UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Desenvolvimento de software para gestão e controlo de elemetos VoIP (*SoftSwitch hiQ8000*)

L3 Cluster Interconnect & Remote Patching Capability: RTP Rolling Upgrade Daemon

Francisco Magalhães Olavo Corrêa de Azevedo

Mestrado em Engenharia Informática

2007

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Desenvolvimento de software para gestão e controlo de elemetos VoIP (*SoftSwitch hiQ8000*)

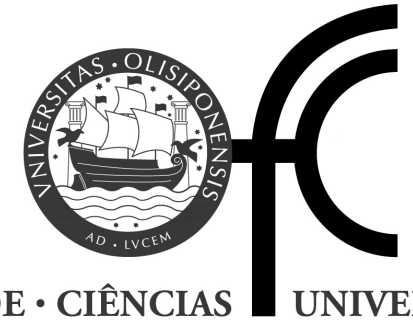
L3 Cluster Interconnect & Remote Patching Capability: RTP Rolling Upgrade Daemon

Francisco Magalhães Olavo Corrêa de Azevedo

Projecto orientado pelo Prf. Dr João Balsa
e co-orientado por Sr. Eng. Pedro Miguel de Carvalho Barroso Capela

Mestrado em Engenharia Informática

2007



FACULDADE · DE · CIÊNCIAS UNIVERSIDADE · DE · LISBOA

Declaração

Francisco Magalhães Olavo Corrêa de Azevedo, aluno nº28714 da Faculdade de Ciências da Universidade de Lisboa, declara ceder os seus direitos de cópia sobre o seu Relatório de Projecto em Engenharia Informática, intitulado ”Desenvolvimeno de software para gestão de controlo de elementos VoIP (softswitch hiQ8000): L3 Cluster Interconnect & Remote Patching Capability: RTP Rolling Upgrade Daemon”, realizado no ano lectivo de 2006/2007 à Faculdade de Ciências da Universidade de Lisboa para o efeito de arquivo e consulta nas suas bibliotecas e publicação do mesmo em formato electrónico na Internet.
FCUL, 17 de Julho de 2007

Pedro Miguel de Carvalho Barroso Capela, supervisor do projecto de Francisco Magalhães Olavo Corrêa de Azevedo, aluno da Faculdade de Ciências da Universidade de Lisboa, declara concordar com a divulgação do Relatório do Projecto em Engenharia Informática, intitulado ” Desenvolvimeno de software para gestão de controlo de elementos VoIP (softswitch hiQ8000): L3 Cluster Interconnect & Remote Patching Capability: RTP Rolling Upgrade Daemon”.
Lisboa, 17 de Julho de 2007

Resumo

Este documento descreve o meu trabalho realizado, num projecto de engenharia de software, que consiste na implementação de um novo processo que permite a execução de actualizações de *software* remotamente e na configuração de um *cluster interconnect* em *Layer* 3 de um VoIP *softswitch* concretizado num conjunto de aplicações *software,* utilizando um *cluster* de sistemas Linux/Solaris.

PALAVRAS CHAVE: *softswitch,* VoIP, *remote upgrade, Layer 3*

Abstract

This document describes my work in a software engineering project, developing a new process which allows the execution of remote upgrades and configuring a layer 3 cluster interconnect of a VoIP softswitch, made of a set of software applications that use Linux/Solaris systems.

KEYWORDS: *softswitch,* VoIP, *remote upgrade, Layer 3*

Índice

Índice de Figuras

Índice de Tabelas

Glossário (Acrónimos)

API – Application Programmable Interface

DQoS – Dynamic Quality of Service

CIP – CF over IP

CF – Cluster Foundation (*software* de *clustering* usado no hiQ)

CLI – Command Line Interface

CMTS – Cable Modem Termination System

GUI – Graphical User Interface

HTTP – HyperText Transfer Protocol

IP – Internet Protocol

IPC – Inter-Process Communication

IPMP – IP Multi Path

ISDN – Integrated Services Digital Network

ISP – Internet Service Provider

ISUP – Integrated Services User Part

JNI – Java Native Interface

MGCP – Media Gateway Controller Protocol

MTA – Media Terminal Adapter

OS – Operating System

PBX – Private Branch eXchange

POTS – Post Office Telephone System

PRI – Primary Rate Interface (ISDN)

PSTN – Public Switched Telephony Network

QoS – Quality of Service

RTP – Real Time Protocol

RTP middleware – Resilient Telco Platform

RSU – Rolling Software Upgrade

SGBD – Sistema de Gestão de Bases de Dados

SIGTRAN – Conjunto de protocolos que providenciam adaptações de protocolos de comunicações SS7 e ISDN

SIP – Session Initiation Protocol

SOAP – Simple Object Access Protocol

SS7 – System Signalling #7

TDM – Time Division Multiplexing

ToS – Type of Service.

UDP – User Datagram Protocol

VoIP – Voice over IP

XML – Extensible Markup Language

Capítulo 1
Introdução

Este documento descreve o trabalho realizado no âmbito na cadeira de Projecto de Engenharia Informática para a finalização do Mestrado em Engenharia Informática (2º ciclo de Bolonha).

O trabalho realizado para a cadeira de Projecto de Engenharia Informática insere-se no contexto de estágio profissional numa instituição externa ou na própria Faculdade de Ciências.

Optei por realizar o estágio numa empresa externa, denominada Siemens Networks S.A., de modo a poder proseguir mais facilmente carreira no meio empresarial. A Siemens providencia tecnologias inovadoras e *know-how* que beneficiam clientes em 190 países. Fundada a mais de 150 anos atrás, a empresa tem áreas activas em Information and Communications, Automation and Control, Power, Tranportation, Medical e Lighting. A Siemens Networks provicencia soluções integradas para operadores de redes e fornecedores de serviços em cerca de 100 países. Durante o meu estágio a Siemens Networks fundiu-se com a Nokia Networks criando a Nokia Siemens Networks. Esta nova empresa tem o objectivo de criar uma empresa inovadora na área das telecomunicações unindo a forte experiência do mundo fixo da Siemens com o mundo móvel da Nokia Networks.

A base do projecto é um produto denominado hiQ8000 da Siemens Networks. O hiQ8000 é um comutador de chamadas, que interliga a rede tradicional telefónica com redes telefónicas sobre IP, concretizado num conjunto de aplicações *software*, utilizando um *cluster* de sistemas Linux/Solaris.

O projecto envolve uma série de tecnologias do mundo das telecomunicações. Um aprofundamento do conhecimento dessas tecnologias vêm complementar e aumentar a minha base científica obtida na licenciatura de engenharia informática numa área que cada vez mais se encontra interligada às tecnologias IP. Dentro

das telecomunicações, o VoIP é uma tecnologia que tem vindo a crescer muito nos últimos anos em consequência do crescimento das redes IP.

Este documento descreve o meu trabalho realizado, que consistiu no desenvolvimento de duas novas funcionalidades. Cada uma dessas funcionalidades teve o seu ciclo de desenvolvimento próprio.

A primeira funcionalidade consiste na concretização de um *link layer 3* para o *cluster interconnect* do hiQ. O hiQ utilizava sómente um *link layer 2* para a comunicação entre os nós do *cluster*, mas tendo em conta o seu elevado custo para longas distâncias foi proposto o estudo e implementação da utilização de um protocolo roteável, neste caso o protocolo IP. A segunda funcionalidade foi a implementação dum método remoto de actualização de *software* do produto que implica uma menor intervenção do operador, para prevenir erros humanos neste processo, a não utilização de um acesso a uma *shell*, pelos impactos que tem na segurança do sistema, e feito de forma a que o sistema nunca seja sujeito a faltas nos serviços que disponibiliza. As funcionalidades que são incorporadas em cada versão são denominadas internamente de “features”. Essas funcionalidades funcionam como sub-projectos, cada uma com o seu ciclo de desenvolvimento próprio.

No âmbito deste relatório irei focar-me mais na segunda funcionalidade tendo em conta o seu maior nível de complexidade e pelo comsumo da maior parte do meu tempo na Siemens Networks. A primeira funcionalidade funcionará mais como um método de me ambientar ao hiQ, ao processo de desenvolvimento e a todos os mecanismos de colaboração cooperativa necessários a um projecto desta envergadura.

Este projecto seguiu o processo de desenvolvimento de *software* da Siemens Networks para o desenvolvimento destas funcionalidades. Esse processo tem uma particularidade interessante pois a equipa envolvida nestas *features* está distribuída pelo mundo. Os *System Engineers*, que estão envolvidos na fase de análise, estão localizados nos Estados Unidos. Os *Developers*, que estão envolvidos da fase de desenho até aos testes de integração, estão em Portugal. Os

System Testers, que estão envolvidos na fase de testes de sistema, estão na Grécia e na China. Para uma descrição mais detalhada do processo de desenvolvimento pode-se consultar o Anexo A, secção "Processo de Desenvolvimento".

1.1 Motivação

O hiQ8000 é uma plataforma líder mundial que conta com um desenvolvimento cooperativo de várias centenas de engenheiros pelo mundo inteiro. De forma a manter o produto competitivo é preciso adicionar funcionalidades novas que aumentem o seu valor no mercado, seja por tornar de alguma forma o preço mais atractivo ou pela criação de funcionalidades que não são existentes nos produtos rivais.

É neste contexto que aparecem as duas novas funcionalidades a serem desenvolvidas: A primeira, é motivada por uma vontade de diminuir o preço final aos clientes que têm *clusters* geográficamente separados. Neste caso a diferença de preço duma implementação sobre *Layer* 2 e *Layer* 3 é bastante grande e pode significar a predominância do nosso produto face aos outros que não conseguem uma *performance* aceitável sobre *Layer 3*. A segunda é motivada por decrescer os gastos nos contractos de manutenção que se tem com os clientes. Ao tornar o processo de *software upgrade* mais automatizado e com menos hipóteses de falha poupam-se muitas horas de operadores Siemens nos clientes.

1.2 Objectivos

No príncipio deste projecto foram definidos os seguintes objectivos:

- Implementação da funcionalidade "Layer 3 Cluster Interconnect" no hiQ8000.

- Implementação da funcionadade "Remote Patching Capability" no hiQ8000.

1.3 Organização do documento

O documento está organizado da seguinte forma:

- Capítulo 2: Contexto do projecto.

- Capítulo 3: Descrição da implementação de “ Layer 3 Cluster Interconnect”

- Capítulo 4: Descrição da implementação de “Remote Patching Capability”

Capítulo 2
Contexto

Este capítulo descreve todo o contexto básico necessário à compreensão do projecto desenvolvido. Para melhor podermos compreender o projecto. No Anexo A podemos encontrar a uma descrição mais detalhada da tecnologia em questão, da plataforma utilizada (hiQ8000), do ciclo de desenvolvimento da Siemens Networks, das ferramentas utilizadas e uma breve descrição da formação dada para a concretização deste projecto.

2.1 Plataforma hiQ8000

O hiQ8000 é um *softswitch.* Um *softswitch* é um comutador que concentra as funções dum *Media Gateway Controller* e dum *Signalling* Gateway num só aparelho. Esta configuração elimina a necessidade dum protocolo de comunicação, como o SIGTRAN, entre a componente *Media Gateway Controller* e *Signalling* Gateway. Este modelo encontra-se descrito na figura 1 (hiQ, 2005).

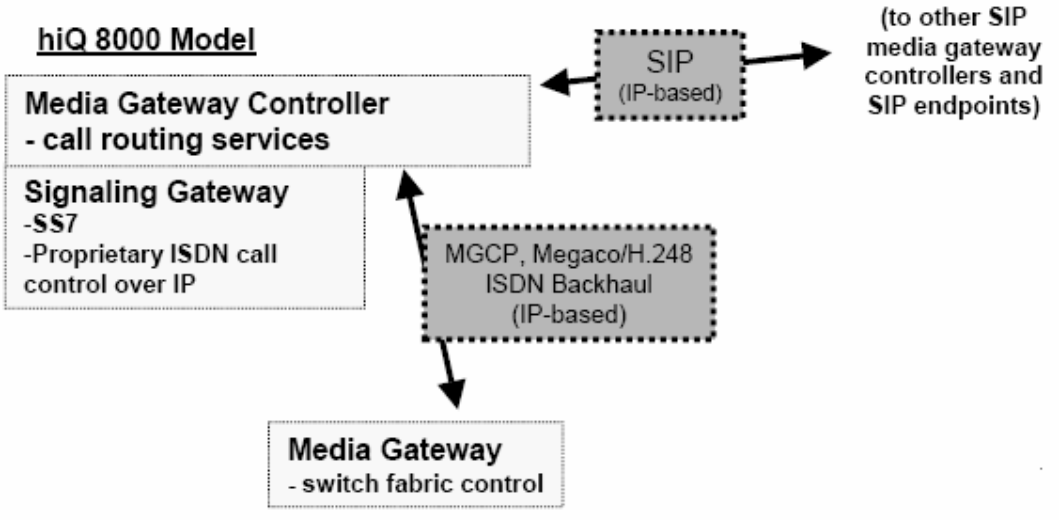


Figura 1 – Modelo Softswitch/hiQ8000

O hardware do hiQ8000 contém elementos que suportam funções de sinalização SS7 de baixo nível, encaminhamento de chamadas e variados tipos de interligações entre protocolos diferentes: ISDN-SS7, SIP-SS7, Voz sobre TDM-Voz sobre RTP/IP.

A função *Media Gateway* é executada por equipamento fisicamente separado que é controlado pelo hiQ8000, utilizando como protocolo primário de sinalização o MGCP. O hiQ8000, pela utilização de protocolos abertos como o MGCP, consegue controlar uma grande variadade de *media gateways* diferentes apesar de em laboratório ser testado com uma solução também da Siemens Networks, denominado hiG1200.

O hiQ8000 é uma solução para operadoras mas também existe uma variante empresarial com menos funcionalidades, como por exemplo inexistência de tratamento de sinais SS7 e um hardware menos potente, denominado HiPath. Tendo em conta a falta de interligação com a rede SS7, o HiPath envia sinalização ISDN (uma linha ISDN tem um canal para voz e um canal para sinalização) num acesso PRI (*primary rate interface*) ou directamente com SIP se a operadora o suportar (hiQ, 2005). Tendo em conta o número limitado de canais de sinalização e voz num acesso PRI ou num acesso IP a uma operadora, é natural que seja uma solução que só seja própria para empresas e não para operadoras, que necessitam duma capacidade muito maior para recepção/envio e tratamento de sinais.

2.2 Arquitectura de Software hiQ8000

Esta secção descreve a arquitectura de *software* do hiQ8000. O HiPath tem uma arquitectura idêntica, tirando os *Signalling Managers* de ISUP e TCAP que são protocolos próprios da rede SS7. Esses protocolos não são incluidos nesta solução pois quem controla a rede SS7 são as operadoras, logo as empresas não têm qualquer interesse em ter essas funcionalidades. Temos também que ter em consideração que o *hardwar,e* que permite a ligação à rede SS7, é extramemente caro.

O hiQ é um sistema que consiste em vários subsistemas (figura 3) (Wood, 2005), que contêm muitos componentes interligados. A plataforma envolvida é constituida por Solaris ou Linux, consoante for hiQ8000 ou HiPath8000, com o RTP *middleware* como fundação operacional, utilizando serviços de Cluster de Solaris/Linux para concretizar distribuição por multiplos nós:

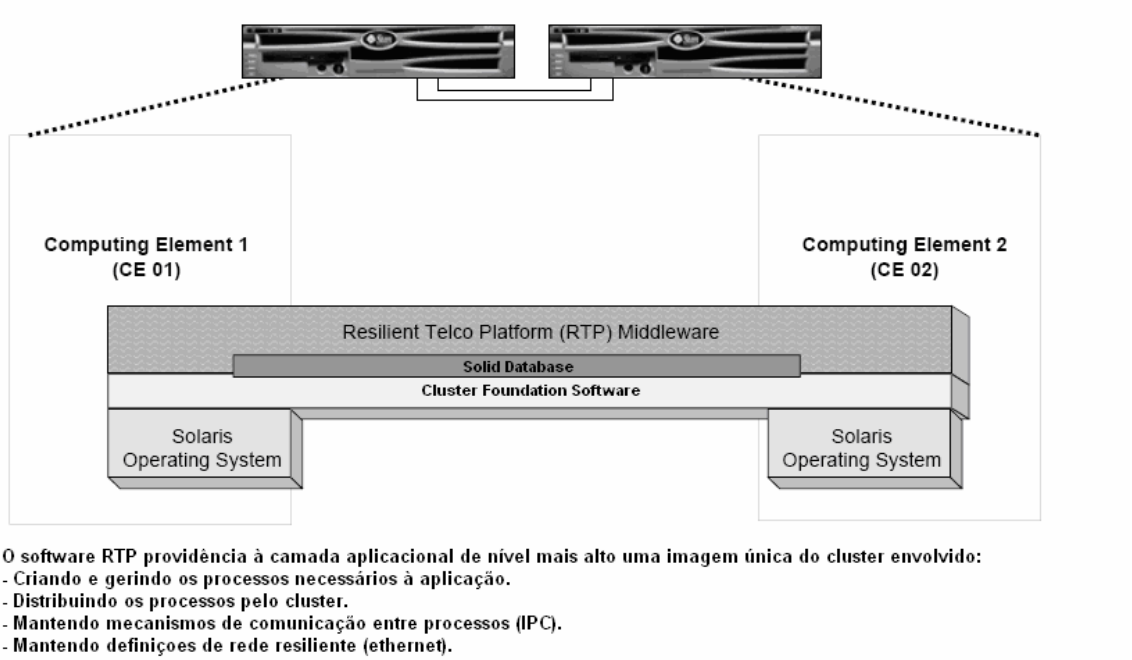


Figura 2 – Cluster, Solid & RTP software

Sistema Operativo: Componente básica de qualquer sistema que permite o acesso ao *hardware* às aplicações. No nosso sistema utilizamos o Solaris ou Linux devido ao seu elevado grau de desempenho e nível de costumização

Cluster Software: Componente que concretiza o *cluster*. O *clustering* do hiQ8000 é paralelo e é feito para aumentar a escalabilidade e disponibilidade. O hiQ consiste num *cluster* de dois nós e é concretizado utilizando o Cluster Foundation Software da Fujitsu Siemens.

Sistema de Gestão de Bases de Dados*:* O SGBD utilizado é o Solid, da Solid Information Technology, que utiliza um modelo não partilhado; ou seja,

cada nó tem uma cópia da base dados inteira e depois vão replicando as mudanças de um nó para o outro. Quem gere esta replicação é o RTP *middleware*.

Middleware: O *middleware* utilizado é o Resilient Telco Platform (RTP). O seu propósito é suportar software de alto nível, como o software aplicacional do hiQ8000, de maneira transparente para os utilizadores do sistema, e fornecer-lhe serviços como criação de contexto de processo, verificação da "saúde" dos processos, gestão de processos de backup, comunicação inter-processos (entre nós inclusivé), e muito mais. Este *software* utiliza o *software* do *cluster*, para comunicar com os nós, e o *software* de base de dados para guardar as variadas configurações essenciais ao funcionamento do sistema. Para administrar este *software* é utilizado um CLI fornecido com o mesmo denominado por "RTP CLI".

Aplicação: O *software* aplicacional do hiQ, na sua totalidade, assenta sobre o RTP *middleware*. As aplicações, utilizando a API do RTP *middleware*, têm uma visão única do *cluster* e têm acesso a vários serviços de comunicação, de redundância e acesso às bases de dados. Esta abstracção permite a alteração das componentes abaixo das aplicações sem que estas tenham que ser modificadas.

No âmbito deste projecto, dividimos a estrutura do hiQ em duas partes: Aplicação e Plataforma.

A Aplicação é o conjunto de todas as aplicações que implementam as funcionalidades dum *softswitch*, como por exemplo os *signalling managers.*

A Plataforma é o conjunto de ferramentas necessárias à instalação, manutenção e alteração de configurações do sistema operativo e respectivo *middleware*.

2.2.1 Instalação hiQ8000

Em termos da instalação da plataforma, ela é feita através de um *Installation Server* e de um conjunto de pacotes de *software*, que são instalados no mesmo, que automatizam o processo todo da instalação do hiQ.

Esses pacotes de *software* podem ser divididos em dois grupos principais:

- *Installation Framework*: Conjunto de aplicações GUI que ajudam o processo de instalação e actualização do hiQ. A aplicação principal deste pacote é o *Node Configuration Profile Editor*.

- *Installation scripts*: Conjunto de *scripts* que automatizam o processo de instalação do hiQ.

Para se instalar um hiQ começa-se por utilizar o *Installation Framework* em modo de instalação. Neste modo só existe uma ferramenta chamada *Node Configuration Profile Editor* (figura 3). Esta ferramenta gráfica, implementada em Java, gera e valida, seguindo um conjunto de regras bem definido, um ficheiro de texto denominado *node.cfg* que contem todos os parâmetros configuráveis do hiQ, como por exemplo IP's das interfaces de rede, nome da máquina, etc.

