Renã Pereira de Oliveira

Um modelo de Virtualização Distribuída

São José – SC

Setembro / 2010

Renã Pereira de Oliveira

Um modelo de Virtualização Distribuída

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Eraldo Silveira e Silva, M.

Co-orientador:
Cleiber Marques da Silva

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Setembro / 2010

Monografia sob o título *"Um Modelo de Virtualização Distribuída"*, defendida por Renã Pereira de Oliveira e aprovada em 03 de setembro de 2010, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Eraldo Silveira e Silva, Ms.
Orientador - IF-SC - Campus São José

Prof. Evandro Cantú, Dr.
IF-SC - Campus São José

Prof. Marcelo Sobral, Ms.
IF-SC - Campus São José

Agradecimentos

Dedico meus agradecimentos àqueles que me apoiaram e incentivaram na realização deste trabalho. Estas pessoas, sem dúvida, facilitaram o caminho me mostrando que eu seria capaz de ir até o fim.

Agradeço à Coordenadoria de Telecomunicações do IF-SC pelos computadores disponibilizados em laboratório, e agradeço principalmente ao Professor Eraldo, por ter acreditado no trabalho, por ter me apoiado, pelo conhecimento transmitido e por sua colaboração nas correções do mesmo.

Resumo

O uso de máquinas virtuais UML para o estudo de redes de computadores vem crescendo gradativamente nos últimos anos. O suporte dado por elas para a simulação de equipamentos de rede é vantajoso, principalmente quando não há componentes reais suficientes para os experimentos realizados. Um problema encontrado com o uso destas máquinas, é que, ao se montar um cenário de rede, ele fica limitado dentro de um único computador(hospedeiro), e dependendo da configuração do hardware, pode ser impossível simular grandes redes. Este documento apresenta um modelo que possibilita usuários de máquinas UML a distribuírem domínios virtuais em diferentes hospedeiros. Esse modelo funcionará de forma transparente, ou seja, não interferirá nas simulações de redes. Além deste modelo, será projetado e implementado um sistema que auxilie o usuário na geração de cenários de rede com este suporte à distribuição de domínios. Ao final do trabalho, alguns testes serão realizados comprovando a eficiência do sistema.

Palavras-chave: User-Mode Linux, Sistemas distribuídos, Virtualização distribuída.

Abstract

The use of UML virtual machines in order to study computer networking is gradually expanding in the last years. The support given by them for the network equipments simulation is advantageous, mainly when there are no enough real components for the experiments. A problem encountered with the use of these machines is that when a network scenario is created, it remains limited within only one computer(host), and depending on the Hardware configuration, it might be impossible to simulate big networks. This paper presents a model that makes possible users of UML machines to distribute virtual domains in different hosts. Besides this model, a system that helps the user to generate network scenarios with this domains distribution support will be projected and implemented. Closing the paper, some tests will be performed proving the system eficiency.

Keywords: User-Mode Linux, Distributed Systems, Distributed Virtualization.

Sumário

Lista de Figuras

Lista de Abreviaturas

CN Correspondent Node

CoA Care-of-Address

GUML GUI Management console for User Mode Linux

GUML4MIP GUI Management console for User Mode Linux for Mobile IP

HA Home Agent

HMIPv6 Hierarchical Mobile Internet Protocol version 6

IPv6 Internet Protocol version 6

MN Mobile Node

MIPv6 Mobile Internet Protocol version 6

RA Router Advertisement

UML User-Mode Linux

VLANS Virtual Local Area Networks

1 Introdução

1.1 Motivação

Dentro da área de redes de computadores, muitas vezes é complicado realizar experimentos, seja pela falta de equipamentos em laboratórios, ou por questões de escalabilidade quando as redes envolvidas são muito grandes. Devido a este fator, buscou-se formas de simular essas redes, sem a necessidade de muitos componentes físicos. Uma das maneiras que acabou se consolidando é o uso de máquinas virtuais, as quais possibilitam que vários ”terminais virtuais”sejam executados dentro de um mesmo terminal real(hospedeiro). Além de trazer mais flexibilidade para a simulação de cenários de rede, trouxe maior segurança para o sistema hospedeiro, já que os processos são executados dentro da máquina virtual, não alterando o sistema de arquivos do hospedeiro.

Um dos tipos de máquinas virtuais utilizado é o User-Mode Linux (UML), o qual basicamente executa um Linux ”virtual”dentro de um Linux hospedeiro. Dentre as funcionalidades desta máquina virtual, uma delas é permitir a simulação de redes de computadoresEsta máquina possui muitas funcionalidades, dentre elas, com a adição de alguns componentes, permite a simulação de redes de computadores.

Em (SILVA; ALMEIDA, 2009b) foi desenvolvida uma plataforma chamada de GUML4MIP, que teve como objetivo proporcionar facilidades para a análise de protocolos de mobilidade na camada de rede utilizando máquinas virtuais como base para os testes. Os terminais móveis se movimentavam em torno de um *switch* virtual, simulando a mobilidade entre roteadores de acesso.

No presente projeto, com o intuito de melhorar a GUML4MIP, pensou-se em mesclar o cenário virtual com terminais móveis reais. Isto exigiria a instalação dos protocolos nestas máquinas. Para flexibilizar este processo, pensou-se distribuir o modelo de virtualização de forma a contemplar as máquinas reais. Adicionalmente, percebeu-se que a virtualização distribuída também favoreceria a execução massiva de máquinas virtuais em redes, que é o caso da

simulação de vários domínios *Ethernet*, onde cada um possui dezenas de roteadores. A partir deste ponto, evoluiu-se para a criação de um modelo geral de distribuição para domínios virtuais utilizando as máquinas UML, assim, percebendo que essa ideia seria mais conveniente e de maior utilidade, o projeto tomou um novo rumo, focando essencialmente na definição e implementação de um modelo de virtualização distribuída.

1.2 Objetivos

O objetivo deste trabalho é propor um modelo e um suporte de virtualização distribuída de redes usando máquinas UML. Como objetivos específicos tem-se:

- definir um modelo de execução distribuída de cenários de redes. Cada parte do cenário, executado em um determinado hospedeiro, será doravante chamado de domínio virtual;
- definir e implementar um suporte para este modelo envolvendo: um protocolo de descoberta de hospedeiros, uma linguagem de descrição de cenários distribuídos, um protocolo de carga de domínios virtuais e um subsistema de execução e monitoramento remoto.

1.3 Organização do texto

Este trabalho está organizado da seguinte forma. O capítulo 2 apresenta conceitos básicos envolvidos no trabalho e a plataforma utilizada para a implementação do modelo de virtualização distribuída, a GUML4MIP. O capítulo 3 apresenta o modelo de virtualização distribuída e fornece uma visão geral do sistema desenvolvido. O capítulo 4 traz detalhes adicionais da implementação e mostra os testes realizados com o sistema. O capítulo 5 conclui o trabalho mostrando algumas perspectivas futuras.

2 *Visão da plataforma GUML4MIP e seu funcionamento*

O uso de máquinas virtuais UML para a simulação de redes de computadores vem se tornando uma ferramenta muito comum entre estudantes, professores e pesquisadores da área. Com apenas um hospedeiro (computador) e com pequenos *scripts* (programas) é possível simular uma quantidade significativa de equipamentos como computadores, roteadores, *switchs*, *hubs*, entre outros, e interligá-los da forma desejada. Sendo um sistema Linux similar ao do hospedeiro, as UMLs permitem a instalação e utilização das mesmas implementações de protocolos utilizados na área de redes.

Para facilitar a rápida implementação de cenários com a UML, foram desenvolvidos ambientes e linguagens específicas de descrição de redes. Neste contexto se insere a aplicação GUML4MIP (SILVA; ALMEIDA, 2009a), uma plataforma desenvolvida com o objetivo auxiliar estudantes de redes de computadores a entenderem o funcionamento dos protocolos para mobilidade na camada de rede. Este mesmo ambiente foi usado como suporte ao desenvolvimento deste trabalho e, por isso, este capítulo será dedicado em parte a descrição da referida plataforma.

O capítulo está organizado da seguinte forma. Inicialmente são apresentados os conceitos básicos associados as máquinas virtuais UML bem como os mecanismos e ferramentas necessários a construção de redes virtuais através destas máquinas. Na sequência é apresentada a plataforma GUML4MIP e um exemplo do uso da mesma.

2.1 Virtualização com máquinas User-mode Linux

2.1.1 Motivações do Uso da UML

A máquina virtual UML possibilita que um Linux com kernel simplificado seja executado juntamente com um sistema de arquivos dentro de um outro Linux chamado de hospedeiro.

Quando uma máquina virtual tem seu sistema danificado, nenhum risco é trazido ao hospedeiro. Além desta segurança, as UMLs oferecem vantagens como:

- **Servidores virtuais** (DIKE, 2005): Inúmeros servidores virtuais podem ser criados dentro de um hospedeiro, diminuindo a necessidade de estações/computadores para isso. Backups podem ser facilmente executados devido ao sistema de arquivos das UMLs ser um simples arquivo dentro do hospedeiro;
- **Desenvolvimento do Kernel do Linux** (DIKE, 2005): Trabalhar com o Kernel do Linux, verificando resultados após a compilação sem trazer problemas ao hospedeiro;
- **Ensino**: Como não há a preocupação com o sistema hospedeiro, utilizando as UMLs os estudantes podem trabalhar livremente com o que foi estudado;
- **Redes de computadores**: As UMLs possuem facilidades para trabalhar com redes de computadores, pois diminuem a necessidade de utilizar um número grande de computadores na rede, possuem excelente suporte para protocolos de rede e são capazes de simular um grande número de interfaces de rede;

Para rodar uma máquina UML, é necessário ter em mãos um Kernel Linux compilado para a arquitetura *um* (modo usuário) e um sistema de arquivos que nada mais é do que um arquivo de imagem contendo toda a estrutura de diretórios do sistema GNU/Linux. A figura 2.1 mostra a execução de máquinas UML no hospedeiro.

