Dissertação de Mestrado

Utilização de Motores de Regras em Sistemas Informáticos

Mestrado Integrado em Engenharia de Comunicações

José Oliveira Nº 39259

27 de Outubro de 2007

Resumo

Nos dias que correm os requisitos de flexibilidade e adaptabilidade aos quais os sistemas informáticos estão sujeitos, são cada vez maiores. Neste cenário, os sistemas desenvolvidos recorrendo a programação convencional, muitas das vezes, tem dificuldades em acompanhar as constantes mudanças de requisitos exigidas. Uma forma de dotar os sistemas da flexibilidade de que necessitam, é integrando motores de regras nestes.

Neste trabalho foi desenvolvido um simulador de um sistema baseado em um motor de regras. O sistema representado pelo simulador, tem como objectivo possibilitar que um conjunto de nós em uma rede local (rede interior), consiga partilhar as suas conexões a Internet (rede exterior) com os seus pares. Sendo o motor de regras a componente utilizada por cada nó para decidir qual a conexão de um dos seus pares, este pretende para o seu acesso à rede exterior. As decisões as quais o motor de regras de cada nó chega são consequência directa do conjunto de regras que este utiliza. Regras estas que podem ser definidas e alteradas sem alterar o sistema, fornecendo assim a flexibilidade necessária à adaptação do sistema a diferentes requisitos.

Abstract

In our days the flexibility and adaptability requisites to which informatics systems are subjected, are becoming increasingly greater. In this scenario, the systems that are developed using conventional programming, sometimes, have difficulties in keeping up whit the constant change of the requisites. One of the possible ways, to give systems the flexibility they need, it's by integrating rule engines on them.

In this work we developed a simulator of a system that is based on a rule engine. The system that the simulator represents, allows for a group of network nodes in a local network (interior network), to share their connections to the Internet (exterior network), with their peers. In this system the rule engine is the component the each node uses to decide which connection of his peers it wants for his access to exterior network. The decisions to which, the rule engine arrives of each node arrives, are the direct consequence of set of rule that it uses. It is possible to define and alter the set of rules that a rule engine uses, without changing the system itself, this provides the needed flexibility, for the system to be adjusted to different requisites.

Conteúdo

Capítulo 1

Introdução

1.1 Enquadramento

No cenário de convergência actual, em que se esbatem as fronteiras entre o mundo das tecnologias da informação, das comunicações e do audiovisual, têm vindo a aumentar os requisitos de flexibilidade e adaptabilidade impostos aos sistemas informáticos e de comunicações que desempenham um papel cada vez mais central na teia global das tecnologias da informação e das comunicações.

Neste cenário, muitas vezes a programação convencional não é suficientemente flexível para acompanhar as constantes mudanças de requisitos. Assim sendo, uma alternativa possível é o recurso aos sistemas baseados em regras, normalmente conhecidos por motores de regras, que permitem resolver problemas complexos de uma forma rápida e simples e de uma mais fácil manutenção. A utilização de regras permite que peritos de um determinado domínio, mesmo sem formação técnica, possam definir e manter a lógica.

Hoje em dia os sistemas baseados em motores de regras são utilizados em diversas áreas, como por exemplo: Segurança de Redes, Telecomunicações, Detecção de Fraudes, Vendas Electrónicas, Bancos, Seguradoras, etc[3].

1.2 Objectivos

Neste trabalho será estudado e implementado um sistema de reconfiguração dinâmica de acessos de rede, baseado em motores de regras. A utilização de motores de regras neste trabalho tem como objectivo fornecer flexibilidade na definição das políticas de encaminhamento, de forma a possibilitar uma adaptação rápida e simples do sistema aos requisitos exigidos. O protótipo construído será testado num simulador modelado de acordo com os requisitos do sistema de distribuição de conteúdos multimédia da Xarevision.

O sistema de distribuição de conteúdos multimédia da Xarevision, é um sistema do tipo *Corporate TV/Digital Signage*, constituído por conjuntos de terminais multimédia (*displays* digitais associados a computadores) que se encontram espalhados em determinados espaços físicos (e.g. centros comerciais, supermercados, centros culturais, etc) onde o objectivo é que cada terminal reproduza conteúdos adequados ao local onde se encontra. Os conteúdos que cada um dos terminais multimédia reproduz localmente são obtidos através de um protocolo peer-to-peer a partir de um repositório central de conteúdos multimédia ou de um dos outros terminais no mesmo espaço, através da Wlan (rede interna) a qual todos os terminais do espaço pertencem.

Para efeitos de optimização de custos e desempenho, pretende-se que dentro de um conjunto de nós (terminais multimédia) em um determinado espaço físico, apenas um número muito reduzido de nós mantenha ligação com o a rede exterior (Internet), enquanto que os restantes nós deverão utilizar a rede interna para encaminhar o seu tráfego até a um dos elos actuais com a rede exterior, para satisfação das suas necessidades de rede.

O sistema deverá reagir dinamicamente as mudanças, quer nas opções disponíveis (ex. Wifi vs Rede Cablada vs 3G), quer em casos de falha dos terminais que tenham assumido o papel de *gateway*. Para cada um destes casos, todos os nós da rede deverão permanentemente monitorar o estado da rede e face à realidade percepcionada e de acordo com as regras do modelo de negócio, reconfigurar o seu próprio acesso dinamicamente e de forma colaborativa com os restantes elementos, por forma a obter uma boa relação custo/desempenho, onde a optimização do custo tem prioridade.

1.3 Estrutura da Dissertação

Esta dissertação é composta por 7 capítulos. O primeiro capítulo pretende enquadrar o trabalho e traçar, na generalidade, os objectivos a atingir. O segundo capítulo é um capítulo de cariz introdutório, que serve para familiarizar o leitor nas áreas chave que este trabalho envolve e também para discutir sobre qual o motor de regras a utilizar, no sistema a desenvolver. No capítulo 3 é apresentada a solução proposta para o sistema, onde primeiro é descrito o modelo conceptual genérico no qual a solução assenta e só depois é descrita a solução propriamente dita. Assim, no capítulo 4 é apresentado o simulador desenvolvido para testar a solução proposta. No capítulo 5 temos um conjunto de cenários de testes, onde é utilizado o simulador para testar e validar a solução proposta. Nos capítulos 6 e 7 temos as conclusões tiradas a partir do trabalho desenvolvido e algumas considerações sobre possíveis melhorias do sistema, incluindo a sua implementação fora de um ambiente simulado.

Capítulo 2

Fundamentos sobre Motores de Regras

2.1 Introdução

Um motor de regras é um sistema que usa regras para derivar conclusões[1].

Uma regra pode ser vista como uma instrução ou comando que é aplicada a determinada situação. Regras como "Não mascar pastilha elástica na escola", "Não correr com tesouras", e outras regras deste tipo são algumas das primeira regras que aprendemos.

Utilizando esta definição muito geral, pode-se pensar que todo o conhecimento que uma pessoa tem sobre o mundo poderá ser traduzido em regras. A experiência mostra que em muitos dos casos tal é possível.

Formalmente as regras podem ser representadas por declarações do tipo *if-then* como nas linguagens tradicionais de programação (como C/C++, Basic, Java, etc).

Podemos escrever uma regra (em pseudo código) sobre a proibição de mascar pastilha elástica na escola, da seguinte forma:

SE Estou na escola E Estou a mascar pastilha elástica

ENTÃO Deitar fora a pastilha
FIM

A parte "SE"das regras escritas desta forma é normalmente designada por *left-hand side* (*LHS*), predicado ou premissa e a parte do "*THEN*"designa-se por *right-hand side*(*RHS*), acções ou conclusões.

O domínio ao qual a regra é aplicada é toda a informação disponível na base de conhecimento do motor de regras.

2.2 Motores de Regras e Expert Systems

A Inteligência Artificial (IA) envolve estudar as formas do pensamento humano e aplicar este à computação. O aparecimento da IA permitiu-nos entender melhor, modelar e resolver problemas no domínio do mundo real. Os vários ramos da IA aplicam-se a diversas áreas, tais como o processamento de linguagem natural, visão por computador, robótica, resolução de problemas, planeamento, aprendizagem e *expert systems*. Os *expert systems* apresentam um bom exemplo do sucesso de IA na resolução de problemas complexos[6].

Os *expert systems* são sistemas que capturam o conhecimento de peritos (*experts*) humanos nas suas áreas de especialidade. Foram uma história de sucesso da investigação sobre inteligência artificial nos anos 70 e 80. Os primeiros *expert system* de sucesso foram construídos com regras baseadas em heurísticas, para diagnósticos médicos, engenharia e química. Um dos primeiros sucessos dos *expert systems* foi o MYCIN2 um programa para diagnosticar infecções bacteriológicas do sangue[1].

Os *expert systems* surgiram da abordagem de manipulação simbólica com o objectivo de modelar o conhecimento de peritos de determinada área. Os *expert systems* são sistemas baseados em conhecimento que emulam o pensamento de um perito de determinada área para resolver problemas significativos dessa área em particular. Estes são construídos para conterem o conhecimento de um perito em uma área bem definida, não sendo desenhados para resolver todos os problemas em geral. Os factos, observações e hipóteses são representados/manipulados como símbolos. Os *expert systems*

têm também a capacidade de se explicarem, conseguindo explicar como chegaram a uma conclusão A, bem como a razão pela qual uma conclusão B foi descartada. Este é um aspecto relevante, visto permitir aos utilizadores relacionarem-se melhor com o sistema ao entender como este funciona[6].

Na figura 2.1 é apresenta a estrutura geral de um *expert system*. O bloco central de um *expert system* é o motor de inferência (*inference engine*), este pode ser baseado em redes neuronais, algoritmos genéticos, motores de regras, etc. É importante distinguir motores de regras de *expert systems* visto não serem a mesma coisa. No entanto na documentação de alguns motores de regras [1] [2], estes são intitulados de *expert systems*. Os motores de regras por si só não são *expert systems*, mas podem ser uma parte destes no papel de motor de inferência (ver figura 2.1).

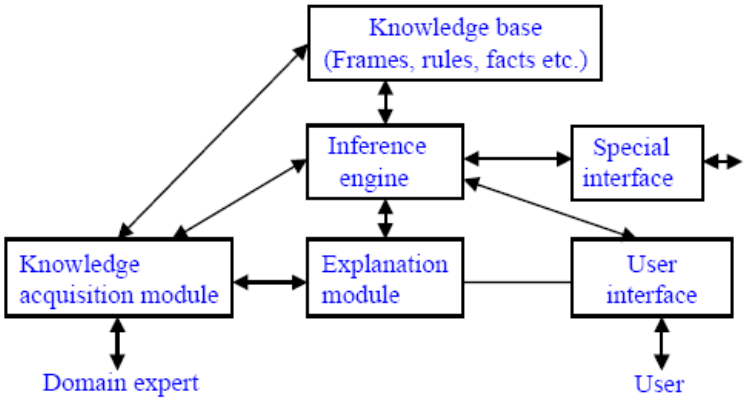

Figura 2.1: Estrutura Geral de um *Expert System*[6]

2.3 Algoritmos de Motores de Regras

Nesta secção serão apresentados os algoritmos mais importantes dos motores de regras. Servindo estes para os distinguir em termos de capacidades.