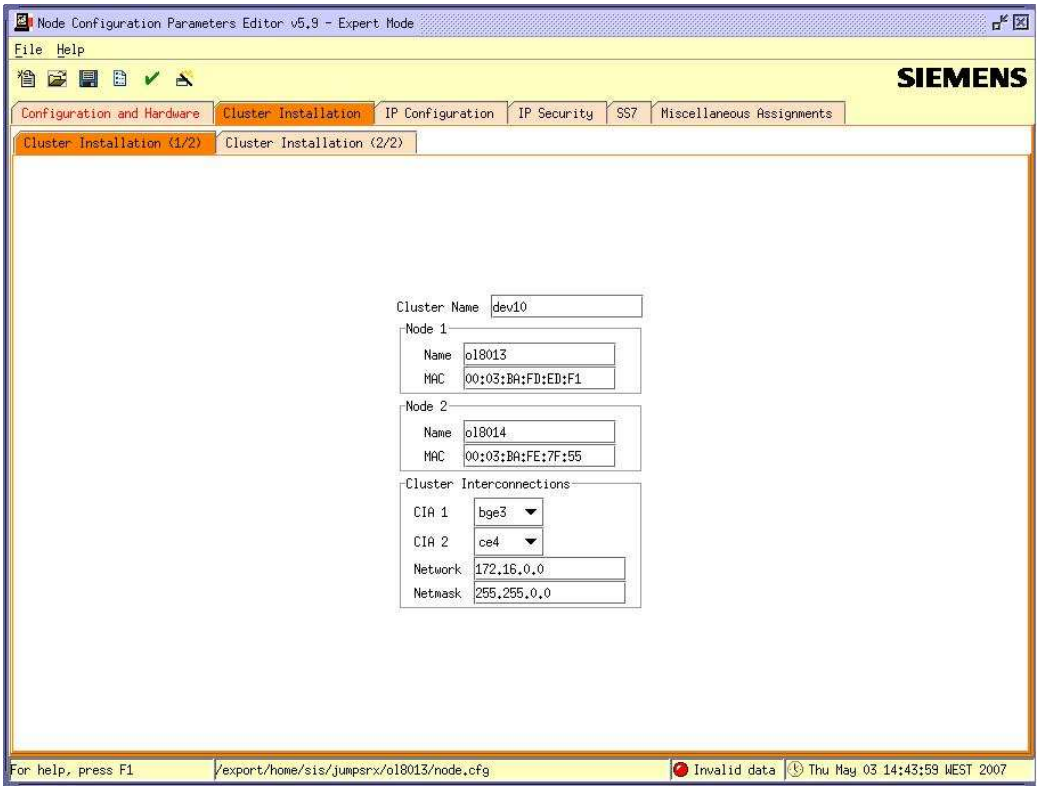


Figura 3 - NCPE

Depois do *node.cfg* ser gerado, são executados uns *scripts* de instalação que vão instalar o sistema operativo e todas as aplicações necessárias ao funcionamento do hiQ. Estes *scripts* vão utilizar os parâmetros que se encontram dentro do *node.cfg* para configurar a máquina.

Quando é preciso alterar estas definições utiliza-se o *Installation* Framework em modo de actualização (figura 4). Neste modo é utilizada uma ferramenta denominada *Installation Framework GUI* (IFGUI), que acede ao hiQ e retira a sua configuração actual. A partir desta configuração utiliza-se o NCPE outra vez, em modo de actualização, para gerar um novo *node.cfg*. De seguida o IFGUI acciona remotamente uns scripts denominados "*update scripts*". Estes *scripts* estão localizados na própria máquina e são responsáveis por fazerem a actualização a todas as áreas de impacto que são afectadas pelas mudanças pretendidas.

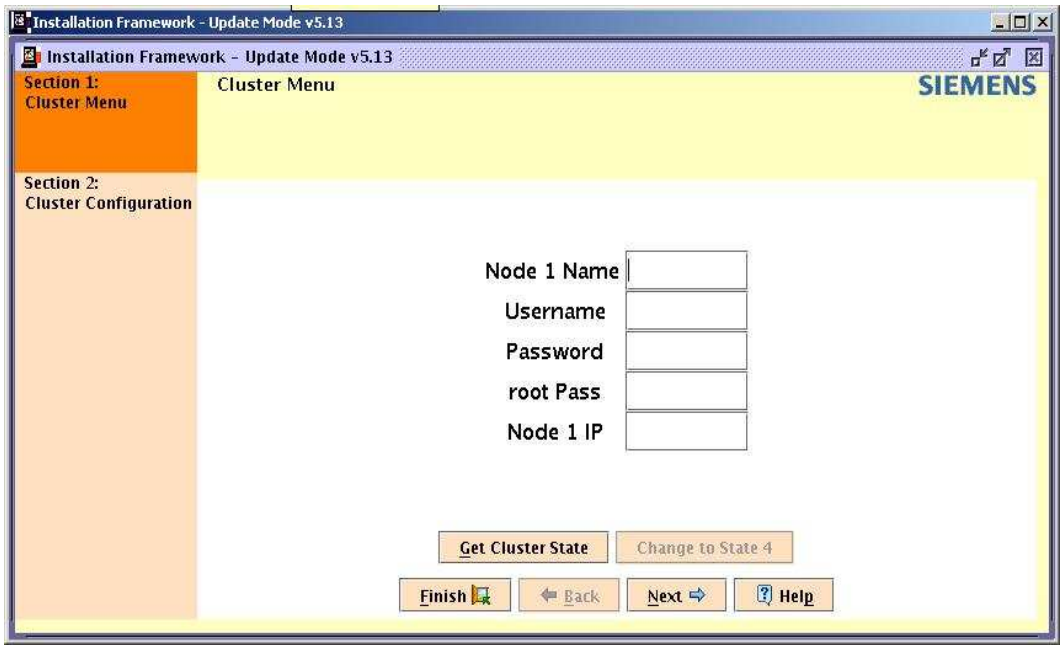


Figura 4 – Installation Framework em modo "update"

Capítulo 3

Layer 3 Cluster Interconnect

Como já foi dito na secção 2.2, o *software* de *clustering* utilizado no hiQ e no HiPath é o Cluster Foundation. A necessidade desta solução vem da alta diponibilidade inerente a um *cluster* pois no mundo das telecomunicações a disponibilidade é um factor crítico. Ninguém aceita o facto de ao tentar fazer uma chamada (VoIP ou não) não o consiga fazer. Assim se houver algum tipo de problema com um nó do *cluster* (ex: o *hardware* falha) o outro nó continua o seu trabalho como não nada tivesse acontecido. Por esta razão existem dois nós por cada hiQ/HiPath. Para quem pretende uma solução mais redudante também é possível separar geográficamente os nós do *cluster* para assim se precaver dos desastres naturais ou outros problemas geográficamente localizados.

O software de *clustering* permite às aplicações terem conhecimento do agrupamento de nós e do estado de cada um dos seus elementos (ex: vivo/morto). A interligação entre os dois nós do *cluster* é feita típicamente com uma ligação directa redudante sobre um canal com uma grande largura de banda, como por exemplo uma ligacação directa de um nó para o outro sobre *ethernet* 1Gbit/s. Isto deve-se principalmente ao facto de o mecanismo usado, para se verificar se um nó está morto ao não, utilizar uns pacotes (*heartbeats)* que vão até ao outro nó e voltam. Se num curto espaço de tempo não se receber resposta alguma do outro nó esse nó é dado como “morto”. Também precisamos juntar a isto o facto de haver uma grande quantidade de informação que é sincronizada entre cada nó. Logo a velocidade é um factor importante. Se pensarmos que o hiQ está normalmente sobre grande carga a fazer processamento de chamadas e a sincronizar a informação resultante num nó e outro, é fácil perceber a necessidade de termos uma interligação o mais estável possível.

Até à data, a interligação dos nós do cluster do HiPath é feita sobre um *link Layer* 2. Esta opção foi tomada por razões de *performance* tendo em conta o menor *overhead* desse tipo de *links*, o grande volume de informação que é trocada entre

cada nó do *cluster* e a necessidade de um tempo de resposta (latência) o mais reduzido possível. Este *performance* é contrabalançado por um custo maior que se evidência bastante em grandes distâncias, tornando um *cluster* geográficamente separado numa solução bastante dispendiosa. Então foi identificada a necessidade de se usar um método mais barato onde se pudesse ter uma solução geográficamente separada com um custo mais moderado. Foi então decidido que se iria implementar uma opção para a interligação do *cluster* funcionar sobre *Layer 3* (ver figura 5), mais especificamente sobre IP, que é bastante mais barato que um *link* em *Layer 2*, a acrescentar na proxima versão do HiPath. Os pacotes IP, além de terem um maior *overhead* que os pacotes em *Layer 2*, usam um método de roteamento *"Best Effort"* que tem grande impacto no tempo de resposta dos *heartbeats* do *cluster*, correndo o risco de algum nó ser dado como "morto" por engano. Para resolver este problema de *performance* foi decidido usar o campo ToS dos pacotes IPs, a serem encaminhados pela interligação, para se poder configurar a rede (ex: *routers*) para dar prioridade aos mesmos. Esta alteração é só para a versão empresarial HiPath pois existem algumas limitações técnicas na solução hiQ que dificultam um pouco mais esta implementação. A maior limitação tem haver com o facto do sistema hiQ usar discos partilhados externos que não se adequam a um cenário geográficamente separado. Apesar disso já se encontra em estudo a concretização duma solução idêntica para o mercado das operadoras.

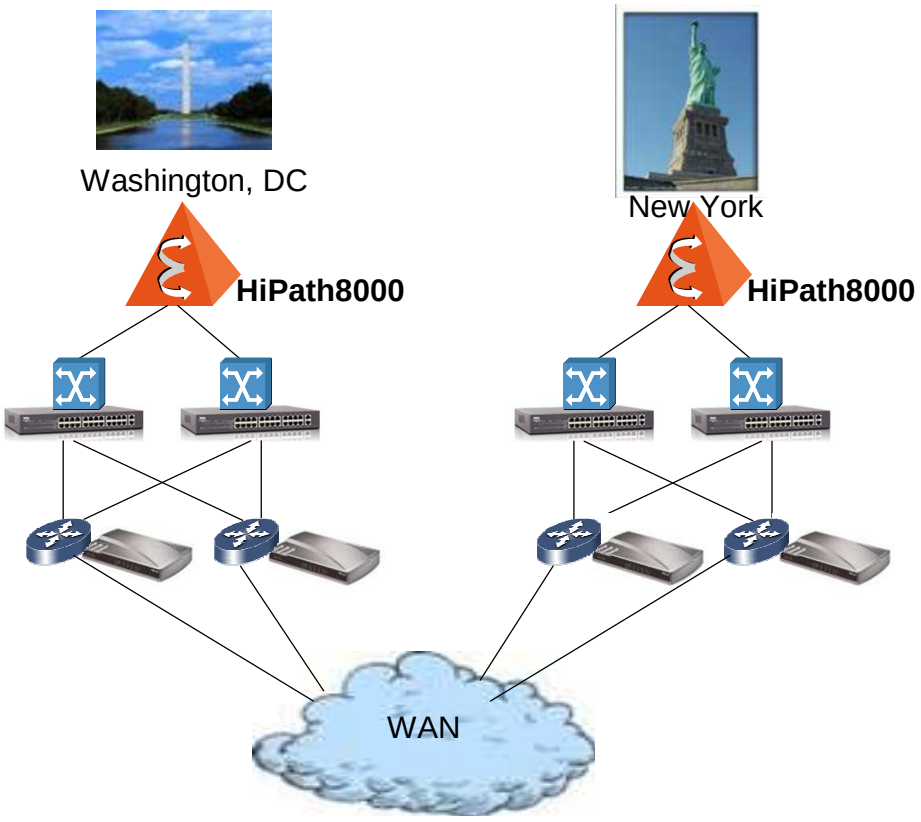


Figura 5 – Cenário Layer 3 *Interconnect*

A minha intervenção nesta feature começou por ser só a alteração da ferramenta NCPE para albergar um conjunto de novos parâmetros a configurar na altura da instalação. Mais tarde foi-me também atribuído a responsabilidade da alteração dos *scripts* de instalação e de *update* do sistema (ver Secção 2.2.1) para que a interligação seja configurada correctamente para o funcionamento sobre *Layer* 3. Além disso também foi necessário modificar certos *scripts* do RTP *middleware*, para criar novos parâmetros relativos à nova funcionalidade.

Os requisitos que seguem na tabela 1 foram definidos pelos *System Enginners* da Siemens Networks (Anexo A, Secção "Processo de Desenvolvimento") e são só os que me foram atribuidos. Existem outros requisitos, como por exemplo de *performance*, que foram entregues a outros elementos do projecto.

Nº	Descrição
1	Tem que haver uma opção no NCPE/node.cfg para o *cluster* interconnect entre ambos os nós seja em *Layer* 3 (roteavel) ex:. para que o Cluster Foundation (CF) seja configurado para correr sobre IP em vez de sobre Ethernet
2	Para a opção "CF over IP", só haverá um *cluster* interconnect configurado.
3	Não poderá haver um único ponto de falha no *cluster interconnect*. (ex: interfaces de rede, cabos, *routers*, etc). As duas interfaces de rede disponiveis para o *cluster interconnect* serão "ligadas" através do "*Linux bonding driver*".
4	Os endereços IP das interfaces de rede usadas para o *cluster interconnect* em ambos os nós farão parte de subnets privadas diferentes.
5	Antes do processo de configuração do CF, as interfaces de rede usadas para o *cluster interconnect* estarão configuradas em ambos os nós e serão capazes de comunicar um com o outro sobre o *cluster interconnect.*
6	O *Installation Framework*, modo "*update"*, tem que ter uma opção para alterar um sistema de *Layer 2* para *Layer 3*.
7	Para satisfazer os requisitos de performance, o seguinte vai ser aplicado: • Layer 2 priority routing via "VLAN *tagging*" e configuração de QoS na porta *Ethernet* do *switch* da rede. • Layer 3 priority routing via configuração de differentiated services code point (DSCP). Isto só pode ser feito pelo *software* do Cluster Foundation.
8	Todos os parâmetros de configuração do CF no node.cfg estarão disponiveis visualmente via CLI/iNMC/Assistant.

Tabela 1 – Requisitos

3.1 Análise e Design

Tendo em conta os requisitos definidos, identificados na tabela 2, podemos concluir que existem três áreas distintas de intervenção:

1 – **Instalação**. Tem que se conseguir instalar um sistema HiPath com uma interligação de *cluster* em *layer 3.*

2 – **Actualização.** Um sistema HiPath em *Layer* 2 tem que poder ser alterado para *Layer* 3.

2 – **Configuração RTP**. Os novos parâmetros de configuração, que são utilizados num contexto de um *link layer 3*, têm que ser visiveis para as aplicações do hiQ através do RTP *middleware*.

Para cada área de intervenção foram identificados todos os elementos a sofrerem alterações:

Área	Elemento	Nº Requisito	Impactos
Instalação	node.cfg	**1**	Acrescentar novos parâmetros de configuração necessários num contexto *layer 3*.
		7	Acrescentar novos parâmetros para o ToS a ser utilizado pelo Cluster Foundation.
	NCPE	**1**	Alteração da interface gráfica para acomodar o novo *node.cfg.*
		4	Acrescentar regras em relação às redes dos *interconnects*.
	Installation Scripts	**2,3**	Criação de um *bonded cluster interconnect* a ser utilizado

			pelo Cluster Foundation.
		5	Antes do Cluster Foundation ser configurado é preciso que os nós consigam comunicar pelo *cluster interconnect*.
		7	O Cluster Foundation tem que ser ser configurado para utilizar um ToS definido pelo utilizador.
Actualização	NCPE	**6**	O NCPE, em modo "*update*", tem que permitir alterar a interligação do *cluster* de *Layer 2* para *Layer 3*.
	Update scripts	**6**	Os *update scripts* têm que permitir alterar um sistema que se encontra em *Layer 2* para *Layer 3*.
Configuração RTP	RTP *parameters*	**8**	Acrescentar novos parâmetros RTP análogos aos novos parâmetros do *node.cfg*

Tabela 2 – Elementos a serem modificados

3.1.1 Análise Layer 2:

Para saber que alterações foram precisas fazer para a concretização de um *cluster interconnect* usando um *link Layer 3*, foi preciso analisar como funciona a implementação actual em *Layer 2* no HiPath8000.

Na configuração em *Layer 2* há dois *interconnects* designados para o Cluster Foundation Software. Um *cluster interconnect* liga os dispositivos /dev/eth1 em ambos os nós e outro conecta os dispositivos /dev/eth3. O Cluster Foundation

(CF) faz o balanciamento dos tráfego sobre os *interconnects* como mostra na figura 5.

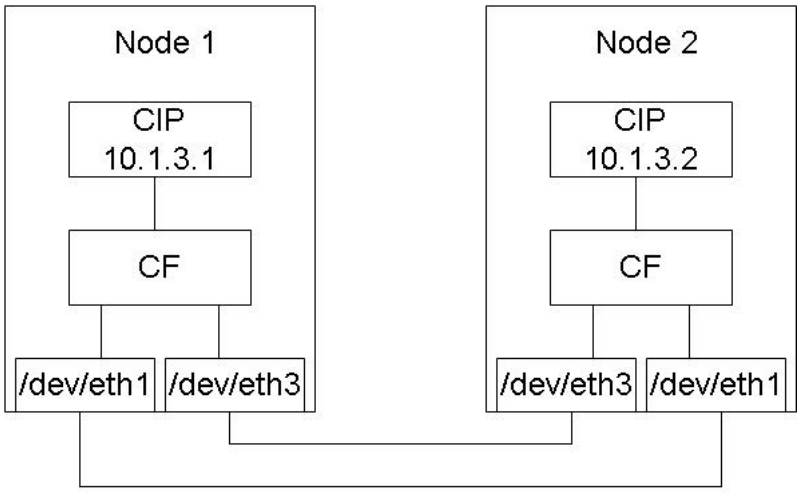


Figura 6 – CF over Ethernet

As interfaces eth1 e eth3 podem ser dispositivos ethernet ou dispositivos IP. O CIP é um *light-weight* IP *stack* que tem como objectivo servir de "ponto de entrada" para as aplicações. Ou seja, sobre o CF podem existir vários CIP's e cada aplicação usa esse IP lógico para diferenciar o tráfego que lhe interessa (figura 7).

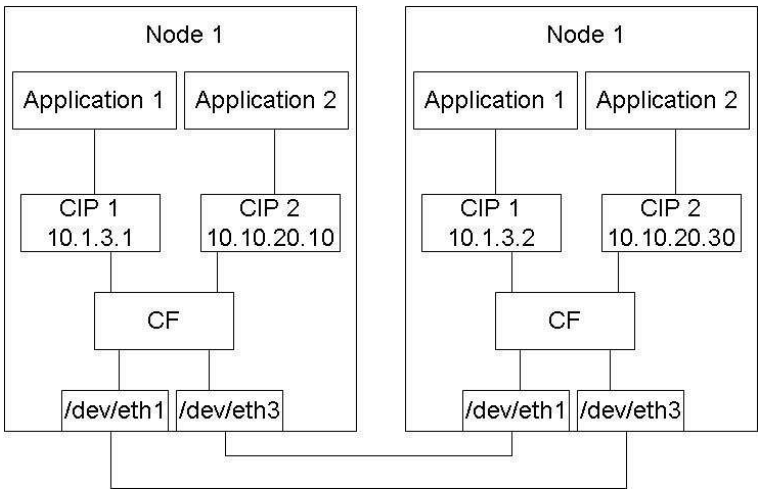


Figura 7 – Multiplos CIPs

No caso do hiQ, como só temos uma aplicação que reside no topo do CF, o RTP middleware, só temos um CIP. As interfaces, usadas pelo CF, existentes no hiQ são interfaces IP logo precisam de um IP para serem configuradas. Isto significa que a configuração específica usada no hiQ pode ser, por exemplo, a que consta na figura 8.

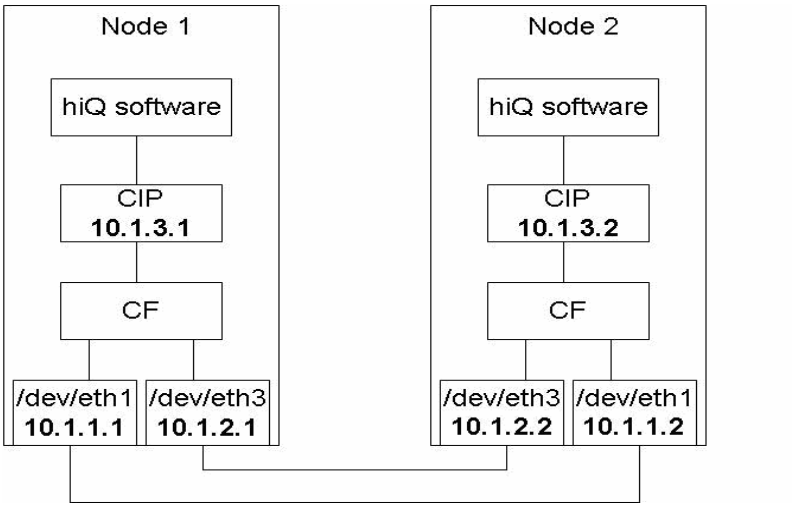


Figura 8 – Exemplo de configuração L2 do hiQ *interconnect*

node.cfg:

O cenário em *Layer 2* implica que na altura da instalação existem vários parâmetros no *node*.cfg para configurar os IPs necessários a configuração apresentada na figura 8. Ou seja o *node.cfg*, tem actualmente os seguintes parâmetros relativos à configuração *Layer* 2:

node_1_cip0 – IP da interface eth1 do primeiro nó. Existe um parâmetro análogo para o segundo nó.

node_1_cip1 – IP da interface eth3 do primeiro nó. Existe um parâmetro análogo para o segundo nó.

node_1_cip_logical – CIP do primeiro nó. Existe um parâmetro análogo para o segundo nó.

cip_network – Rede do CIP. É idêntica em ambos os nós.

cip_netmask – *Netmask* do CIP. É identical em ambos os nós.