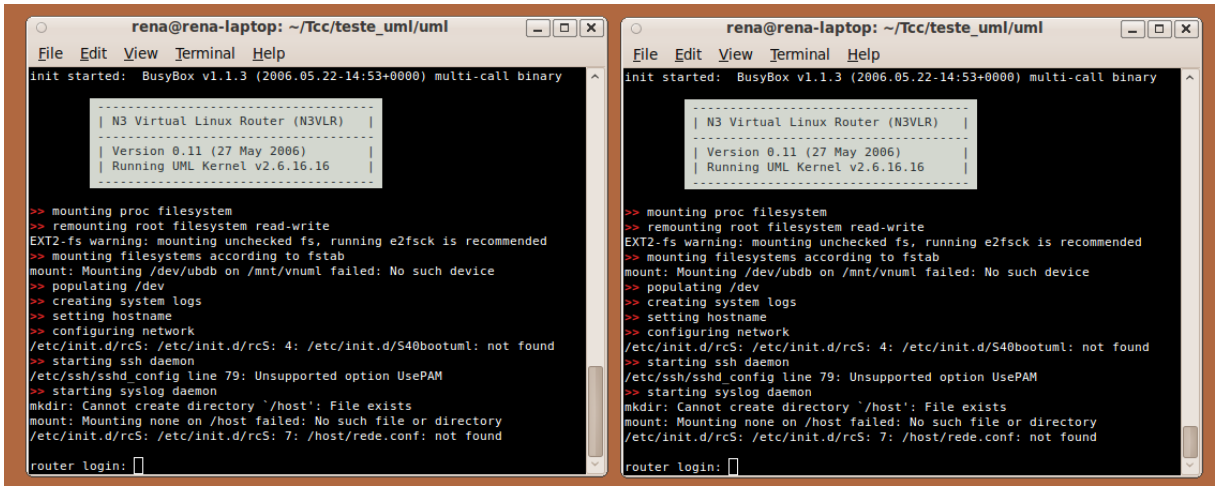

Figura 2.1: Execução de Máquinas UML em hospedeiro.

2.1.2 Arquitetura da UML

Como dito anteriormente, uma máquina UML não trás danos ao sistema hospedeiro. Isto ocorre devido à maneira que sua arquitetura foi formulada. Na figura 2.2 é mostrado como a UML se executa no sistema operacional hospedeiro. Sua execução se iguala a outros processos comuns do sistema, ou seja, é executada como uma aplicação. Contudo, como a UML também possui um Kernel Linux, também possui seus próprios processos internos, os quais não sofrem interferência do Linux hospedeiro. (DIKE, 2005)

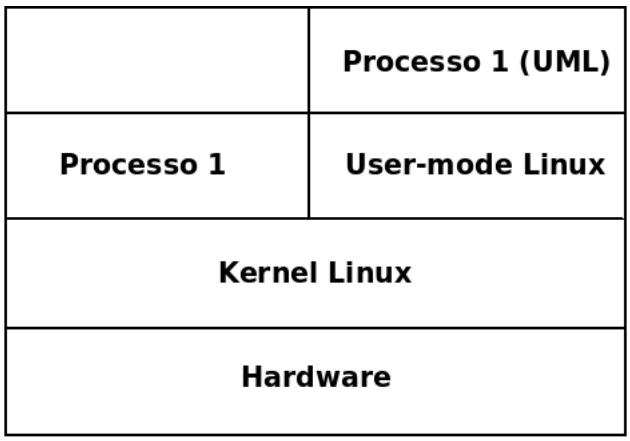

Figura 2.2: Processo de execução da UML no hospedeiro.

2.1.3 Construindo Redes Virtuais com a UML

A UML oferece dois tipos de redes para as máquinas virtuais: uma que permite conectar UML com o hospedeiro e uma outra que permite apenas instâncias UML, denominada de rede virtual isolada. No primeiro tipo de rede as tecnologias de transporte utilizadas pela UML são: *TUN/TAP*, *Ethertap*, *SLIP* e *Slirp*. No segundo tipo: *switch* virtual e *Multicast*. (SILVA; ALMEIDA, 2009b)

A figura 2.3 mostra como a máquina UML se conecta ao hospedeiro através dos dois tipos de redes virtuais.

Devido à necessidade deste trabalho, as redes virtuais isoladas foram utilizadas na simulação dos cenários de rede. Neste caso, para a conexão entre o hospedeiro e a máquina UML, é necessário a criação de uma rede virtual simulando uma rede real. Este domínio é criado no hospedeiro através da aplicação *uml_switch*. Esta aplicação cria um *socket* da família UNIX em forma de arquivo no próprio hospedeiro. Este arquivo é então caracterizado como um domnínio de colisão virtual, e todos que se conectarem a ele farão parte deste domínio. Basicamente, este

socket faz a retransmissão de dados entre as máquinas, podendo se comportar como um *switch* ou como um *hub*.

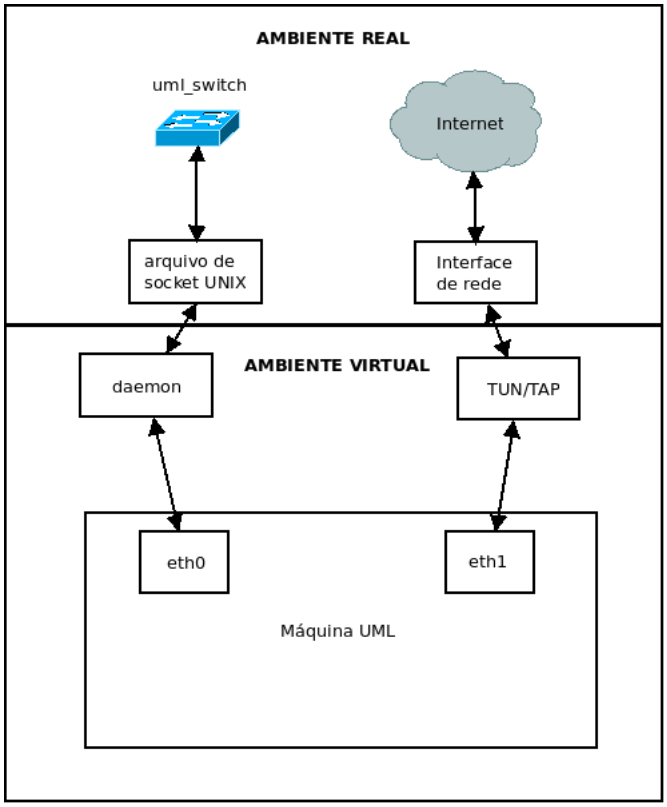

Figura 2.3: Redes Virtuais da máquina UML.

2.2 GUML4MIP - GUI Management console for User Mode Linux for Mobile IP

2.2.1 Visão Geral

Devido ao simples carregamento das máquinas UML, pode ser um pouco exaustivo ter que finalizá-las uma a uma por linha de comando. A partir deste inconveniente, uma aplicação chamada *GUI Management console for User Mode Linux (GUML)* (PALMER, 2005) foi desenvolvida para oferecer um controle gráfico aos terminais UML. Apesar de ser usável, o projeto ainda é muito simples, podendo o usuário apenas carregar e fechar máquinas UML sem ter que fazer por linha de comando. A aplicação GUML foi escrita com a linguagem de programação *Python*[1](ROSSUM, 1990-2010b).

[1] `http://www.python.org`

No ano de 2009, os alunos Cleiber Marques da Silva e Filipe Medeiros de Almeida, trabalhando no seu Projeto de conclusão de curso, criaram uma plataforma para o estudo da mobilidade IP chamada de *GUI Management console for User Mode Linux for Mobile IP (GUML4MIP)* (SILVA; ALMEIDA, 2009a). Esta plataforma tem como objetivo auxiliar estudantes de redes de computadores a entenderem o funcionamento dos protocolos para mobilidade na camada de rede, mais especificamente o *Mobile Internet Protocol version 6 (MIPv6)* (JOHNSON; PERKINS; ARKKO, 2004) e o *Hierarchical Mobile Internet Protocol version 6 (HMIPv6)* (SOLIMAN et al., 2005). Para começar a plataforma, utilizaram o código fonte do projeto GUML, pois ele possui as funções básicas para o sistema. A GUML4MIP utiliza máquinas UML que possuem suporte à mobilidade em seu kernel, sendo que os protocolos são instalados diretamente no sistema de arquivos. Um *switch* móvel virtual (*uml_switch_mobile*) foi criado a partir do código do *uml_switch* para ter suporte à *Virtual Local Area Networks (VLANS)*, tornando possível a mobilidade.

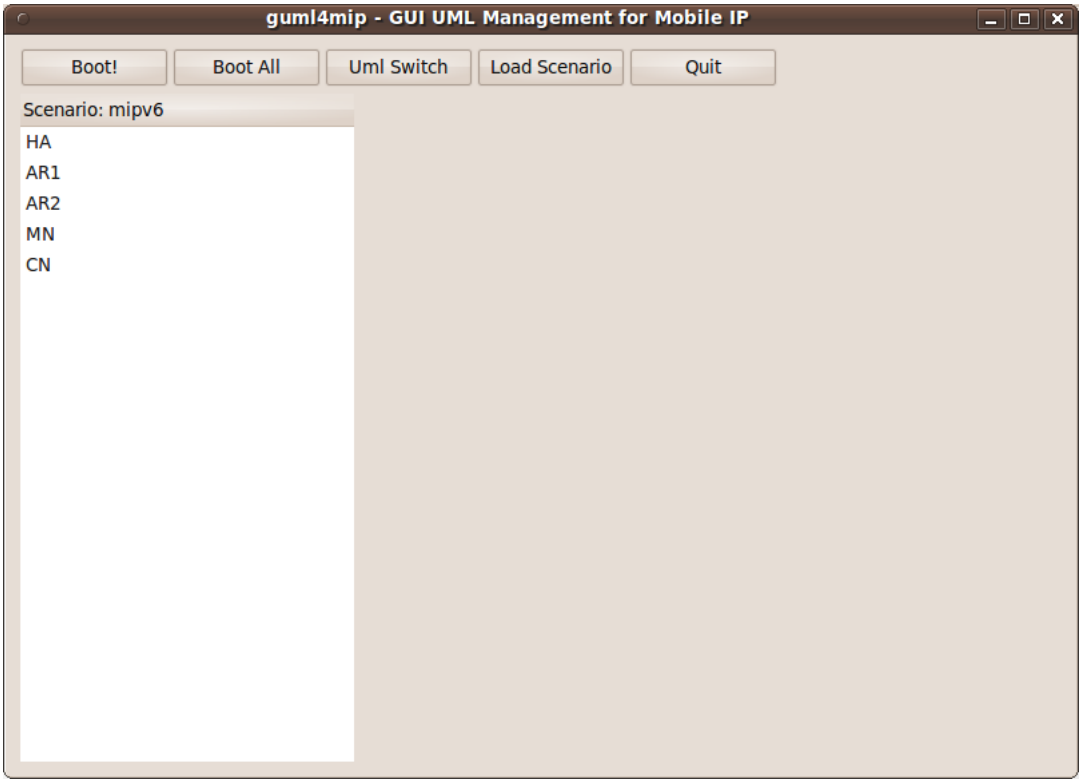

Figura 2.4: GUML4MIP.