Vamos começar por apresentar os diferentes métodos de inferência (*forward chaining* e *backward chaining*) que podem ser utilizados pelos motores de

regras. Seguidamente será apresentado um tipo de lógica diferente da lógica clássica (lógica difusa ou *fuzzy logic*) que serve para trabalhar com conceitos vagos, imprecisos, incertos ou ambíguos (e.g. jovem, alto, bom, quente).

2.3.1 Forward Chaining

Para exemplificar o algoritmo de inferência *forward chaining*, podemos pensar em uma das formas que Sherlock Holmes teria para resolver um mistério. Imaginemos que o Sherlock Holmes tinha um conjunto de provas (um lenço, uma impressão digital e um corpo). Uma forma de resolver o mistério seria tirar conclusões a partir do conjunto de provas apresentado e adicionando as conclusões às provas e continuando a inferir conclusões até encontrar uma ligação entre as provas do crime. O método utilizado pelo Sherlock Holmes neste caso seria método de inferência *forward chaining*.

No método de inferência *forward chaining* uma regra é semelhante a uma declaração *if-then* numa linguagem procedimental (como C/C++, Basic, Java, etc), no entanto não é utilizada de uma forma procedimental. Nas linguagens procedimentais as declarações *if-then* são executadas em uma ordem definida pelo programador. No algoritmo *forward chaining* a parte *then* de uma declaração *if-then* é executada quando a parte *if* é satisfeita pelos factos (dados) inseridos, o que resulta em uma ordem de execução menos determinística que nas linguagens procedimentais, isto porque o motor de regras é quem decide a ordem pela qual as regras são executadas[1]. O *forward chaining* caracteriza-se por ser orientado aos dados (*data driven*) e reactivo.

A figura 2.2 demonstra através de um fluxograma o funcionamento do método de inferência *forward chaining*.

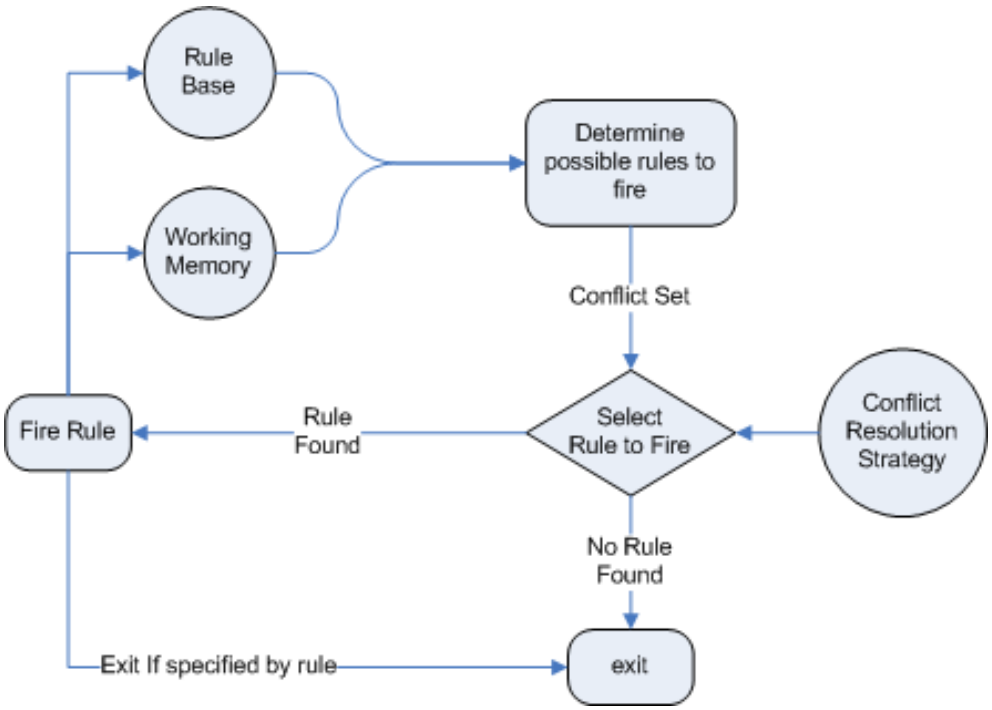

Figura 2.2: Forward Chaining[2]

2.3.2 Backward Chaining

Para exemplificar o algoritmo de inferência *backward chaining* podemos pensar em uma das outras formas que o Sherlock Holmes poderia utilizar para resolver o mesmo mistério da secção 2.3.1. O mistério poderia ser resolvido a partir da formulação de hipóteses sobre o que aconteceu, com base nas circunstâncias do crime e posteriormente procurando as pistas que suportassem as hipóteses formuladas. O método utilizado pelo Sherlock Holmes neste caso seria método de inferência *backward chaining*.

Alguns sistemas baseados em regras, notavelmente o Prolog e seus derivados, suportam *backward chaining*. Um sistema *backward chaining* é orientado ao objectivo (começamos com uma conclusão que o motor de regras tenta satisfazer). Em um sistema deste tipo, as regras são também declarações *if-then*, mas neste caso o motor tenta activamente chegar ao objectivo de uma regra (parte *then*). Se a parte *if* de uma regra apenas for parcialmente cumprida, o motor poderá concluir que provar o objectivo de uma outra regra (sub objectivo) poderá fazer com que a primeira regra seja totalmente cumprida, caso conclua isto, este vai tentar chegar ao objectivo da segunda regra, podendo esta ter os seus próprios sub objectivos (*sub goals*). Este processo continua até o objectivo inicial seja provado, ou não existam mais sub objectivos, concluindo-se assim que a regra não se verifica. Este comportamento é normalmente denominado de procura de objectivo (*goal seeking*). O *backward chaining* é caracterizado por ser orientado ao objectivo (*goal driven*)[1][4].

A figura 2.4 demonstra através de um fluxograma o funcionamento do método de inferência *backward chaining*.

Para melhor entender a diferença entre os algoritmos de *backward* e *forward chaining* vamos utilizar duas regras simples (ver figura 2.3)e aplicar os diferentes algoritmos a estas. Na figura 2.3 temos a ilustração da execução da cada algoritmo para as regras. O algoritmo *forward chaining* começa com os dados de **a**=1 e **b**=2 e usa as regras para derivar **d**=4. O algoritmo *backward chaining* começa com o objectivo de achar o valor de **d** e utiliza as duas regras para reduzir o problema a achar os valores a e b (*sub-goals*).

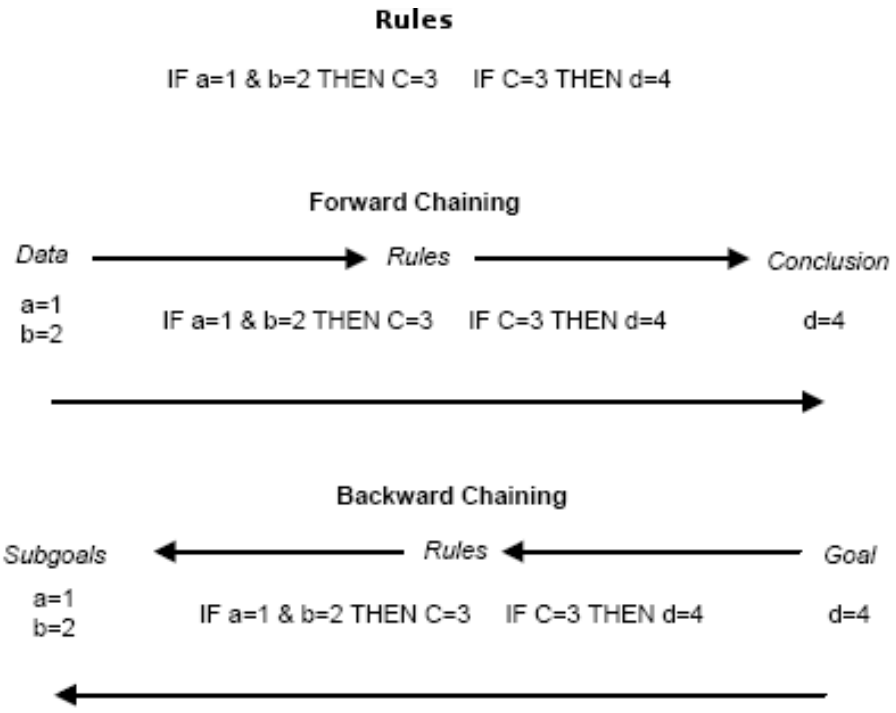

Figura 2.3: Diferença entre Forward e Backward Chaining[8]

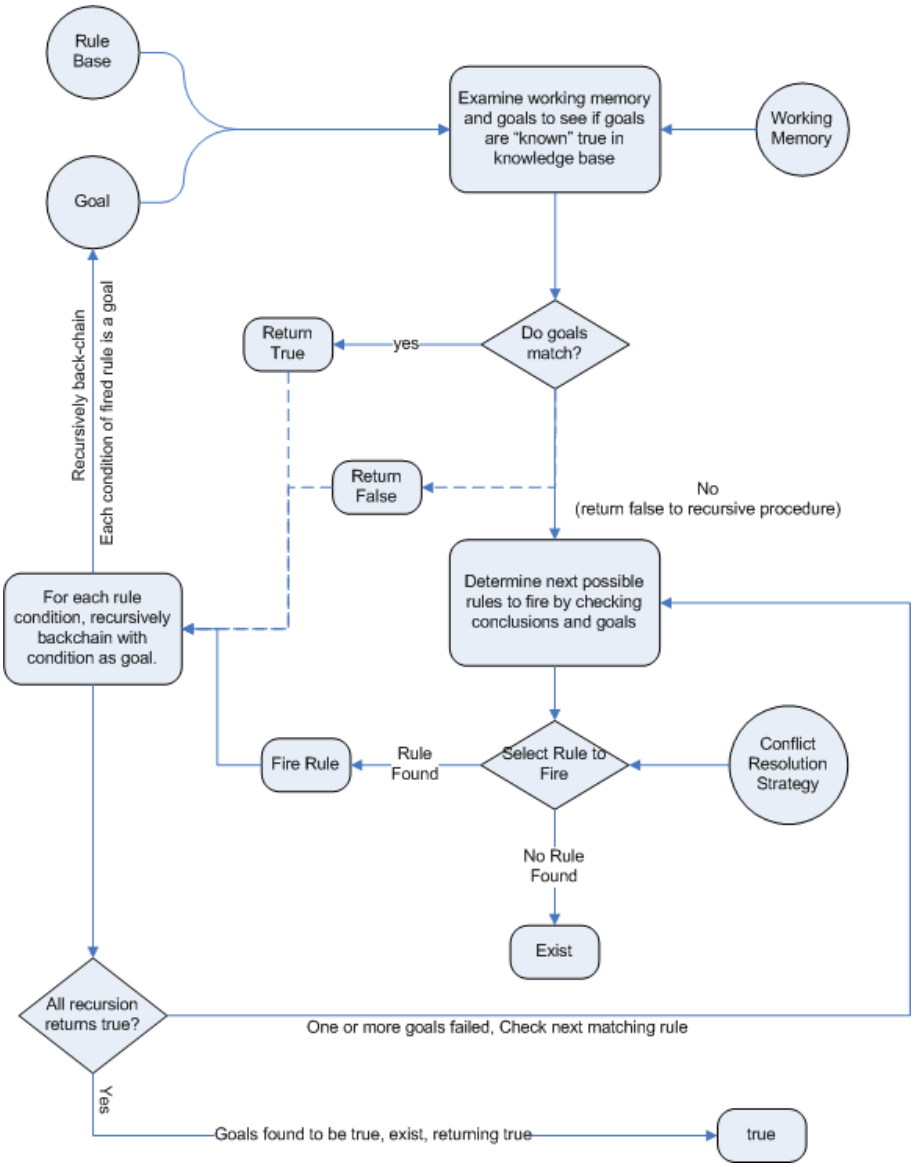

Figura 2.4: Backward Chaining[2]

