioned as the example of the enormous difference between the two Packaging Systems being compared. In the Former Packaging System one should find the JAR file and run a specific command to get its content (without version control). In the new system, after the compilation, a dialog box pops up showing the content of the JAR file as well as its manifest file with information about the version of the JAR file (module).

The creation of the RPM packages was also improved in terms of control over the process but especially in terms of simplicity. The workings of the Shell script responsible for the creation of RPM packages in the Former Packaging System were far from simple as the creation of multiple types and quantities of packages was allowed with the help of consecutive menus. In the new Packaging System, the creation of RPM packages is done with a simple interaction with the GUI and the control is all done through the easy to access RPM Specification file and the Package Information File made available through the GUI. Moreover, all the results are showed in the log area or with popup windows.

The build of the Install Anywhere project is the most improved packaging step as a lot of actions were taken manually. The new Packaging System transformed this step (including the package publication) in a collection of actions performed with some mouse clicks in the GUI. The automated version update and package publication are the most important improvements in this step.

It's important to describe what kind of work loads are involved in this process. The number of packages to be done in a week may vary a lot but, in a day, may go beyond a dozen. This fact is very important to the discussion being taken as the improved process is repeated many times. This explains why a small enhancement in one of the packaging steps can make a large difference in the time consumed with the packaging of a module.

To show the quickness improvement of the Packaging System a small example can be described. While in the beginning of the internship the complete packaging process could take nearly one hour for a couple of modules (this process involved a lot of file manipulation and manual update of values). Now, with the new Packaging Tool, the process can be done in less than 15 minutes and these are spent launching actions in the GUI, checking execution traces and the work being done by the tool.

The amount of flow control over the process that the new Packaging system offers is also very significant. This control is often given by automated checks but also in the form of control points where results can be visually checked. The amount of flow control offered is always inversely proportional to the error tax. This last statement is crucial to this analysis.

As conclusion, it can be said that all these changes bring a lot of advantages to the Packaging System. The major ones are related to process control, speed and automation that led to a high increase of productivity. The real difference between past and present can be shown by stating that in the beginning of the internship two persons were needed to do all Imagine Broadband packaging and in its end, only one person was assuring all the packaging with the help of the new "Imagine Broadband Packager".

### 4.7 Conclusion and further work prespectives

The position in which the intern was placed in the company enabled him to communicate with other sectors of the company. This happens because of the interactions described in picture 1. These interactions are considered very important as they enable the real contact with other members of the company and the understanding of the culture of the company.

Although, it was impossible to predict whether or not the available time (beyond packaging duties) would permit the intern to efficiently develop the required system, all the objectives were successfully accomplished.

Since the beginning of the internship packaging duties were frequent. New packages had to be produced in a high rate. With the new packaging system, the time consumed in the execution of these tasks was gradually decreased. Nevertheless, the productivity, related to the development of the new tool, was deeply affected by the packaging duties themselves. It is very important to notice that the development of the packaging tool was done in between the execution of the packaging duties.

The most important point regarding the work done is related to effectiveness. In a company like Imagine Broadband the ultimate goal is effectiveness and this internship was rich in increasing it as the packaging resources productivity.

It is time to consider some future work perspectives related to the Packaging System. This work is based on the maintenance of the application developed which does not appears to be a complex task. Other maintenance task of the tool would be the adaptation to the new Source Control System that will soon be used at the company: PVCS.

The real improvement to the tool is the complete automation of the process, which is to say, the removal of the user interface. This new Packaging System would just report errors occurred during the packaging process for an administrator to analyse those reports. This would transform the tool in an intelligent agent that would do everything alone just reporting occurred bugs and logs to an administrator.

## 5 Imagine Broadband Configuration Management Tools

Configuration Management consists basically in every task done to manage the state of a given system. This section will describe in full detail the Configuration Management tools developed during the internship.

There was much to do about System Configuration on Imagine Broadband products in the beginning of the internship. It was defined that some tools should be developed in order to handle tasks like package de-installation, installation report, installation backup and rollback functionalities, among others.

As seen in the previous chapters, the major products of Imagine Broadband are deployed in the form of RPM packages (enclosed in Install Anywhere graphical installers). These RPM packages usually enclose product modules. As modules were delivered, they were installed on target platforms and put together with other modules already installed there. This is the basic context in which Configuration Management tools should be thought. Versions management as well as system backups are the main ingredients for these tools.

The primary activities of Configuration Management are identification, change control, status accounting, and configuration audit. [32] During the internship two Configuration Management tools were developed conjointly, the Rollback Tool and the Rpm Browser. The RpmBrowser does identification and status accounting and the Rollback tool does change control. Next sections will explain these facts.

The only platform requirement for these tools is the existence of RPM (see chapter 3 Used Technology) installed in the target platform.

The target products of these tools are Imagine Broadband products that are installed using RPM, in other words, any product installed in the form of a RPM package may be the target of these tools.

These tools were developed in Java because Java was considered the most powerful language to develop Graphical User Interfaces to integrate in the specific target platform. One of the first tasks of the intern was to develop a simple interface between a Java application and the Shell. The fact that this task was achieved easily also reinforced this choice.

The tools were developed before the "Imagine Broadband Packager". In this report, it is described after it, mainly, because packaging should serve as background to this part of the internship. Besides, these tools were done after the study of the packaging procedures and after the first improvements of the packaging system, still based on Shell scripts.

The next two sections will describe each of the Configuration Management tools.

## 5.1 Rpm Browser

### 5.1.1 Introduction

The RpmBrowser is a simple tool that enables anyone to report the installation status of a given platform.

This tool should permit a user, in the target platform, to get a report of the installation status. This report can then be sent to the development center for debugging and can be used to track some problems in a specific release of a module.

The usual procedure would be the detection of a problem in the client platform, the use of the RpmBrowser to report the configuration of the system and sending of that report to the development center where the report could be used to replay the configuration in a debugging environment. With this, the problem could be easily detected and a correction would originate a new delivery which would end the process where the RpmBrowser accomplished an essential role.

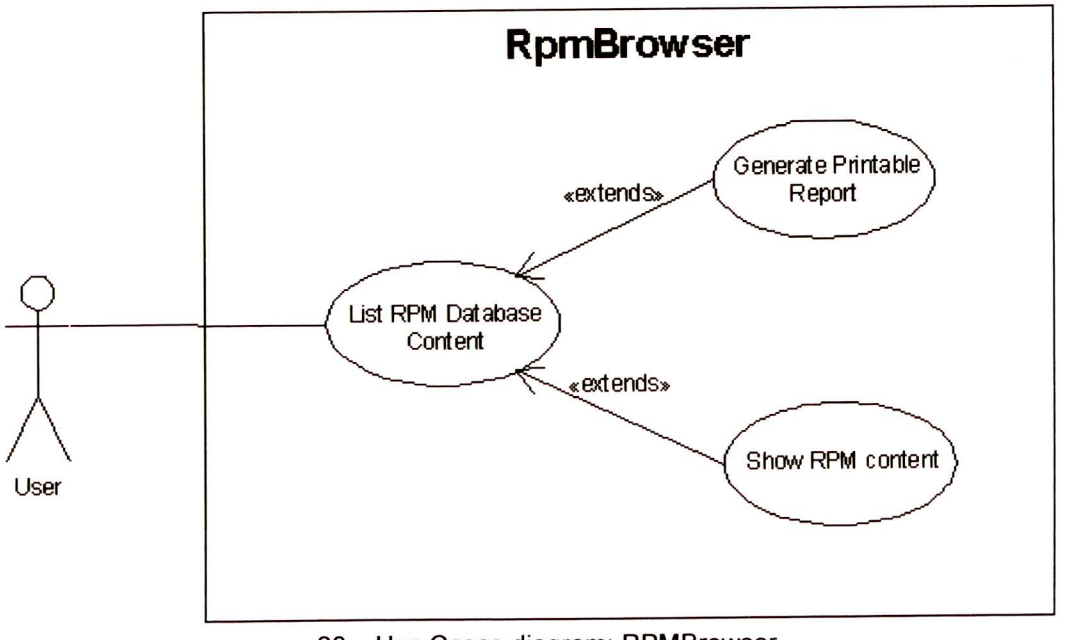
The RpmBrowser was required to improve the visibility over the configurations of target platforms. This stands for any product of Imagine Broadband that is deployed with RPM.