Exemplo (seguindo a figura 8):

```
node_1_cip0 – 10.1.1.1

node_1_cip1 – 10.1.2.1

node_1_cip_logical – 10.1.3.1

node_2_cip0 – 10.1.1.2

node_2_cip1 – 10.1.2.2

node_2_cip_logical – 10.1.3.2

cip_network – 10.1.3.0

cip_netmask – 255.255.255.252
```

NCPE:

O NCPE tem um ecrân para configurar os parâmetros do *cluster interconnect*. Continuando a seguir o exemplo da figura 8, podemos ver na figura 9 como essa configuração é feita.

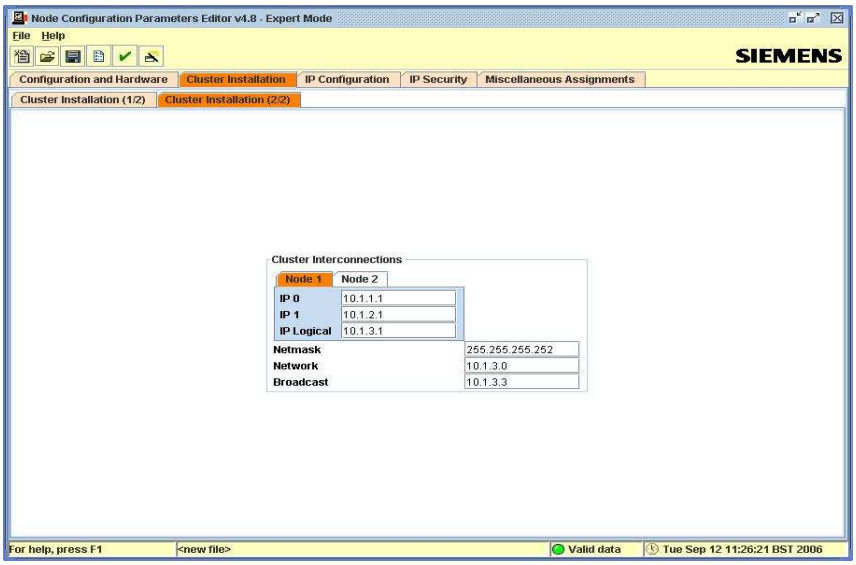


Figura 9 – NCPE: configuração L2 do *cluster interconnect*

No modo "*update*" o NCPE permite alterar todos os IPs atribuídos na altura da instalação. Logo o ecrân em modo "*update*" é idêntico ao ecrân utlizado na instalação (figura 10).

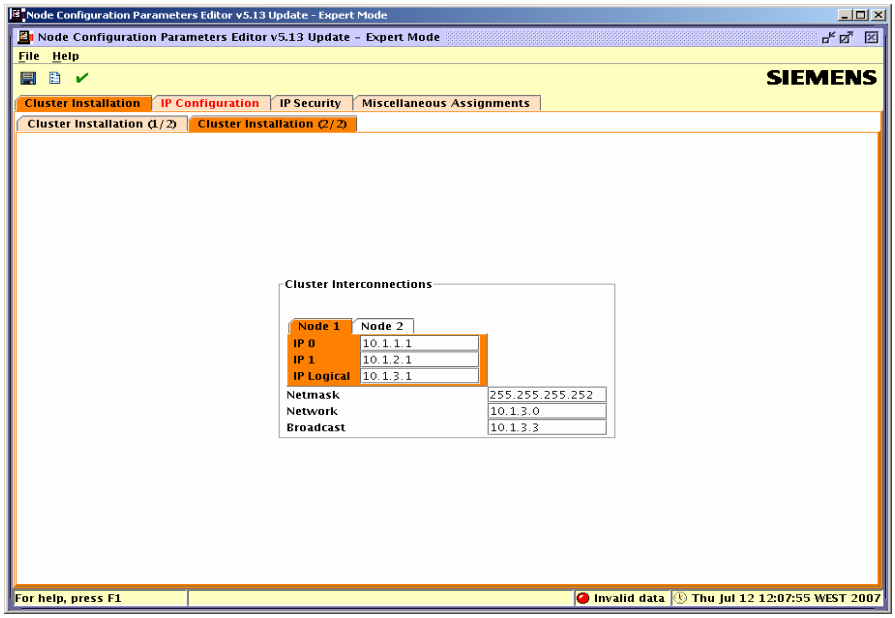


Figura 10 – NCPE em modo "update" (L2): Alteração de IPs do *cluster interconnect*

Installation Scripts:

O *script* principal de instalação, e o que nos interessa, é um denominado de "rtpinstall.sh". Nesse *script* podemos encontrar a lógica que configura os interfaces de rede e configura o CF para *Layer 2*.

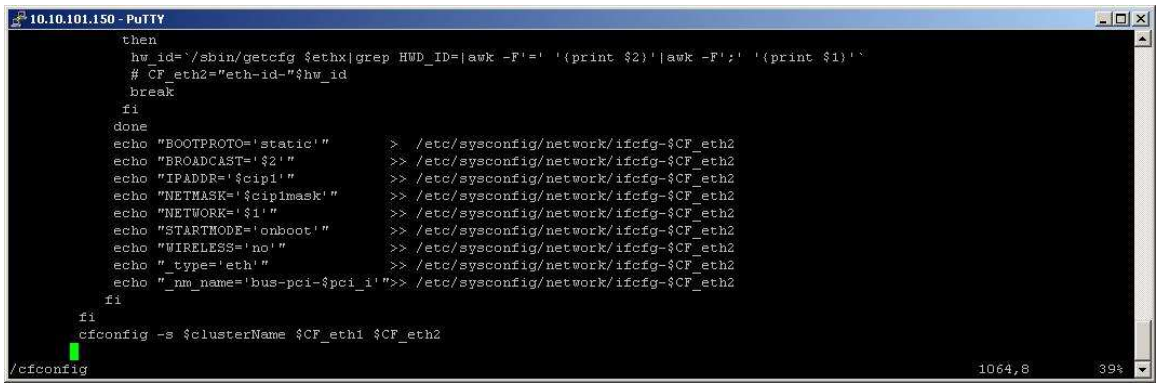


Figura 11 – Extracto do rtpinstall.sh: configuração interfaces & CF

Como podemos ver pelo extracto do rtpinstall.sh (figura 11), as interfaces são configuradas criando um ficheiro de configuração próprio de Linux para esse efeito, na directoria `/etc/sysconfig/network.` O CF é configurado utilizando um comando na linha de comando (figura 11), fornecido pelo próprio CF, chamado "`cfconfig`".

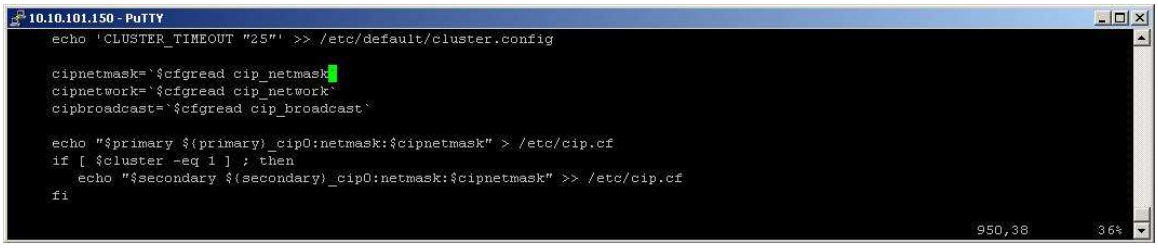


Figura 12 – Exctracto do rtpinstall.sh: configuração CIP

Em relação ao CIP, existe um ficheiro `/etc/cip.cf` que contem os parâmetros necessários à sua configuração. Este ficheiro também é editado no *script* rtpinstall.sh (figura 12).

Update Scripts:

Em relação aos *scripts* de actualização existe um *script* no próprio sistema na directoria `/opt/unisphere/srx3000/ifw/bin/Linux_2_6/` denominado `as_update_lib.sh.` Esse *script* tem uma lógica idêntica à utilizada nos *scripts* de instalação.

RTP *parameters*:

Os parâmetros do RTP são guardados num ficheiro `/opt/SMAW/SMAWrtp/cust_conf/SrxSystemDependant.parm.` Esses parâmetros são colocados nesse ficheiro por um *script*: `/etc/hiq8000/config/bin/dependent_alias_configuration.sh` que lê os parâmetros do *node.cfg*.

3.1.2 Análise Layer 3:

No caso de *Layer 3*, existe sómente um *interconnect* configurado utilizando um conceito conhecido em Linux como "*bonded interface*". Criar um *bonded interface* é dar um único IP a dois dispositivos físicos. O que acontece é que sómente um dispositivo está activado e se, por acaso, houver alguma falta nesse dispositivo o sistema operativo detecta-a, desactiva esse dispositivo e activa o outro. A outra grande alteração é devido ao facto de usar o IP *stack* do sistema operativo não precisamos da componente CIP (ver figura 13) como se usa no caso de *Layer 2*.

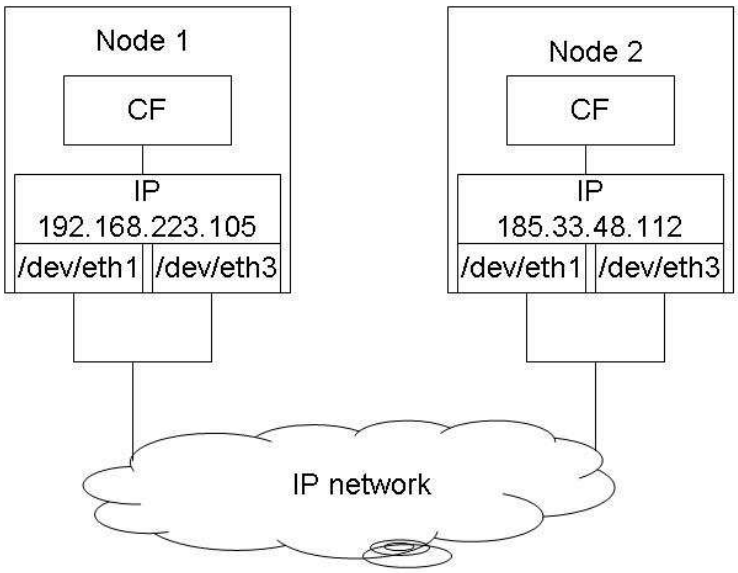


Figura 13 – CF over IP

node.cfg:

Em relação ao *node.cfg* foram precisos novos parâmetros para um contexto em *Layer 3*:

cf_type – Parâmetro que identifica se o *cluster* está a utilizar um *link layer* 2 ou 3. Pode ter o valor "2" ou "3" consoante o tipo de *layer* a adoptar.

cip2_network – Parâmetro que identifica a rede da interface de *interconnect* do segundo nó do *cluster*. Este parâmetro só tem significado num contexto *Layer* 3 e é um IP.

cip2_netmask – Parâmetro que identifica a *netmask* da rede da interface do *interconnect* do segundo nó do *cluster*. Este parâmetro só tem significado num contexto *Layer* 3 e é um IP.

cip_gateway – Parâmetro que identifica o *gateway* a ser utilizado pelo *interconnect* do primeiro nó para o segundo. Este parâmetro só tem significado num contexto *Layer 3* e é um IP.

cip2_gateway – Parâmetro que identifica o *gateway* a ser utilizado pelo *interconnect* do segundo nó para o primeiro. Este parâmetro só tem significado num contexto *Layer 3* e é um IP.

tos_data – Parâmetro que define o valor do campo *Type of Service* dos pacotes IP de dados a serem transmitidos pelo *cluster interconnect*. Este parâmetro só tem significado num contexto *Layer 3* e é constituido por dois números hexadecimais.

tos_control – Parâmetro que define o valor do campo *Type of Service* dos pacotes IP de controlo (ex: *heartbeat)* a serem transmitidos pelo *cluster interconnect*. Este parâmetro só tem significado num contexto *Layer 3* e é constituido por dois números hexadecimais.

NCPE:

Foi preciso alterar o ecrân de configuração do *cluster interconnect* para acomodar os novos parâmetros do *node.cfg*. Para esse efeito foi criado um protótipo do NCPE que podemos ver nas figuras 14 e 15.

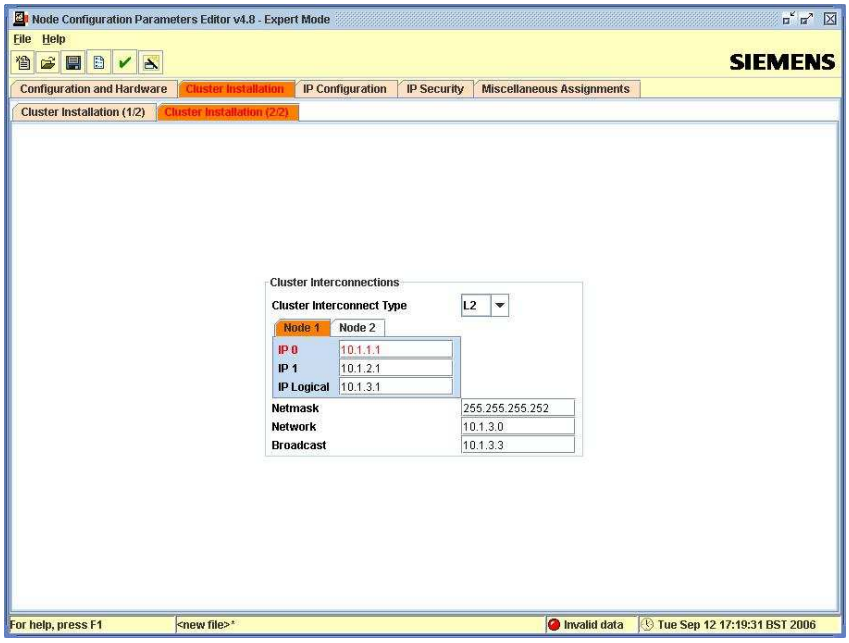


Figura 14 –NCPE proposto: configuração *Layer* 2

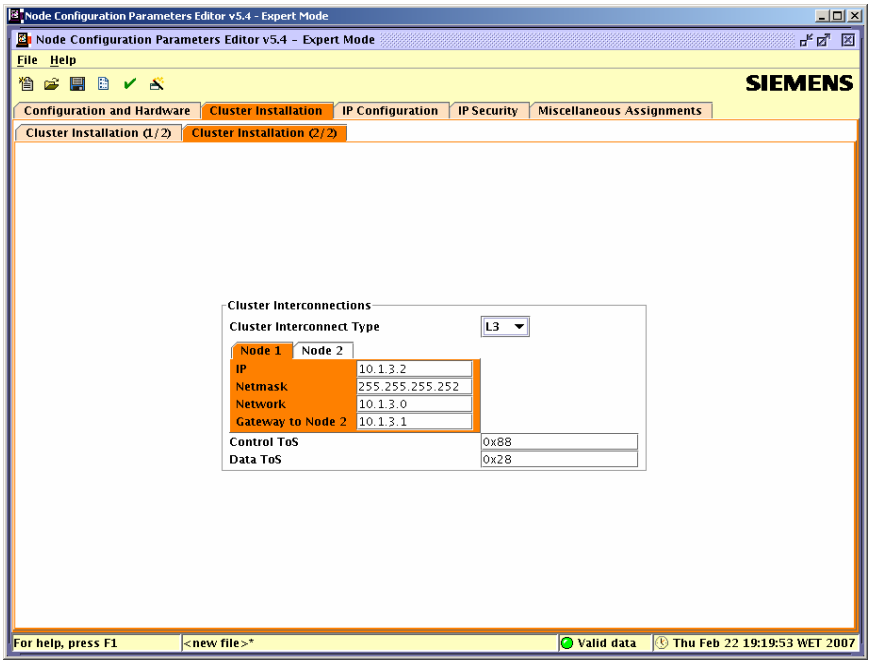


Figura 15 - NCPE proposto: configuração *Layer* 3

Para o NCPE em modo "*update"* foi utilizado uns ecrâns iguais aos apresentados anteriormente com a única diferença que só é permitido alterar de *Layer 2* para *Layer 3*. Esta restrição está relacionada directamente com o requisito 6 da tabela 1.

Installation Scripts:

Foi preciso alterar o *script* rtpinstall.sh de forma a que ele reconheça a diferença entre uma instalação *Layer 2 e Layer 3*. Isto é concretizado lendo o parâmetro "cf_type" do *node.cfg*. A partir daqui temos dois planos de accção:

1 – Se for *Layer* 2, configurar exactamente como "antigamente".

2 – Se for *Layer* 3, criar um ficheiro de configuração para o *bonded interface* na directoria `/etc/sysconfig/network`, excluir qualquer configuração relativa ao CIP, configurar o CF com os ToS definidos no *node*.cfg e configurar o CF utilizando o "`cfconfig`", como analisado no caso *layer* 2.

Para configurar o ToS do CF foi preciso editar um ficheiro de configuração desse *software* chamado "`cluster.config`" que reside em `/etc/default/`. Este valor depois é lido na altura do arranque desse *software*.

RTP *parameters*:

Neste caso simples foi só preciso editar o *script* `/etc/hiq8000/config/bin/dependent_alias_configuration.sh` e modificá-lo de forma a que tenha em consideração os novos parâmetros do RTP.

Os parâmetros novos que foram adicionados encontram-se na tabela 3 e têm uma relação directa com os parâmetros do *node.cfg* relativos ao *cluster interconnect*. Como ainda não existia nenhum parâmetro relacionado com o caso *Layer 2* também foram precisos adicionar parâmetros relacionados com esse caso.

RTP Parameters	Node.cfg Parameters
Srx/Main/ClusterName	*cluster_name*
Srx/Main/node1Name	*node_1_name*
Srx/Main/node1PhyIpAddr	*node_1_ip*
Srx/Main/node2Name	*node_2_name*
Srx/Main/node2PhyIpAddr	*node_2_ip*
Srx/Main/ClusterType	*cf_type*
Srx/Main/node1CIP0	*node_1_cip0*
Srx/Main/node1CIP1	*node_1_cip1*
Srx/Main/node1CIPLogical	*node_1_cip_logical*
Srx/Main/node2CIP0	*node_2_cip0*
Srx/Main/node2CIP1	*node_2_cip1*
Srx/Main/node2CIPLogical	*node_2_cip_logical*
Srx/Main/node1CIPNetmask	*cip_netmask*
Srx/Main/node1CIPNetwork	*cip_network*
Srx/Main/node2CIPNetmask	*cip2_netmask*
Srx/Main/node2CIPNetwork	*cip2_network*
Srx/Main/CIPBroadcast	*cip_broadcast*

Tabela 3 – Mapeamento entre parâmetros RTP e *node.cfg*

3.2 Implementação

O processo de desenvolvimento correu em conformidade com o planeado durante a fase de análise. Não há qualquer tipo de ocorrência a reportar e todas as *milestones* até à fase de testes de integração foram respeitadas.

3.3 Testes de Componente e Integração

Como a forma mais básica de testar uma componente desta *feature* é uma instalação do sistema ou um alteração de *Layer 3* para *Layer 2*, que por sí só já implica o interfuncionamento das várias componententes, podemos concluir que os testes de componente e integração são idênticos.

Para esse efeito foram executados os seguintes testes:

1 – *Install Layer 2:* Isto é um teste onde o *cluster* interconnect é configurado para utilizar *Layer* 2.Este teste teve que ser executado para testar se as alterações feitas não estragaram o funcionamento antigo do sistema.

2 – *Install Layer 3*: Neste teste o *cluster inteconnect* é configurado para utilizar *Layer 3*. Neste teste todas as componentes alteradas relativas à instalação são testadas. Se a implementação estiver correcta, este teste configura o sistema em *Layer 3* correctamente.

3 – *Upgrade Layer 3:* Neste teste altera-se um sistema que foi instalado em *Layer 2* para *Layer 3*. Neste teste todas as componentes relativas ao processo de *update* são testadas. Se a implementação estiver correcta, este teste altera o sistema de *Layer 2* para *Layer 3* correctamente.

4 – *Check RTP parameters:* Neste teste verifica-se que se os parâmetros introduzidos no *node.cfg* aparecem na base de dados do RTP. Assim verificamos se a alteração relativa aos parâmetros RTP foram bem concretizadas.

Nas tabelas 4 , 5, 6 e 7 seguem as descrições detalhadas dos testes executados.

Install Layer 2 design steps		
Step Name	**Description**	**Expected Result**
Criar node.cfg com NCPE	Usar NCPE para criar um node.cfg com "cluster interconnect type" como "Layer 2"	node.cfg com "cf_type == 2"
Instalar sistema	Utilizar procedimento oficial para instalar o sistema.	sistema instalado e a funcionar correctamente.

Tabela 4 –Passos do teste de integração "Install Layer 2"

Install Layer 3 design steps		
Step Name	**Description**	**Expected Result**
Criar node.cfg com	Usar NCPE para crier um	node.cfg com "cf_type == 3"

NCPE	node.cfg com "cluster interconnect type" como "Layer 3".	
Instalar sistema	Utilizar procedimento oficial para instalar o sistema.	sistema instalado e a funcionar correctamente.

Tabela 5 – Passos do teste de integração "Install Layer 3"

Upgrade Layer 3 design steps		
Step Name	**Description**	**Expected Result**
Executar IFGUI	Utilizar *Installation Framework* em modo "*update*"	IFGUI em modo "update"
Alterar para *Layer 3*	Utilizar IFGUI para alterar a configuração do sistema para *Layer 3*.	*node.cfg* gerado com parâmetros relativos a um cenário *Layer 3*.
Executar *Update*	Utilizar o IFGUI para executar um *update*.	Sistema configurado correctamente com *cluster interconnect* em *Layer 3*.