Dentre as facilidades da plataforma GUML4MIP, as que mais se destacam são:

- Linguagem de descrição para máquinas UML, ou seja, o usuário edita um arquivo de configuração para cada nó com as seguintes configurações:
  - Nome;

- Tamanho de memória RAM;
- Tipo de máquina: Nó/Roteador;
- Interface de rede;
- Endereço IP/Máscara de rede;
- Socket UNIX ao qual a UML estará conectada;
- Ponto de VLAN;
- Sistema de arquivos;
- Serviços adicionais que serão iniciados no boot da máquina UML;

```
[general]
name = AR2
mem = 128
type = router

[eth0]
type = daemon
mac = random
ip = 2001:d::1/64
socket = /tmp/net.ctl
vlan = 4

[disks]
ubd0 = /usr/share/guml4mip/vm/lenny.img

[daemons]
daemon0 = /usr/local/sbin/radvd -d 10 -C /host/conf/1/radvd.conf.AR2 &
```

Figura 2.5: Linguagem de Descrição de máquinas para GUML4MIP

- Interface gráfica para *uml_switch_mobile*, fazendo com que o usuário além de visualizar o estado do *switch*, possa mover os nós entre VLANs em tempo real;

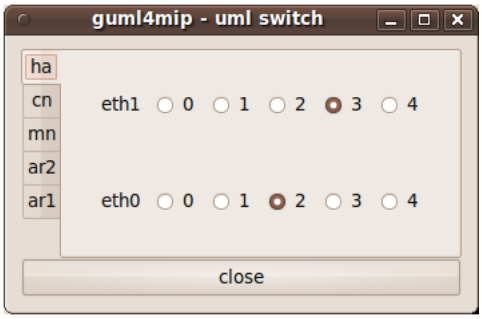

Figura 2.6: Interface Gráfica para UML Switch Móvel

- Captura de pacotes com a aplicação tcpdump;
- Roteamento dinâmico com o Zebra;

2.2.2 Funcionamento da GUML4MIP

O programa GUML4MIP possui um funcionamento bem simples, por esse motivo se torna uma aplicação versátil e facilmente alterável(SILVA; ALMEIDA, 2009b). Na figura 2.7 é possível verificar o diagrama de blocos da aplicação.

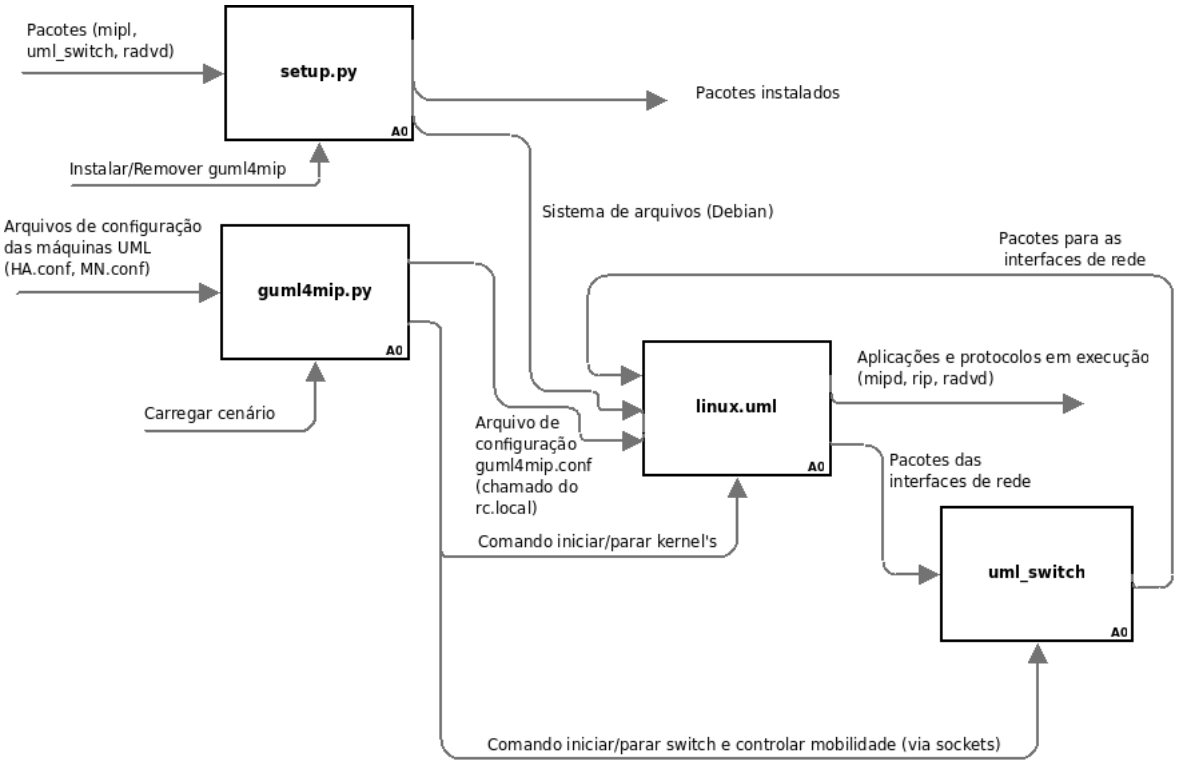

Figura 2.7: Diagrama de blocos da GUML4MIP. (SILVA; ALMEIDA, 2009b)

- **setup.py** - É o instalador e desinstalador da aplicação. Também desempenha as funções de criar o sistema de arquivos utilizado pelo kernel da máquina UML, instalar os pacotes dos protocolos neste mesmo sistema de arquivos e por fim instalar o pacote que contém o *uml_switch_mobile* no hospedeiro;

- **guml4mip.py** - É o programa principal, monta todo o sistema da aplicação e possui a interface gráfica para a interação com o usuário;

- **linux.uml** - É o Kernel da UML, chamado no carregamento de cada nó/roteador, o qual é configurado pelo arquivo de configuração criado pelo usuário;

- **uml_switch** - O *uml_switch_mobile* é inicializado antes do carregamento dos cenários. Após sua inicialização, a interface gráfica de controle para o uml_switch_mobile é criada;

Quando um nó ou um cenário de rede são totalmente carregados no sistema, dois terminais são disponibilizados ao usuário, o terminal ”console”mostrando informações sobre a máquina UML e o terminal ”VC”o qual fica esperando comandos do usuário, basicamente um terminal Linux. Neste ponto, o usuário então pode começar a capturar pacotes nos nós e utilizar o *uml_switch* para efetuar a mobilidade.

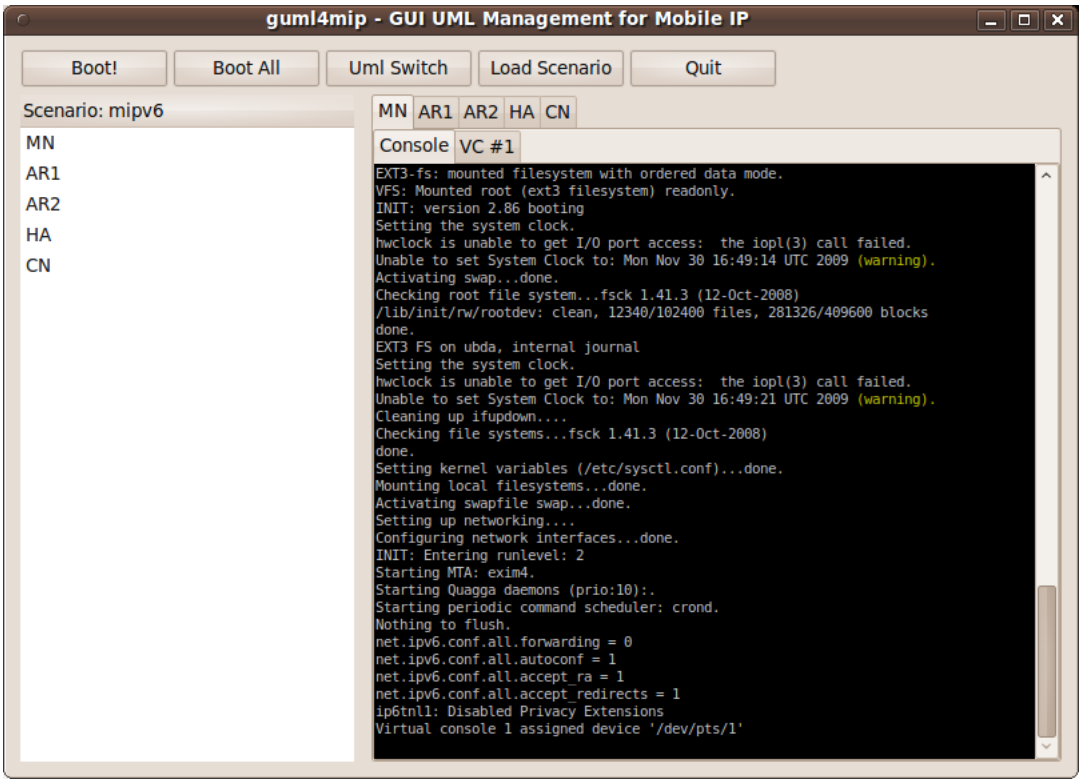

Figura 2.8: GUML4MIP: Execução das UML

2.2.3 Exemplo de Uso da GUML4MIP

Para iniciar os testes com a GUML4MIP, o primeiro passo é selecionar um cenário ou configurá-lo. Como mostrado na figura 2.4, por padrão a plataforma vem com um cenário pronto para testes com o protocolo IPv6 móvel. Este cenário está dividido em 3 redes, a rede do *Correspondent Node (CN)* com o prefixo 2001:a::, a rede do *Mobile Node (MN)* 2001:c:: (sendo que o roteador será o agente domiciliar), e a rede a qual o nó móvel se locomoverá, 2001:d::. Todos os IPs apresentados nos testes estarão de acordo com esta configuração, a qual está sendo ilustrada na figura 2.9. Os IPs globais e locais são convenções utilizadas no *Internet Protocol version 6 (IPv6)*.