This section will detail the design and the implementation of the RpmBrowser.

### 5.1.2 Requirements

To achieve the pretended goals the tool should implement a Graphical User Interface in which the content of the RPM database could be shown and easily archived.

The following Use Cases diagram can illustrate the functionalities the RpmBrowser should offer.




20. Use Cases diagram: RPMBrowser

There is only one actor as it can be called "User" that represents the person working in the client side platform.

The basic Use Case is identified as "List RPM Database Content" (see chapter 3 Used Technologies for further information on these concepts, as RPM packages and RPM database). This Use Case consists in the graphical visualization of a list of RPM packages installed in the platform where the tool is running. This list should be enriched by additional details on each RPM package.

This main Use Case is extended in two more Use Cases, "Generate Printable Report" and "Show RPM package content".

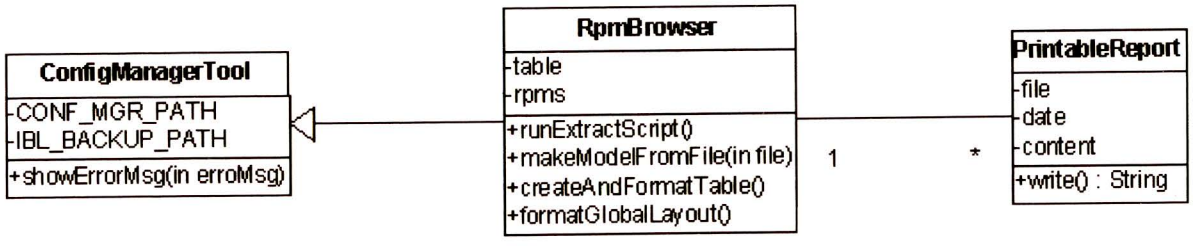
The Use Case "Show RPM content" represents the possibility of getting the complete information about the content of each of the RPM packages listed.

The Use Case "Generate Printable Report" consists in the generation of the same list as in the main Use Case but in a printable format, i.e., a file.

After the description of the requirements for this small tool a description of the design and then of the implementation will follow.

### 5.1.3 Object Data Model

This tool was implemented in a single Logical Package as it is rather simple. The Logical Package can be called RpmBrowser and its classes are the following:




21. Classes diagram: RPMBrowser

In this diagram we can see the structure of the implementation of the tool. There is a main class that offers some basic methods for error reporting and constants definition called ConfigManagerTool. This class will be also used in the Rollback tool as it is a generic class.

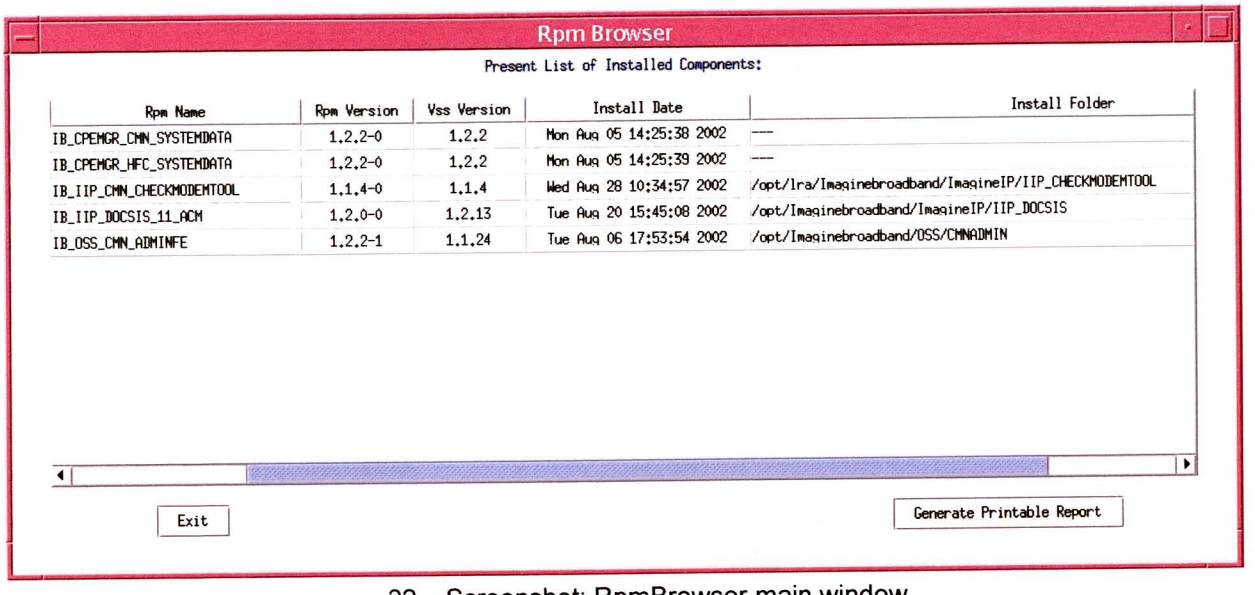
The class RpmBrowser is a subclass of the class ConfigManagerTool and is the class that implements almost all the functionalities of the tool. The exception is the PrintableReport class that implements all the functionalities related to the Use Case "Generate Printable Report". The PrintableReport class enables the class RpmBrowser to easily create and show a Printable Report as required.

The methods defined in the class RpmBrowser are used almost sequential and the only ones worth mention are the method runExtractScript that reads all the content of the RPM database to a file and the method makeModelFromFile that converts that file to a JTable (JFC component).

### 5.1.4 Development Details

The Dialog classes from the GUIUtil logical package that belong to Imagine Broadband Packager (see section 4.5.4 Object Data Model of Imagine Broadband Packager) were developed and used for the first time in the RpmBrowser. They were omitted from the class diagram for conciseness.

The following picture shows a screenshot of the RpmBrowser interface.




22. Screenshot: RpmBrowser main window

As seen in the picture, the RpmBrowser shows a table with the list of all RPM packages installed on the machine where the tool is running. This table contains the most significant fields related to the RPM packages. For each entry in the table the user can get all the information of the selected RPM package, this includes fields like: Build Date, Development Project Reference or Build Dependencies, among others.

The button "Generate Printable Report" seen in the lower right corner of the picture enables the user to get a formatted report in a file or in a printer.

No major complexity is involved in the implementation of this tool. The scripts written for this tool (RPM database read, etc.) were the first scripts elaborated during the internship and, as such, the time consumed for their implementation was slightly larger than usual.

### 5.1.5 Conclusion and further work prespectives

There are some improvements that could be made to this tool, not related to the implementation, but to functionalities, such as, RPM packages de-installation, file checking or even dependencies checking on each RPM package.

This tool is not yet delivered to the client despite it is a simple task to do it. The main reason for this is that it is not a priority. The delivery to the client or even to internal use is planned for soon.

This tool is considered a prototype as it encloses the basis for a complete Configuration Management framework.