Tabela 6 – Passos do teste de integração "Upgrade Layer 3"

Check RTP parameters design steps		
Step Name	**Description**	**Expected Result**
Executar RTP CLI	Executar o RTP CLI.	Menu do RTP CLI
Verificação	Utilizar o menu do RTP CLI para verificar se os novos parâmetros.	Novos parâmetros relativos ao *cluster interconnect*.

Tabela 7 – Passos do teste de integração "Check RTP parameters"

Capítulo 4

Remote Patching Capability: RTP Rolling Upgrade Daemon

As correcções do *software* do hiQ são feitas a partir de *patchs*. *Patchs* são pacotes, que contêm versões corrigidas de ficheiros binários ou de texto, que são instalados no sistema. De tempo a tempo esses *patchs* são agrupados em *patchsets* e são esses *patchsets* que são utilizados nas actualizações de *software* do hiQ. O facto dos *patchs* conterem binários que estão a ser utilizados pelo sistema obriga-nos a ter que parar o sistema antes de instalar os mesmos. Num sistema como o hiQ, onde a disponibilidade é um factor crítico, é necessário concretizar actualizações de *software* de forma a que o sistema nunca pare de funcionar. Para isso é utilizado um método chamado *Rolling Upgrade*, que é disponibilizado pelo *middleware* RTP.

Para explicar como o método de *Rolling Upgrade* funciona, vamos imaginar que queremos fazer uma actualização do *software* do hiQ da versão A para uma versão B:

1 – Começamos por ter um *cluster* com a aplicação hiQ, versão A, a funcionar correctamente dos dois nós como está representado na figura 16.

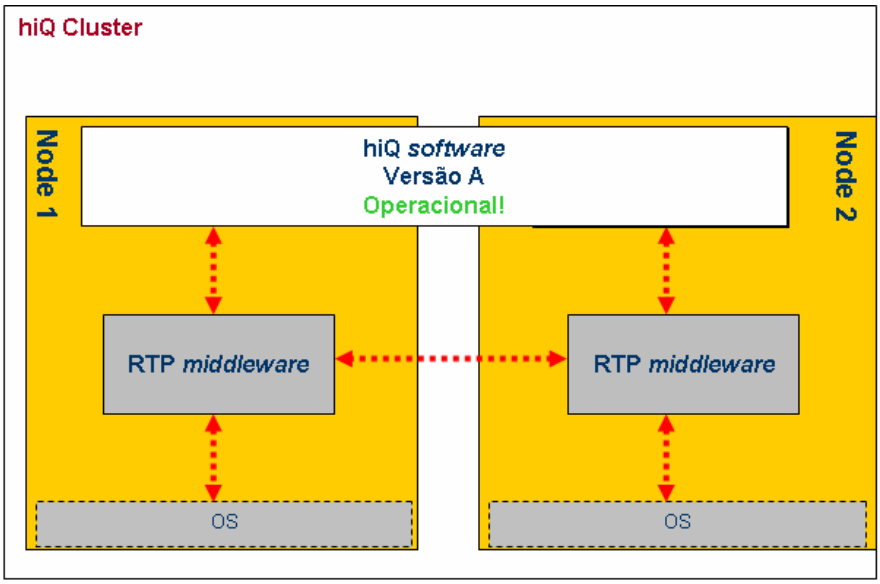


Figura 16 – hiQ a funcionar com versão A do *software*.

2 – O *Rolling* Upgrade utiliza pacotes próprios chamados "*MASTER UNIT*". *Logo* é preciso preparar os *patchsets* para serem utilizados pelo *Rolling Upgrade*. Para isso é utilizado um *script* chamado "setup_rtp_unit". Este *script* além de preparar os *patchsets* também faz alguns testes de validação, como por exemplo ver estamos a instalar uma versão correcta para a nossa base corrente. Como "input" utilizamos o *patchset* que queremos instalar e como "output" é nos dado a directoria onde o "*MASTER UNIT*" está localizado.

3 – Utiliza-se o RTP CLI para accionar o *Rolling Upgrade* dando como "input" a localização do "*MASTER UNIT*" a utilizar. A partir deste passo o *Rolling Upgrade*, propriamente dito, começa.

4 – O *Rolling Upgade* começa. Todos os processos que fazem parte da aplicação do hiQ do primeiro nó são desligados. Todo o processamento de chamadas é agora feita sómente no segundo nó (figura 17).

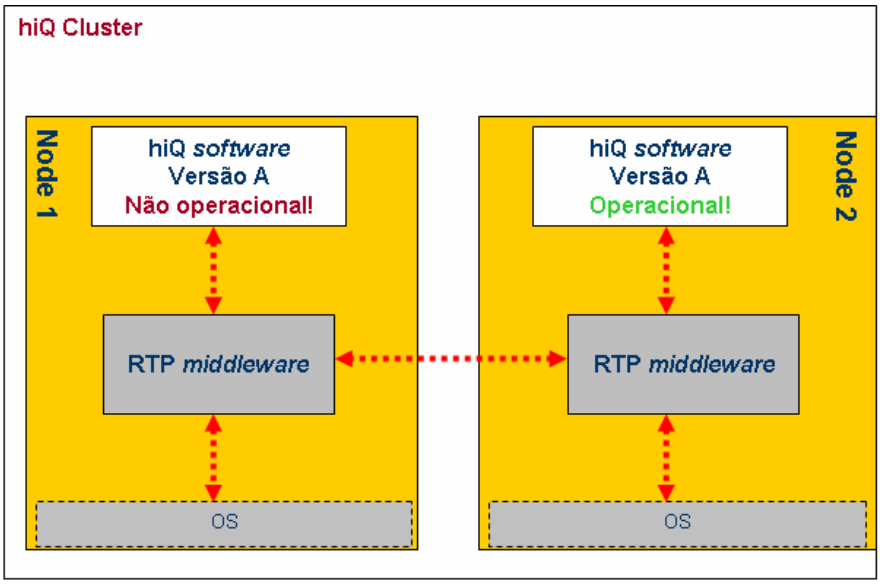


Figura 17 – hiQ a funcionar sómente no segundo nó.

5 – O *software* é actualizado no primeiro nó. O segundo nó continua a funcionar utilizando o *software* antigo (figura 18).

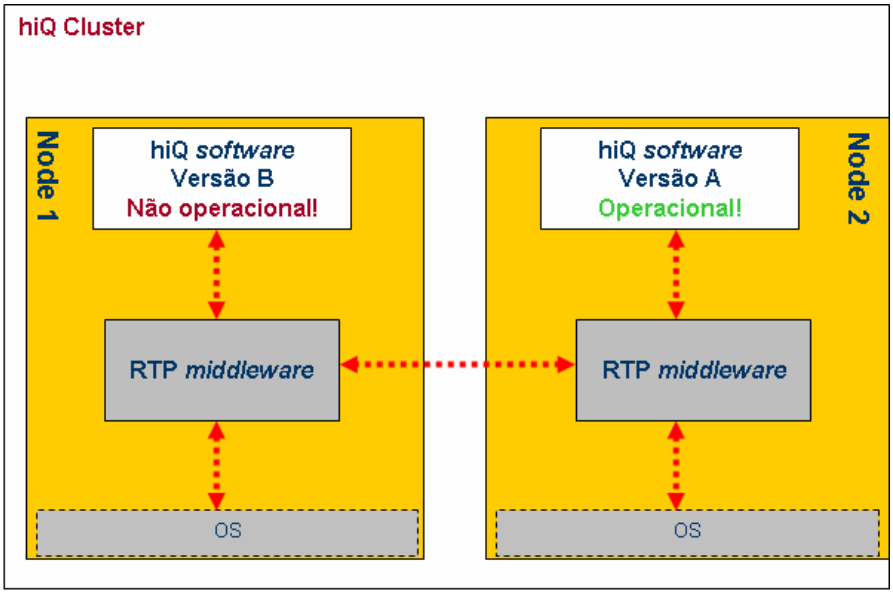


Figura 18 – hiQ com o primeiro nó actualizado e segundo nó a funcionar com versão antiga.

6 - A aplicação no primeiro nó é posta em funcionamento. Nesta altura são feitos testes básicos no primeiro nó para verificar se a actualização de *software* correu bem antes de se continuar (figura 19).

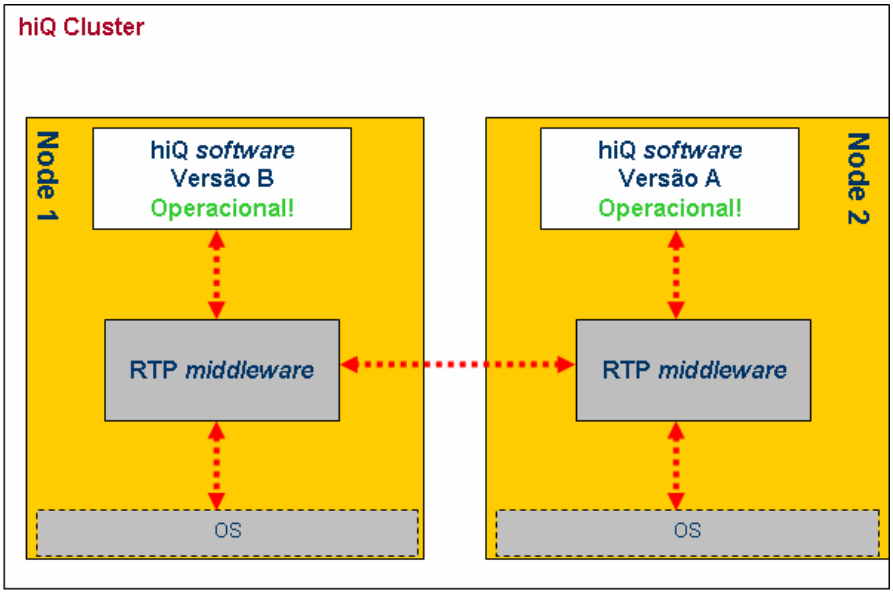


Figura 19 – hiQ a funcionar em ambos os nós com versões diferentes.

7 – Se tudo correu bem no primeiro nó então todos os processos da aplicação do segundo nó esão desligados. Todo o processamento de chamadas é agora feito pelo primeiro nó que utiliza a nova versão de *software* (figura 20).

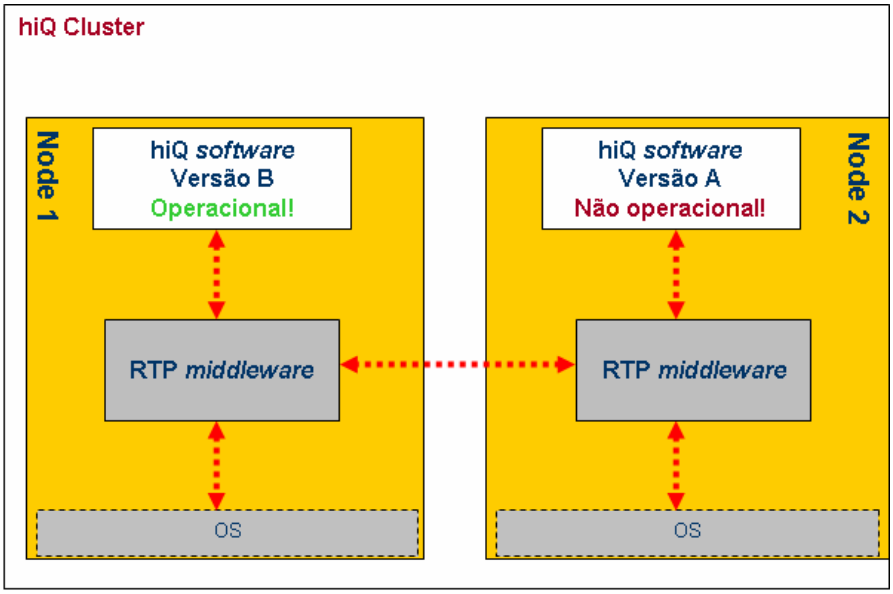


Figura 20 – hiQ a funcionar sómente no primeiro nó.

8 – O *software* é actualizado no segundo nó. O primeiro nó continua a funcionar utilizando o *software* novo (figura 21).

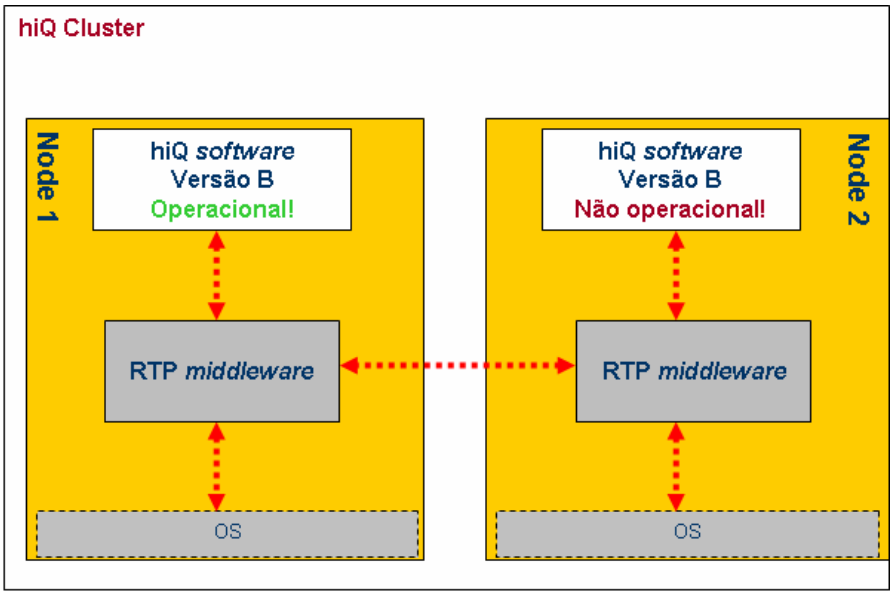


Figura 21 – hiQ com o segundo nó actualizado e o primeiro nó a funcionar com a versão nova.

9 - A aplicação no segundo nó é posta em funcionamento. Nesta altura são feitos testes básicos no *cluster*, como um todo, para verificar se a actualização de *software* correu bem (figura 22).

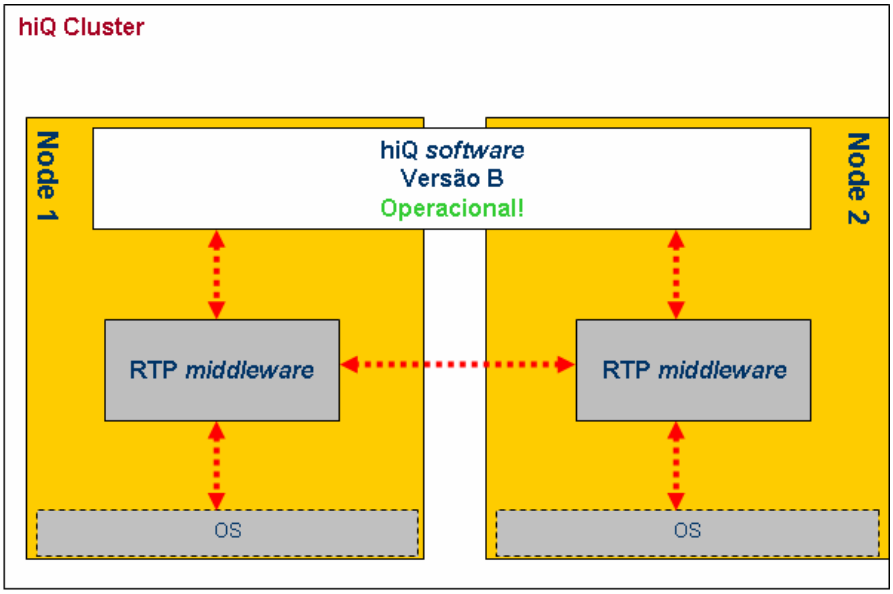


Figura 22 – hiQ a funcionar com a versão B do *software*.

Como podemos verificar o método de *Rolling Upgrade* permite fazer uma actualização de *software* sem nunca ter o sistema totalmente em baixo. Também verificámos que é necessário um *cluster* com, no mínimo, dois nós. Além disso também é necessário que as versões sejam compatíveis entre sí pois existe uma altura onde ambas as versões estão a funcionar ao mesmo tempo (figura 19).

As grandes desvantagens deste método é o facto de ter que se ter acesso remoto ao sistema, via uma *shell*, e a grande quantidade de interacções que são precisas pela parte do operador até o *Rolling Upgrade* estar concluído. O facto de se precisar de acesso por uma *shell* tem implicações a nível de segurança, pois estamos a abrir o sistema a um operador que só queremos que faça a actualização do *software*, e também implica que estamos a dar hipótese a que o operador cometa algum erro que possa afectar o sistema. A grande quantidade de interacções dá muito espaço de manobra para o operador cometer erros e também aumenta o tempo da actualização, pois o operador tem que estar 100% do tempo

com atenção à consola. Estas desvangens, principalmente o facto de termos várias janelas de oportunidade para o operador cometer algum erro, têm um custo associado de manutenção que se pretende evitar ou minimizar.

A iniciativa por de trás desta *feature* foi criar um método automatizado, com a menor intervenção possível dum operador, e que possa ser feito remotamente. Assim espera-se que a taxa de erros nas actualizações de *software* seja muito menor e que se possa poupar largas quantias em custos de manutenção por parte dos clientes da Nokia Siemens Networks.

Dum ponto de vista geral (figura 23) existe uma aplicação denominada "*Patching Tool*" que controla todo o processo de preparação dos *patchsets* e de actualização do sistema utilizando o método já existente de *Rolling* Upgrade. O *Patching Tool* disponibiliza uma interface, implementada em SOAP, para as aplicações gráficas de configuração do hiQ puderem utilizar (ex: iNMC e Assistant. Anexo A: secção "Configuração hiQ").

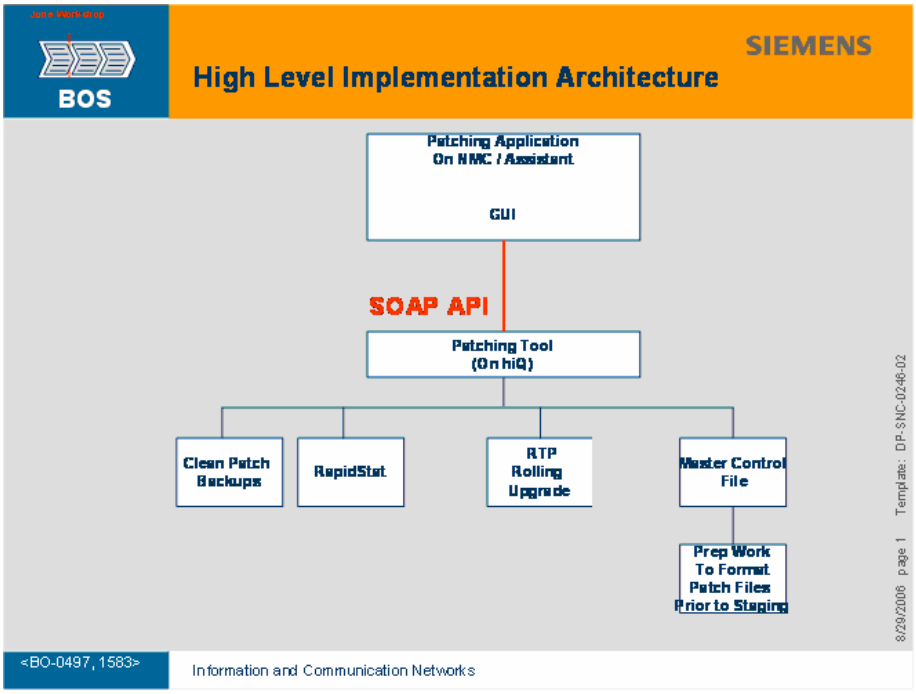


Figura 23 – Arquitectura do *Patching Tool*

O *Patching Tool* depois utiliza quatro componentes apresentadas também na figura 23:

1 – *Clean Patch Backups:* Módulo que permite apagar *patchsets* que estejam gravados no disco e já não sejam necessários (ex: já foram instalados).

2 – *RapidStat:* Módulo que permite verificar a saúde do sistema. Este módulo limita-se a correr um *script* chamado "RapidStat" que faz um relatório sobre a saúde dos vários componentes do sistema (ex: espaço no disco, problemas em processos, etc). Após cada actualização de *software* é preciso correr esta ferramenta para ter a certeza da boa saúde do sistema.

3 – *RTP Rolling Upgrade:* Este módulo controla e automatiza o processo de *Rolling Upgrade*.

4 – *Master Control File:* Este Módulo corre o *script* "setup_rtp_unit". Ou seja, este módulo faz a preparação dos *patchsets* para o formato "*MASTER_UNIT*" utilizado pelo *Rolling Upgrade*.

Para entrar mais no detalhe da estrutura dos módulos do *Patching Tool* podemos olhar para a figura 24. Na figura indiquei com um quadrado encarnado a minha área de intervenção.