Após o botão ”Boot All”ser acionado, todas as máquinas são carregadas deixando o cenário pronto para os testes. Um requisito para se entender o cenário, é conhecer os pontos de

conexão(VLANS).

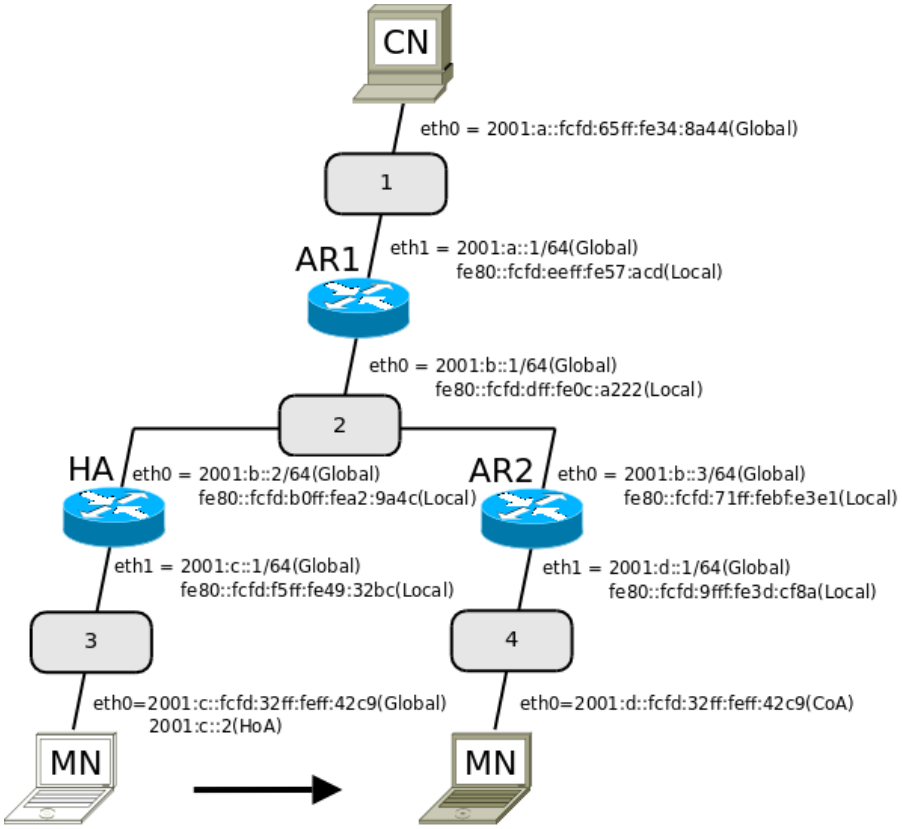

Figura 2.9: Topologia para o cenário do MIPv6 (SILVA; ALMEIDA, 2009b)

```
mn:~# tcpdump -n
tcpdump: WARNING: eth0: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:50:12.914801 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
11:50:14.745859 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
11:50:17.711981 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
11:50:20.630831 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
11:50:20.739501 IP6 fe80::fcfd:f5ff:fe49:32bc.521 > ff02::9.521:  ripng-resp 4: 2001:a::/64 (2)[|ripng]
11:50:23.466932 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
11:50:24.951761 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
11:50:26.075667 IP6 fe80::fcfd:f5ff:fe49:32bc > ff02::1: ICMP6, router advertisement, length 56
```

Figura 2.10: Captura de pacotes no Nó móvel

Para iniciar o teste, deve-se efetuar a captura de pacotes no nó móvel(MN) através da aplicação tcpdump. Com esta captura de pacotes, é possível enxergar as mensagens de Router Advertisement (RA) que são enviadas periodicamente pelo Home Agent (HA)(agente domiciliar) para todos os nós da rede informando quem é o agente com controle sobre o roteamento. Após essa verificação, deve-se então efetuar a mobilidade.

A interface gráfica do *uml_switch_mobile* deve ser carregada e então, deve-se selecionar uma nova posição para o nó móvel como mostra a figura 2.11. Este movimento simula uma máquina se desconectando de uma rede e se conectando a outra.

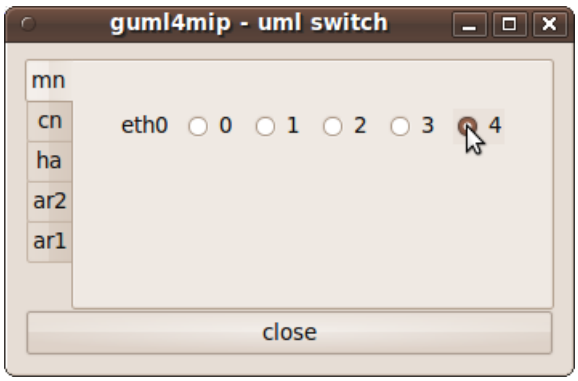

Figura 2.11: Efetuando a mobilidade.

```
12:03:27.686565 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 56) fe80::fcfd:71ff:febf:e3e1 > ff02::1: ICMP6, router advertisement, leng
th 56 hop limit 64, Flags [none], pref medium, router lifetime 9s, reachable time 0s, retrans time 0s[ndp opt]
12:03:27.702239 IP6 (hlim 1, next-header Options (0) payload length: 56) :: > ff02::16: HBH (rtalert: 0x0000) (padn)ICMP6, multicast listener rep
ort v2, length 48, 2 group record(s) [gaddr ff02::1:ff00:2 to_ex { }][|icmp6]
12:03:27.839007 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 24) :: > ff02::1:ffff:42c9: [icmp6 sum ok] ICMP6, neighbor solicitation, l
ength 24, who has fe80::fcfd:32ff:feff:42c9
12:03:28.316380 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 24) :: > ff02::1:ffff:42c9: [icmp6 sum ok] ICMP6, neighbor solicitation, l
ength 24, who has 2001:d::fcfd:32ff:feff:42c9
12:03:29.356114 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) 2001:d::fcfd:32ff:feff:42c9 > fe80::fcfd:71ff:febf:e3e1: [icmp6 sum ok
] ICMP6, neighbor advertisement, length 32, tgt is 2001:d::fcfd:32ff:feff:42c9, Flags [solicited, override] destination link-address option (2),
length 8 (1): fe:fd:32:ff:42:c9   0x0000:  fefd 32ff 42c9
12:03:29.310641 IP6 (hlim 64, next-header unknown (60) payload length: 56) 2001:d::fcfd:32ff:feff:42c9 > 2001:c::1: DSTOPT (padn)(homeaddr: 2001:
c::2)mobility: BU seq#=31290 AH lifetime=262140(padn)(alt-CoA: 2001:0:3100::)
12:03:30.357031 IP6 (hlim 63, next-header Routing (43) payload length: 40) 2001:c::1 > 2001:d::fcfd:32ff:feff:42c9: srcrt (len=2, type=2, segleft
=1, rsv=0x0, [0]2001:c::2) mobility: BA status=0 seq#=31290 lifetime=262140(padn)
```

Figura 2.12: Captura de pacotes.

Após a mobilidade, percebe-se através das mensagens trocadas, que o nó móvel agora recebe *Router Advertisements* de um outro roteador, e não mais do Agente domiciliar. Em seguida o nó móvel envia uma mensagem de *Neighbor Solicitation* para este roteador formando o endereço *Care-of-Address (CoA)*, ou seja, um novo endereço para que seja encontrado dentro desta rede. Agora, com o nó móvel fazendo parte desta rede, seu próximo passo é informar ao agente domiciliar sua nova posição. Ele faz isso, enviando uma mensagem de *Binding Update* para o Agente domiciliar, o qual responde com a mensagem de *Binding Acknowledgement* confirmando o registro deste novo endereço. A mobilidade enfim ocorreu.

Este simples teste mostrou a eficiência da plataforma para o estudo da mobilidade. Várias coisas além do que foi mostrado aqui podem ser visualizadas, como por exemplo o nó móvel efetuando o caminho reverso, o uso de otimização de rota e sem otimização de rota, entre outros.

2.3 Conclusões

Neste capítulo foram apresentados os conceitos básicos associados as máquinas virtuais UML e a forma de construção de redes a partir destas máquinas. Conforme mostrado, a construção de cenários mais complexos de redes exige um esforço adicional na forma de construção

de *scripts* para partida das máquinas, dos *switches* virtuais e das configurações de rede.

Para contornar estes problemas foram criadas plataformas adicionais que facilitam a construção de cenários mais complexos de redes. Inclui-se neste ponto a GUML4MIP desenvolvida com intuito de facilitar o ensino de protocolos de mobilidade.

A execução de cenários ainda mais complexos, que exigem grande quantidade de máquinas pode vir a ser um fator limitante no uso das UMLs. É o caso da simulação de vários domínios administrativos de rede que executam protocolos e aplicações que exigem desempenho da máquina hospedeira. É dentro deste contexto que será mostrado no próximo capítulo um modelo que permite a execução distribuída das UMLs.