## 5.2 Rollback Tool

### 5.2.1 Introduction

The Roll Back tool was also developed to improve Configuration Management in target platform of products of Imagine Broadband.

The idea that led to the development of this tool was that after an unsuccessful installation of a module, there was no way to get back to the previous configuration. And, as the configuration of the modules (after installation) is a time consuming task, this problem was considered important.

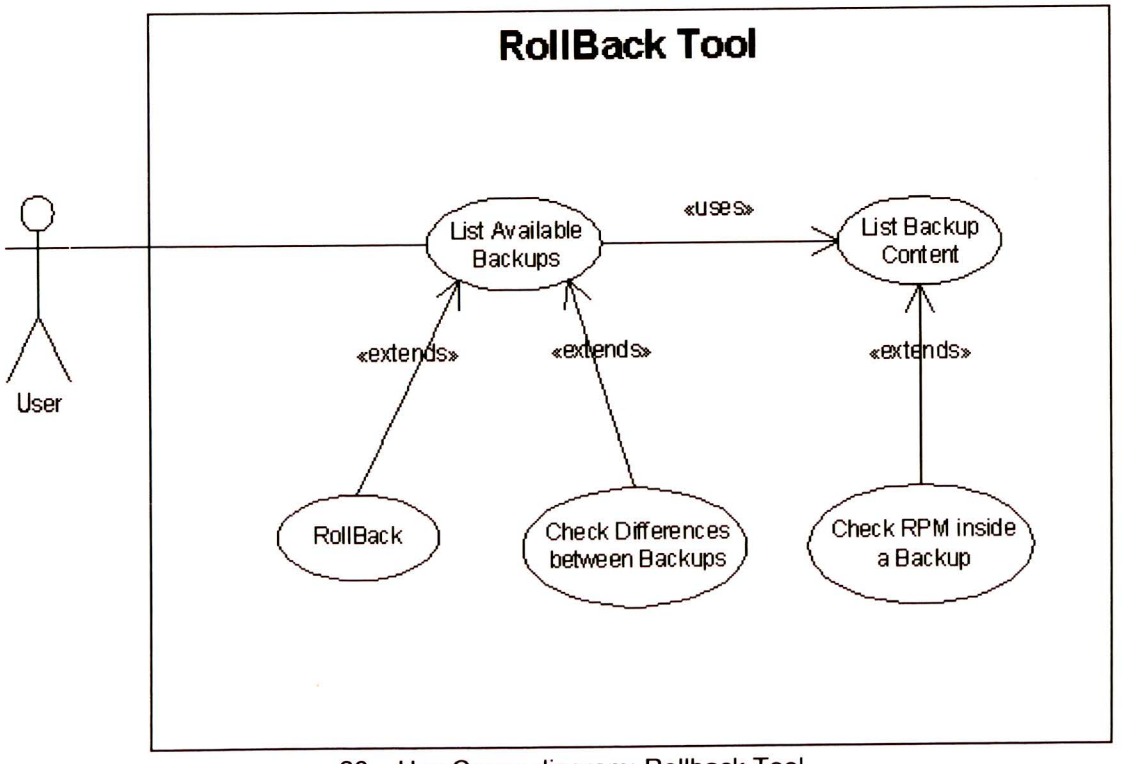
The developed tool basically enables the user to rollback a complete installation to a previous state, this includes, all Imagine Broadband products installed in the platform. To achieve this, another major task had to be done: System Backup.

This section will briefly present the requirements and architecture of the Rollback Tool as its main development issues.

### 5.2.2 Requirements

It was written a document that specifies the requirements of the rollback tool. This document only included the lowest level requirements (functionalities). Graphical User Interfaces and extra functionalities were conceptualized later (see appendix E: IBL Rollback Tool Specification).

The following Use Cases diagram represents the basic requirements defined in that document for the Rollback Tool and also includes extra functionalities defined later.




23. Use Cases diagram: Rollback Tool

There is only one actor that represents the person installing the software and using the tool.

The Use Cases presented are:

**List Available Backups**

This Use Case consists in the simple presentation of the list of available Backups of precedent system states.

**Rollback**

For each Backup, the user should be able to Rollback to the system state enclosed in it. As the name says, this is the main Use Case of the tool.

**Check Differences between Backups**

This Use Case consists in the presentation of a comparison table between two (or more) available backups. This table should list all the modules in backups and the versions in each backup. This functionality is an extra and it is not fundamental for the workings of the tool, as such, this Use Case extends the basic one: "List Available Backups".

**List Backup Content**

This Use Case is "used" by the first Use Case and it consists in the listing of each of the modules inside a backup.

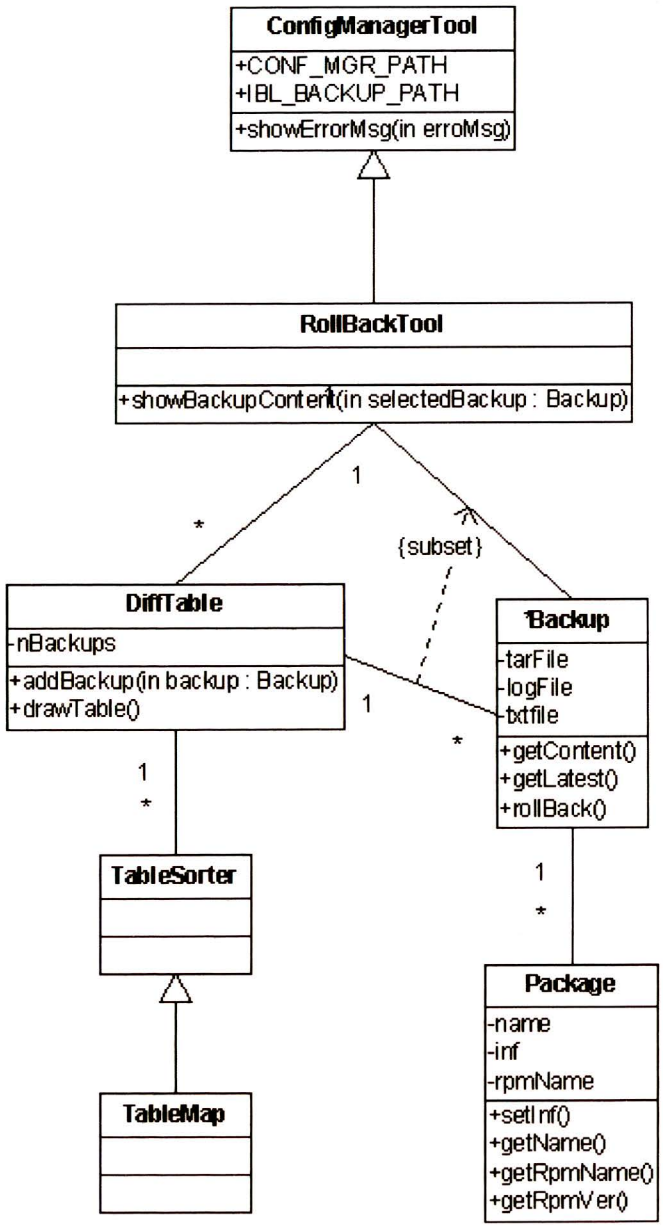
**Check RPM package inside a Backup**

This Use Case extends the previous as it consists not in the list of the modules of a backup but in the listing of the properties of a given RPM package representing a module.

With this list of Use Cases the requirements are clearly defined and the approach to the design and implementation will follow.

### 5.2.3 Object Data Model

The next picture shows the classes diagram that presents the logical model of the Rollback tool. As with the RpmBrowser, this tool consists in just one Logical Package where all the implementation is done.