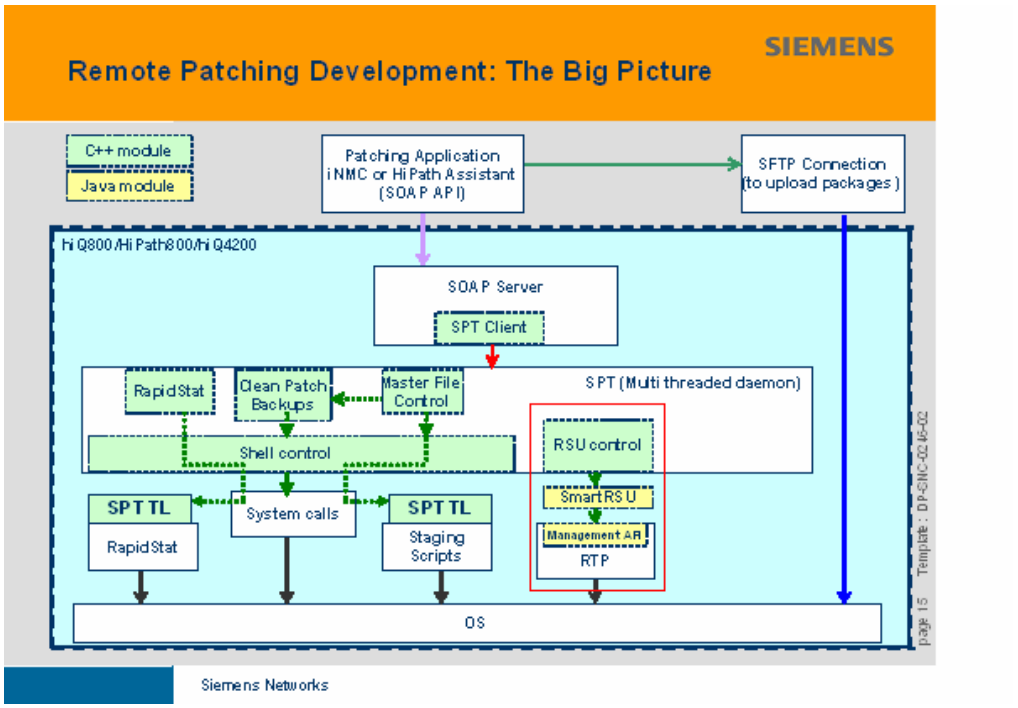


Figura 24 – Componentes do *Patching Tool* (encarnado indica área de intervenção)

O meu trabalho nesta *feature* foi a concretização dum módulo em Java que se interliga ao RTP *middleware*, que controla o processo de *Rolling Upgrade* e que o automatiza. Depois criei um módulo em C++ que encapsula as funcionalidades do módulo Java e que fornece essas mesmas para o *Patching Tool*. A este módulo chamaremos, a partir de agora, de *SmartRSU* (diminuitivo para *Smart Rolling Software Upgrade*).

Na tabela 3 estão definidos os requisitos, definidos pelos *System Engineers*, que são relevantes para a concretização do *Smart Patching Tool.*

Nº	Descrição
1	O Patching Tool pode ser utilizado para remover/desinstalar *patchsets*, depois do upgrade mais recente ter acabado com sucesso.
2	O Patching Tool deve permitir cada nó ser actualizado em sequência. Isto é para permitir a execução de testes de verificação manuais (ex: comportamento do processamento de chamadas, saude do sistema) a serem feitos quando um nó acabou de ser actualizado e antes de proceder com o nó seguinte. O utilizador deve ser avisado para executar os testes manuais e dado uma opção para continuar ou fallback.

3	Mensagens de progresso devem ser mostradas ao utilizador durante o processo todo de actualização.
4	O utilizador tem que perceber se o nó está ou não em condições de começar uma actualização (ex: se um *upgrade* já está em curso e o utilizador tenta começar um)
5	O utilizador deve saber o resultado final de uma actualização.(Sucesso, Falha/porque).
6	Não se deve pedir input ao utilizador depois de um processo de actualização ter começado. Todo o input deve ser introduzido antes do upgrade começar. Serão providenciados pontos Y/N (ex: continuar, voltar para trás), e avisos para contactar Customer Service.
7	O Patching Tool deve recolher toda informação necessária ao upgrade antes do procedimento começar.
8	Upgrades têm que continuar a ser possiveis utilizando o antigo método manual. Este método providencia "expert-mode".
9	*Rolling Upgrade* continua a ser o método primário e preferido de instalar patches. Tem que poder ser activado manualmente como também pela interface SOAP.

Tabela 8 – Requisitos

4.1 Análise e Design

O SmartRSU é um componente totalmente novo. Logo foi preciso desenhar uma aplicação que obedeça aos requisitos apresentados na tabela 8.

Tendo em conta que a ferramenta utiliza uma funcionalidade já existente no *middleware* utlizado, comecei por analisar o processo de *Rolling Upgrade* existente. Para tal foi utilzado o RTP CLI descrito no Anexo A, secção "Configuração hiQ".

O RTP CLI utiliza como *input* a localização de um pacote chamado "*MASTER UNIT"*. Um *master unit* é um ficheiro de controlo num formato específico e o conjunto de *patches* a instalar. Para criar esse *master* unit é utilizado uma ferramenta chamada "setup_rtp_unit". Esta preparação vai ser concretizada pela aplicação *Patching Tool*, que depois fornece a localização do mesmo ao SmartRSU.

É preciso também salientar que uma das funcionalidades existentes no processo de *Rolling Upgrade,* fornecido pelo *middleware,* é que se houver um erro em qualquer altura do processo, pode-se executar um "fallback". Isto é, retroceder

todos os passos feitos, desde o momento onde ocorreu o erro, até voltar ao estado inicial antes da tentativa de *upgrade*.

Analisando o processo de *Rolling Upgrade* chegou-se à conclusão da existência dos estados indicados na tabela 9 como "Estados RTP RSU". Para melhor se perceber o significado de cada estado RTP RSU pode-se consultar a tabela 10, onde estão as mensagens de texto que o utilizador lê aquando cada estado.

Os estados marcados a **negrito,** na tabela 9, são os estados onde o processo pede algum tipo de interactividade por parte do utilizador. O estado a sublinhado é o estado que marca o fim da instalação de um nó. O estado a itálico indica o ultimo estado onde o RTP consegue concretizar uma operação de *fallback* automaticamente.

Tendo em conta o requisito que indica que só se deve pedir *input* da parte do utilizador no final da instalação de cada nó, mapeou-se cada fase entre *inputs* para um estado de alto nível que depois possa ser utilizado e automatizado pelo *Smart Patching Tool*. Esses estados estão indicados como "Estados *SmartRSU"* na tabela 9.

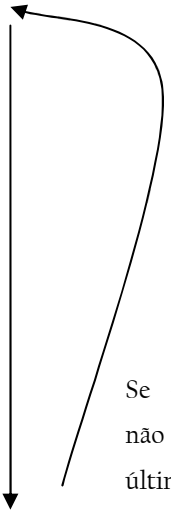
Estados SmartRSU	Estados RTP RSU	Fluxo
UPDATE_PREPARATION_STARTED	CONTROL_INIT	
	CHECK_UNIT	
	GLOBAL_PROCESS_CHECK	
	SEND_UPGRADE_MSG_TO_PROCESSES	
	APPLICATION_SCRIPTS_RTP_RUPG_BEGIN	
	CREATE_UPGRADE_ENVIRONMENT	
	COPY_UNIT_TIMEOUTS	
	CHECK_MIRROR_TAB	
	UNMIRROR_SHARED_DISKS	
UPDATE_STARTED or UPDATE_CONTINUE_OTHER_NODE	CONTEXT_SYNC_CHECK	
	STOP_RTP	
	UNMIRROR_LOCAL_DISKS	
	INSTALL_UNITS	
	APPLICATION_SCRIPTS_RTP_RUPG_INSTALL_DONE	
	CONFIGURE_NEW_SW	
	START_RTP	
	SUBSYSTEM_CHECK	
UPDATE_NODE_COMPLETED or UPDATE_ALL_NODES_COMPLETED	**HALT**	
UPDATE_CONTINUE_ACTIVATING_NEW_CONTEXT	APPLICATION_SCRIPTS_RTP_RUPG_COMPLETE	
	CONTEXT_CHECK	
	ACTIVATE_NEW_CONFIGURATION	
UPDATE_COMMITING	*DISTRIBUTE_CONFIG_FILES*	
	APPLICATION_SCRIPTS_RTP_RUPG_CLEANUP	
	CLEANUP	
	MIRROR_LOCAL_DISKS	
	MIRROR_SHARED_DISKS	
UPDATE_FINALIZING	DELETE_UPGRADE_ENVIRONMENT	
UPDATE_COMPLETED	INSTALL_COMPLETE	

Tabela 9 – Tabela de estados RTP RSU e SmartRSU

Depois de encontrado os estados necessários para um *Rolling Upgrade* como especificado pelos requisitos, criei a seguinte máquina de estados (figura 25):

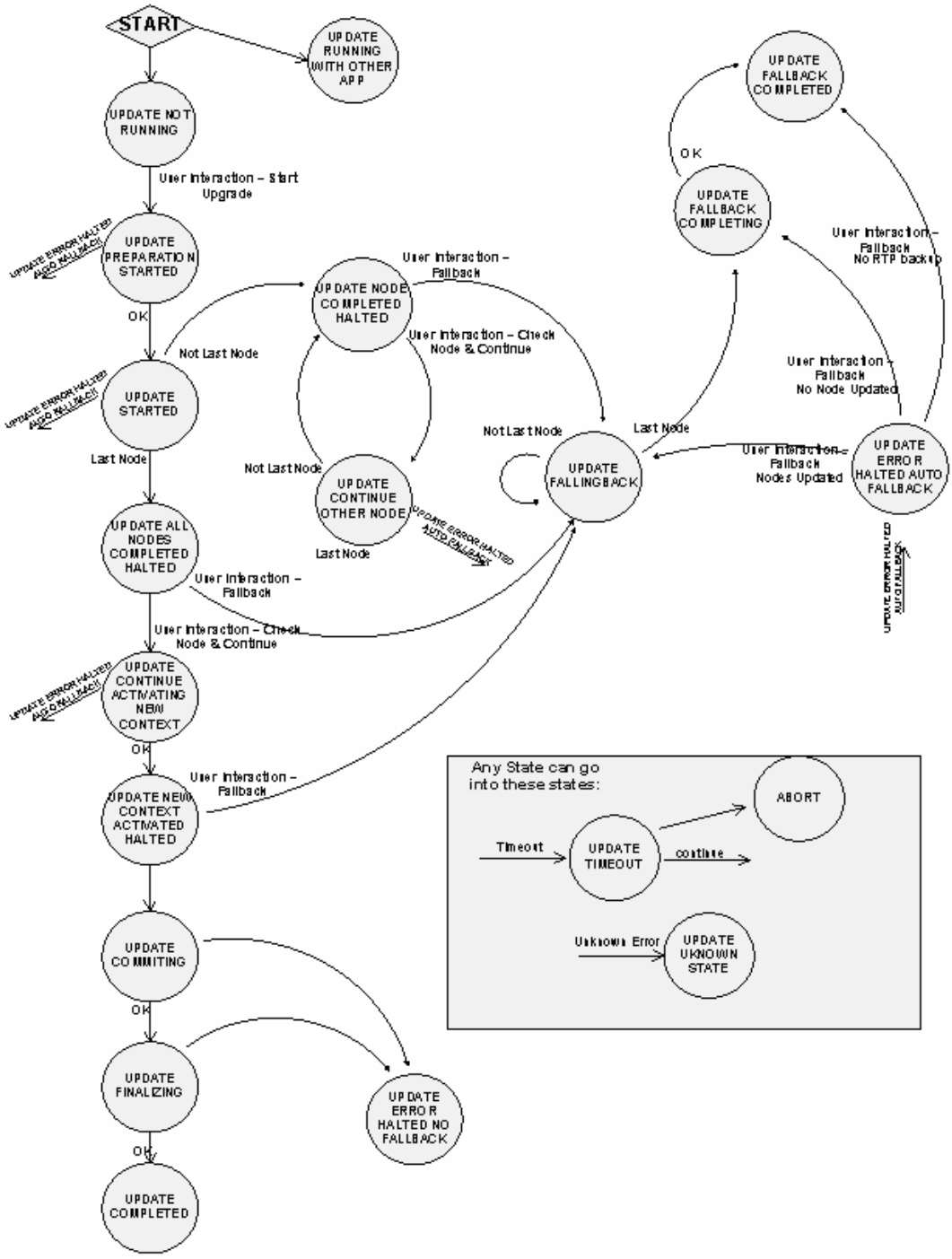


Figura 25 – Estados do SmartRSU

Para a concretização das mensagens, a serem enviadas ao utilizador (requisito 3 da tabela 8), de progresso, mapeou-se uma mensagem de texto para cada estado encontrado. Este mapeamento encontra-se na tabela 10.

RTP RSU State	String
CONTROL_INIT	control initialization!
CHECK_UNIT	checking install unit!
GLOBAL_PROCESS_CHECK	global process restart!
SEND_UPGRADE_MSG_TO_PROCESSES	sending upgrade msg to registed processes!
APPLICATION_SCRIPTS_RTP_RUPG_BEGIN	running application specific scripts with RTP_RUPG_BEGIN flag!
CREATE_UPGRADE_ENVIRONMENT	creating rolling upgrade environment!
COPY_UNIT_TIMEOUTS	copying unit specific command timeouts to update environment!
CHECK_MIRROR_TAB	checking mirror disk configuration!
UNMIRROR_SHARED_DISKS	disabling configured shared mirror disks!
CONTEXT_SYNC_CHECK	*node_name*: checking context synchronization!
STOP_RTP	*node_name*: stopping RTP!
UNMIRROR_LOCAL_DISKS	*node_name*: disabling configured local disks!
INSTALL_UNITS	*node_name*: installing SW units!
APPLICATION_SCRIPTS_RTP_RUPG_INSTALL_DONE	*node_name*: running application specific scripts with RTP_RUPG_INSTALL_DONE flag!
CONFIGURE_NEW_SW	*node_name*: configuring new SW!

START_RTP	*node_name*: starting RTP!
SUBSYSTEM_CHECK	*node_name*: local subsystem and process check!
APPLICATION_SCRIPTS_RTP_RUPG_COMPLETE	running application specific scripts with RTP_RUPG_COMPLETE flag!
CONTEXT_CHECK	checking context!
ACTIVATE_NEW_CONFIGURATION	activating new configuration and restarting all processes!
DISTRIBUTE_CONFIG_FILES	distributing configuration files to all nodes!
APPLICATION_SCRIPTS_RTP_RUPG_CLEANUP	running application specific scripts with RTP_RUPG_CLEANUP flag!
CLEANUP	cleaning up system!
MIRROR_LOCAL_DISKS	enabling all disabled local mirrored disks on all nodes!
MIRROR_SHARED_DISKS	enabling all disabled shared mirrored disks on all nodes!
DELETE_UPGRADE_ENVIRONMENT	removing backup environment!
Other	Unknown state!

Tabela 10 – Mensagens para cada estado da aplicação a implementar.

Como a funcionalidade de *Rolling Upgrade* é fornecida pelo *middleware* por uma Java API, o SmartRSU foi desenvolvido em Java.

Como o *Patching Tool* foi desenvolvido em C++ foi preciso arranjar uma interface com o SmartRSU. Logo foi definado um *header file*, que segue no Anexo B. No corpo dessa classe C++ foi utilizado o Java Native Interface (JNI) como forma de utilizar as classes e métodos concretizados em Java.

A funcionalidade básica do SmartRSU é a seguinte:

1 – *Wrapper* implementado em C++, seguindo o *header file* no Anexo B, é chamado pela aplicação *Patching Tool* para começar um *Rolling Upgrade*.

2 – O *wrapper* invoca o JNI que cria uma *Java Virtual Machine* e cria o processo ControlRSU.

3 – A partir daqui, o processo ControlRSU, cria outro processo chamado ExecuteRSU que controla o RTP *Rolling Upgrade* directamente. Isto é feito para que o ControlRSU possa controlar e alterar o ambiente Java em que o RTP *Rolling Upgrade* é executado.

Esta funcionalidade encontra-se descrita na figura 26:

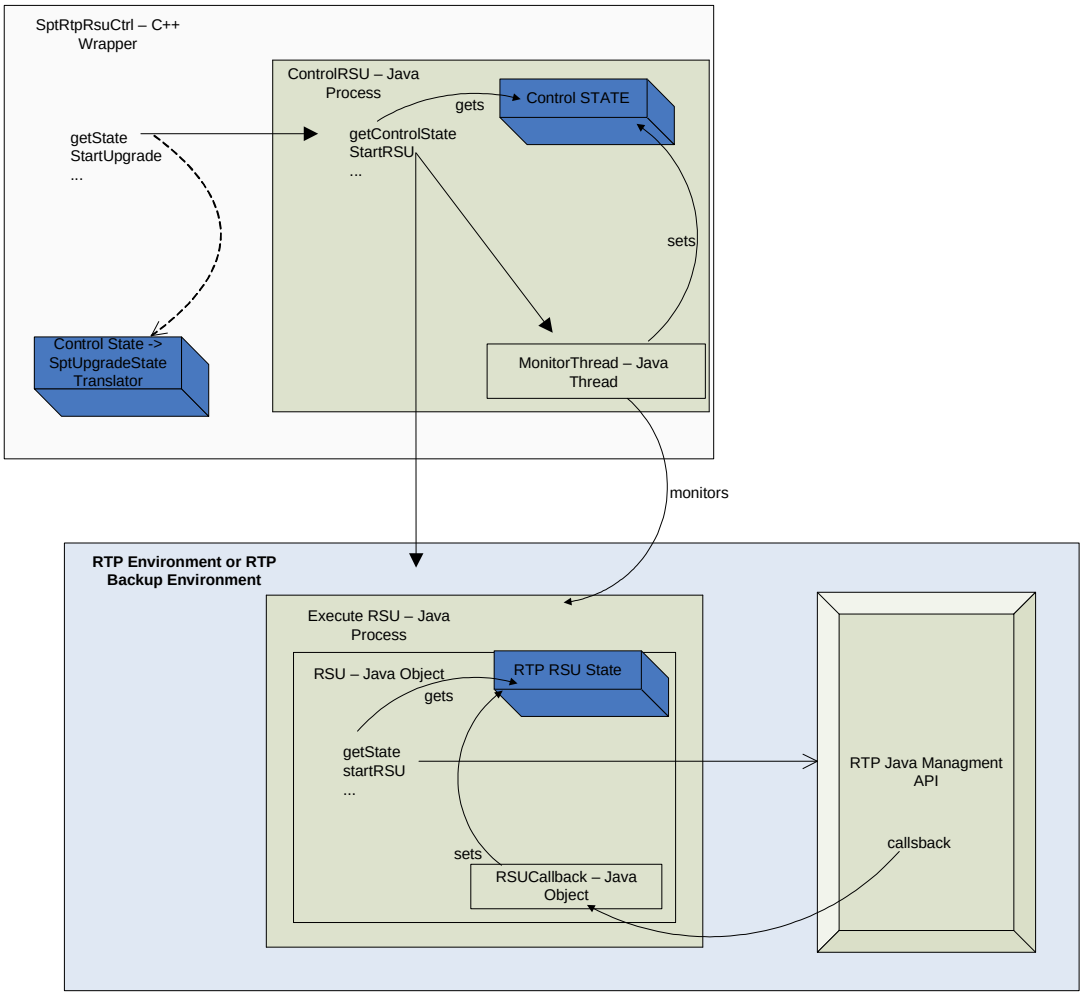


Figura 26 – Funcionalidade básica SmartRSU

Logo, as classes implementadas são:

- ControlRSU.cc
- ControlRSU.java
- MonitorThread.java
- ExecuteRSU.java
- RSU.java

4.2 Implementação

Durante o desenvolvimento desta *feature* ocorreu o seguinte problema no processo de desenvolvimento: foram escolhidos dois processos de desenvolvimento completamente diferentes para o *Patching Tool* e para o Assistant (GUI). O processo de desenvolvimento do *Patching Tool* começou primeiro que os GUIs e de forma independente.

Em consequência desta escolha, na altura em que o *Patching Tool* começou os seus testes de integração, o iNMC apareceu para testar as interfaces externas. Todas as interfaces externas (SOAP) foram testadas utilizando este GUI. começou Entretanto desenvolveu-se a *feature* toda e chegou-se à fase de testes de sistema. Descobriu-se nesta altura que a interface externa definida e implementada, em SOAP, pelo *Patching Tool* era incompatível com a biblioteca de interpretação de SOAP que o Assistant utilizava. O problema é que o Assistant é que era o GUI oficial da solução empresarial do hiQ e não o iNMC usado inicialmente Esta descoberta levou a alterações de última hora na interface externa do *Patching Tool* para tentar arranjar uma interface compatível com o *software* que eles tinham. Esta redefinição da interface externa resultou em atrasos que puseram em causa as datas determinadas no planeamento da *feature*. Para tentar minorizar o atraso foram feitas reuniões bidiárias para controlo do estado do desenvolvimento.

Com um esforço extra da parte das várias equipas, e da gestão do projecto todas as datas foram respeitadas com sucesso.

Todo o código fonte feito por mim está disponibilizado no Anexo C.