2.3.3 Fuzzy Logic

No mundo existe muito conhecimento que não é claro, conhecimento que é vago, impreciso, incerto, ambíguo, não exacto ou de natureza probabilística. O raciocínio humano frequentemente envolve informação imprecisa (*fuzzy*), que possivelmente teve origem em conceitos humanos inerentemente não exactos e associações a experiências semelhantes em vez de idênticas. Em sistemas baseados em lógica clássica onde existem apenas dois valores possíveis (i.e. verdadeiro e falso), é difícil responder a algumas questões, visto estas não terem respostas completamente verdadeiras ou completamente falsas. Os humanos, porém conseguem normalmente dar respostas satisfatórias a este tipo questões[1].

A falta de clareza (*fuzziness*) ocorre quando o limite de uma informação não se encontra bem definido. Por exemplo, palavras como jovem, alto, bom, quente representam conceitos difusos. Não existe um valor quantitativo que defina o termo "jovem". Para algumas pessoas, alguém como a idade de 25 anos é considerado jovem, para outros 35 é considerado jovem. O conceito "jovem"não tem um limite bem definido. Na maior parte dos casos pode-se dizer que a idade de 1 ano é definitivamente considerada jovem e a idade de 100 anos definitivamente considerada não jovem. Porém a idade de 35 anos tem alguma possibilidade de ser considerada jovem, bem como de ser considerada não jovem, dependo do contexto em que esta a ser considerada. Uma idade tem sempre alguma possibilidade de ser considerada jovem bem como de ser considera idosa (ver as tabelas 2.1 e 2.2). A representação deste tipo de informação é tratável pela lógica difusa (fuzzy logic).

Idade	Grau de Associação
25	1.0
30	0.8
35	0.6
40	0.4
45	0.2
50	0.0

Tabela 2.1: Termo Difuso "jovem"

Idade	Grau de Associação
25	0.0
30	0.2
35	0.4
40	0.6
45	0.8
50	1.0

Tabela 2.2: Termo Difuso "idoso"

Contrariamente à lógica clássica (que trabalha com objectos cujo grau de associação a um conjunto pode ser claramente descrito como pertencente ou não pertencente) em lógica difusa um elemento pode pertencer parcialmente a varios conjunto com um certo grau de associação. O grau de associação ao conjunto é normalmente representado por valores de 0 a 1 como podemos ver nas tabelas 2.1 e 2.2 (onde 0 significa não pertence ao conjunto e 1 significando pertence completamente ao conjunto). Por exemplo na figura 2.5 é mostrado a distribuição do grau de associação das diferentes idades para o termo difuso jovem. Na figura 2.5 podemos ver que para as idades entre 0 e 25, o grau de associação é 1 (definitivamente jovem) e entre as idades de 25 a 50 o grau de associação decresce (idades com alguma possibilidade de serem consideradas jovens).

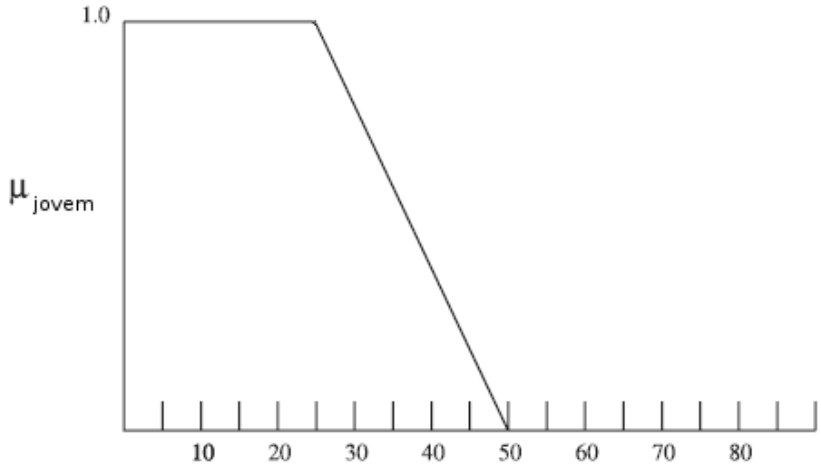

Figura 2.5: Distribuição do Grau de Associação ao Termo Difuso "jovem"[1]

A distribuição de um grau de associação de um conceito difuso como "de certa forma jovem" ou "muito jovem" podem ser obtida aplicando operações aritméticas especificas a distribuição de "jovem" (ver figura 2.5), denominados do modificadores (*modifiers*). Por exemplo, a distribuição do grau de associação do conceito difuso "de certa forma jovem" pode ser calculado pela raiz quadrada dos valores da distribuição de "jovem" (ver figura 2.6). Este modificadores são normalmente chamados de *edges*. Os *edges* possíveis incluem entre outros: não, mais ao menos, muito, pouco, extremamente.

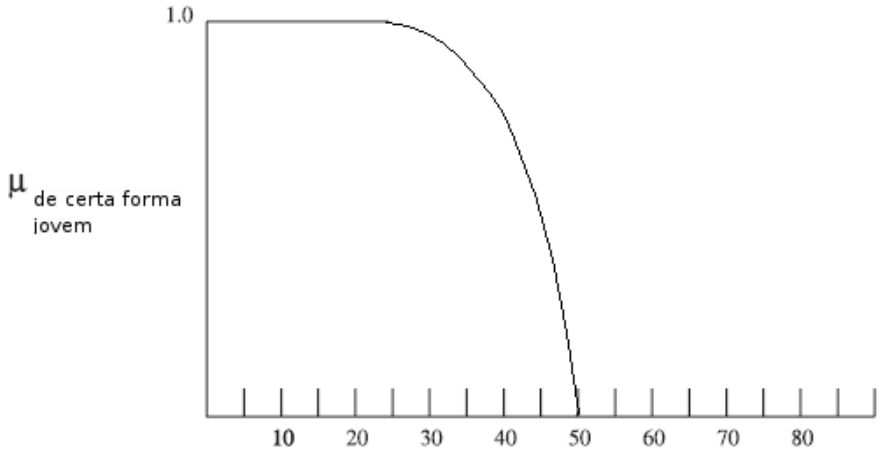

Figura 2.6: Distribuição do Grau de Associação ao Termo Difuso "de certa forma jovem"[1]

2.4 Principais Implementações de Motores de Regras para Java

Nesta secção serão apresentadas as principais implementações de motores de regras Java.

2.4.1 Jess

O Jess é um motor de regras comercial (excepto para uso académico) baseado no algoritmo de casamento (*matching*) RETE. Este é na sua base uma implementação Java de algumas das parte mais populares do CLISP, onde ao longo dos últimos anos foram adicionadas algumas funcionalidades, como por exemplo o suporte para *Java Beans*.

O Jess é o motor de regras que serviu de base para implementação de referência de motores de regras do Java (API JSR 94). A implementação de referencia é na verdade um *wrapper* que utiliza o motor de regras Jess [5]. A API JSR 94 é uma tentativa para normalizar as implementações de motores de regras em Java[5]. Apartir da versão 7 o Jess suporta nativamente a API JSR 94.

Este motor de regras suporta os métodos de inferência *backward chaining* (ver subsecção 2.3.2) e *forward chaining* (ver subsecção 2.3.1). Permite trabalhar com lógica clássica bem como com lógica difusa (ver 2.3.3)[1].O Jess pode ser utilizado de modo autónomo (*standalone*) na forma de uma *shell* ou integrado com o Java. No modo autónomo é também possível trabalhar com objectos Java.

2.4.2 JBoss Rules

O JBoss Rules é um dos mais directos concorrentes do Jess, devido ao seu continuo e rápido crescimento, em muito devido ao facto de ser totalmente *open source*. Este é semelhante ao Jess em muitos pontos, incluindo o algoritmo em que se baseia (RETE). Actualmente o JBoss Rules suporta também a API JSR 94 à semelhança do Jess.

Este suporta somente o método de inferência *forward chaining* (ver 2.3.1) e permite trabalhar apenas com lógica clássica, não suportando lógica difusa (ver subsecção 2.3.3)[2].

O JBoss Rules só pode ser utilizado integrado com Java contrariamente ao Jess, suportando também *Java Beans*.

2.4.3 JLog (Prolog)

O JLog é a implementação de um interpretador do Prolog escrito em Java. Este é totalmente *open source* [7]. Sendo o JLog uma implementação de Prolog para Java o método de inferência que este suporta é forçosamente *backward chaining* (ver 2.3.2)[8].

O JLog apenas pode ser utilizado integrado com Java. Para o suporte a utilização de objectos Java este inclui facilidades de mapeamento entre termos Prolog e objectos Java standard[7].

2.5 Analise Comparativa

Nesta secção vamos comparar dois dos mais conhecidos motores de regras para Java, sendo estes o Jess e o JBoss Rules (Drools). O objectivo desta analise será seleccionar um destes para a implementação do sistema de reconfiguração dinâmica de acessos de rede para a rede de distribuição de conteúdos multimédia da Xarevision.

2.5.1 Jess Vs JBoss Rules

Vamos começar por apresentar as semelhanças entre o Jess e o JBoss, passado de seguida a apresentar as suas diferenças na forma de vantagens e desvantagens de cada um.

Semelhanças

No texto que se segue são enumeradas as principais semelhanças entre os dois motores de regras.

- Suporte para a API JSR-94, sendo esta uma tentativa para normalizar as implementações de motores de regras em Java.
- Suporte do método de inferência *forward chaining*.
- Manual *on-line* mantido actualizado.
- Utilização do algoritmo RETE, para o casamento das regras com os factos.
- Abordagem *open source*, que faz com que programadores de todo o mundo estejam constantemente a encontrar e consertar problemas, sugerir novas funcionalidades e a escrever novo código (além de o Jess na verdade não ser *open source*).
- IDE baseado no eclipse, fornecendo este um bom ambiente de desenvolvimento e correção de problemas (*debugging*). Em ambos os casos o IDE fornece ainda uma representação gráfica da rede do algoritmo RETE para as regras definidas.
- Linguagem alternativa em formato XML.