3 *Um Modelo para a Execução de Virtualização Distribuída*

Conforme colocado no capítulo anterior, as máquinas UML, por possuírem grandes vantagens, vem se tornando uma ferramenta muito comum entre profissionais da área de redes de computadores. No entanto, devido aos recursos limitados de memória e processamento do hospedeiro, o hardware real pode se tornar um fator limitante para a simulação de um número maior de equipamentos. É com foco neste problema que esse trabalho apresenta um modelo de distribuição de redes virtuais utilizando máquinas UML. O objetivo principal do sistema é, através de uma linguagem de descrição do sistema, alocar partes das redes simuladas em outros hospedeiros disponíveis na rede real.

Neste capítulo, é inicialmente apresentado de forma geral, o mecanismo utilizado para a distribuição de domínios virtuais. Na sequência é descrito um sistema que consiga carregar cenários de rede com suporte à distribuição.

3.1 Modelo para execução de Virtualização Distribuída

3.1.1 Modelo Centralizado

Em condições normais de uso, as redes construídas com as UMLs são formadas de máquinas virtuais, roteadores virtuais e *switches* virtuais, estes últimos chamados de *uml_switches*. Os *uml_switches* se comportam exatamente como *switches* (ou *hubs* dependendo da configuração) físicos, delimitando um domínio *ethernet*. Os roteadores são nada mais do que máquinas virtuais com funções de roteamento habilitadas. Eles se conectam a hospedeiros virtuais através dos *uml_switches*, podendo ser associados endereços de rede IP a todas estas máquinas. A função de um *switch* virtual, da mesma forma que um *switch* real, é receber e encaminhar quadros (*frames*) *ethernet* para os seus destinos que se encontram no mesmo domínio e conectados a uma porta física do dispositivo. Em substituição aos cabos, os *sockets* no domínio UNIX são

utilizados para a conexão com os hospedeiros/roteadores. *Sockets* são basicamente mecanismos de comunicação entre processos.

Neste contexto, a simulação de redes dentro de um único hospedeiro será chamada de "**Virtualização centralizada**"e a Figura 3.1 ilustra este modelo.

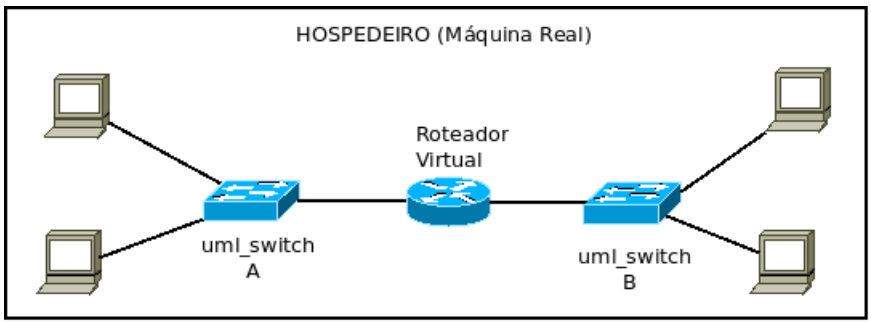

Figura 3.1: Modelo de Virtualização Centralizada.

3.1.2 Modelo Distribuído

A questão para um "modelo distribuído"se resume em como distribuir um domínio *ethernet* virtual entre mais de um hospedeiro, fazendo com que a comunicação entre os terminais deste domínio não seja quebrada. A ideia então é separada em dois tópicos: a distribuição de *switches*, e o uso de um mecanismo que faça a comunicação entre eles.

Distribuição de *switches*

Como o *uml_switch* é o retransmissor de dados padrão de um domínio, a ideia pensada foi de separá-lo em partes e posicionar cada uma nos hospedeiros desejados. Essas "partes"são basicamente outros *uml_switches* que somados representam um só. Como mostrado na figura 3.2, um *uml_switch* A é separado em duas partes, cada uma posicionada em um hospedeiro.

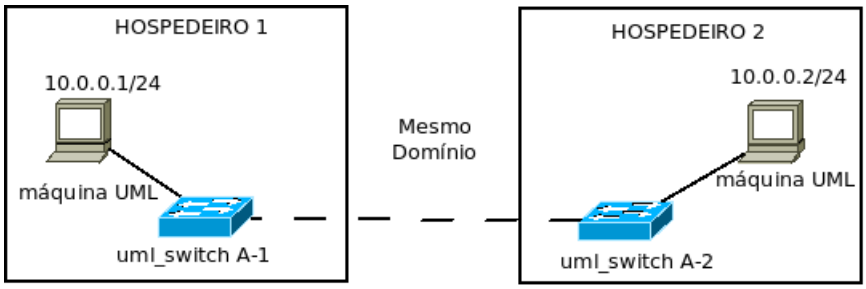

Figura 3.2: Modelo para Distribuição.

O modelo pode ser considerado um "sistema distribuído"que segundo Tanenbaum "Um

sistema distribuído é uma coleção de computadores independentes que aparenta ao usuário ser um único sistema"(TANENBAUM; STEEN, 2007). No nosso caso a coleção de computadores seria a coleção de *uml_switches*.

Ponte de comunicação para os *uml_switches*

Um outro mecanismo deve ser adicionado para fazer a "ponte"entre os *switches*, enviando os dados de um para o outro, viabilizando a comunicação. Este mecanismo deve ter a capacidade de capturar todos os pacotes que saem de um *uml_switch* e enviá-los para outro *uml_switch* em outro hospedeiro, fazendo com que todas as máquinas conectadas ao *switch* se "enxerguem"como se estivessem dentro da mesma rede.

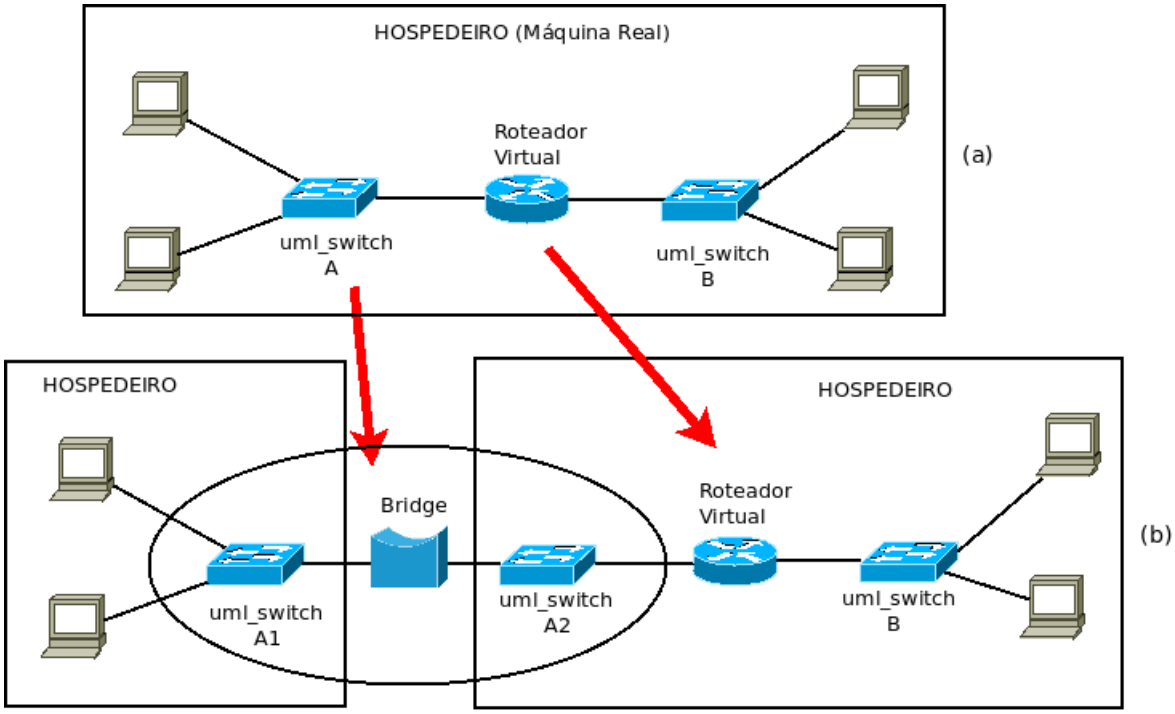

Figura 3.3: Modelo Centralizado versus Modelo Distribuído.

A figura 3.3 b ilustra a versão distribuída da virtualização do mesmo cenário da Figura 3.1. Pode-se observar que o *uml_switch* A é decomposto em dois *switches* interligados por uma *bridge* virtual, que é então o mecanismo utilizado. A função da *bridge* é garantir que todos os hospedeiros conectados ao *switch* A se "enxerguem". A *bridge* é composta por dois ou mais componentes posicionados nas máquinas hospedeiras sobre as quais foi colocado o *switch* distribuído. Cada componente captura todos os pacotes entrantes e saintes do *uml_switch* e envia para o componente no outro hospedeiro e vice-versa. Neste caso, o *uml_switch* deve estar trabalhando no modo *hub* para que todos os nós conectados à ele recebam todos os pacotes por ali trafegados. Como as *bridges* estão virtualmente conectadas nas portas do *uml_switch*, elas não

receberão as informações se o mesmo estiver trabalhando em modo *switch*. O funcionamento da *bridge* é ilustrado na Fig. 3.4.

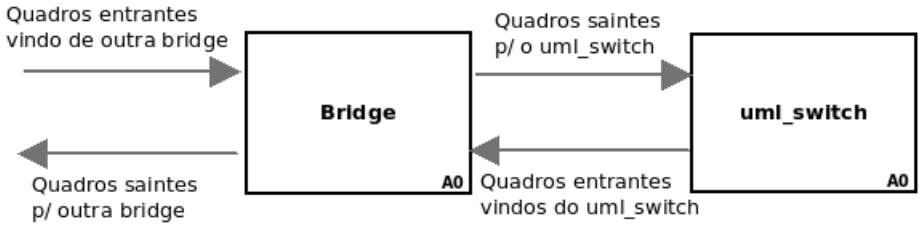

Figura 3.4: Funcionamento da Bridge.