4.3 Testes de Componente

Para a concretização destes testes foi criado uma aplicação de teste, chamada a partir de agora de JavaDEBUG (figura 27) para testar todas as funcionalidades da componente:

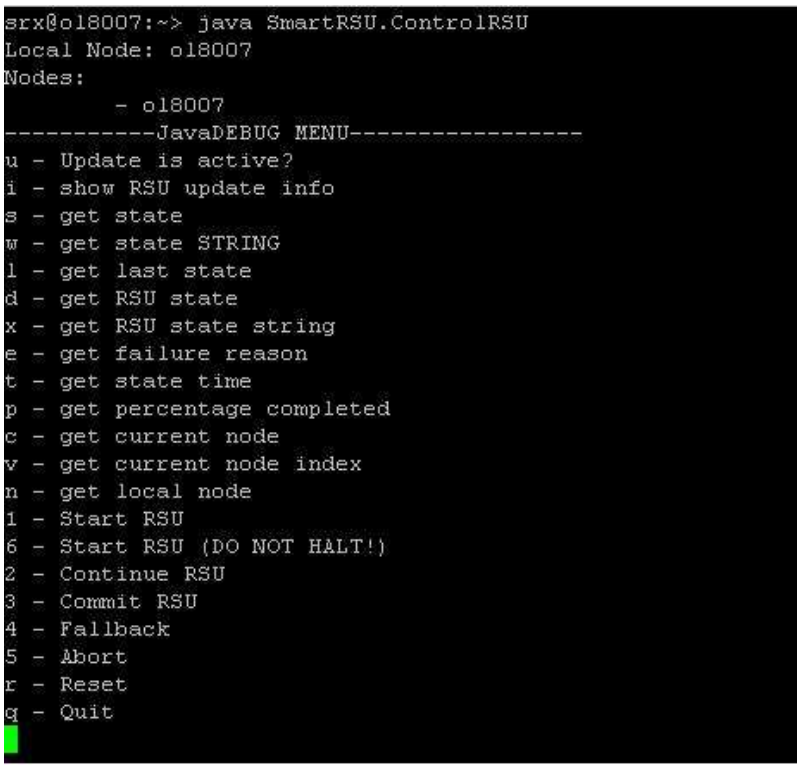


Figura 27 – Consola de testes de componente

Para a criação dos *master units* foi utilizada a ferramenta oficial "setup_rtp_unit".

Então, foram definidos os seguintes testes de componente:

1 – *Complete Upgrade*: Teste que consiste na concretização de um *Rolling Upgrade* completo, consoante os requisitos definidos na tabela 8. Este teste

comprova o correcto funcionamento do meu componente para concretizar um *Rolling Upgrade* do princípio ao fim.

2 – *Fallback after first node upgraded:* Teste que consiste em testar a funcionalidade de "fallback" após o *upgrade* do primeiro nó.

3 – *Fallback after second node upgraded:* Teste que consiste em testar o "fallback" após upgrade do Segundo nó.

Nas tabelas 11, 12 e 13 seguem as descrições detalhadas dos testes executados.

Complete Upgrade design steps		
Step Name	**Description**	**Expected Result**
Criar "MASTER UNIT".	executar "/unisphere/srx3000/srx/bin/setup_rtp_unit" como root	MASTER UNIT criado.
Executar JavaDebug.	Executar consola de *debugging*: java -cp SmartRSU.jar:$CLASSPATH SmartRSU.ControlRSU	Menu JavaDEBUG.
Começar *Rolling Upgrade*.	escolher opção 1 do menu e colocar caracteristicas do MATER UNIT.	*upgrade* começa sem problemas.
Verificar progresso do *Rolling Upgrade*.	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU.Callback.log. Cada novo estado ou erro é registado neste ficheiro.	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára depois do primeiro nó estar actualiazdo.
Continuar *Upgrade*	Escolher opção 2 do menu	*upgrade* continua normalmente.
Verificar progresso do *Rolling Upgrade*.	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU.Callback.log. Cada novo estado ou erro é	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára depois do segundo nó estar actualiazdo.

	registado neste ficheiro.	
Continuar *Upgrade*	Escolher opção 2 do menu	*upgrade* continua normalmente.
Verificar progresso do *Rolling Upgrade.*	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU. Callback.log. Cada novo estado ou erro é registado neste ficheiro.	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára depois de distribuir configuração pelos dois nós.
Commit Upgrade	Escolher opção 3 do menu	*Upgrade* começa a fase de finalização
Verificar progresso do *Rolling Upgrade.*	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU. Callback.log. Cada novo estado ou erro é registado neste ficheiro.	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára quando acabar.

Tabela 11 – Passos do teste de componente "Complete Upgrade"

Fallback_After_First_Node_Completed design steps		
Step Name	**Description**	**Expected Result**
Criar "MASTER UNIT".	executar "/unisphere/srx3000/srx/bin/ setup_rtp_unit" como root	MASTER UNIT criado.
Executar JavaDebug.	Executar consola de *debugging*: java -cp SmartRSU.jar:$CLASSPATH SmartRSU.ControlRSU	Menu JavaDEBUG.
Começar *Rolling Upgrade.*	escolher opção 1 do menu e colocar caracteristicas do MATER UNIT.	*upgrade* começa sem problemas.
Verificar progresso do *Rolling Upgrade.*	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU. Callback.log.	Estados do *Rolling Upgrade* progridem normalmente.

	Cada novo estado ou erro é registado neste ficheiro.	*Upgade* pára depois do primeiro nó estar actualiazdo.
Fallback Upgrade	escolher opção 4 do menu	*upgrade fallbacks*
Verificar progresso do *Rolling Upgrade.*	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU.Callback.log. Cada novo estado ou erro é registado neste ficheiro.	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára depois do *fallback* finalizar.

Tabela 12 – Passos do teste de componente "Fallback after first node completed"

Fallback_After_Seconde_Node_Completed design steps		
Step Name	**Description**	**Expected Result**
Criar "MASTER UNIT".	executar "/unisphere/srx3000/srx/bin/setup_rtp_unit" como root	MASTER UNIT criado.
Executar JavaDebug.	Executar consola de *debugging*: java -cp SmartRSU.jar:$CLASSPATH SmartRSU.ControlRSU	Menu JavaDEBUG.
Começar *Rolling Upgrade.*	escolher opção 1 do menu e colocar caracteristicas do MATER UNIT.	*upgrade* começa sem problemas.
Verificar progresso do *Rolling Upgrade.*	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU.Callback.log. Cada novo estado ou erro é registado neste ficheiro.	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára depois do primeiro nó estar actualiazdo.
Continuar *Upgrade*	Escolher opção 2 do menu	*upgrade* continua normalmente.
Verificar progresso do *Rolling Upgrade.*	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU.Callback.log.	Estados do *Rolling Upgrade* progridem normalmente.

	Cada novo estado ou erro é registado neste ficheiro.	*Upgade* pára depois do segundo nó estar actualiazdo.
Fallback Upgrade	escolher opção 4 do menu	*upgrade fallbacks*
Verificar progresso do *Rolling Upgrade*.	Verificar ficheiro de log: /log/JavaDEBUG.SmartRSU.Callback.log. Cada novo estado ou erro é registado neste ficheiro.	Estados do *Rolling Upgrade* progridem normalmente. *Upgade* pára depois do *fallback* finalizar.

Tabela 13 – Passos do teste de componente "Fallback after second node completed"

4.4 Testes de Integração

Para os testes de integração era preciso testar a minha interface com o resto da aplicação *Patching Tool*. Então foi utilizada uma ferramenta, criada para a integração, para testar as funcionalidades do *Patching Tool*, denominada SPT CLI (figura 28):

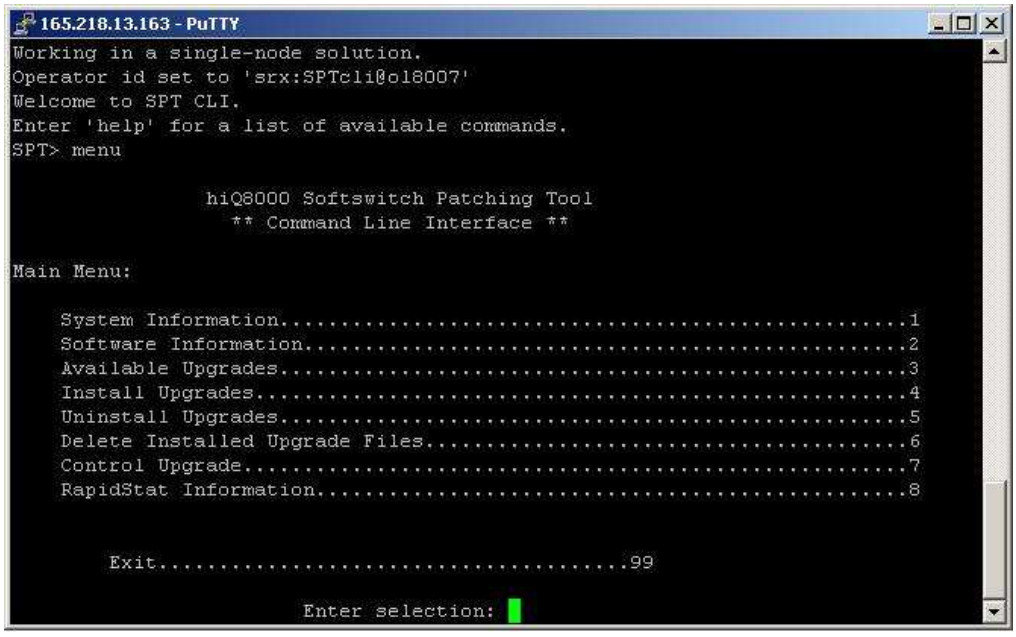


Figura 28 – Consola de testes de integração

Ao utilizarmos esta ferramenta para fazer *upgrades* estariamos efectivamente a testar a integração com a minha componente. Logo, foram definidos os seguintes testes de integração:

1 – *Simultaneos Upgrades Fail*: Este teste consiste em começar um *Rolling Upgrade* e depois tentar executar outro durante o decurso do primeiro. Este teste permite verificar se a aplicação detecta e avisa o utilizar que já existe um *upgrade* em curso.

2 – *Upgrade Complete:* Este teste consiste na execução de um *Rolling Upgrade* do princípio ao fim. Este teste permite verificar o funcionamento correcto, da interface entre a minha componente e o resto da aplicação, de todo o processo de *Rolling Upgrade*.

3 – *Upgrade Fails and Fallback:* Este teste consiste em introduzir um erro no processo de *Rolling Upgrade* de forma a que este falhe para se testar a funcionalidade de "fallback" após uma falha no processo.

4 – *Upgrade Fallback in continue:* Este teste consiste em testar a funcionalidade de "fallback" após o *upgrade* do primeiro nó sem ter havido qualquer erro no processo.

5 – *Uninstall Patchsets:* Este teste consiste em executar um *Rolling Upgrade* onde *patches* são removidos do sistema em vez de introduzidos e serve para verificar o correcto funcionamento da interface com o resto da aplicação neste caso específico. Este teste é referente ao requisito 1 da tabela 5.

Nas tabelas 12 a 16 seguem as descrições detalhadas dos testes executados.

Simultaneous Upgrades Fail design steps		
Step Name	**Description**	**Expected Result**
Começar *Rolling Upgrade*	Utilizar qualquer outro teste de integração para começar um *Rolling Upgrade*.	*Upgrade* começa.
Começar segundo	Equanto o primeiro *upgrade* está a	Segundo *upgrade* falha.

Rolling Upgrade.	corer, arranca-se um segundo *upgrade*.	
Executar comandos fora de contexto	executar "continue". "commit" e "fallback" no SPT CLI, enquando o *upgrade* está a correr.	Todos os comandos falham.

Tabela 14 – Passos do teste de integração "Simultaneous Upgrades fail"

Upgrade Complete design steps		
Step Name	**Description**	**Expected Result**
Colocar *patchset* a instalar.	copiar *patchset* a install em: /software/patch/LOAD_ID	patchset em: /software/patch/LOAD_ID
Executar SPT CLI	executar "sptCli"	SPT CLI.
Instalar *patchset*	executar comando "installps" dentro do SPT CLI.	lista de *patchsets* a instalar.
Escolher *patchset* a instalar.	Escolher um *patchset* do menu e carregar no ENTER.	*upgraede* começa.
Verificar progresso.	Verificar o progresso da isntalação executando o comando "status" dentro do SPT CLI	*upgrade* a progredir normalmente.
Continuar *upgrade*.	Quando o *upgrade* parar para "Node Checking", executar "continue" dentro do SPT CLI.	*upgrade* continua.
Verificar progresso.	Verificar o progresso da isntalação executando o comando "status" dentro do SPT CLI	*upgrade* a progredir normalmente.
Commit Upgrade	Quando o *upgrade* parar para "Waiting for commit", executar "commit" dentro do SPT CLI.	*ugpgrade* acaba normalmente.

Tabela 15 – Passos do teste de integração "Upgrade Complete"

Upgrade Fails and Fallback design steps

Step Name	Description	Expected Result
Colocar *patchset* a instalar.	copiar *patchset* a install em: /software/patch/LOAD_ID	patchset em: /software/patch/LOAD_ID
Executar SPT CLI	executar "sptCli"	SPT CLI.
Corromper *patchset*	ir para /software/patch/LOAD_ID , e alterar os pacotes (rpm or pkg) com uns privados que não possam ser instalados.	*patchset* corrompido.
Instalar *patchset*	executar comando "installps" dentro do SPT CLI.	lista de *patchsets* a instalar.
Escolher *patchset* a instalar.	Escolher um *patchset* do menu e carregar no ENTER.	*upgraede* começa.
Verificar progresso.	Verificar o progresso da isntalação executando o comando "status" dentro do SPT CLI	*upgrade* falha porque "MASTER UNIT" está corrompido.
Fallback upgrade	Quando o *upgrade* falhar, executar "fallback" no SPT CLI.	*upgrade fallbacks.*

Tabela 16 – Passos do teste de integração "Upgrades Fails and Fallbacks"

Upgrade Fallback in Continue design steps		
Step Name	**Description**	**Expected Result**
Colocar *patchset* a instalar.	copiar *patchset* a install em: /software/patch/LOAD_ID	patchset em: /software/patch/LOAD_ID
Executar SPT CLI	executar "sptCli"	SPT CLI.
Instalar *patchset*	executar comando "installps" dentro do SPT CLI.	lista de *patchsets* a instalar.
Escolher *patchset* a instalar.	Escolher um *patchset* do menu e carregar no ENTER.	*upgraede* começa.
Verificar progresso.	Verificar o progresso da isntalação	*upgrade* a progredir

	executando o comando "status" dentro do SPT CLI	normalmente.
Fallback upgrade	Quando *upgrade* parar para "continuar", executar "fallback" no SPT CLI.	*upgrade fallbacks.*

Tabela 17 – Passos do teste de integração "Upgrade Fallback in continue"

Upgrade Uninstall Patchsets design steps		
Step Name	**Description**	**Expected Result**
Colocar *patchset* a instalar.	copiar *patchset* a install em: /software/patch/LOAD_ID	patchset em: /software/patch/LOAD_ID
Executar SPT CLI	executar "sptCli"	SPT CLI.
Desinstalar *patchset*	executar comando "uninstallps" dentro do SPT CLI.	lista de *patchsets* a desinstalar.
Escolher *patchset* a deinstalar.	Escolher um *patchset* do menu e carregar no ENTER.	*upgraede* começa.
Verificar progresso.	Verificar o progresso da isntalação executando o comando "status" dentro do SPT CLI	*upgrade* a progredir normalmente.
Continuar *upgrade*.	Quando o *upgrade* parar para "Node Checking", executar "continue" dentro do SPT CLI.	*upgrade* continua.
Verificar progresso.	Verificar o progresso da isntalação executando o comando "status" dentro do SPT CLI	*upgrade* a progredir normalmente.
Commit Upgrade	Quando o *upgrade* parar para "Waiting for commit", executar "commit" dentro do SPT CLI.	*ugpgrade* acaba normalmente.

Tabela 18 – Passos do teste de integração "Upgrade Uninstall Patchsets"

Capítulo 5
Conclusões

Durante a execução deste projecto de engenharia de software foram concretizadas duas funcionalidades novas para o *softswitch* hiQ8000 da Siemens Networks. Essas duas funcionalidades foram:

- Layer 3 Cluster Interconnect: Este trabalho consistiu em modificar o transporte dos pacotes transmitidos entre os dois nós do *cluster* de *layer 2* para *layer 3*. Para esse efeito foi preciso modificar os *scripts*, utilizados durante o processo de instalação do sistema, para configurar as interfaces de rede e o *software* de *clustering* para suportar este novo cenário. Durante o desevolvimento desta funcionalidade foi preciso aprender *shell scripting* como também vários aspectos essenciais à administração de sistemas Linux como também o sistema específico utilizado.

- Remote Patching Capability: Este trabalho consistiu em criar uma aplicação que automatiza o processo de *upgrade*, já disponibilizado pelo *middleware* presente no sistema utilizado no projecto. Para o desenvolvimento dessa aplicação foi preciso utilizar Java, para comunicar com o dito *middleware* e assim controlar o processo de *upgrade*, e C++ para o resto da aplicação envolvente. Aqui foi preciso aprender a trabalhar com JNI para o interfuncionamento do Java e C++. Durante o processo de desenvolvimento desta aplicação foi também possivel aprender a importância dum processo de desenvolvimento adequado, correndo o risco de ter grandes impactos no esforço previsto (ver capítulo 4.2).

Durante o desenvolvimento destas duas funcionalidades também aprendi a seguir um processo de engenharia de *software* onde é necessário colaborar com outros elementos distribuídos pelo mundo fora, ou seja um processo de desenvolvimento colaborativo.

5.1 Trabalho Futuro

Existe uma componente corrente de manutenção das *features* implementadas. Ao longo da vida essas *features* vão sendo encontradas pequenas falhas que são corrigidas utilizando o processo de *patching* do projecto.

Em relação a outros trabalhos futuros, a serem concretizados por mim, para cada *feature:*

Layer 3 Cluster Interconnect:

Vai ser preciso implementar uma solução semelhante para o mercado das operadoras de telecomunicações. Ou seja, concretizar uma implementação semelhante ao que foi feito mas em ambiente Solaris e com os desafios que lhes são próprios.

Remote Patching Capability:

Vão ser feitas várias melhorias no código do *Patching Tool*. Foi encontrada uma questão de segurança que tem que ser endereçada o mais rapidamente possível e também vai ser preciso "portar" o código para ambiente Solaris.

Referências

1. (hiQ, 2005) hiQ8000 OverView, Siemens Networks, 2005
2. (Wood, 2005) J.M. Wood, *Suprpass hiq8000 Software Architecture,* Siemens Networks, 2005.

Anexos

ANEXO A – Informação contextual

Voz sobre IP

A voz tem sido transmitida sobre a rede tradicional telefónica desde 1878 e o negócio das chamadas a longa distância tem crescido cerca de 100 mil milhões por ano só nos Estados Unidos. O desejo dos consumidores e das empresas de reduzirem este custo, assim como o grande investimento na ultima década em redes IP, produziu um interesse substancial em transmitir voz sobre redes IP. A possível re-emergência de Internet Service Providers (ISP's) e outros fornecedores de serviços telefónicos pela internet é provavel que aumente a competição entre todos os fornecedores de serviços telefónicos. Operadoras tradicionais e recentes começam a oferecer conectividade Voz sobre IP a ambos consumidores residenciais e empresariais.

O VoIP envolve transmitir voz como pacotes de dados, usando o *Internet Protocol* (IP), onde a voz do utilizador é convertida num sinal digital, comprimido (codificado), e "desmantelado" numa série de pacotes. Esses pacotes são então transportados sobre uma rede IP pública ou privada e reassembladas e descodificadas no lado de recepção (ver figura 29).

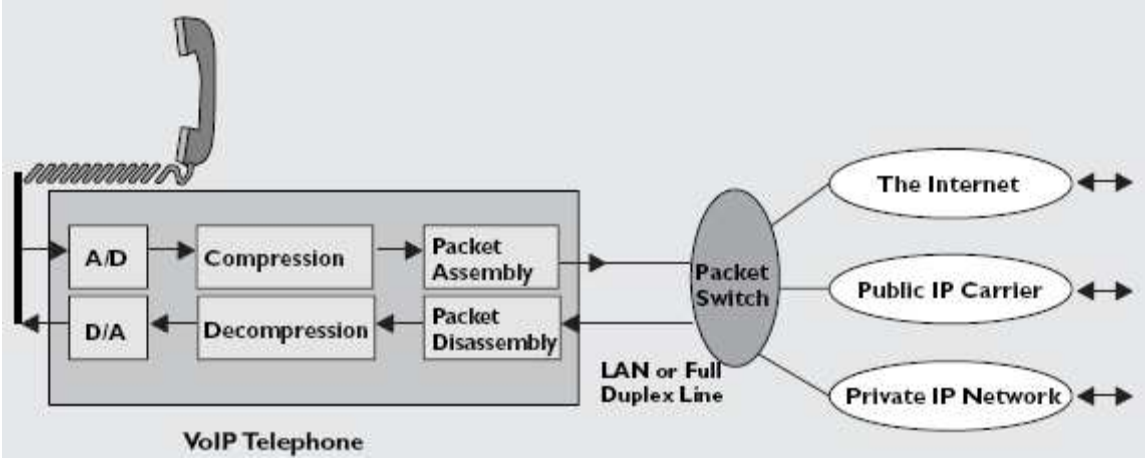


Figura 29 - Cenário VoIP

A Voz sobre IP, tem sido discutida, pelo menos, desde o príncipio dos anos 70 quando a ideia e tecnologia foi desenvolvida [1]. Independetemente disto, a tecnologia VoIP não estabeleceu um nicho comercial até aos meados de 1990. Este desevolvimento gradual pode ser atribuido à falta de insfrastruturas IP e pelo facto que chamadas comutadas em circuitos (*circuit switched*) eram, e ainda são, uma alternativa muito mais fiável, especialmente tendo em conta a falta de qualidade das primeiras chamadas VoIP. Em 1995, Vocaltec produziu o primeiro produto VoIP comercialmente disponível, requerendo que ambos os participantes numa chamada tivessem o *software* num PC como também acesso à internet. Infelizmente não permitia chamadas tradicionais pela PSTN.