Vantagens do Jess

No texto que se segue são enumeradas principais vantagens do motor de regras Jess.

- Possibilidade de ser utilizado como uma aplicação autónoma, na forma de uma *shell*, tal não é possível em JBoss Rules.
- Suporte ao uso de lógica difusa para a resolução de problemas onde os conceitos com que se trabalha são inerentemente imprecisos (ex: quente, frio, etc). Em JBoss Rules não existe suporte para este tipo de lógica.

- Suporte do método de inferência *backward chaining* (orientado ao objectivo), em adição ao método *forward chaining* (orientado aos dados).
- A licença do Jess é gratuita para uso académico.
- Existência de alguma documentação especifica sobre Jess, como o manual *on-line* e um livro[1].

Desvantagens do Jess

No texto que se segue são enumeradas principais desvantagens do motor de regras Jess.

- Não é *open source* contrariamente ao JBoss Rules, sendo necessário comprar uma licença para a sua utilização, excepto para uso académico.
- Sintaxe relativamente mais difícil de compreender que a do JBoss Rules em relação a linguagens como C/C++, Java e C#. Isto é relativamente importante visto que o Jess é um motor de regras para Java (além de ser possível utilizar de modo autónomo).

Vantagens do JBoss Rules

No texto que se segue são enumeradas principais vantagens do motor de regras JBoss Rules.

- O JBoss Rules ao contrário do Jess é completamente *open source*, o que significa que é gratuito e pode ser modificado/adaptado caso seja necessário.
- Além de suportar o algoritmo RETE para o casamento a semelhança do Jess, este suporta ainda o algoritmo LEAPS, este é ainda considerado experimental, visto ser muito recente[2].
- Sintaxe relativamente mais fácil de compreender que a do Jess para programadores habituados a linguagens como C/C++, Java e C#. No JBoss em vez de *if-then* temos *when-then*.

Desvantagens do JBoss Rules

No texto que se segue são enumeradas principais desvantagens do motor de regras JBoss Rules.

- Algumas funcionalidades, como por exemplo operadores, ainda se encontram por implementar (e.g. forall e accumulate)[2].
- Contrariamente ao Jess, o JBoss Rules não tem suporte para lógica difusa.
- Não suporta o método de inferência *backward chaining* (orientado ao objectivo), apenas suporta o método *forward chaining* (orientado aos dados). Em Jess existe suporta para os dois métodos.
- Não é possível utilizar o JBoss Rules de forma autónoma como acontece no Jess, assim sendo, apenas é sendo possível utilizar integrado com o Java.
- O manual *on-line* é praticamente a única fonte de documentação, este encontra-se em algumas partes um pouco incompleto e confuso devido a alguns erros ortográficos e semânticos.

2.5.2 Conclusão

O Jess e o JBoss Rules são ambos semelhantes em termos de desenho e funcionalidade, assim sendo, a escolha de um destes motores de regras, depende de alguns factores:

- Se o preço a pagar pela utilização de um destes motores de regras for relativamente importante então neste caso a melhor opção será o JBoss Rules visto este ser *open source*, o que significa que é gratuito.
- Se for preferencial uma abordagem *open source* onde o foco está no desenvolvimento, então mais uma vez a melhor opção será o JBoss Rules.

- Se o objectivo for um motor de regras mais maduro e com maior tempo de existência e um maior número de provas dadas, então neste caso o Jess será opção mais acertada.

- Se o projecto em questão necessitar de lógica difusa e/ou *backward chaining* então neste caso o Jess é a única opção visto que o JBoss Rules não ter suporte para tal.

No caso concreto do sistema de reconfiguração dinâmica de acessos de rede a implementar para a rede de distribuição de conteúdos multimédia da Xarevision, foi seleccionado o motor de regras JBoss Rules, visto que para resolver o problema ao qual este se propõem, não ser necessário recorrer a lógica difusa nem ao método de infêrencia *backward chaining* e também porque o preço a pagar é uma questão relativamente importante para a Xarevision, visto esta ser ainda uma empresa muito jovem, e assim sendo os seus recursos monetários são limitados.

Capítulo 3

Solução Proposta

Neste capítulo será descrita a solução proposta para a implementação do sistema de reconfiguração dinâmica de acessos de rede referido na secção 1.2.

Antes de passarmos à descrição da solução proposta em concreto, vamos descrever, na secção 3.1 o seu modelo conceptual genérico. A descrição do modelo conceptual tem como objectivo uma melhor compreensão da solução proposta. Uma vez descrito o modelo conceptual, passamos à descrição da solução propriamente dita na secção 3.2.

3.1 Descrição do Modelo

O modelo conceptual que passamos a apresentar nesta secção, tem como propósito descrever uma rede formada por um conjunto de nós pares (ver figura 3.1), interligados entre si através de uma rede local (rede interna). Estes nós, além do seu acesso à rede interna podem ainda ter um conjunto de acessos a uma rede externa (Internet). Os nós na rede interna são considerados pares entre si, visto compartilharem de uma mesma configuração.

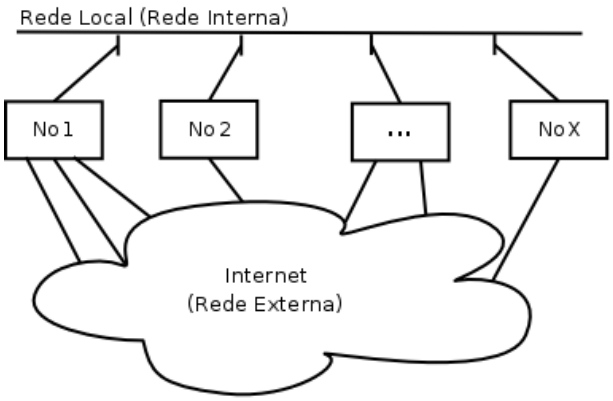

Figura 3.1: Rede de Nós Pares

Um nó, para o acesso à rede externa utiliza um *gateway*. Cada nó pode apenas ter um *gateway*, mas um *gateway* pode servir vários nós em simultâneo (ver figura 3.2). Um *gateway* é ele próprio um nó par, que pode ser inclusive o próprio nó. No entanto para que um nó possa ser considerado um *gateway*, este deve de ter uma interface com acesso a rede interna e uma ou mais interfaces com acesso a rede externa.

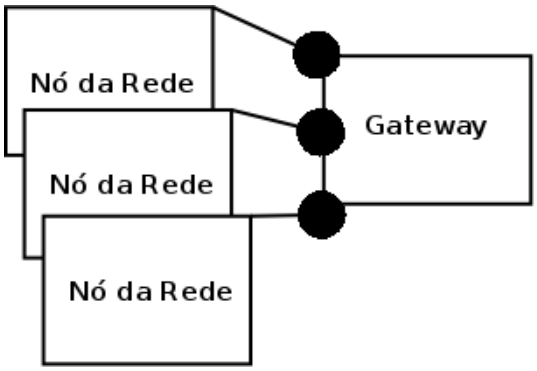

Figura 3.2: Nós da Rede e o *Gateway*

Cada nó par na rede interna (ver figura 3.1), é constituído por um identificador (que identifica o nó univocamente, em relação aos seus pares) e um número variável de interfaces de rede (ver figura 3.3).

Figura 3.3: Nó da Rede

Por sua vez as interfaces de rede de cada nó são constituídas por um conjunto de campos que as caracterizam (ver figura 3.4).

Figura 3.4: Interface de Rede

Para uma melhor compreensão dos campos que caracterizam uma interface de rede, vamos passar à descrição de cada um destes em pormenor.

- **Up:** Este campo indica se a interface de rede se encontra "activa" (*up*) ou "desactiva". Os valores que o campo pode tomar são do tipo boleano, onde "verdadeiro"significa "activa" e "falso" significa "desactiva".

- **Alias:** Este campo indica um nome alternativo para a interface (ex: 3G, Wifi, Adsl, ...). O objectivo deste campo, é uma mais fácil identificação do tipo de interface em questão, por parte do utilizador. A atribuição de um valor a este campo é opcional, e o valor atribuído, não necessita de ser único, no nó onde se encontra.

- **If:** Este campo representa o nome de uma interface de rede, nome este, que identifica a interface univocamente no nó onde se encontra.

- **Ext:** Este campo indica se a interface tem acesso a rede interna ou a rede externa (ver figura 3.1). Os valores que o campo pode tomar são do tipo boleano, onde "verdadeiro" significa "interface com acesso a rede externa" e "falso" significa "interface com acesso a rede interna". É de notar que cada nó apenas pode ter uma interface para a rede interna (Ext=falso), podendo ter várias para a rede externa (Ext=verdadeiro).

- **Ip:** Este campo representa o endereço do Internet Protocol v4 (IPv4) da interface de rede em questão.

- **Lb Total Disponível:** Este campo é um valor real positivo que representa a largura de banda total disponível em Megabits, que uma interface tem reservada para o papel de *gateway*. É de notar que este campo representa apenas uma parte da largura de banda total de uma interface.

- **Lb Disponível:** Este campo é um valor real positivo que representa em Megabits a porção da "Lb Total Disponível", que uma interface de rede dispõem em dado instante do tempo.

- **Custo:** Este campo é um valor real positivo que representa o custo monetário em Euros de cada unidade de tráfego que passa pela interface de rede. A unidade de tráfego considerada é o Megabyte. É de notar que este campo apenas tem significado no caso em que o campo "Ext" tem um valor verdadeiro (que significa, interface com acesso a rede exterior), caso contrario o valor não tem significado e deverá tomar o valor zero.

3.2 Descrição da Solução

A solução encontrada para o sistema de reconfiguração dinâmica de acessos de rede referido na secção 1.2, enquadra-se no modelo conceptual genérico descrito na secção anterior. Assim sendo tudo o que foi descrito para o modelo também se aplica à solução que aqui será descrita. Está previsto que a solução proposta seja implementada utilizando a linguagem de programação Java.

A descrição da solução divide-se em duas partes, na primeira parte é exposta a arquitectura do sistema, descrevendo-se as diferentes partes que a compõem (subsecção 3.2.1), passando-se na segunda parte à descrição do algoritmo que implementa as políticas de encaminhamento dos nós pares (subsecção 3.2.2).

3.2.1 Arquitectura da Solução

A figura 3.5 representa a arquitectura geral da solução proposta. Na figura podemos identificar um conjunto de nós ligados via uma rede local (rede interna) a um servidor. Este servidor tem como função, armazenar informação

sobre todos os nós pares actualmente na rede interna. O servidor além de armazenar a informação, também distribui esta, a cada um dos nós. A informação sobre um determinado nó no servidor é adicionada pelo próprio nó.

A informação que cada um dos nós pares na rede interna obtêm do servidor, é utilizada por estes para escolher o seu *gateway* de acesso a rede exterior.