Note que em cenários complexos podem existir vários *switches* distribuídos e um *switch* pode estar distribuído sobre vários hospedeiros. Um cenário mais complexo é mostrado na Fig. 3.5. Neste exemplo, 3 domínios *ethernet* são distribuídos em hospedeiros distintos. Em particular, um domínio (representado por S4) com os roteadores R2,R3 e com o terminal H3 é distribuído em 3 hospedeiros distintos. Uma *bridge* deve interconectar os *switches* resultantes da decomposição de S4 que foi distribuído entre estas máquinas. Como regra geral se coloca: se existem n máquinas virtuais interconectadas através de um *switch* virtual (um domínio *ethernet*) e estas máquinas são distribuídas em m hospedeiros distintos ($m \leq n$) então o *switch* deve ser repartido pelos m hospedeiros sendo associado uma *bridge* com m componentes interconectados.

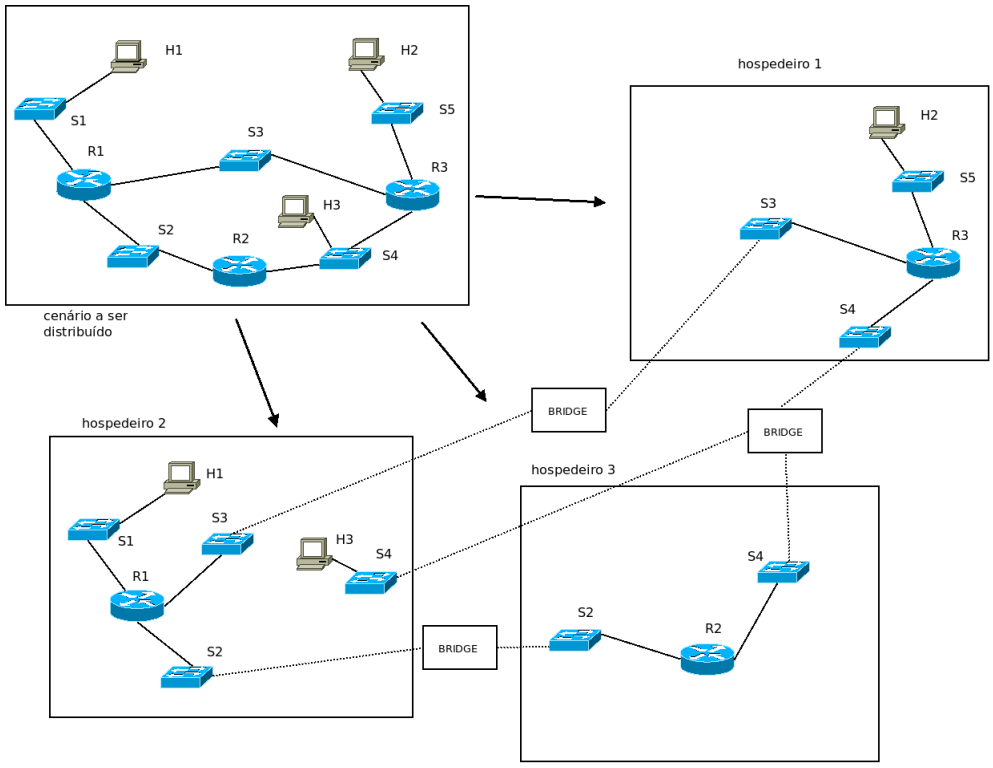

Figura 3.5: Cenário com complexidade adicional.

Na prática, um domínio distribuído em mais de dois hospedeiros necessita de um *switch* central. Esse *switch* se conecta a um número de *bridges* equivalente ao número de partes do *switch* menos um (nºp -1), como mostra na Figura 3.6. Neste exemplo, um *uml_switch* é separado em 4 partes, sendo que o *switch* central se conecta à três *bridges*. Todos os dados provenientes ou com destino aos outros *switches* deste domínio passarão por ele. Para um domínio distribuído em apenas dois hospedeiros, não haverá *switch* central, pois a *bridge* é empregada para comunicação fim-a-fim.

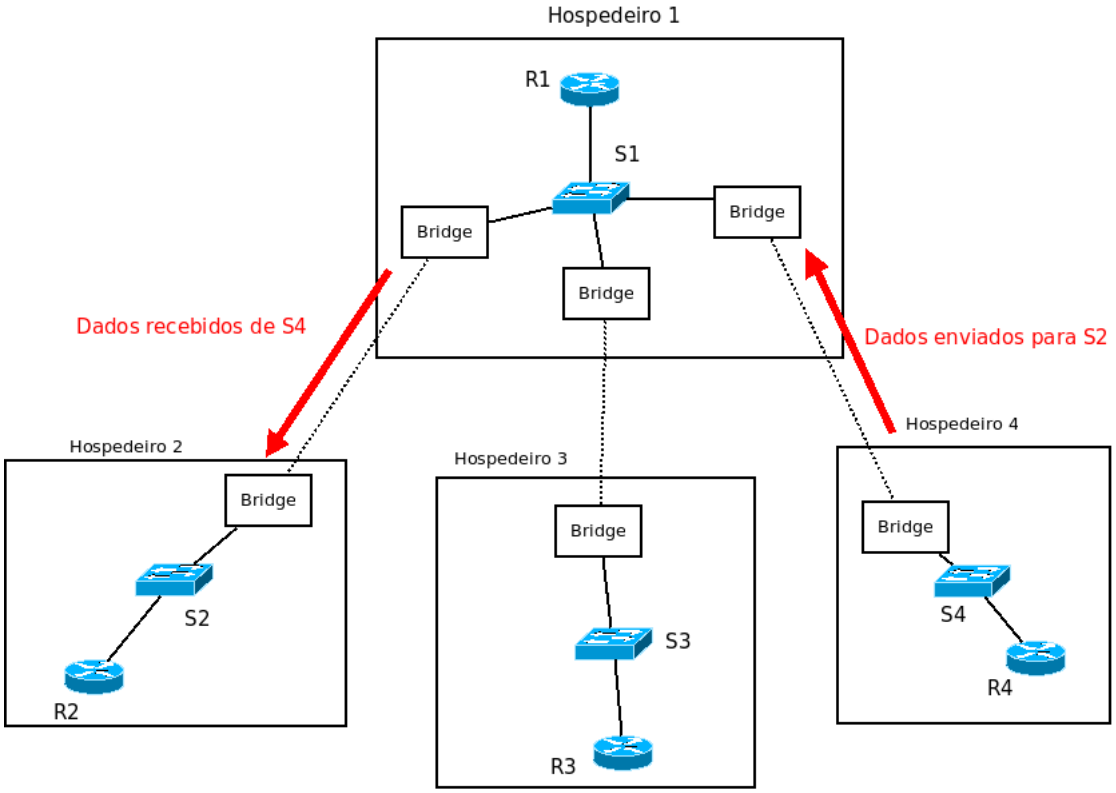

Figura 3.6: Componente central em domínio distribuído.

3.2 Sistema para a execução de cenários com suporte à distribuição de domínios

3.2.1 Visão Geral do Sistema

A execução automática de um cenário com distribuição de domínios é realizada a partir de um suporte que permite ao usuário descrever as máquinas (roteadores, terminais e *switches*), as interconexões entre elas e, na versão atual, de forma manual a alocar máquinas UML em hospedeiros reais. Usando o suporte, o usuário pode verificar em quais hospedeiros de uma rede ele pode carregar o cenário. Uma vez pronta a configuração, o suporte se encarrega de

carregar e executar o cenário distribuído, bem como acompanhar a execução do mesmo.

Utilizar mais de um hospedeiro não significa que haverá um domínio distribuído, a distribuição fica a escolha do usuário, podendo ele criar um cenário com um domínio distribuído como mostra a Figura 3.7a, ou mesmo cenários distintos em diferentes hospedeiros, Figura 3.7b.

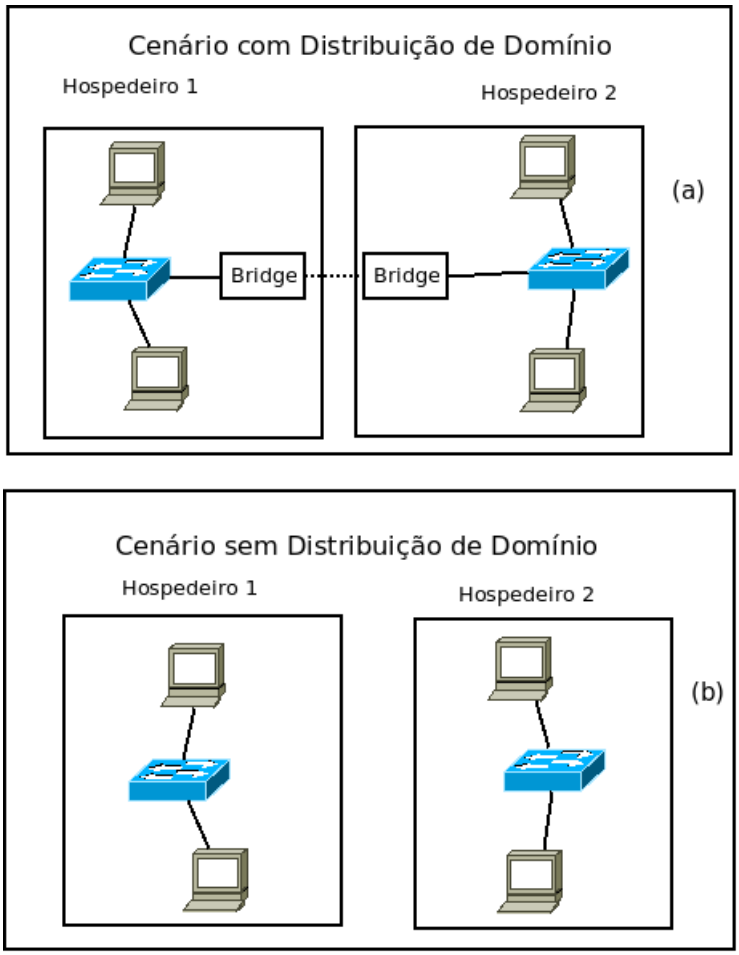

Figura 3.7: Cenário com Distribuição de domínio vs. Cenário sem distribuição.