24. Classes diagram: Rollback Tool

The top class in this diagram is a common class called ConfigManagerTool. It was also used for the RpmBrowser and it implements basic methods for error reporting and constants definition.

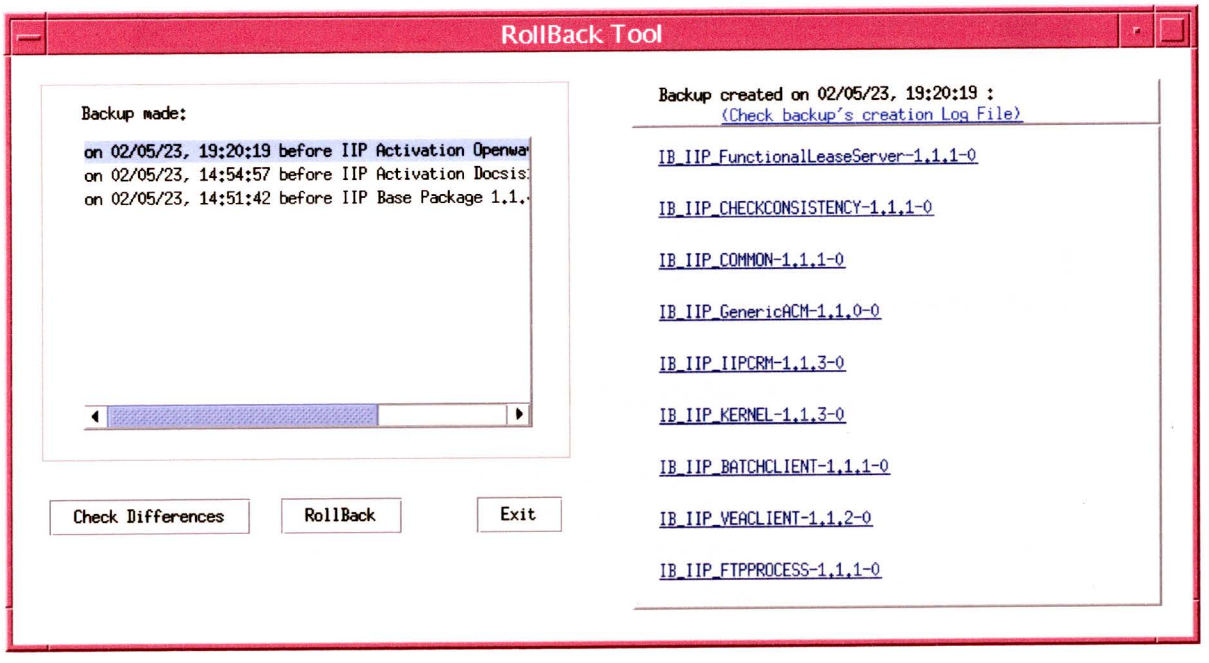
The main class of the tool (called RollbackTool) defines, for itself, all the basic interface components and actions. The class RollbackTool encloses more than one Backup as it lists all available backups. Each Backup contains more than one Package which is used to enclose information about each RPM package.

The class RollbackTool contains a collection of possible DiffTables. This class is responsible for the implementation of the Use Case "Check differences between Backups". This table is sorted and contains a subset of the Backups list (user selection). The classes TableSorter and TableMap are used to control not only the order of the table components but also to control table actions as the move of columns, resize, etc.

The next section will present the most important details related to the implementation of this model.

### 5.2.4 Development Details

The implementation of the Rollback Tool is based in a Graphical User Interface consisting in a single window that centralizes all the functionalities. The next picture shows the main window of the application.



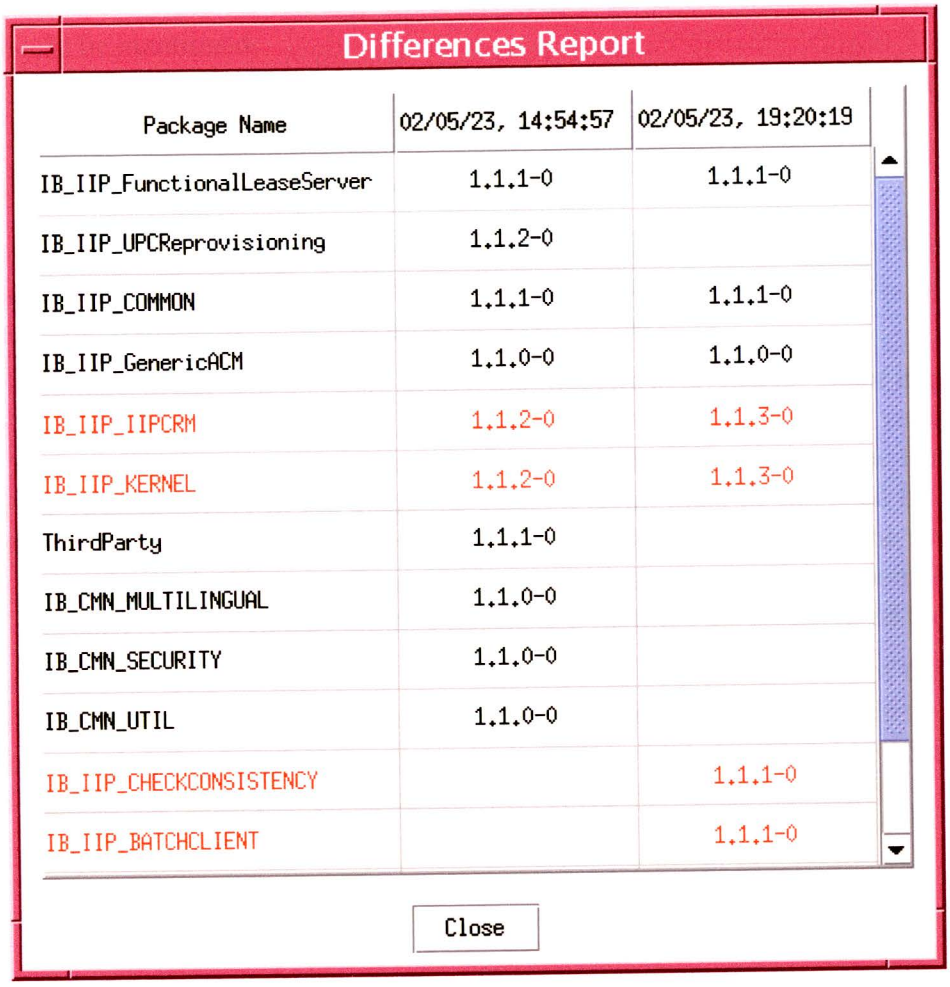

25. Screenshot: Rollback Tool main window

In the left side of the window, the list of all available backups is shown. The list at the right side of the window contains the Packages in the presently selected Backup. The information of each Package can be visualized by clicking in the name of the Package (a dialog box will appear).

This screen already implements three Use Cases defined before. The other two Use Cases can be evoked with the help of the two buttons provided in the interface.

The DiffTable class implements a special dialog box that shows the differences between two (or more) Backups. These differences are expressed in terms of modules contained and

version of those modules. The following picture shows a DiffTable that is presented when two Backups are selected in the list of Backups and the button "Check Differences" is clicked.




26. Screenshot: Rollback Tool DiffTable

The Rollback button makes the Rollback to the selected Backup. This rollback is the major functionality implemented by the tool. It is performed by a Shell script using the data contained in the Backup. This script removes all Imagine Broadband products from the current platform and installs all the modules enclosed in the selected Backup.