Seguindo o crescimento rápido da Internet pública, especialmente a Web, durante os anos 90, e do investimento em infrastruturas de rede IP pelas empresas e operadoras, VoIP começou finalmente a ser uma alternativa viável para enviar voz sobre a PSTN. Um numero de factores estão a influenciar a adopção da tecnologia VoIP. Em primeiro lugar, o custo das redes *packet-switched* (comutação de pacotes) para VoIP podem ser tanto como metade do custo duma rede tradicional *circuit-switched* (comutação de circuitos) para transmissões de voz [2]. Esta redução de custos é o resultado do uso eficiente da largura de banda necessitando de menos *trunks* (conjunto de circuitos de voz) de longa distância entre *switches*. A rede tradicional *circuit-switched*, ou PSTN, tem que dedicar um canal *full-duplex* de 64Kbps para uma única chamada. Redes VoIP necessitam

aproximadamente de 14Kbps, pela compressão da voz e pela largura de banda, que só é usada quando qualquer coisa necessita de ser transmitida. Um uso mais eficiente da largura de banda quer dizer mais chamadas podem ser levadas num único *link* (circuito de voz), sem que a operadora tenha que instalar novas linhas ou aumentar a capacidade da rede; a tabela 19 compara voz sobre PSTN e sobre IP.

Concept	Voice over PSTN	Voice over IP
Switching	Circuit switched (end-to-end dedicated circuit set up by circuit switches)	Packet switched (statistical multiplexing of several connections over links).
Bit rate	64Kbps pr 32Kbps	14Kbps with overhead*
Latency	< 100ms	200–700ms depending on the total traffic on the IP network. Lower latencies possible with private IP networks.
Bandwidth	Dedicated	Dynamically allocated
Cost of access/billing	**Business customers.** Monthly charge for line, plus per-minute charge for long distance, cost of PBX, and other telephony equipment. **Residential customers.** Monthly charge for line, plus per-minute charge for long distance, cost of simple phone.	**Business customers.** Cost of IP infrastructure, Hybrid IP/PBX, and IP phones. **Residential customers.** Monthly charge for line, plus monthly charge for ISP, cost of computer, and other equipment.
Equipment	Dumb terminal (less expensive); intelligence in the network	Integrated smart programmable terminal (expensive); intelligence not in the network.
Additional features and services	Requires reprogramming or changes in the network design but fast enough to add if advanced intelligent networks (AIN) are in use.	Easy to add without major changes, due to flexible protocol support, but standards are needed for traditional user services.
Quality of service	High (extremely low loss)	Low and variable, but traffic is sensitive depending on packet loss and delay experienced.
Authorization and authentication	Only once when the service is installed	Potentially required, per-call basis
Regulations	Many at federal and state levels	Few yet, but regulatory uncertainty; future regulations may reduce the cost advantages of VoIP.
Network availability	99.999% up time	Level of reliability is not known.
Electrical power failure at customer premises	Not a problem; powered by a separate source from phone company.	Will have problems, as equipment may be down. Power from other sources is not easy to obtain.
Security	High level of security because one line is dedicated to one call.	Possible eavesdropping at routers.
Standards/status	Mature (simplified interworking among equipment from different vendors).	Emerging possible problems in interworking.

*Only when speaker is talking

Tabela 19- Voice over PSTN vs Voice over IP

Cenário típico hiQ8000: IP –> PSTN

Um tipo de cenário bastante usual da utilização de um hiQ8000 é um utilizador numa operadora cabo, que disponibiliza serviço VoIP, que faz uma chamada telefónica para um número que se encontra na rede PSTN (figura 30):

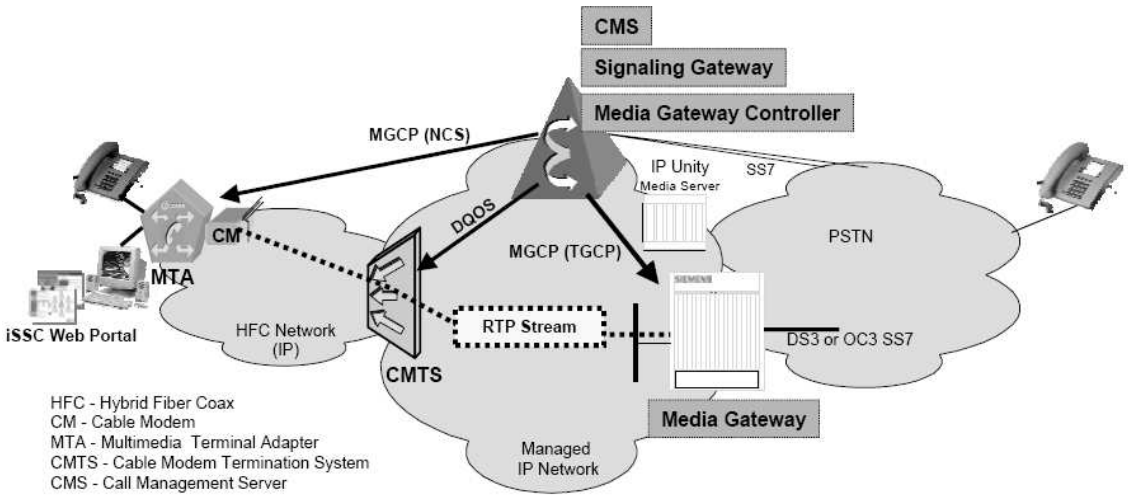


Figura 30 – Cenário cabo IP para PSTN

O utilizador ao fazer a chamada, o seu MTA comunica via MGCP sobre IP com o hiQ8000, que verifica se o utilizador está registado e se tem o serviço de chamadas activo e de seguida verifica o destino. Tendo em conta que o número é da rede PSTN, o hiQ8000 apercebe-se que tem que encaminhar a chamada duma rede cabo IP para a rede PSTN TDM, e envia uma mensagem de DQoS para o CMTS, para este reservar uma certa largura de banda entre ele e o MTA (caso especial para cenários em redes cabo), e uma mensagem MGCP para o media gateway, para este reservar um *stream* RTP do lado da rede IP e um *time slot* no lado da rede TDM, e uma mensagem ISUP para a rede de sinalização SS7. Entrentanto o telefone do destinatário toca, por causa da sinalização enviada pela rede SS7, e quando este o atende é enviado um sinal de volta ao hiQ8000 que indica que a chamada vai ser concretizada. Aquando a reserva do dito canal RTP e TDM, o *media gateway* comunica de volta com o hiQ800, via MGCP, indicando a reserva dos mesmos, e depois o hiQ8000 comunica com o MTA, via MGCP também, indicando que pode começar com a ligação RTP. Neste momento é concretizado um canal RTP, entre o MTA e o *media gateway*, onde é enviada a

voz, codificada por *codecs*, por IP. Depois o *media gateway* descodifica a voz que vem sobre IP e coloca a voz na rede TDM, que já se encontra préviamente preparada para receber a chamada por causa do sinal enviado na rede SS7. Quando algum dos participantes do telefonema terminar a chamada, é enviado uma sinalização para o hiQ8000 (MGCP ou ISUP) que por sua vez envia uma mensagem MGCP ao *media gateway* para desreservar os canais utilizados.

No cenário com telefones SIP o funcionamento é idêntico mas em vez de ser utilizado o protocolo MGCP, para toca de mensagens de sinalização entre o telefone e o hiQ8000, é utilizado o protocolo SIP. A voz será transportada via RTP sobre IP como no caso descrito anteriormente.

Media Gateways

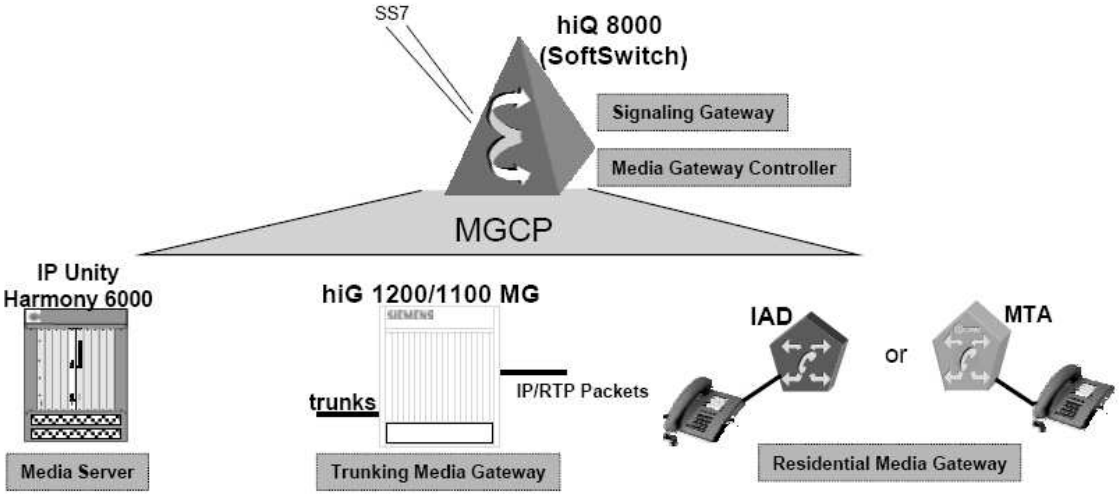


Figura 31 – Tipos de Media Gateways

Existem três tipos básicos de *media gateways* que o hiQ8000 pode controlar (ver figura 31):

Media Server – Oferece serviços de anuncios de chamadas e outras gravações de voz (ex: atendedor de chamadas), e de ponto de acesso para *conference bridges* (ex: conferências telefónicas com vários participantes). No laboratório utilizamos o Harmony 6000 da IP Unity.

Trunking Gateway – Oferece um ponto de convergência entre redes *circuit-switched* (TDM) e *packed-switched* (IP). No laboratório utilizamos o hiG1200.

Residencial Megia Gateway - Oferece controlo a uma linha analogica tradicional via MGCP, onde estará ligado um telefone tradicional. No laboratório utilizamos um conjunto de MTAs ligados a um CMTS.

Configuração hiQ

Em termos de configuração da plataforma, já depois de ela estar instalada, existem três aplicações para o efeito: iNMC, iSMC e CLI. Na figura 32 podemos verificar o método como estas aplicações comunicam com o hiQ8000.

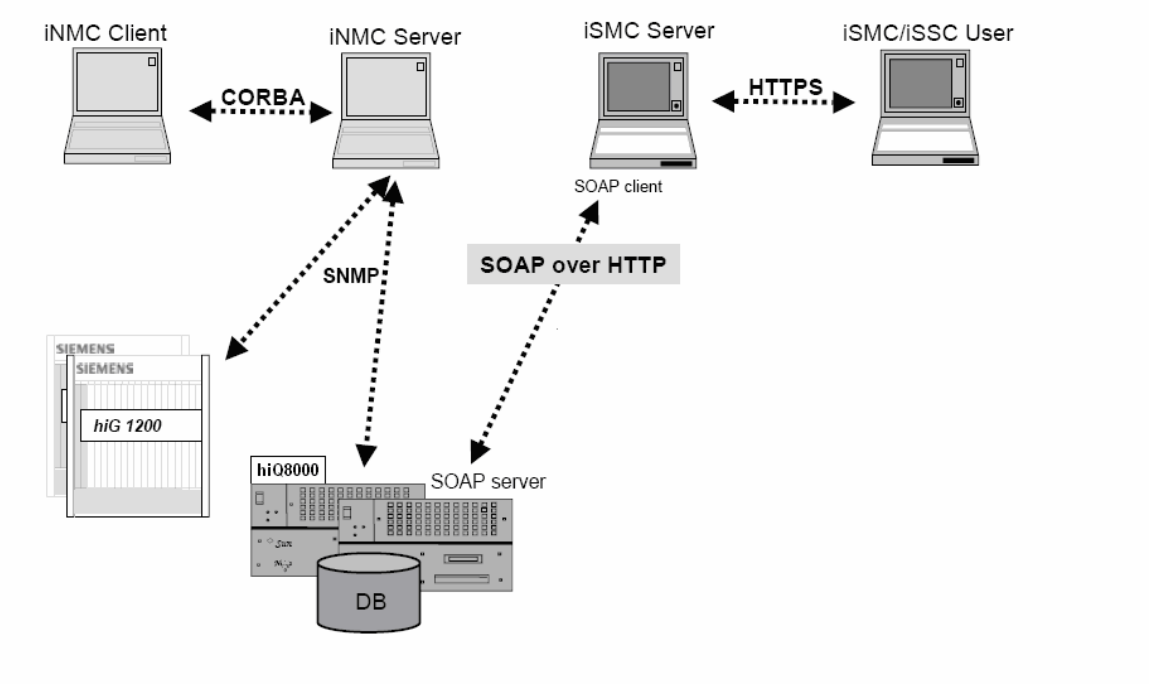


Figura 32 – iNMC e iSMC

O iNMC é usado para a configuração dos elementos de rede, como por exemplo a configuração dos *links* SS7, a configuração de *media gateways* e MTA's controlados pelo hiQ8000, o plano de numeração utilizado e grupos de números de telefones suportado pelo hiQ. Esta ferramenta consiste numa aplicação cliente e numa aplicação servidor que intercomunicam entre si utilizando a norma CORBA. Por sua vez, a aplicação servidor comunica com o hiQ utilizando gets e sets SNMP. Recentemente o servidor iNMC também está preparado para enviar algumas mensagens SOAP para o HiQ, numa forma idêntica ao servidor iSMC.

O iSMC trata da configuração de subscribers e de serviços, como por exemplo assignar um número de telefone disponível a um telefone SIP dum utilizador que se quer subscrever. Os utilizadores desta ferramenta serão pessoas com pouca

formação que só estão interessadas em tarefas relacionadas com a associação de serviços a números de telefone que já foram configurados pelo iNMC, logo a ferramenta foi desenhada para simplificar e restringir o seu campo de acção. A ferramenta consiste numa aplicação web residente num servidor Web, que por sua vez comunica com o HiQ por mensagens SOAP (XML over HTML).

Para além desta ferramentas, existe o CLI a ser utilizado para funções de manutenção e de resolução de problemas, e que dá acesso a todas as funcionalidades do hiQ8000, seja regras de segurança, seja upgrades, seja qualquer outra funcionalidade suportada por qualquer das outras duas ferramentas. O CLI é fornecido como parte do RTP *middleware*. e depois é extendido pela equipa aplicacional para fornecer serviços relacionados com o hiQ, e basicamente é uma aplicação de consola baseada em menus, mas que também aceita *scripts* para automatizar certos processos de configuração.

Também está em desenvolvimento uma ferramenta de gestão só para HiPaths denominada de Assistant que tem um comportamento similar ao iNMC mas é uma ferramenta Web.

Processo de Desenvolvimento

A Siemens Networks segue o seguinte modelo de desenvolvimento para criar novos produtos:

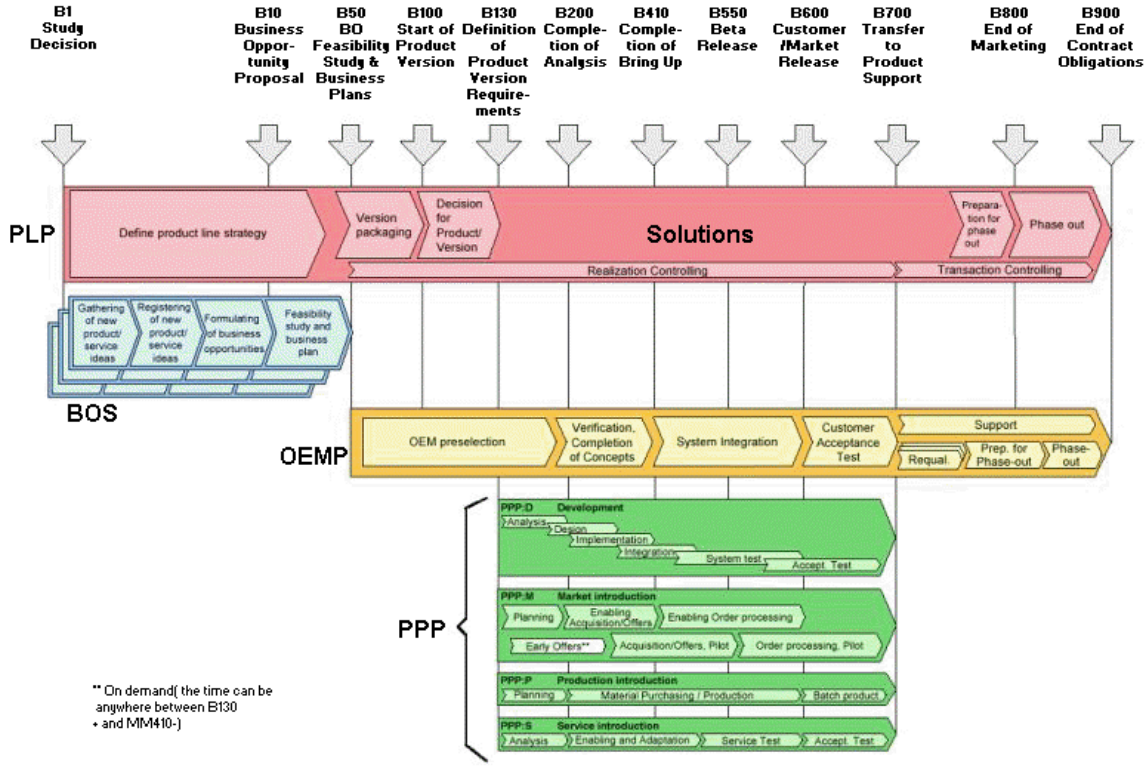


Figura 33 – Processo de desenvolvimento do produto

Na figura 33 conseguimos ver todo o processo de criação, desde à concepção da ideia em si (milestone B1) até ao fim do suporte do produto (milestone B900).

Podemos verificar pela figura 33 que, antes do desenvolvimento do *software/hardware* do produto (indicado como PPP na figura) existem variados passos, previamente tomados, relacionados com o planeamento e exequabilidade do produto proposto. Na fase do começo do desenvolvimento, o produto encontra-se na milestone B130, que requer que estejam definidos um conjunto de requisitos a seguir pelos engenheiros de desenvolvimento de *software*. Estes requisitos são definidos pelos *System Engineers* da Siemens Networks.

O modelo que mais nos interessa é o do desenvolvimento de *software*, ampliado e extendido na figura 34. O modelo contempla em simultâneo o desenvolvimento do Software, Hardware e Documentação, mas tendo em conta que o trabalho realizado é só em software vamos nos focar nas milestones do desenvolvimento de

software:

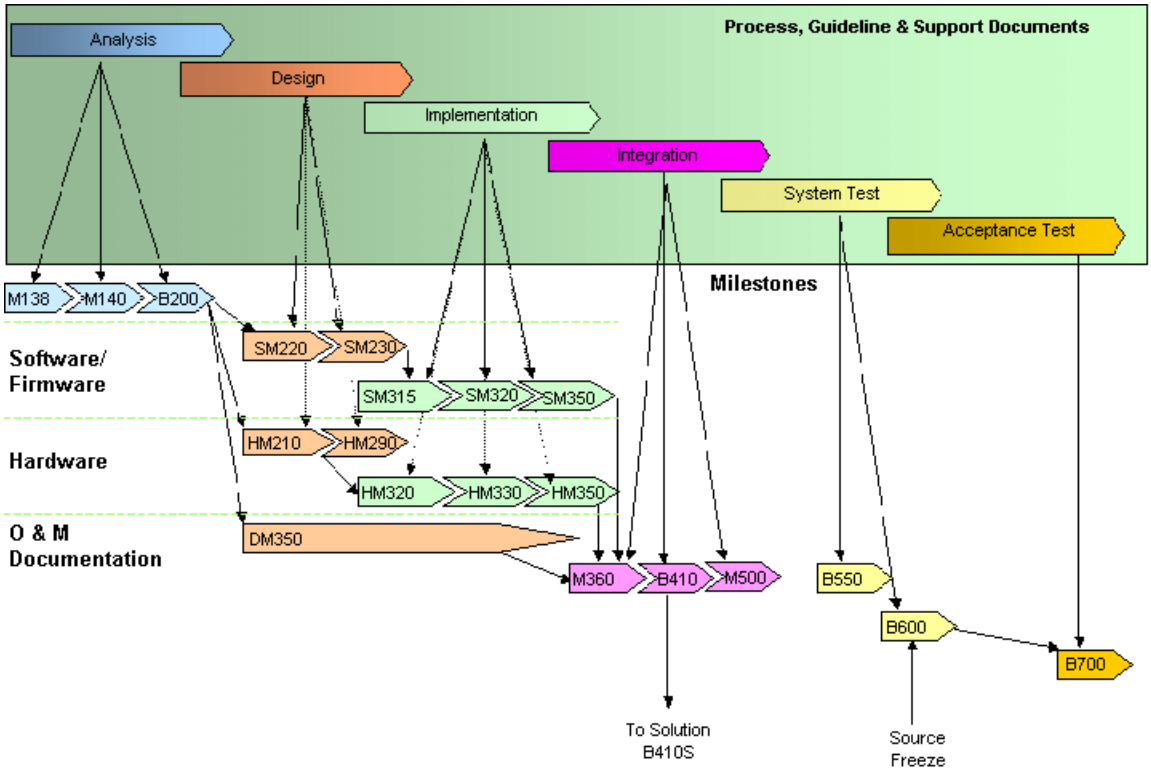


Figura 34 – Processo de desenvolvimento do *software*

B200 – Análise

SM230 – Design

SM320 – Implementação e especificação dos Testes de Componentes.

SM350 – Testes de Componentes

M360 – Especificação dos Testes de Integração

M500 – Testes de Integração

B600 – Testes de Sistema

B700 – Testes de Aceitação

É necessário referir que o envolvimento dos engenheiros de *software* vai desde a *milestone* B200 até M500. Os testes de sistema são concretizados por uma equipa própria denominada *System Testers*. De qualquer das formas, é preciso dar suporte à equipa de testes durante essa fase, para o caso de ser encontrado algum defeito no *software* implementado. Os testes de aceitação são executados por uma equipa especializada que acompanha a vertente comercial do produto. Essa equipa pode ser interna ou externa, dependendo se as funcionalidades a testar são personalizadas para alguma empresa ou gerais.