A plataforma GUML4MIP, descrita no capítulo anterior mostrou-se eficaz no processo de descrição de cenários de rede de forma local. Aproveitando a experiência, esta plataforma foi utilizada para a implementação do suporte de distribuição. Deve-se salientar que a GUML4MIP foi modificada para implementar o modelo cliente/servidor, e com isso a questão da mobilidade foi desconsiderada momentaneamente para que o foco do trabalho ficasse somente no modelo de virtualização distribuída.

Como dito anteriormente, o sistema é dividido em parte cliente e parte servidora. Na parte cliente, chamada de **agente cliente**, o sistema é descrito e, a partir deste agente, é realizada a carga do cenário nas aplicações servidoras, então chamadas de **agentes servidores**. Todo o controle é feito pelo usuário na parte cliente. Em um hospedeiro pode coexistir um agente

cliente com um agente servidor, ou mesmo, mais de um agente servidor.

O processo de execução do sistema pode ser decomposto nas seguintes etapas:

- Descoberta de agentes Servidores;
- Descrição e geração de arquivos de configuração;
- Processo de distribuição e execução do cenário;
- Processo de monitoramento e encerramento do sistema;

O diagrama da figura 3.8 ilustra as principais funções do agente cliente envolvidos no suporte de execução às máquinas distribuídas e a figura 3.9 ilustra as funções do agente servidor. Os processos citados nas figuras serão explicados em seguida.

3.2.2 O Processo de Descoberta de Agentes Servidores

O primeiro passo para a virtualização distribuída é descobrir os agentes servidores disponíveis na rede. Os agentes servidores ativos na rede enviam seus endereços IPs periodicamente para um endereço de grupo *multicast* (função de Anunciação). O agente cliente (função Descoberta de Agentes Servidores) os recebe e cria uma lista com estes IPs, mostrando ao usuário.

Optou-se pelo uso do *multicast* pois é uma maneira para transmissão de dados para múltiplos destinos, que, diferentemente do *Broadcast*, a transmissão é feita para determinados destinos e não para todos conectados em uma subrede (KUROSE; ROSS, 2002). Desta forma haverá intervenção somente nos receptores que estão esperando em um grupo *multicast* específico, reduzindo levemente o *overhead* nos computadores, mas ocupando a rede igualmente.

A função de “Descoberta de Agentes Servidores” no agente cliente basicamente escuta as mensagens no endereço *Multicast*. Cada IP diferente recebido é salvo em uma lista, e paralelamente através do uso de *Threads* (ROSSUM, 1990-2010a) é executado um *Timer* que possui um tempo um pouco maior que o intervalo entre o envio periódico no agente servidor. Se ao final desse tempo não for recebido o mesmo IP novamente, o ”**Descobridor Multicast**”concluirá que o agente servidor com este IP não está mais ativo na rede, e o mesmo será removido da lista de IPs. O pseudocódigo abaixo resume o funcionamento da função.

```
LOOP
    Espera por evento
    CASO evento seja recebimento IP de servidor ENTÃO
```

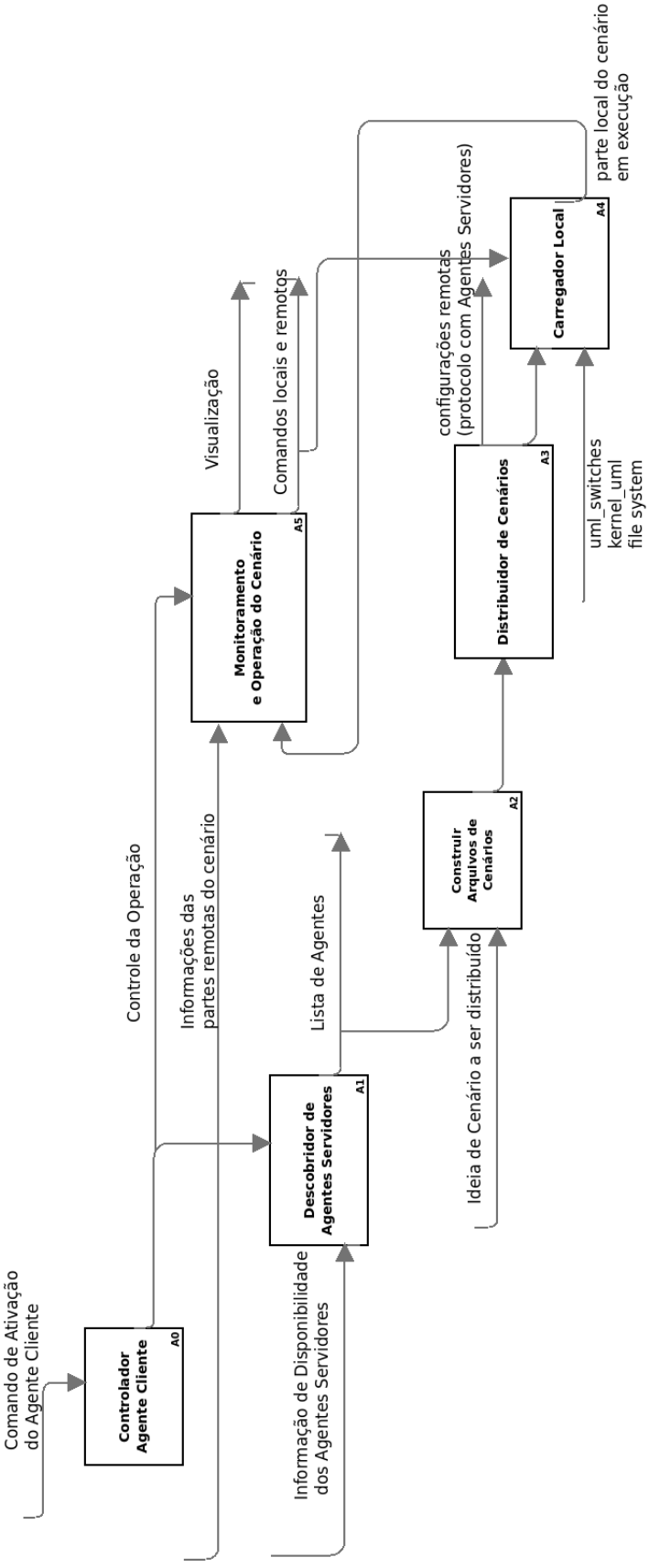

Figura 3.8: Diagrama Funcional do Agente Cliente

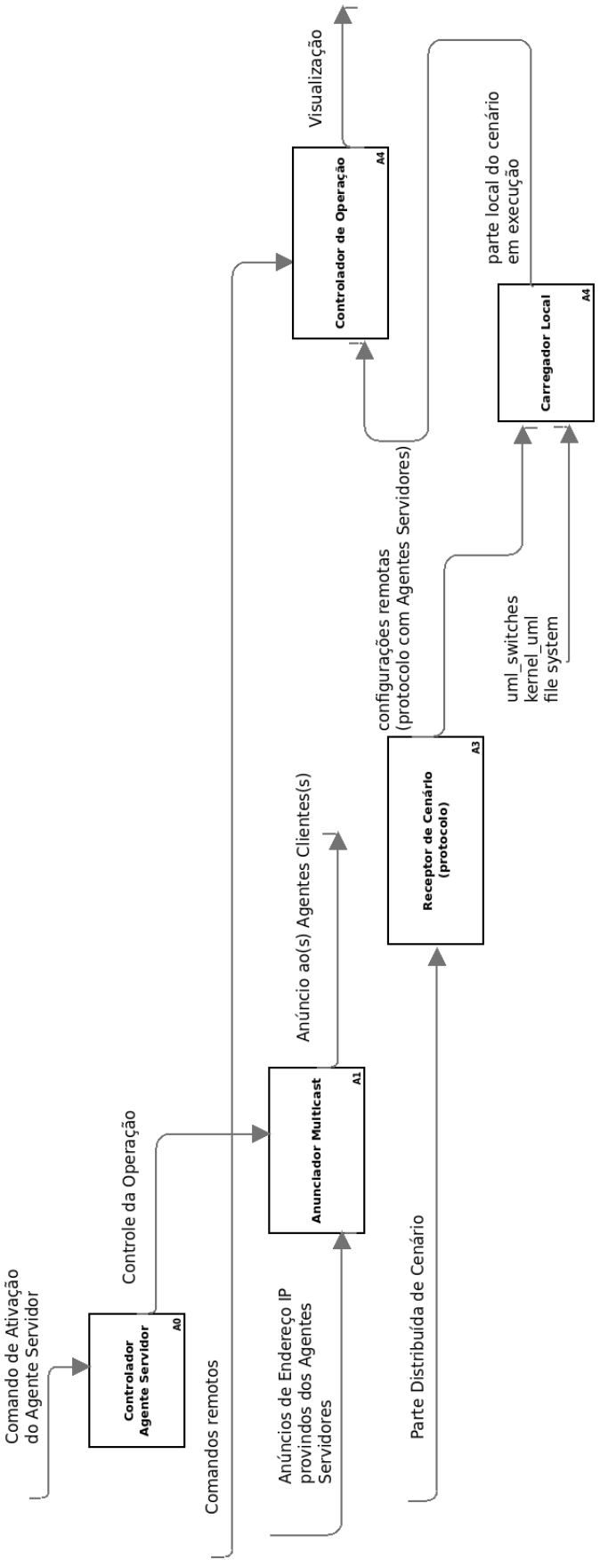

Figura 3.9: Diagrama Funcional do Agente Servidor

```
            SE IP não incluido na lista de IPs ENTÃO
                  inclui na lista de IPs
                  liga timer associado ao IP
            SENAO
                  SE timer ligado para este IP ENTÂO
                       reinicia timer
                  FIM_SE
            FIM_SE
      CASO evento seja estouro de timer ENTAO
            retira ip correspondente ao timer
FIM_LOOP
```

3.2.3 O Processo de Descrição e Geração de Arquivos de Configuração

A ideia principal é fazer com que o usuário que tenha controle sobre o agente cliente possa configurar cenários de rede virtuais e que possa escolher em quais hospedeiros cada máquina ou máquinas UML residirão, ficando a cargo do agente cliente de distribuir as informações e cada agente do cenário de executar suas máquinas UML. Para fazer isso, o usuário precisa editar um simples arquivo de configuração de cenários, escolhendo quais agentes servidores serão utilizados e, dentro de cada agente, os nós da rede juntamente com seus IPs/máscaras. A figura 3.10 ilustra um exemplo para o arquivo de configuração.

```
[local]

name: A1
eth0: 10.0.0.1/24

name: A2
eth0: 10.0.0.2/24

[172.18.22.211]

name: A3
eth0: 10.0.0.3/24
```

Figura 3.10: Modelo de arquivo para configuração de cenário.