To create the Backups, all the installers delivered after the Rollback tool should include a procedure to make the Backups as defined in appendix E: Rollback Tool Specification. This procedure was written as a Shell script that would run as a background process during the installation of any new module.

The most common use of the Rollback Tool starts with the installation of a new module in a given platform with some modules already installed. During the installation the Backup is done. If the installation, for some reason fails, the Rollback tool is used. With its GUI the tool can show the user the content of the backup done during the last installations and the user can rollback the state of the system to that previous state (annealing the failed installation).

The time consumed by the implementation of the Rollback Tool is larger than the time consumed with the implementation of the RpmBrowser. This tool is much more complex and contains much more features. Among these, the scripts to do the rollback and to create the backups should be mentioned.

### 5.2.5 Conclusion and further work prespectives

This tool is not considered a prototype as the RpmBrowser, because it implements a specific required feature for Configuration Management. Despite more complex, this tool may be included in the framework represented by the RpmBrowser.

It is important to say that this tool, unlike the RpmBrowser, has no major improvements to be done as it implements the required functionalities. Nevertheless, it is not yet delivered to the client. In this case, the reason may be considered slightly different from the RpmBrowser. To deliver the Rollback Tool to the client the Backup script must be included in all the new modules delivered by Imagine Broadband. The fact that these tasks should be synchronized makes the delivery process more difficult. This integration is planned for soon. The lack of priority of this delivery is obviously another reason that justifies the undelivered tool.

The development of the Rollback Tool was required in the beginning of the internship and it was successfully developed by the intern as one of his first challenges in the company. It should be said that the fact that the tool is not yet delivered should not be regarded as an evaluation point of the intern performance.

# 6 Conclusion

In this conclusion each of the main chapters conclusion will be summarized, namely, chapters 2, 4 and 5. Each of the three main chapters described one of the main three parts of this internship.

Chapter 2 Imagine Broadband Products and Processes besides presenting some company details also described the brief work done at the Integration/Testing team. In section 2.3 Imagine Instant Provisioning Pre-Integration it was concluded that although this small work was not fully executed, it was a brief but profoundly rich experience as it permitted the acquisition of vast and completely new knowledge about the product, the technologies, and generally, about the company processes and culture.

Chapter 4 Imagine Broadband Packaging described the largest share of the work done during the internship. The themes of this work are Software Packaging and Packaging Process Automation. The result of this work was the developed tool called "Imagine Broadband Packager" that substituted a Former Packaging System. This tool enhanced the automation capabilities of Packaging System of Imagine Broadband and turned it into a framework easy to use and maintain. In this chapter, it is also described that this part of the internship included packaging tasks. These tasks involved the practical application of the available tools to build packages according to the currently settled deployment methodology and architecture.

With this new tool, all the packaging steps were improved in terms of easiness of use as the usability of the new Graphical User Interface enhancement is enormous compared to the difficulty and slowness of use of the former Packaging System. This new Packaging System also enhanced the automation of the process as before there were a lot of actions to be performed manually and now all the process is completely automated. The process control that the new Packaging System offers was also improved with multiple automatic checks, but specially, with the possibility of the visualisation of all the work done by the tool during its execution. Finally, the speed of the process was considered the biggest improvement offered by the new Packaging System mainly because it was the responsible for the high increase of productivity.

The most important point regarding the work done is related to effectiveness. In a company like Imagine Broadband the ultimate goal is effectiveness and productivity enhancement and this internship was rich in increasing them.

Another important point is related to the last month of work that was spent improving the already delivered tool. With this final work, the intern had the opportunity to transform the Packaging Tool in an application that fulfilled all the requirements settled. Moreover, some additional functionalities were implemented. This work, made the list of improvements to do very small, but specially, with very specific items like PVCS adaptation or even the total automation of the tool (GUI removal, see section 4.7 Conclusion and further work perspectives).

Chapter 5 Imagine Broadband Configuration Management Tools described the work done in the beginning of the internship regarding System Configuration on Imagine Broadband products. These tools should perform tasks like package de-installation, installation report, installation backup and rollback functionalities, among others. Not all of these functionalities were considered priorities, and so, only some of these functionalities were implemented. These tools were not yet delivered to the client mainly because it is not a priority to do it. The delivery of these tools to the client or even for internal use is planned for soon.

The RpmBrowser was considered a prototype as it encloses the basis for a complete Configuration Management framework. The Rollback Tool may be seen as a specific part of that framework.

One of the main issues regarding the work done in the internship is related to the overlapping of tasks such as, the packaging duties, development of the new Packaging System, development of Configuration Management tools or even the participation in other teams like the Integration Team. Since the beginning of the internship, packaging duties were frequent. New packages had to be produced in a high rate. With the new packaging system, the time consumed in the execution of these tasks was gradually decreased. Nevertheless, the productivity, related to the execution of tasks as the ones mentioned above, was deeply affected.

**Links and References**

[1] Sommerville, Ian "Software Engineering", Addison-Wesley Pub Co, 2000

[2] Solaris Official Web Site: http://wwws.sun.com/software/solaris/index.html

[3] Compaq Computer Corporation, "Tru64 UNIX, Technical Overview, version 5.1A", 2001 Online: http://www.tru64unix.compaq.com/docs/base_doc/DOCUMENTATION/V51A HTML/ARHXDTE/TITLE.HTM

[4] Petersen, Richard "Linux: The Complete Reference, Fifth Edition", McGraw-Hill Osborne Media, 2002

[5] Linux Documentation Project: http://www.linuxdoc.org

[6] Minasi, Mark "Mastering Windows 2000 Professional", Sybex, 2000

[7] Microsoft Windows Web Site: http://www.microsoft.com/windows2000/default.asp

[8] Veeraraghavan, Sriranga "Teach Yourself Shell Programming in 24 Hours", Sams, 1999

[9] Korn shell Web Site: http://www.kornshell.com/

[10] Winsock RSHD/NT Web Site: http://www.denicomp.com/rshdnt.htm

[11] Murata, M.; S., Laurent; Kohn, D. "Request for Comments 3023: XML Media Types", 2001; Online: http://www.ietf.org/rfc/rfc3023.txt

[12] Deitel, Harvey; M. Deitel, Paul J. "Java How to Program", Prentice Hall, 2001

[13] Java Web Site: http://java.sun.com/

[14] Java Tutorial: http://java.sun.com/docs/books/tutorial/

[15] Java API Specification: http://java.sun.com/j2se/1.4/docs/api/

[16] Java Virtual Machine Specification: http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html

[17] Java Foundation Classes: http://java.sun.com/products/jfc/index.html

[18] Remote Method Invocation: http://java.sun.com/products/jdk/rmi/

[19] Stallman, Richard; McGrath, Roland "GNU Make, a Program for Directing Recompilation", 2000

[20] Ant User Manual: http://jakarta.apache.org/ant/index.html

[21] Java Archives: http://developer.java.sun.com/developer/Books/javaprogramming/JAR/

[22] DOM standard: http://www.w3.org/DOM/

[23] SAX definition: http://www.saxproject.org/

[24] Xerces Web Site: http://xml.apache.org/

[25] VSS User Guide: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/guides/html/vstskUsing_VSS.asp?frame=true

[26] RPM Web Site: http://www.rpm.org/

[27] Bailey, Edward "Maximum RPM", Sams, 1997, Online: http://www.rpm.org/max-rpm/

[28] Install Anywhere Web Site: http://www.zerog.com/

[29] SAMBA Web Site: http://www.samba.org/

[30] Carter, Gerald; Allison, Jeremy "Sams Teach Yourself Samba in 24 Hours", Sams, 2001

[31] Bugzilla Web Site: http://bugzilla.mozilla.org/

[32] Configuration Management Definition: http://www.enteract.com/~bradapp/acme/scm-defs.html

[33] Duchet, Eric "IIP Packaging", Imagine Broadband, 2002

[34] Duchet, Eric "IBL Rollback Tool specification", Imagine Broadband, 2002

[35] Russell, Bertrand Arthur "The Principles of Mathematics", 1903, chapter VI (W.W. Norton & Company, 1996)

## Appendix A: ReleaseNotes.xml structure

This appendix describes the structure of the ReleaseNotes.xml file. As said in section 4.3.3 Source Code Compilation, this file is the most important item in the communication process between the development section and the packaging section.