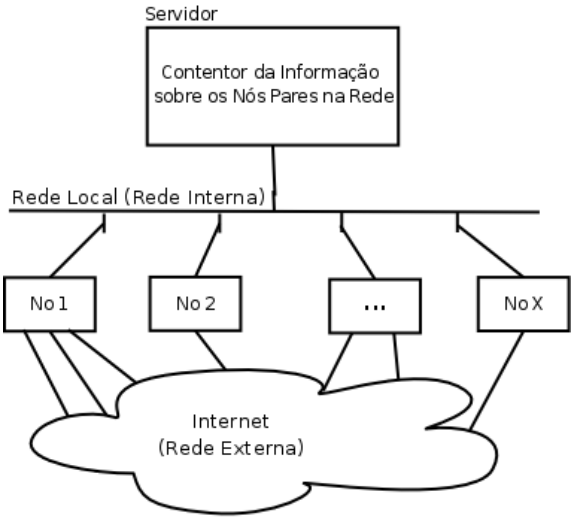

Figura 3.5: Arquitetura Geral

A tecnologia utilizada para a implementação do servidor que armazena e distribui a informação dos nós que se encontram na rede é a plataforma de *middleware* Java, EQUIP2 (ver figura 3.6). No centro desta tecnologia encontra-se a noção de *dataspace*, sendo este um contentor lógico de dados. Este pode ser visto como uma base de dados ou um *tuplespace*. No entanto um *dataspace* comporta objectos de linguagens de programação em vez de tabelas e linhas (base de dados relacional) ou tupulos (*tuplespace*). Esta integra também um sistema de notificação das alterações dos objectos, que é comparável a um sistema de distribuição publicador/subscritor[9].

Figura 3.6: Contentor da Informação do Nós

Para a interface com o *dataspace* (servidor), cada um dos nós utiliza a API Java disponibilizada pelo EQUIP2 (ver figura 3.7). Esta API possibilita aos nós partilharem o seu estado no *dataspace* e receberem notificações deste, sobre o estado dos outros nós. O estado partilhado por cada nó da rede no *dataspace* utilizando a API, resume-se a um objecto para armazenar a informação sobre o acesso do nó à rede exterior e um conjunto de objectos que representam as interfaces de rede do nó segundo a estrutura definida na secção 3.1 (ver figura 3.4).

Além do bloco de interface com o *dataspace*, existe ainda um outro bloco que faz parte da constituição de um nó (ver figura 3.7). Este outro bloco é o motor de regras para Java, JBoss Rules, apresentado na subsecção 2.4.2, e escolhido para a constituição dos nós da rede na subsecção 2.5.2. A função deste bloco, neste contexto, é utilizar a informação sobre as interfaces de

rede de todos os nós, contida no *dataspace* para decidir (segundo o algoritmo na subsecção 3.2.2), qual a melhor interface a utilizar para o acesso à rede exterior, escolhendo assim indirectamente, o *gateway*. É de notar que um *gateway* pode ter várias interfaces com acesso a rede exterior.

Figura 3.7: Nó da Rede

3.2.2 Algoritmo de Selecção do Gateway

O algoritmo que vamos descrever nesta subsecção, é o algoritmo encontrado para implementar as políticas de encaminhamento, onde o objectivo é achar uma boa relação custo/desempenho para o tráfego com destino ao exterior da rede de nós pares onde a optimização do custo é prioritária.

Antes de apresentar o algoritmo é importante referir que cada nó da rede tem dois parâmetros que caracterizam as suas necessidades de largura de banda (desempenho), que são os seguintes:

- **Largura de Banda Desejada:** Parâmetro que indica a largura de banda que um nó gostaria ter disponível, para satisfação plena das suas necessidades de acesso à rede exterior.
- **Largura de Banda Mínima:** Parâmetro que indica o limite mínimo de largura de banda aceitável para a satisfação das suas necessidades de acesso à rede exterior.

O algoritmo encontrado para a selecção do *gateway*, tem como resultado uma interface com acesso à rede exterior de um *gateway*. O algoritmo foi

implementado como um conjunto de regras para motor de regras JBoss Rules, este divide-se em duas fases distintas:

- **Fase 1:** Nesta fase o algoritmo começa por identificar quais as interfaces dos *gateways*, com acesso à rede exterior que possuem largura disponível suficiente para satisfazer a largura de banda desejada. É de notar que apenas são consideradas as interfaces de *gateways*, onde a interface de acesso a rede interior tem também largura de banda suficiente para satisfazer a largura de banda desejada, isto porque o tráfego que um *gateway* encaminha para à rede exterior chega através sua interface de acesso a rede interior.

  Uma vez identificadas as interfaces de acesso a rede exterior que satisfazem a largura de banda desejada, passamos a comparar estas relativamente ao seu custo monetário, sendo escolhida a interface de menor custo. Caso existam várias interfaces com o mesmo custo, para o desempate são aplicadas algumas preferências que reflectem algumas optimizações, sendo estas as seguintes:

  - **1º:** Preferir a interface de acesso a rede externa que estamos a utilizar actualmente (caso exista), de modo a manter esta, visto não existir nenhuma vantagem em termos de custo, em escolher outra, garantindo assim uma maior estabilidade e evitando gastos desnecessários de recursos computacionais e de rede.

  - **2º** Preferir as interfaces de acesso a rede exterior, do próprio nó (caso existam), isto para evitar que dois nós possam ser o *gateway* um do outro, e também evitar também alguma sobrecarga na rede interna.

  - **3º** Preferir as interfaces de acesso à rede exterior, com a menor largura de banda disponível, como forma de tentar evitar que as interfaces tenham largura de banda livre não utilizável, que no seu total poderia ser suficiente para satisfazer outros nós, mas como se encontra distribuída por varias interfaces, não dá para mais nenhum. Está preferência possivelmente, não será a melhor estratégia para o objectivo ao qual se destina, além de se ter mostrado eficaz nos testes realizados.

Caso nenhuma destas preferências se verifique, é escolhida uma interface arbitrariamente.

- **Fase 2:** Esta fase apenas é executada caso na fase anterior, não se encontrem interfaces de acesso à rede exterior, com largura de banda suficiente para satisfazer a largura de banda desejada. Nesta fase o algoritmo passa a identificar as interfaces, que possuem largura disponível entre o valor dos parâmetros de largura de banda mínima e largura de banda desejada.

  Uma vez identificadas as interfaces com largura de banda disponível entre a mínima e a desejada, o procedimento que resulta na escolha de uma das interfaces é em tudo semelhante ao da fase anterior, excepto na ùltima preferência de desempate para as interfaces com custos idênticos, onde em vez de se preferir interfaces com menor largura de banda disponível, preferem-se interfaces com maior largura de banda, visto que o valor destas é mais próximo da largura de banda desejada, resultando assim em uma melhor satisfação das necessidades do nó.

Capítulo 4

Simulador da Solução

Para a demonstração da solução apresentada no capítulo anterior, foi desenvolvido um simulador em Java, que será apresentado neste capítulo. O objectivo do simulador é tornar possível testar a solução proposta utilizando apenas um computador pessoal, evitando assim a utilização de toda uma infraestrutura.

Vamos agora passar a apresentar, os principais elementos que constituem o simulador, elementos estes que se encontram numerados na figura 4.1.

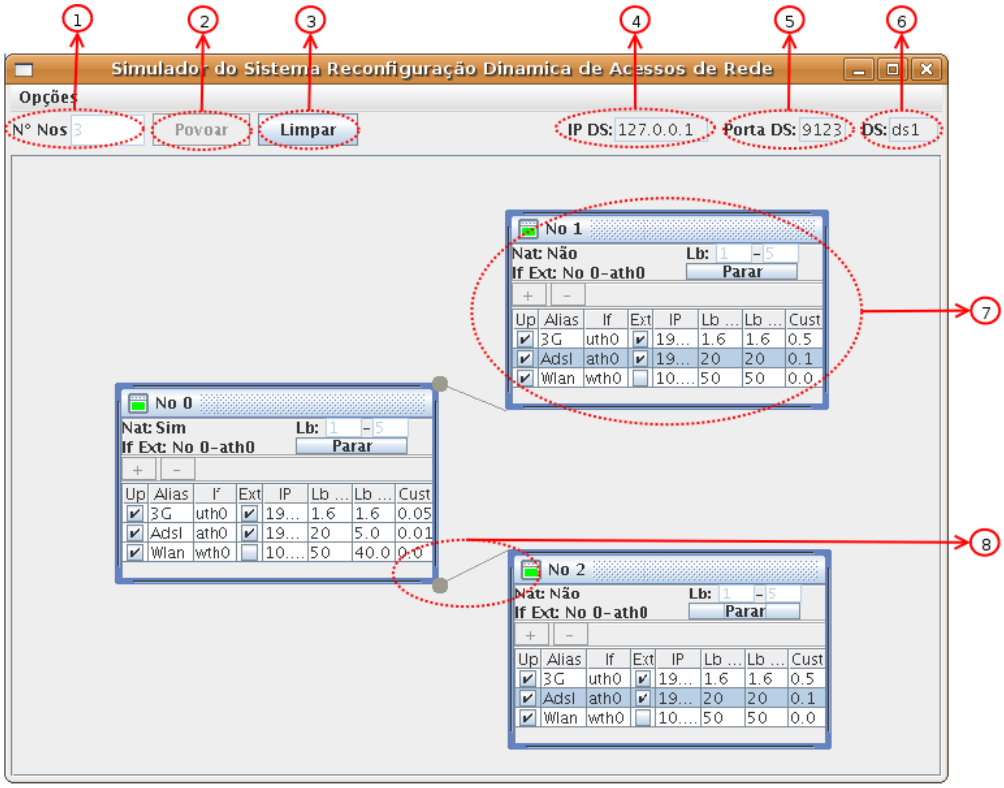

Figura 4.1: Simulador da Solução

Descrição dos elementos constituintes do simulador:

- **1)** Numero de nós que pretendemos ter numa simulação.
- **2)** Butão que povoa o simulador com o número de nós, que se pretende ter na simulação.
- **3)** Butão que limpa os nós actualmente no simulador.
- **4)** Endereço IP do servidor onde se encontra o *dataspace*.
- **5)** Porta TCP do servidor do *dataspace*.
- **6)** Identificador do *dataspace* no servidor (visto que, um servidor pode ter vários *dataspaces*)

- **7)** Representação de um nó da rede.
- **8)** Seta que representa a relação entre um nó e o seu *gateway*, sendo o *gateway* o que se encontra na ponta da seta (circulo).

Uma vez descritas os principais elementos presentes no simulador, vamos agora passar descrever em pormenor um nó da rede (ver figura 4.2).