O projecto hiQ8000 está em constante desenvolvimento e cada release (versão) disponibilizada tem um ciclo de desenvolvimento próprio de análise, implementação e testes. Cada versão incorpora várias funciondalidades diferentes (*features*), que são os requisitos para a respectiva versão atingir a milestone M500.

Ferramentas

No âmbito deste projecto, utilizamos variadas ferramentas:

- LiveLink

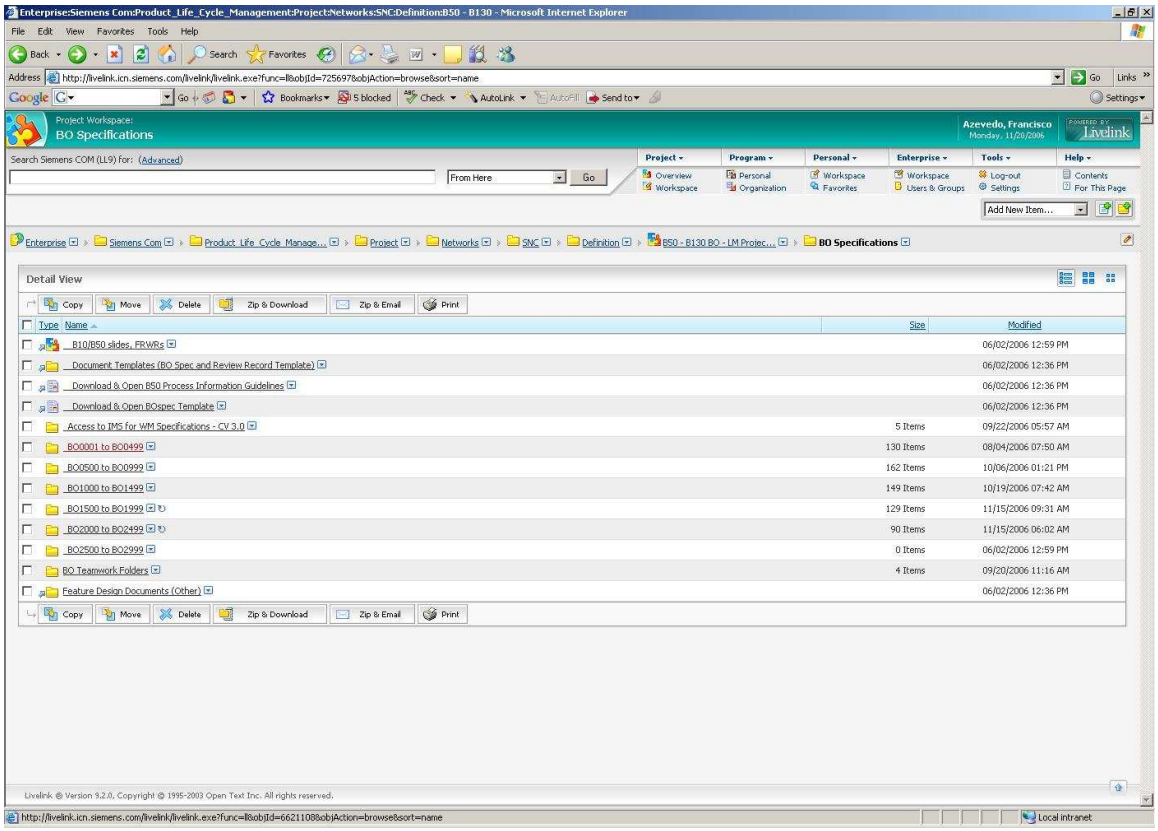


Figura 35 - LiveLink

Esta aplicação é uma ferramenta web, ou seja é acedida via um *web browser*, e é usada como um repositório partilhado de documentação. Qualquer documentação referente ao projecto e às features é colocada aqui.

- ClearCase:

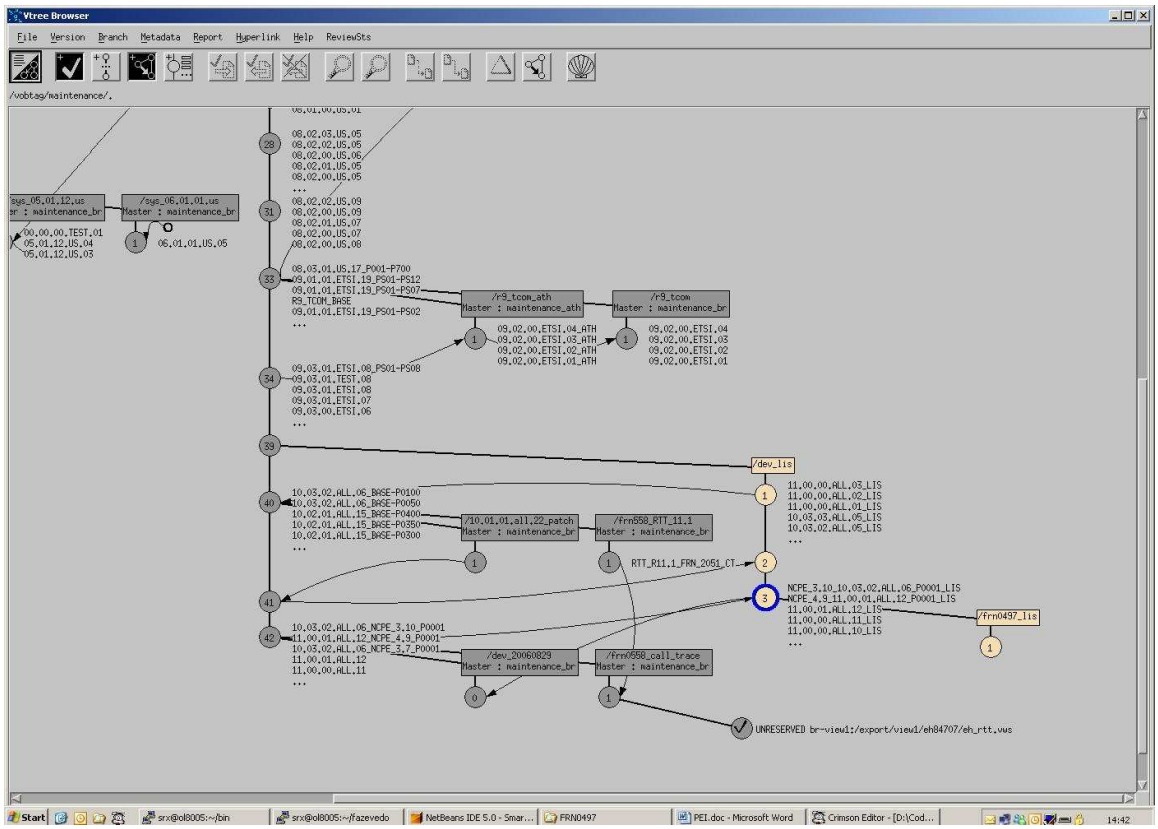


Figura 36 - ClearCase

Esta é uma aplicação de controlo de versões bastante avançada que permite o desenvolvimento paralelo de *software*. Esta aplicação tem duas componenetes: componente CLI e componente gráfica. Para facilitar a utilização utilizo sobretudo a versão gráfica, excluindo quando quero fazer mudanças avançadas. Esta aplicação também é utilizada para compilar o código fonte desenvolvido.

- TestDirector:

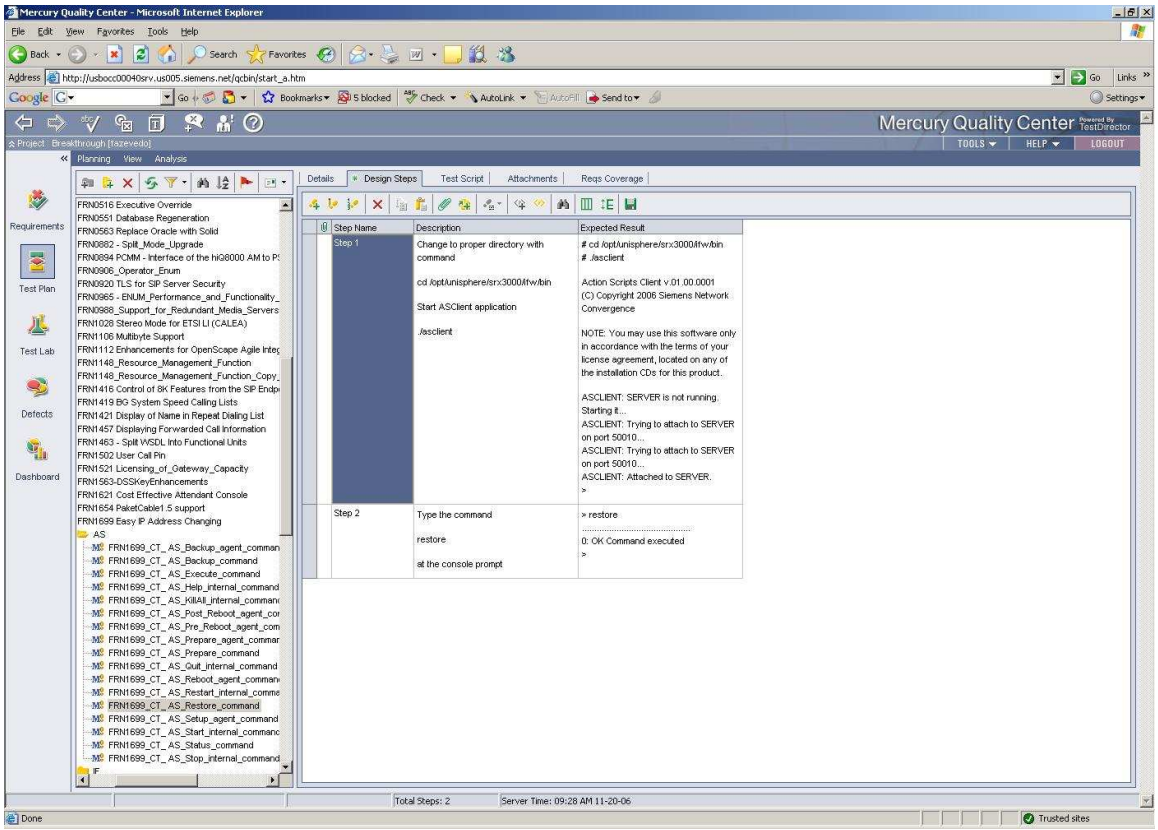


Figura 37 – Test Director

É uma aplicação *web* de suporte à fase de testes. A aplicação permite definir um conjunto de requisitos e depois definir um conjunto de planos de testes que cobrem esses mesmo requisitos. Existe ainda uma forma de indicar se os testes foram concretizados ou não, e se passaram com sucesso ou não. Esta informação encontra-se disponibilizada numa base de dados partilhada para qualquer colaborador do projecto.

- ClearQuest:

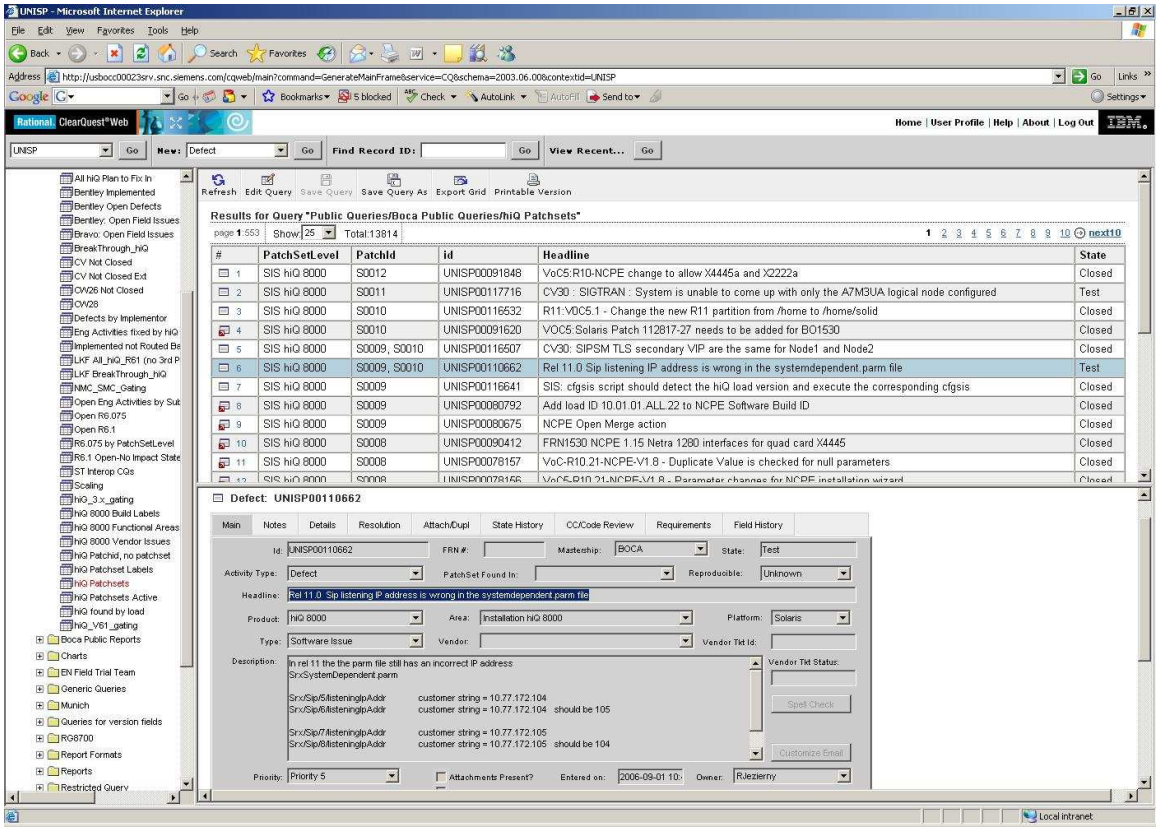


Figura 38 - ClearQuest

É uma aplicação, também *web*, para suporte ao rastreamento de *bugs*. Quando algum *bug* é encontrado acede-se a esta aplicação e abre-se um "*ticket*" novo descrevendo o problema e a forma como foi encontrado e indica-se a pessoa que está responsável por essa área. O responsável é contactado automaticamente por email, que indica a existência dum novo *bug*, e delega a resolução do problema a um dos colaboradores que está afectado a essa área, que utiliza esta aplicação para verificar a descrição do problema para o tentar recriar. Quando o problema é resolvido indica-se a resolução e que está resolvido na aplicação e depois, automaticamente, quem abriu o "ticket" é informado, por email também, para poder verificar se o problema realmente ficou corrigido. Se estiver, o autor do "ticket" fecha-o. Qualquer colaborador do projecto pode percorrer a base de dados de todos os *bugs* existentes utilizando esta aplicação.

- Netbeans

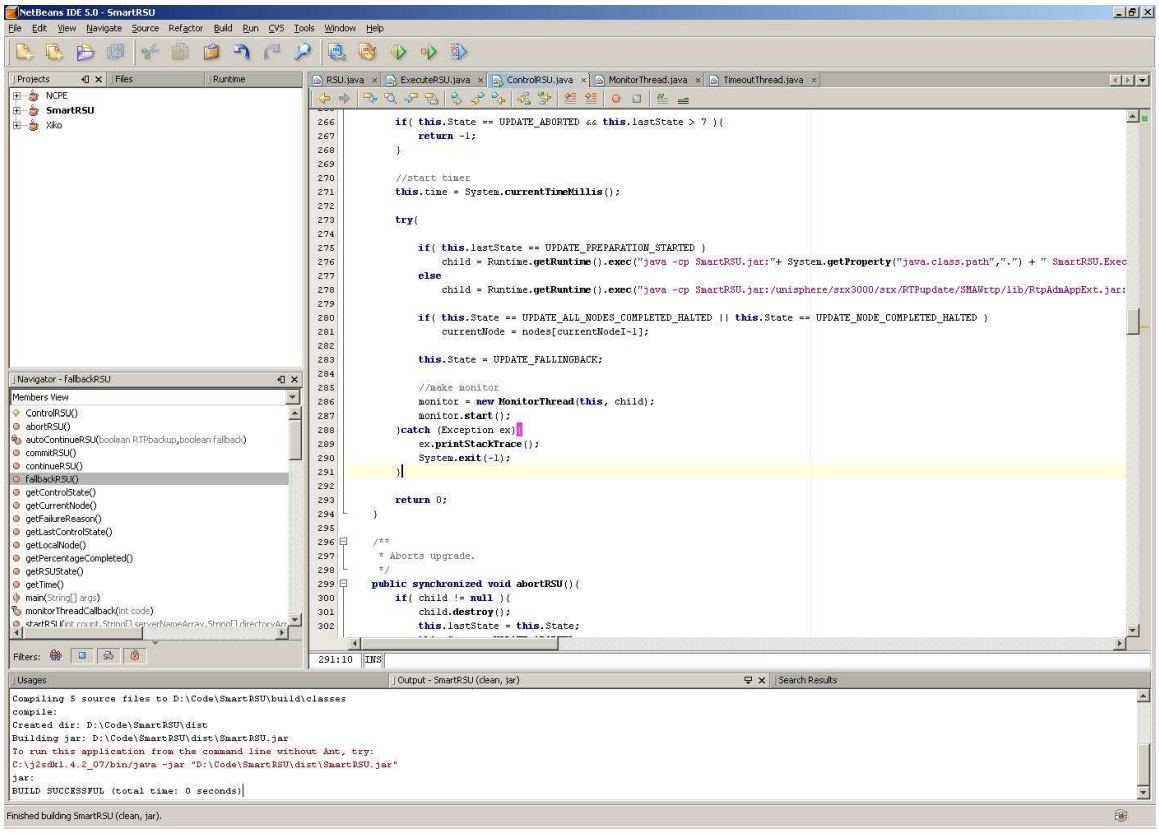


Figura 39 - NetBeans

Esta aplicação é usada como ferramenta para editar, escrever e compilar código Java. A ferramenta é globalmente conhecida, pois é disponibilizada pela própria Sun, e tem variadas funcionadades de grande utilidade. Como por exemplo *highlighting*, *auto-completion*, *refactoring*, etc.

Formação

Foram incluidos, na formação inicial do projecto, os tópicos descritos na tabela 20.

Tópico	Descrição
Internet Workings	Redes Layer 2 e Layer 3 (IP). QoS e VLAN Tagging.
VoIP	Rede PSTN e SS7. Protocolos VoIP.
RTP *middleware*	Visão geral e específica do *middleware* utilizado.
hiQ8000	Visão geral, configuração, manutenção e provisonamento.
ClearCase	Estudo da ferramenta utilizada para desenvolvimento paralelo.
ClearQuest	Estudo da ferramenta utilizada para "*bug tracking*".
Linux	Instalação, configuração e administração de sistemas Linux.
C++	Estudo da linguagem de programação.
Shell Scripting	Aprendizagem da linguagem interpretada.
CISCO	Configuração e manutenção dos *switchs/routers*.

Tabela 20 – Tópicos de formação

ANEXO B – SmartRSU header file

SptRtpRsuCtrl.h // HEADER FILE

```
#ifndef INCLUDE_SPTRTPRSUCTRL_H
#define INCLUDE_SPTRTPRSUCTRL_H
#include "SptEnum.h"
#include "SptTypes.h"


namespace SPT {
    class SptRtpRsuCtrl {
        public:
             //Constructor
            SptRtpRsuCtrl();
            //Desctructor
            ~SptRtpRsuCtrl();

             //Starts Upgrade with specified unit
            SptReturnCodeEnum StartUpgrade(SptString operatorId,
                                           SptString operatorAddress,
                                           SptString nodeName,
                                           SptString sourceDir,
                                           SptString unitName,
                                           bool globalRestart,
                                           bool haltOnEachNode);

            //Starts upgrade with multiple units
            SptReturnCodeEnum StartMultiUpgrade(SptString operatorId,
                                                SptString peratorAddress,
                                                UInt32 upgradeCount,
                                                SptString nodeName[],
                                                SptString sourceDir[],
                                                SptString unitName[],
                                                bool globalRestart,
                                                bool haltOnEachNode);



            //Continues Upgrade (after node checking)
```

```
            SptReturnCodeEnum ContinueUpgrade();

            //Fallbacks Upgrade (in case auto-fallback is viable)
            SptReturnCodeEnum FallbackUpgrade();

            //Aborts Upgrade (kills PID, dangerous)
            SptReturnCodeEnum AbortUpgrade();

            //Gets time since last state change (in seconds)
            UInt32                GetLastTimeChangedState();

            //Gets current node upgrading
            SptReturnString       GetCurrentNode();

            //Gets current SPT Upgrade state
            SptUpgradeStateEnum   GetUpgradeState();

            //Gets current Smart RSU Control tate (for debug)
            SptRsuCtrlStateEnum   GetRsuControlState();

            //Gets last Smart RSU Control state (for debug)
            SptRsuCtrlStateEnum   GetLastRsuControlState();

            //Gets percentage completed
            UInt8                 GetPercentageCompleted();

            //Gets string explaining where it failed (if it has failed)
            SptReturnString       GetFailureReason();

        private:
            //Operator Id
            SptString128      m_operatorId;

            //Operator address
            SptString32       m_operatorAddress;

    }; // end class SptRtpRsuCtrl

} // end namespace SPT
```

ANEXO C – Código fonte SmartRSU

Código retirado por razões de confidencialidade!