The following picture shows a typical ReleaseNotes.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReleaseNotes>
  <Name>IIP_DOCSPROVFE</Name>
  <Summary>Imagine Instant Provisioning CableCom Front End</Summary>
  <Label>Version_1.1.20</Label>
  <Description>
        The Goal of this application is to transform files generated by the Docsprov module in RMI calls to the IIP Backend. The application is supposed to be run as a cron job.
  </Description>
  <Dependencies>
        <Package>
              <Name>IIP_COMMONFE</Name>
              <Label>Version_1.1.6</Label>
        </Package>
        <Contrib>
              <Name>crimson.jar</Name>
              <Label></Label>
        </Contrib>
  </Dependencies>
  < Functionalities > Provision ACCOUNT,SUBSCRIBER,MODEM,SERVICE </Functionalities>
  <WhatIsNew>
        <Version_1.1.20>Add top net services.</Version_1.1.20>
        <Version_1.1.18>Update code in order to use last LDManager.</Version_1.1.18>
  </WhatIsNew>
  <KnownBugs></KnownBugs>
  <FixedBugs>
        <Version_1.1.18>Fix BUG 2094</Version_1.1.18>
        <Version_1.1.13>Fix BUG 1653, 1677, 1708 and 1740</Version_1.1.13>
  </FixedBugs>
  <KnownIssues>
        Mac Address format handling to review and still uses some deprecated methods (LDManager)
  </KnownIssues>
</ReleaseNotes>
```

27. File example: ReleaseNotes.xml

In this example all the tags are shown with reduced content for conciseness. The most important tags in this file are <Name>, <Label> and <Dependencies>. The tag <Label> encloses the name of the Label (version) of the development project. It is from the list of dependencies enclosed in the tag <Dependencies> that the compilation environment is setup. The other tags are used to produce the reports to be provided to the Integration/Test Team.

### Appendix B: Ant's build.xml automated generation

This appendix will describe the automated generation of the build.xml file in the new Packaging System (see section 4.5.6 Development Details).

The script that automatically generates the build.xml file in the Former Packaging System was totally transformed as now it uses a XML configuration file to get all the details required for a given package to be built. This XML configuration file is named pkgNames.xml (see section 4.5.5 Physical Architecture, package Config). It encloses a list of modules with the required information to build a project like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<pkgNames>
  <!-- Backend -->
    <Module>
      <Vss prod="IIP" jarFile="IIP_KERNEL">IIP_KERNEL</Vss>
      <Rpm prod="IIP" platform="Sun"
            relocPath = "/opt/Imaginebroadband/ImagineIP"
            staticPath = "/IIP_KERNEL">IB_IIP_KERNEL</Rpm>
      <Iaw rpmSource = "/opt/InstallAnyWhere/IIP_Packages/IIP_Base/RpmSource" />
  </Module>
</pkgNames>
```

28. File example: pkgNames.xml

In this example, there is information regarding only one module. In a typical file a few tens of modules are detailed.

Inside the tag <Module> there are three sub tags called <Vss>, <Rpm> and <Iaw>. The last is used to update the RPM package in the Install Anywhere project directory (see section 4.3.5 Install Anywhere project). The other two are used to generate the build.xml file.

The application, when building a project, parses this pkgNames.xml file and gets all the information about the project to build (this XML file is parsed using some specific Java libraries like SAX; see chapter 3 Used Technologies for further details). After retrieving the information from the configuration file the application runs a Shell script that generates the build.xml file.

The information contained in the tag <Vss> is related to VSS and the attributes are the JAR file name and the VSS database. The main value in <Vss> is the name of the project used to seek in the configuration file for the correct project to build.

The <Rpm> tag provides information related to the RPM package to build. This information is needed as the build of a project consists not only in the compilation of the Source Code, but also in the organization of the directories structure to build the RPM package. As such, the name of the RPM package, the path where it will be installed and the platform need to be defined at the time of the build.

The Shell script that generates the build.xml uses a rather simple mechanism as it just constructs a default build.xml with the values provided by the application (taken from the configuration file) substituted in specific places.

## Appendix C: RPM Specification File

A RPM Specification File contains all the information required to build a RPM package: package information, package upgrade type and files list/types.

The following picture shows a reduced example of a RPM Specification file.

```
# Spec file IB_IIP_CMN_IIPCRM-1.1.21-0.spec 091202_151625
Name: IB_IIP_CMN_IIPCRM
Version: 1.1.21
Release: 0

# Descriptive tags
Summary: Imagine Instant Provisioning functional layer with APIs for general use
Distribution: IIP-1.1.0-0 Sun FE
Group: IBL/IIP
Copyright: Imagine broadband -2002-

# Directory tags
Prefix: /opt/Imaginebroadband/ImagineIP

%description
The IIPCRM package includes APIs performing searches and updates in the CRM database relative to generic entities, not specific to a service or technology. Packaged for host FE on platform Sun for product IIP-1.1.0-0.
VSS Project Reference: IIP_IIPCRM Version_1.2.17
VSS Build Dependencies: CMN_MULTILINGUAL-Version_1.1.0 CMN_UTIL-Version_1.1.15 IIP_COMMON-Version_1.1.11 IIP_KERNEL-Version_1.1.29

%files
%defattr(-,root,other)
%config /opt/Imaginebroadband/ImagineIP/IIP_IIPCRM/etc/ConfigIIP_CRM.partial
/opt/Imaginebroadband/ImagineIP/IIP_IIPCRM/lib/IIP_IIPCRM.jar
%dir /opt/Imaginebroadband/ImagineIP/IIP_IIPCRM/resources/scripts
/opt/Imaginebroadband/ImagineIP/CRM/resources/stubs/CRMstub.jar
```

29. File example: RPM Specification File

With this file, the rpm command can be evoked and the RPM package is created with the characteristics described in the file, i.e. information tags like Name or Distribution and listed files and their attributes (see 3 Used Technologies for more information regarding RPM packages).

## Appendix D: IIP Packaging

This appendix contains selected parts of the document IIP Packaging written by Eric Duchet in 2002-03-22. [33] The selected parts are the ones that are considered significant to the description of the Former Packaging System. This document was written in the beginning of the internship to make the transmission between resources doing Packaging, i.e. to introduce the intern in the Packaging section.

# Introduction

### Purpose

This document intends to explain the building process of Imagine Broadband products.

### Targeted Audience

The target audience of this document is any person who intents to make packaging.