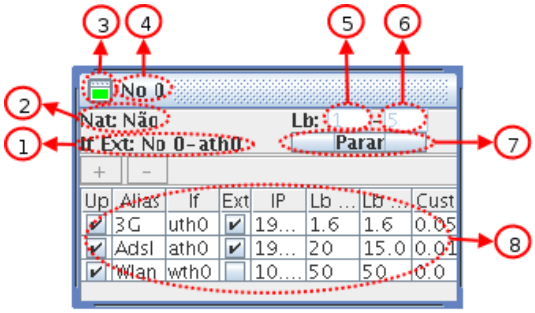

Figura 4.2: Nó da Rede

Descrição dos elementos constituintes de um nó da rede:

- **1)** Identificador da interface do *gateway*, pela qual o tráfego do nó sai para o rede exterior (o formato do identificador é o seguinte: "identificador do nó onde se encontra a interface"-"nome da interface").
- **2)** Indicação sobre se o nó está ou não a realizar NAT, para um ou mais dos seus nós pares (um nó apenas realiza NAT quando desempenha o papel de *gateway* perante outros nós, que não ele próprio)
- **3)** Icon que pode assumir quatro cores diferentes (vermelho, laranja, verde e amarelo), onde cada uma destas, representa um estado do nó. O significado associado a cada cor é o seguinte:
  - **Vermelho:** O nó encontra-se desligado.
  - **Laranja:** O nó encontra-se ligado mas não tem um *gateway*
  - **Verde:** O nó encontra-se ligado e tem um *gateway* que lhe alocou a largura de banda desejada.

  - **Amarelo:** O nó encontra-se ligado e tem um *gateway* que lhe alocou largura de banda abaixo da desejada (mas acima da mínima)

- **4)** Identificador que identifica univocamente um nó, em relação os seus pares.
- **5)** Largura de banda mínima, que o nó aceita para a satisfação das suas necessidades
- **6)** Largura de banda desejada, pelo nó, para a satisfação em pleno das suas necessidades.
- **7)** Butão que inicia e pára o nó (liga e desliga)
- **8)** Interfaces de rede do nó (estrutura descrita na secção 3.1).

Capítulo 5

Testes e Validação da Solução

Para a demonstração da solução proposta, foi desenvolvido o simulador apresentado no capitulo anterior. Neste capitulo vamos sujeitar esse mesmo simulador a um conjunto de cenários, como forma de validar a solução proposta no capitulo 3.

Para cada um dos cenários além da sua caracterização, será também apresentado o resultado obtido no simulador, bem como a sua interpretação. É importante referir a ordem pela qual são iniciados (ligados) os nós presentes no simulador para os cenários apresentados, isto porque esta pode em alguns casos influenciar o resultado. A ordem que vamos convencionar, aplica-se a todos os cenários, esta tem como referencia o identificador dos nós, sendo os nós iniciados ascendentemente de acordo com o seu identificador (e.g. "No 1","No 2", ..., "No X").

5.1 Cenário 1

Este cenário caracteriza-se, pela existência de apenas um interface de acesso à rede exterior com o melhor custo, onde esta tem largura de banda disponível suficiente para satisfazer a necessidade de todos os nós da rede, incluindo a do próprio nó. O nome desta interface é "ath0" e pertence ao "No 0". Os restantes nós têm todos interfaces de acesso a rede exterior idênticas as do "No 0", mas com custos superiores.

Na figura 5.1 é apresentado o resultado do simulador para este cenário, depois de todos os nós serem iniciados de acordo com a ordem convencionada.

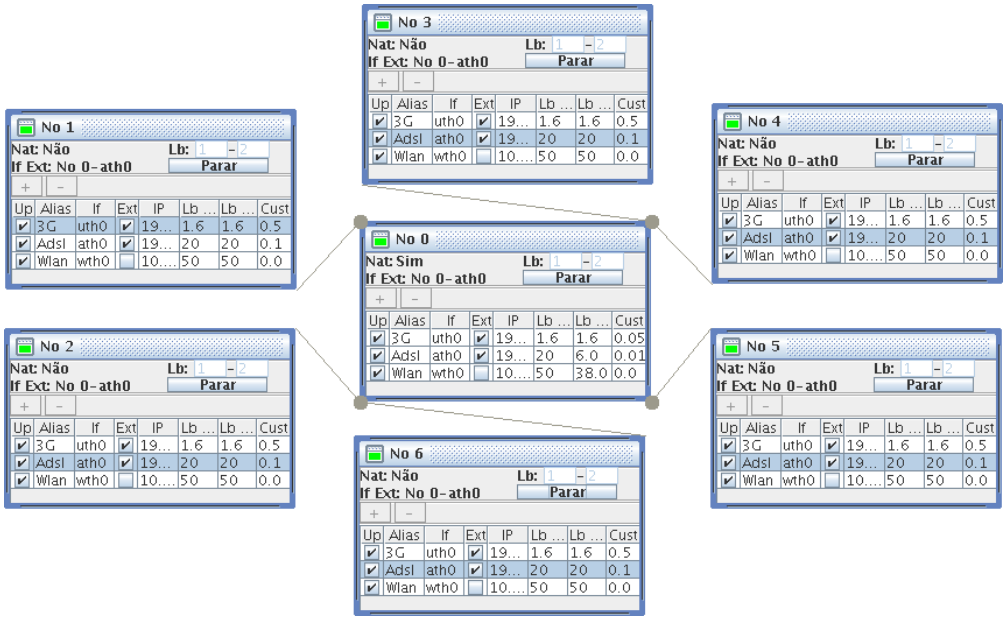

Figura 5.1: Cenário 1

Como podemos ver na figura 5.1, todos os nós escolheram o "No 0" como *gateway*, em particular a interface "ath0" deste para o seu acesso a rede exterior, visto esta ter o menor custo e largura de banda disponível suficiente para satisfazer a largura de banda que cada um dos nós desejava.

5.2 Cenário 2

Este cenário caracteriza-se pelo facto, de as interfaces de acesso à rede exterior de todos os nós serem idênticas e largura de banda disponível nestas, ser suficiente para se considerar, que qualquer um dos nós pode assumir o papel de *gateway* perante todos os seus pares, incluindo o próprio.

Na figura 5.2 é apresentado o resultado do simulador para este cenário, depois de todos os nós serem iniciados de acordo com a ordem convencionada.

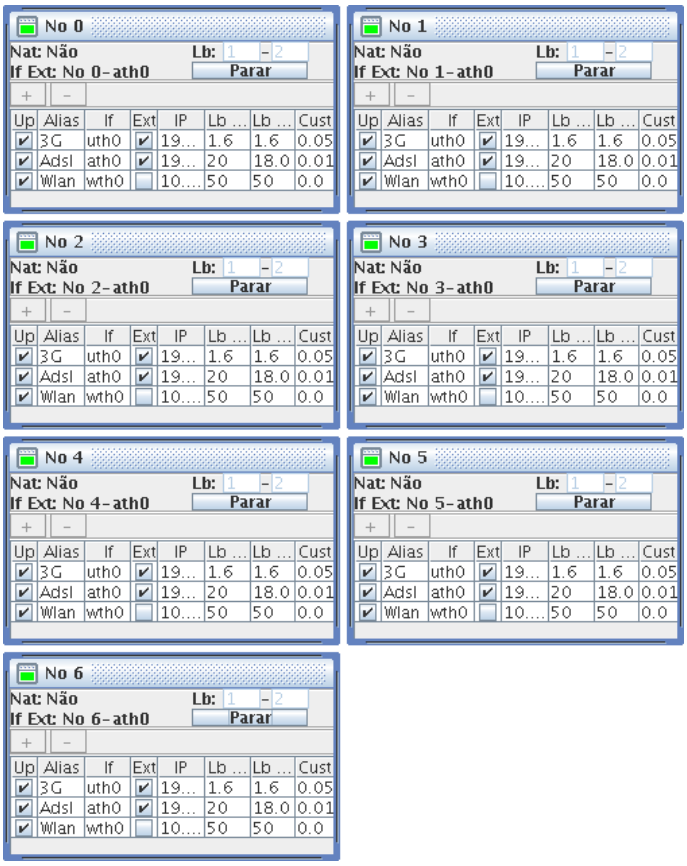

Figura 5.2: Cenário 2

Em princípio, neste cenário, cada um dos nós da rede poderia escolher uma interface de qualquer nó para o acesso a rede exterior, visto que as interfaces têm todas o mesmo custo. No entanto, como se pode ver na figura 5.2, nenhum dos nós escolheu uma interface de um outro nó. Este resultado deve-se ao facto de cada nó dar preferência às suas próprias interfaces, para o caso em que várias interfaces tem um mesmo custo, isto como forma de evitar sobrecarga na rede interna (lan), quando não existem vantagens em termos de custo (ver a 2ª preferência da 1ª fase do algoritmo na subsecção 3.2.2), entre utilizar uma interface própria e uma de outro nó.

5.3 Cenário 3

Neste cenário à semelhança do que acontece no cenário 1, a interface "ath0" do "No 0", é a interface de acesso à rede exterior com o melhor custo, a diferença é, que neste caso a largura de banda disponível não é suficiente para satisfazer a necessidade de todos os nós. Já a interface "ath0" do "No 6" tem largura de banda suficiente para satisfazer as necessidades de todos os nós, mas o seu custo é um pouco superior, ao da interface do "No 0". Quanto aos restantes nós, estes tem todos interfaces semelhantes as do "No 6", excepto no custo, que é superior.

Na figura 5.3 é apresentado o resultado do simulador para este cenário, depois de todos os nós serem iniciados de acordo com a ordem convencionada.

Figura 5.3: Cenário 3

Como se pode constatar na figura 5.3, os primeiros nós a serem iniciados ("No 0" a "No 3"), obviamente escolheram a interface "ath0" do "No 0",

como acesso a rede exterior, visto o seu custo ser o menor. A partir do momento em que a interface "ath0" do "No 0", deixou de ter largura disponível, os restantes nós ("No 4" a "No 5") tiveram de procurar alternativas, estes inicialmente começaram por utilizar uma interface própria (devido a 2ª preferência da 1ª fase do algoritmo na subsecção 3.2.2), até ao momento em que o "No 6" é iniciado e se apercebem que a interface "ath0" deste é a segunda melhor opção em termos de custo e que esta tem largura de banda disponível suficiente, para satisfazer em pleno as suas necessidades.

5.4 Cenário 4

Neste cenário o "No 0" é em principio o mais forte candidato a melhor *gateway*, visto este ter duas interfaces de acesso a rede exterior ("ath0" e "ath1"), com o menor custo. A largura de banda disponível nestas interfaces é diferente, sendo a da interface "ath0" menor. A largura de banda nas duas interfaces, no total, é suficiente para satisfazer em pleno as necessidades de todos os nós na rede interna, incluindo o próprio. Os restantes nós tem interfaces semelhantes entre si, onde estas tem custos superiores aos das interfaces presentes no "No 0".

Na figura 5.4 é apresentado o resultado do simulador para este cenário, depois de todos os nós serem iniciados de acordo com a ordem convencionada.