# The building process

### Introduction

The different steps to package a module are:

1. Extract the Source Code from VSS
2. Compile the package with Ant
3. Build the RPM package
4. Integrate the RPM packages in a "commercial package" generated with Install Anywhere
5. Generate the installer
6. Deliver the installer to the test team

### VSS Extraction

This operation is done on a UNIX machine by using the script extract.ksh. In order to work properly this script needs:

1. An extraction directory,
2. A connection to a PC running a rshd service (This daemon is the interface with the VSS repository)

The distinct steps are:

1. Select the name of the User for extraction,

```
++++ VSS Extract ++++


++ User Login ++
1) Eric_Duchet
2) Olivier_Depaye
3) Pierre_Walter
Select user name: 1
```

1. If you want to clean the extraction directory, answer y, otherwise answer n.

```
User name      : Eric_Duchet
Host name      : Ws-educhet.ib.sophia.com
Working folder: /export/home/hbo/edu

Should I clean extraction directory /export/home/hbo/edu/vss/ (y/n): n
```

1. Select the product to extract. Each product corresponds to a VSS repository.

```
++ Product Selection  ++

1) CMN
2) ISA
3) IIP
4) NEXT
5) EXIT
Select a product for extraction: 3
```

1. Select the project you want to work with. The 'All Latest_Label' menu extracts all projects with the last version of each file. You can select multiple projects. When you are done, select the 'done' menu.

```
++ Projects Selection for IIP ++
 1) IIP_ADMIN/IIP_ADMINFE                                    13) IIP_COMMON
 2) IIP_ADMIN/IIP_LOGMGR                                     14) IIP_CSR
 3) IIP_ADMINTOOL                                            15) IIP_SELFCARE/demo/DSL
 4) IIP_BACKEND/IIP_KERNEL
 5) IIP_BACKEND/IIP_TOOLKIT
 6) IIP_BACKEND/MODULES/Activation/HSD/IIP_ActivationHSDCommon
 7) IIP_BACKEND/MODULES/Activation/HSD/IIP_ActivationHSDDSL
 8) IIP_BACKEND/MODULES/Activation/HSD/IIP_ActivationHSDHFC
 9) IIP_BACKEND/MODULES/Functional                           16) ALL_Latest_Label
10) IIP_BACKEND/MODULES/IIP_GenericACM                       17) RESET
11) IIP_BACKEND/MODULES/IIP_IIPCRM                           18) DONE
12) IIP_BATCHCLIENT
Select projects of IIP: 20
```

1. Select the label you want to work with. The 'Latest' option extracts all files for the project in their latest version.

```
Getting label(s) of IIP_SERVICEMGR
1) 1.1
2) Package v1.0
3) Version_1.0.5
4) Version_1.1.0
5) LATEST
6) BACK
Select Label for IIP_SERVICEMGR product IIP:4
```

After selecting the label, the script extracts the project in the destination directory.

1. Select done when you have finished the project extraction.

```
++ Projects Selection for IIP ++
Extracted project: IIP_SERVICEMGR=>Version_1.1.0


 1) IIP_ADMIN/IIP_ADMINFE                                    12) IIP_BATCHCLIENT
```

```
 2) IIP_ADMIN/IIP_LOGMGR                                    13) IIP_COMMON
 3) IIP_ADMINTOOL                                           14) IIP_CSR
 4) IIP_BACKEND/IIP_KERNEL
 5) IIP_BACKEND/IIP_TOOLKIT
 6) IIP_BACKEND/MODULES/Activation/HSD/IIP_ActivationHSDCommon
 7) IIP_BACKEND/MODULES/Activation/HSD/IIP_ActivationHSDDSL
 8) IIP_BACKEND/MODULES/Activation/HSD/IIP_ActivationHSDHFC
 9) IIP_BACKEND/MODULES/Functional                          15) RESET
10) IIP_BACKEND/MODULES/IIP_GenericACM                      16) DONE
11) IIP_BACKEND/MODULES/IIP_IIPCRM
Select projects of IIP: 21
```

1. Select the 'next' menu when you have finished the production extract.

```
++ Product Selection  ++


1) CMN
2) ISA
3) IIP
4) NEXT
5) EXIT
Select a product for extraction: 4
```

After the interactive steps, the script regenerates every pinfo.cfg file for the projects located under the VSS root directory. The generation is based on the Release Note file. If the Release Notes have syntax errors (XML parsing failure) the pinfo.cfg file can not be generated. You have to report a bug to the development team.

**Compiling the project**

*Packaging directory structure*

On the root directory of the packaging directory, you have the following structure:

1. Build: Directory for temporary java compilation
2. contribs: package export directory.
3. lib: java libraries used for compilation. It contains third party jar libraries and package dependant jar files.
4. rpmdata: Destination directory for JAR generation. This directory is the base of the RPM package structure and the source directory for making RPM packages.
5. tmp: contains log files generated by the extract and buildrpm script.
6. VSS: contains the product, projects and files extracted from VSS.

*Java compilation*

To compile each project, go into the directory vss/'Product Name'/'Project Name' and run Ant.

Ant is a java based build tool. It uses the Build.xml file located in the Project root directory.

**Structure of the build.xml file**

The build.xml file has five sections: environment variables, init, compile, build and clean sections.

*Definition of environment variables*

This section is used to define the environment variable (xml property field) used in the script. A typical section looks like:

```
<project name="IB_IIP_VEACLIENT" default="clean" basedir=".">
<property name="HBOROOT" value="/export/home/hbo/edu"/>
<property name="HBODATA" value="${HBOROOT}/rpmdata"/>
<property name="PROJ" value="IIP"/>
<property name="VSSROOT" value="${HBOROOT}/vss"/>
<property name="PKG_NAME" VALUE="IIP_VEACLIENT"/>
<property name="CONTRIBS" value="${HBOROOT}/contribs"/>
<property name="PKG_SRC" value="${VSSROOT}/${PROJ}/${PKG_NAME}"/>
```

*Init section*

The map of the initial section is:

1. Delete of destination directories in order to be sure not to have bad files from previous packaging work.
2. Creation of destination directories
3. Copy of required files from VSS extraction directory to the contrib./<pkgname> destination directory

This is a sample of Initial section

```
<!-- INIT -->
   <target name="init">
      <delete dir="${PKG_ROOT}"/>
      <delete dir="${PKG_CONTRIBS}"/>
      <mkdir dir="${PKG_CONTRIBS}"/>
      <copy file="${PKG_SRC}/pinfo.cfg" todir="${PKG_ROOT}"/>
      <copy todir="${HBOROOT}/lib/">
         <fileset dir="${PKG_SRC}/lib"/>
      </copy>
      <copy todir="${PKG_CONTRIBS}/bin">
         <fileset dir="${PKG_SRC}/bin"/>
      </copy>
```

*Compile section*

The map of the compile section is:

- Compile all java files from the src directory
- If an interfaces directory is present, launch the rmic compile for classes ending with Impl.java residing in that directory.

This is a sample of the compile section

```
<!-- COMPILE -->
   <target name="compile" depends="init">
      <echo message="${PKG_NAME} : compile"/>
      <javac>
         <classpath>
            <fileset dir="${HBOROOT}/lib/">
               <include name="**/*.jar"/>
            </fileset>
         </classpath>
         <src path="${PKG_SRC}/src/"/>
      </javac>
   </target>
```

*Build section*

The goal of the build section is to build the package JAR file. The map of the build section is:

- Move all class files into the build directory
- Constitute a jar file of the package.

This is a sample of the build section:

```
<!-- BUILD -->
   <target name="build" depends="compile">
      <echo message="${PKG_NAME} : build"/>
      <move todir="${HBOROOT}/Build/">
         <fileset dir="${PKG_SRC}/src/" excludes="**/*.java"/>
      </move>
      <jar jarfile="${HBOROOT}/lib/${PKG_JAR}" basedir="${HBOROOT}/Build/" />
      <copy file="${HBOROOT}/lib/${PKG_JAR}" todir="${PKG_CONTRIBS}/lib"/>
      <chmod dir="${PKG_DEST}/bin" perm="ugo+rx"/>
   </target>
```

*Clean section*

The clean section deletes the temporary directory used to make the jar file.