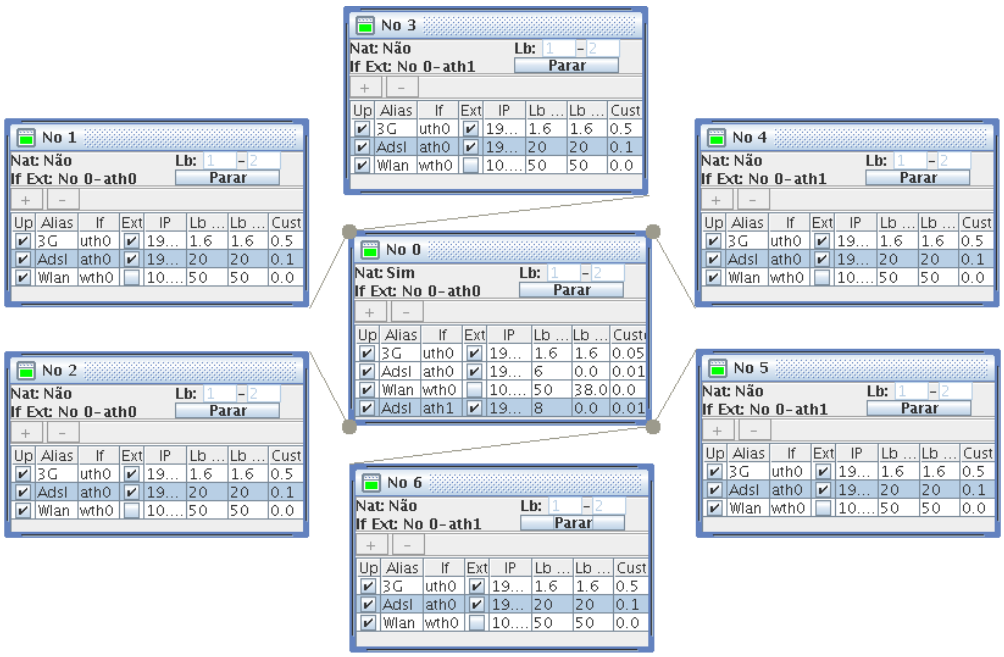

Figura 5.4: Cenário 4

Como seria de esperar o "No 0", foi o *gateway* escolhido por todos os nós, incluindo o próprio (ver figura 5.4). Os primeiros três nós a serem iniciados ("No 0" a "No 2") escolheram para satisfação das suas necessidades a interface "ath0", esgotando a largura de banda que esta disponibilizava. Uma vez esgotada a largura de banda disponível em "ath0", os restantes nós escolheram a interface "ath1", visto o custo desta ser igual ao de "ath0" e ainda dispor de largura de banda para partilhar. É de notar que o facto dos três primeiros nós a serem iniciados terem escolhido uma mesma interface quando existia uma outra com o mesmo custo, não é um acaso. Tal comportamento deveu-se a terceira preferência da 1ª fase do algoritmo de selecção de *gateway* na subsecção 3.2.2, para o caso de existirem várias interfaces com um mesmo custo.

5.5 Cenário 5

Neste cenário vamos demonstrar como um conjunto de nós recupera, após a falha do seu *gateway*. Para tal vamos apresentar o resultado do cenário antes (figura 5.5) e depois (figura 5.6) da falha do *gateway*.

No resultado do cenário para a situação "antes", podemos ver na figura 5.5, que "No 0" é o *gateway* escolhido por todos os nós, sendo a sua interface "ath0" a escolhida, visto ter o menor custo e largura de banda disponível suficiente para satisfazer plenamente todos os nós.

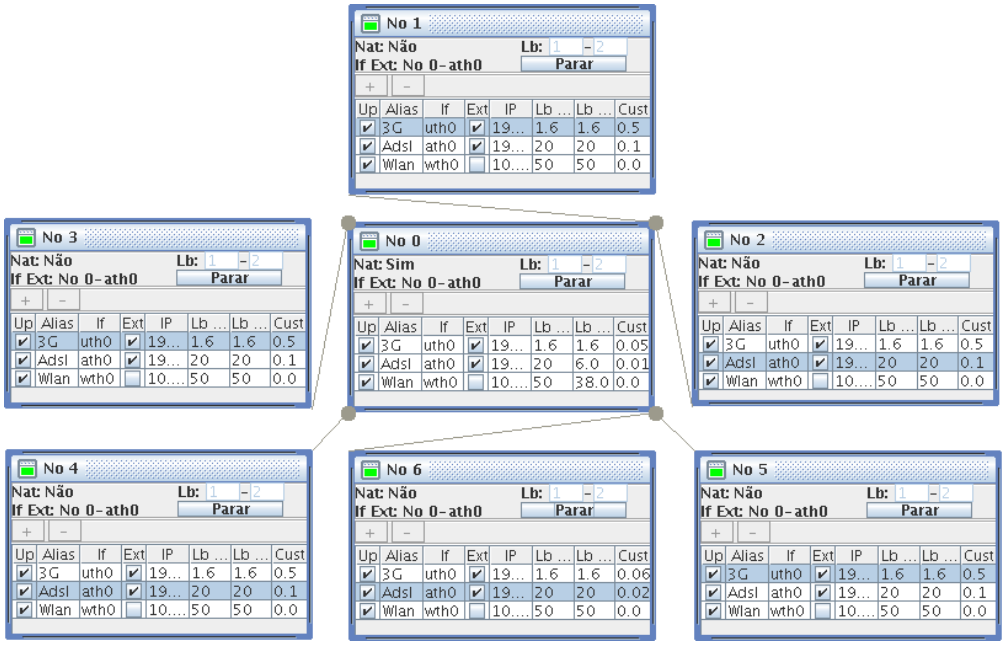

Figura 5.5: Cenário 5 - Antes da Falha do "No 0"

Para simular a falha do "No 0" vamos carregar no botão "Parar", fazendo com que o nó seja desligado e assim deixe de estar presente na rede (icon vermelho). Uma vez desligado, todos os nós rapidamente se vão aperceber que perderam o seu *gateway*, reiniciando assim todo o processo de selecção do *gateway*, considerando para tal, todos os nós actualmente presentes na rede. Uma vez terminado o processo, como se constata na figura 5.6, cada um dos nós concluiu, que actualmente a melhor opção para *gateway* é o "No 6" e a

sua interface “ath0”, visto que esta actualmente possui actualmente o menor custo e largura de banda disponível suficiente para satisfazer plenamente todos os nós.

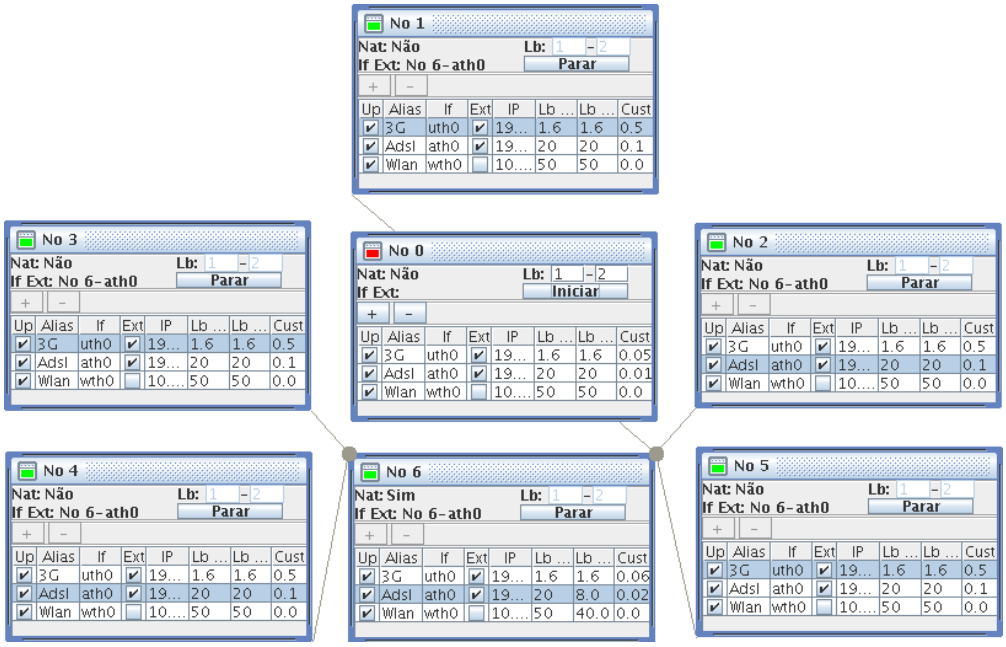

Figura 5.6: Cenário 5 - Depois da Falha do “No 0”

5.6 Cenário 6

Este cenário, caracteriza-se pela existência de apenas uma interface de acesso à rede exterior com o menor custo, onde a largura de banda que esta tem disponível para partilhar, não é suficiente para satisfazer a largura de banda desejada por todos os nós. Esta interface pertence ao “No 0” e é designada de “ath0”. Uma outra particularidade deste cenário, é a inexistência da interface de acesso à rede exterior “ath0”, em todos os outros nós (“No 1” a “No 6”). Assim sendo, aos nós de 1 a 6, resta-lhes apenas a interface “uth0”, com largura de banda total disponível entre a mínima e a desejada (à semelhança da “uth0” do “No 0”) e de custo superior ao da interface “uth0” do “No 0”.

Na figura 5.7 é apresentado o resultado do simulador para este cenário, depois de todos os nós serem iniciados de acordo com a ordem convencionada.

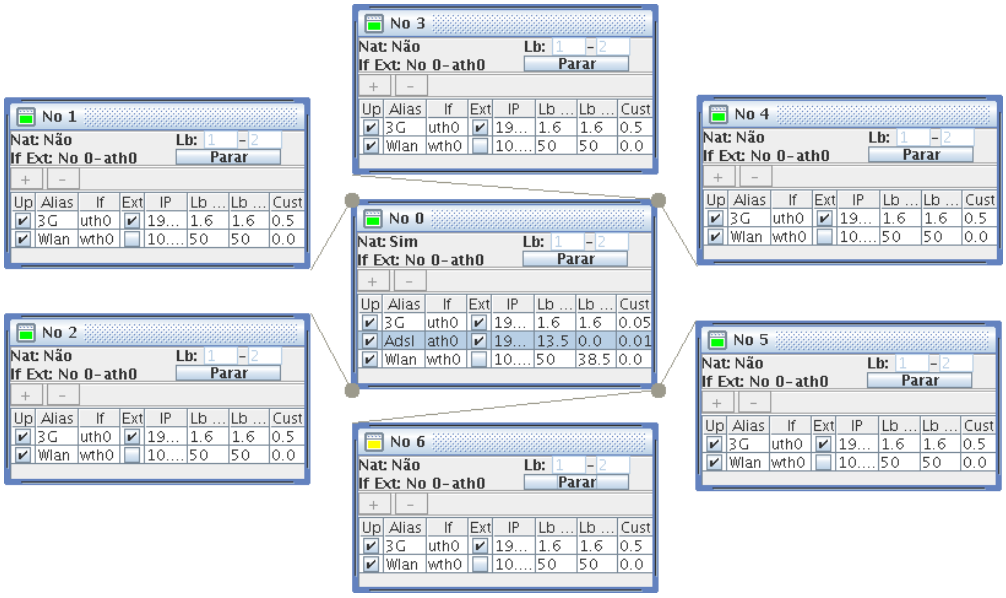

Figura 5.7: Cenário 6

Até agora, nos cenário apresentados anteriormente os nós escolhidos para *gateway*, tinham sempre largura de banda disponível para satisfazer a largura de banda desejada pelo nós, com os quais partilhavam a sua interface. Como podemos na figura 5.7, todos os nós escolheram a interface "ath0" do "No 0", para o seu acesso a rede exterior, no entanto o icon do "No 6" (o ultimo nó a ser ligado) em vez de apresentar a cor verde como os seus pares, apresenta a cor amarela, o que significa que este na 1ª fase do algoritmo de selecção do *gateway* na subsecção 3.2.2, não encontrou uma interface com acesso a rede exterior, que satisfizesse a largura de banda que desejava, tendo assim de passar a 2ª fase do algoritmo onde considerou interfaces com largura de banda disponível entre a largura de banda mínima aceitável e a desejada. A interface escolhida pelo "No 6", apesar de tudo, é também a interface "ath0" do "No 0", devido ao seu menor custo, apesar da existência de outras opções (interfaces "uth0" dos nós de 1 a 6), onde a largura de banda disponível é superior, mas que em contrapartida o custo também é superior. Em suma o

icon amarelo significa, que apesar de o nó ter encontrado um *gateway*, este não o satisfaz plenamente, mas por outro lado consegue no mínimo satisfazer as suas necessidades básicas.

É de notar que caso os nós, excepto o "No 0", não se encontrassem desprovidos das suas interfaces "ath0", possivelmente o "No 6", escolheria a interface de um destes nós, mesmo que esta fosse mais cara que a do "No 0", desde que esta satisfizesse a largura de banda desejada.

5.7 Cenário 7

Este cenário tem como ponto de partida o resultado do cenário anterior (ver figura 5.7). Neste cenário vamos apenas desligar (carregar em "Parar") o "No 5" e ver o que acontece.

Na figura 5.8 é apresentado o resultado do simulador para este cenário, depois de se desligar o "No 5".

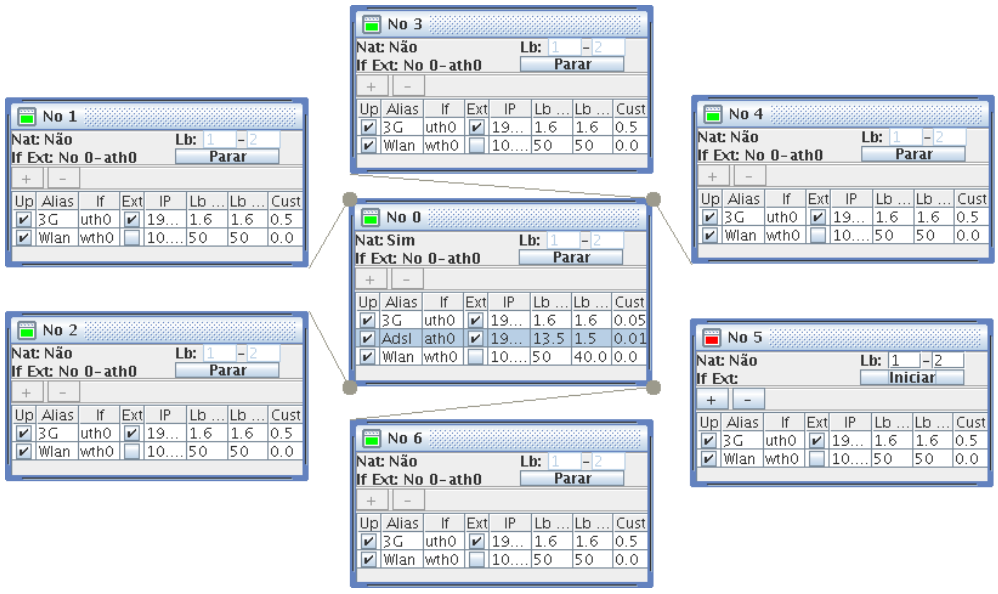

Figura 5.8: Cenário 7

No cenário anterior como podemos ver na figura 5.7, o "No 6" tinha o seu icon amarelo, que indicava que o *gateway* que escolheu, não lhe tinha alocado a largura de banda que desejava. No resultado deste cenário o icon que estava a amarelo passou a verde, no entanto o *gateway* escolhido manteve-se. O que se passou, foi que quando o "No 5" foi desligado, a largura de banda que havia sido alocada a este na interface "ath0" do "No 0" foi libertada e o "No 6" apercebeu-se disto e renegociou a largura de banda que lhe havia sido alocada, conseguindo assim a largura de banda desejada, passando assim o icon deste da cor amarela para a verde.

É de notar que a largura de banda libertada juntamente com a largura de banda já alocada pelo "No 0" ao "No 6", na interface "ath0", poderia não ter sido suficiente para satisfazer a largura de banda que este desejava, mas mesmo neste caso o "No 6" iria tentar renegociar a largura de banda alocada e possivelmente obter mais um pouco de largura de banda alocada, mantendo o seu icon a cor amarela, mas no entanto mais próximo da largura de banda desejada.

Capítulo 6

Conclusões

Neste trabalho foi desenvolvido um sistema que permite simular a utilização de um sistema de reconfiguração dinâmica de acessos de rede, por conjunto de nós de uma rede. O objectivo do simulador é tornar possível testar a solução proposta utilizando apenas um computador pessoal, evitando assim a utilização de toda uma infraestrutura.

Como especificado na descrição da solução proposta para o sistema (ver secção 3.2), cada um dos nós presente no simulador utiliza uma instância do motor de regras para Java, JBoss Rules, no qual são implementadas as políticas de encaminhamento (algoritmo de selecção do *gateway*) na forma de regras.

Neste trabalho verificou-se, que o motor de regras utilizado se mostrou adequado, para a implementação da solução proposta. Não só porque este permite que as politicas de encaminhamento possam ser alteradas sem que o código fonte da aplicação seja alterado, bastando para tal alterar o ficheiro de texto onde estas estão definidas, mas também pelo seu bom desempenho no simulador, no qual, em vez de uma instancia do motor de regras como é previsto na solução, tínhamos várias, a serem executadas concorrentemente em um único computador.

A utilização de um motor de regras neste trabalho, resultou em uma implementação mais rápida da lógica da aplicação, mas que, mesmo assim se mostrou robusta na fase de testes, na qual, os problemas encontrados quase

nunca se relacionavam com as regras que definem a lógica da aplicação (politicas de encaminhamento).

Neste trabalho, verificou-se que os motores de regras são realmente ferramentas que têm as suas vantagens, sendo assim, em certos casos uma opção a considerar, principalmente no que toca, à definição da lógica de uma aplicação. No entanto é preciso considerar se realmente é necessário recorrer a uma ferramenta deste tipo, visto que o seu desempenho não é tão bom, como o de uma linguagem imperativa e de lhe estar associada uma curva de aprendizagem considerável.

As principais dificuldades encontradas durante a realização deste trabalho, encontram-se relacionadas com escassa documentação existente sobre o motor de regras JBoss Rules bem como sobre a plataforma de *middleware* Equip2. No entanto, a documentação sobre o JBoss Rules além de não ser abundante, provou ser suficiente para este trabalho, já o mesmo não se pode dizer acerca do EQUIP2. A insuficiência de informação sobre a plataforma EQUIP2, fez com que esta, além de ser uma componente secundária, fosse a componente à qual foi dedicada mais tempo, visto que a sua aprendizagem muitas das vezes se baseava no método de tentativa erro.

Capítulo 7

Trabalho Futuro

Esta secção apresenta um conjunto de aspectos susceptíveis a dar continuação ao trabalho realizado. Nele incluem-se alguns dos aspectos que podem ser refinados através de estratégias e implementações mais elaboradas, e todos aqueles que por não serem fundamentais ao funcionamento do sistema, dentro do âmbito do trabalho, não foram abordados. Estes aspectos são apresentados de seguida.

- Transposição da solução proposta de um ambiente simulado, para um ambiente real, aproveitando praticamente todo trabalho realizado até este ponto. Sendo os passos seguintes principalmente na direcção do desenvolvimento de uma interface que permita a cada um dos nós realizar as configurações de rede necessárias ao funcionamento do sistema, como a configuração do NAT para o papel de *gateway* e a configuração da tabela de encaminhamento para o caso de o *gateway* ser um nó par e não o próprio.

- Implementação do suporte para os outros tipos de taxação (tarifários), além da taxação por volume de tráfego, visto que hoje em dia existem vários outros tipos de taxação, que estão cada vez mais em voga.

- Optimização do algoritmo de selecção do *gateway*, por forma a este tolerar que um acesso que não disponibilize a largura de banda desejada seja escolhido em vez de outros a disponibilizem, caso o custo seja bastante mais reduzido e a largura de banda disponível se encontre entre a mínima e a desejada.

- Possibilitar a um nó que assuma o papel de *gateway*, reatribuir as interfaces de acesso a rede exterior aos nós com os quais partilha estas, por forma a minimizar o espaço não utilizado em cada interface.
- Identificar outros parâmetros que possam influenciar/indicar o desempenho dos acessos de rede de cada nó, e alterar o algoritmo de selecção do *gateway* por forma a considerar estes também, de acordo com o tipo de ligação pretendida.

Bibliografia

[1] Friedman-Hill, Jess in Action: Java Rule-based Systems,
Ernest, Manning Publications, 2003

[2] JBoss Rules User Guide,
http://labs.jboss.com/file-access/default/
members/jbossrules/freezone/docs/3.0.6/html_single/index.html,
visitado em: 19/05/07

[3] MGJUG JbossRules,
http://www.mgjug.com.br/palestras/20070308-MGJUG-JBossRules-BrenoBarros.pdf,
visitado em: 19/05/07

[4] Jess the Rule Engine for the Java Platform,
http://herzberg.ca.sandia.gov/jess/docs/71/,
visitado em: 19/05/07

[5] Getting Started With the Java Rule Engine API (JSR 94):
Toward Rule-Based Applications,
http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html,
visitado em: 13/07/07

[6] Venkat Venkatasubramanian, Sourabh Dash, Mano Ram Maurya,
Priyan Patkar and Chunhua Zhao,
EXPERT SYSTEMS - PRINCIPLES AND APPLICATIONS
Purdue University,
Laboratory for Intelligent Process Systems, School of Chemical Engineering

[7] JLog Homepage,
http://jlogic.sourceforge.net/
visitado em: 13/08/07

[8] Dennis Merritt, Building Expert Systems in Prolog,
Amzi! inc., 2000

[9] EQUIP2 Overview,
http://equip.sourceforge.net/equip2/docs/EQUIP2_Overview.html,
visitado em: 12/10/07