A typical clean section looks like:

```
<!-- CLEAN -->
   <target name="clean" depends="build">
      <echo message=" : clean"/>
      <delete dir="${HBOROOT}/Build/"/>
   </target>
```

**Automated xml generation**

To avoid constructing every time a new or updated build.xml file, the command GenXml.ksh helps you to generate automatically the xml build file.

The script looks the directory structure of the package and determines what to do.

Actions where a directory is present:

| Directory | Action |
|---|---|
| Bin | Change the access right to execution in Building Section |
| Etc | Nothing |
| Lib | Put the jar file from the src lib directory to the third-party directory |
| Resources | Take every files except the 'content' subdirectory (destined for the population package) |

Actions taken for a specific package

| Package | Action |
|---|---|
| KERNEL | Generates two jar files IIP_KERNEL and IIP_ACTIVATIONMODULE |
| Activation modules | Creates a stub directory to create interfaces<br>Launch the RMI command to compile the interfaces |

To construct a new build xml file, just launch the GenXml.ksh command under the project you want.

**Directory used for compilation**

The directories used during the compilation are:

- Build: Temporary directory for java building.
- Contribs: Put the directory structure to be included into the rpm file (in fact, it contains all files included in the rpm file except the jar files generated)
- Lib: The lib java files used for compilation of the module (class path)
- Rpmdata: Directory structure for rpm generation

**Building rpm**

When the compilation is successful, you can build a new RPM package for that package.

The building repository for the rpm command is located in rpmdata from the user root directory. The script used for building RPM packages is buildrpm.ksh.

The command line option is:

1. –b: Internal build
2. –u: Software upgrade (increment by 1 the second digit of the software version)

Depending of what you want to do, you must choose one of the command line options to build correctly the package.

After the rpm package generation, it is recommended to check it with the command:

- rpm –i –vv –test <pkg name>

This command makes the test of the installation, but does not install the package.

**Commercial package**

The commercial packages are made with Install Anywhere. The goal is to have a user friendly graphical installation.

The steps of installation can be resumed as follow:

- Welcome screen of installation,
- Licence agreement,
- Select the components to be installed on the system (if any)

- Choose the destination installation path,
- Extract the rpm files and the installation scripts into the RpmSource directory of base installation
- Run the pre installation script called Install.sh, which tests the installation of the rpm files,
- Installs the packages,
- Run the post installation script.

In order to build a new version of commercial package, you have to:

- Put the new RPM files generated with the buildrpm.ksh script under the right RpmSource directory (it depends on the project),
- Remove the previous version of the RPM file from the RpmSource directory,
- Open the Install Anywhere project,
- Change the version number of the installation,
- Generate the new Installer,
- Try the installer to look if it works properly.

The directory IIP_Commands contains IIP specific commands as the post installation scripts.

The directory structure for an Install Anywhere project is:

- <Projectname>.iap: Main file of the project
- <Projectname>_Web_Installer: Location of Installation software (generated by Install Anywhere after any build)
- RpmSource: RPM location directory where RPM packages are placed after their generation.

**Delivery**

Once the generation of the installation is done, it's time to deliver the product the test team.

To do that, you have to:

- Put the new commercial on the delivery web site (use the command /opt/packaging/bin/updateDelivery.sh).
- Put the delivery on the delivery server (\\Srvdev01\Integration\Testing\Delivery\IIP)
- Write a mail to test team with the new versions available,
- Update the excel sheet for changes.

## Appendix E: IBL Rollback Tool Specification

This appendix contains selected parts of the document IBL Rollback Tool Specifications elaborated by Eric Duchet in 2002-04-03. [34] The selected parts are the ones that are considered significant to the understanding of the tool. The aim of this document was to define the major requirements for the tool and also the most important implementation details.

# Introduction

**Purpose**

This document intends to define a way to rollback the installation of an IBL product and go to a previous state. The scope of the rollback is limited to the IBL software installed by an Install Anywhere package.

**Limitation of the RPM command**

The RPM is great for upgrading a system. Unfortunately, this utility does not offer any rollback possibilities and does not track the historic on an installation.

**Solution**

Due to the limitation seen above, we have to find a simple and secure manual solution to do the rollback operation.

The constraints after a rollback are:

- Have the consistency of the rpm database,
- Have the exact image of the IBL product, including the configuration files.

So, before any new installation, a backup of some set of files has to be done.

The rollback operation will consist in removing the new installation and use the backup.

We need two procedures to implement that solution:

- One for the backup of the system before any new installation,
- One for the recovery of a previous version after the new installation

# Backup of IBL software

**General map of the backup**

The backup procedure is made during the installation of a new component, before the Rpm package installation itself.

A script called ibl_backup.sh has to be developed.

In our Install Anywhere structure, the backup script has to be integrated into the script called "Install.sh" before the Rpm installation section.

The backup is made with the tar command. Tar files are located in a backup directory.

An additional file also has to be generated to resume the content of the tar file.

The files generated are indexed with the date and time

- iblsave-YYMMDDHH24MISS.tar for the tar file
- iblsave--YYMMDDHH24MISS.txt for the information file.

With that system, the complete history of the system is made, and it is possible to come back to any version already backuped.

The roadmap for the backup looks like:

- Save the IBL product in a tar file.

- Generates the information files
- Moves the both files into the backup directory.
- Install the new packages.

**Files to save**

*Introduction*

The files to save are in fact:

- The Rpm packages,
- The RPM database.
- Some frontend configuration files

*Rpm package*

The backup procedure has to save all files from Imagine Broadband products. Almost all the files of Imagine Broadband products are installed with an rpm package. A couple of files, such as some configuration files or symbolic links, are installed during the post installation procedure.

In order to correctly make the backup, we have to make the backup of each IBL product in its base installation.

The steps for the backup of the IBL product will be:

- Browse the RPM database to obtain the list of all IBL packages.
- Determine the base directory for each package.
- Tar each base directory and its sub directories.

*Rpm database*

The rpm database contains all information related to rpm packages installed on the machine.

It is located in <rpminstallationpath>/db

The rpm installation path is not easy to guess. The best way is to look in the /usr/bin directory and looks for the symbolic link with the rpm command.

*Frontend configuration*

For the frontend packages, some links are made with tomcat and apache. We need to backup those links.

**Information file**

The information file contains all information of the RPM Database. This file will be only used for information purpose to list in term of IBL product the content of a backup file.

The option '–qai' of the rpm command will be used to generate this file.

**Backup repository**

The backup repository will be located into the rpm installation directory in a directory called 'iblbackup'

# Rollback of IBL software

### General map of a rollback

The rollback procedure consists to go back to a previous version of IBL software. In order to do that, the prerequisite is that a backup exists on the machine.

This operation is done by a script called ibl_rollback.sh.

The steps for the rollback operations are:

- Remove all IBL rpm installed on the machine and its base directories
- Restore the backup tar file which contains the old IBL software and the rpm database

### Remove of the current IBL product

The remove of IBL product consists in removing all IBL products from the machine. In order to do that, the following operations have to be made:

- Browse the rpm database to identify the Imagine Broadband rpm product.
- Delete the product for the rpm database
- Delete the packages base directories

### Restore the backup

The backup made with the ibl_backup.sh script contains a complete older version of the IBL product and its RPM database. The restore of the file is simply made with the tar command.

### Parameters

The script has got one parameter: the name of the backup file to restore.

### Improvement of the rollback

A graphical interface can be made to select the tar file to restore and automatically print on the screen the content of the tar file (information extracted from information backup file)

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000065328