Após a seleção dos agentes servidores e a configuração do arquivo de cenários, um interpretador é chamado para ler este arquivo, e então gerar outros na linguagem de descrição de máquinas utilizada pela plataforma GUML4MIP para subir as UMLs, e por fim também são gerados arquivos *shell* para a execução dos *uml_switchs* e das *bridges*.

Interpretador

O interpretador é uma das partes mais importantes para o sistema genérico de distribuição, pois é ele quem lê arquivos de configuração e cria outros informando como a distribuição será executada. Seu trabalho se divide em duas partes. A primeira é de tratar os endereços IPs, separando e classificando redes, a segunda é de gerar os arquivos de configuração para a GUML4MIP, e os arquivos *shell* para subir *uml_switches* e *bridges*.

Os passos do interpretador são descritos da seguinte maneira:

1. Primeiramente, o interpretador lê o arquivo de configuração criado pelo usuário, trata os IPs de cada nó e classifica as redes como não distribuídas e distribuídas:
   - Rede Não Distribuída - Rede em que todos os nós estão dentro do mesmo agente. Este tipo de rede não necessita de *bridge* já que todos os nós estão no mesmo *uml_switch*;
   - Rede Distribuída - Rede em que os nós estão espalhados em agentes diferentes. Este tipo de rede necessita do uso de *bridges*;
2. No próximo passo, é verificado quais destes tipos de rede estão presentes. Se apenas redes não distribuídas, é chamado o método ”Montar redes não distribuídas”, o qual cria os arquivos de configuração na linguagem de máquina para a GUML4MIP e gera arquivos *shell* para carregar os *uml_switches*.
3. Se detectar redes distribuídas, o método ”Montar redes distribuídas”será chamado, e além de fazer o que o outro método faz, cria arquivos para subir as *bridges*.

Os métodos são independentes, podendo ser chamado apenas um como também ambos dependendo dos tipos de redes criadas. Ao final do processo, uma pasta contendo os arquivos de configuração é criada para cada agente com o próprio IP no nome. Os métodos são mostrados na Figura 3.11.

Os arquivos para subir os *uml_switches* e as *bridges* precisam ser devidamente configurados. Os *uml_switches* são executados em forma de arquivo e devem ter nomes diferentes dentro de um mesmo agente.

Para configurar uma *bridge*, as seguintes informações são necessárias: o nome do *uml_switch* que a *bridge* se conectará, o IP do agente o qual ela espera os dados, e a porta desejada para abrir a conexão de espera de dados. Essa configuração deve ser executada em ambos os agentes comunicantes.

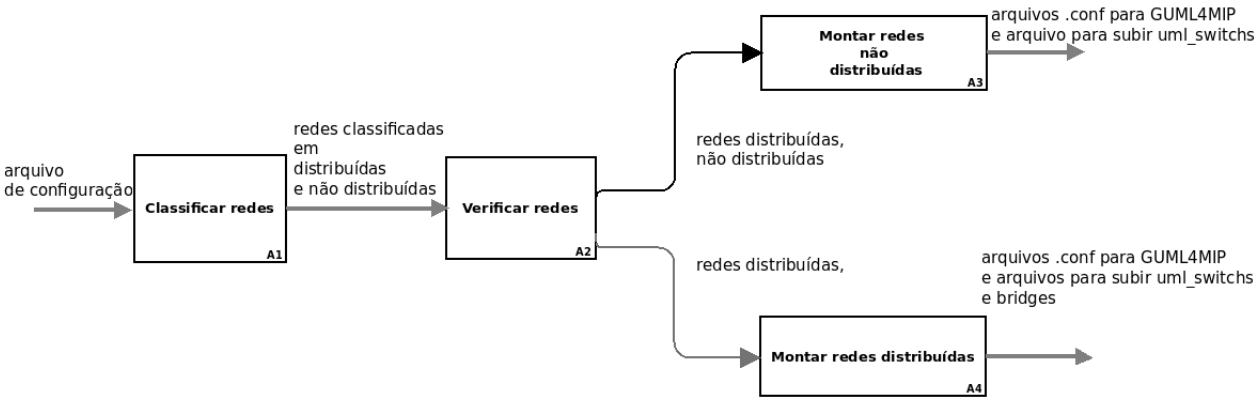

Figura 3.11: Interpretador.

Após estas configurações, os arquivos de linguagem de descrição de máquina utilizados pela plataforma GUML4MIP são gerados, pois já é possível determinar em quais *sockets* eles se conectarão.

3.2.4 O Processo de Distribuição e Execução do Cenário

O último passo antes do carregamento das máquinas UML é o envio dos arquivos para seus respectivos hospedeiros através do "Entregador", pois todos já foram configurados previamente. O entregador é o componente da distribuição com o trabalho mais simples. Ele lê o nome das pastas, que são endereços IPs dos agentes, e então cria *sockets* para esses endereços, assim envia os arquivos de configuração que estiverem dentro da pasta. É esperado que os agentes servidores estejam com seus *sockets* abertos prontos para receberem os arquivos.

Após todos os agentes envolvidos receberem os arquivos de configuração, inicia-se o processo de execução do cenário. Os *uml_switches* são executados primeiramente e logo em seguida as *bridges*, deixando os domínios prontos para receberem conexões de máquinas. Finalizando o processo, as máquinas UML são carregadas sobre os devidos *sockets* UNIX e tem seus IPs configurados.

Figura 3.12: Funcionamento do Entregador.

3.2.5 O Processo de Monitoramento e Encerramento do Sistema

Neste ponto, as máquinas são carregadas e as que estão em agentes servidores tem seu *display* configurado para aparecer no agente cliente. O usuário tem então total controle para fazer testes nos seus domínios virtuais com a distribuição rodando transparentemente. Esta etapa ainda está em projeto, mas pretende-se incluir o monitoramento automático de tráfego nos *switches* e o gerenciamento do encerramento das máquinas de forma ordenada.

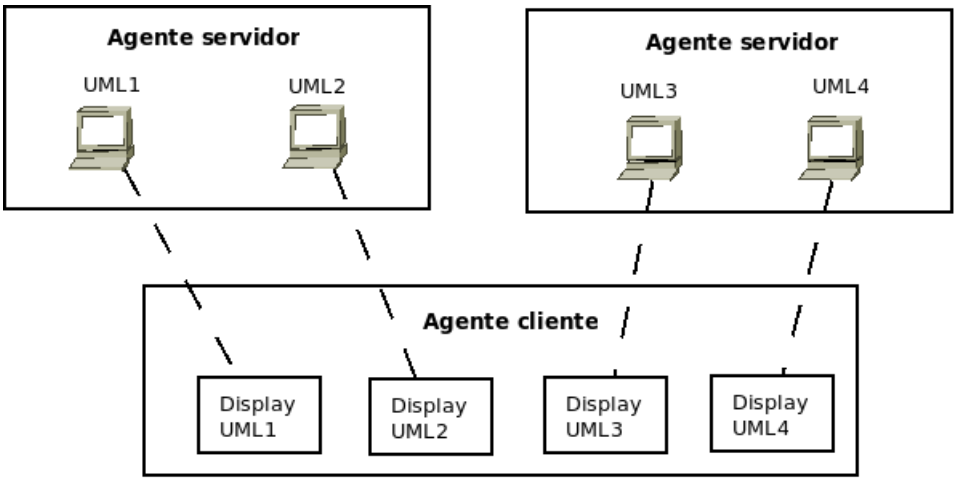

Figura 3.13: Funcionamento dos Displays remotos.

3.3 Conclusões

Neste capítulo, foi apresentado um modelo de virtualização distribuída juntamente com um sistema genérico para o carregamento de máquinas virtuais com suporte à distribuição de domínios. Esse sistema foi separado em quatro processos. Cada processo é dependente do processo anterior, se um não for corretamente executado, o próximo não funcionará. O modelo foi apresentado com o intuito de mostrar que o uso da distribuição de domínios com máquinas UML é viável e pode ser utilizado para a massificação das mesmas.

No próximo capítulo serão apresentados mais informações sobre o sistema, como os detalhes de implementação e os testes finais.

4 *Detalhes de implementação e Testes do Sistema*

No capítulo 3, foi apresentado de forma geral, um modelo para a distribuição de domínios virtuais juntamente com um suporte que facilita a execução de cenários com essa distribuição. Neste capítulo, mais informações sobre o que foi feito serão abordadas, como detalhes de implementação mostrando como cada parte do sistema foi desenvolvida. Ao final, serão apresentados alguns testes realizados, mostrando a eficiência do sistema.

4.1 Detalhes de implementação

4.1.1 Bridge

A *Bridge*, diferentemente do resto dos códigos utilizados, foi feita com a linguagem de programação *C* e não com *Python*. O motivo está nas estruturas as quais o *uml_switch* recebe para o registro de máquinas UML. Este é o único meio em que o *uml_switch* efetua os registros. Então, a *bridge* passa a utilizar desta mesma estrutura em *C* (KERNIGHAN; RITCHIE, 1988), simulando a conexão de uma máquina UML(figura 4.1). Como o *uml_switch* está trabalhando em modo *hub*, ele envia todos os dados para todos que estiverem conectados à ele. A *bridge* como explicado no capítulo 3 simplesmente captura estes dados e repassa para outra *bridge* e vice-versa.

4.1.2 Interpretador

Por padrão, a linguagem de programação *Python*, utilizada na plataforma GUML4MIP, não possui suporte para o tratamento de IPs. Uma biblioteca de código aberto chamada de ipaddr-py(GOOGLE, 2009), desenvolvida pela empresa *Google*[1], foi utilizada por possuir este suporte. A biblioteca, além de conseguir distinguir endereços IPs para terminais(*IPv4Adrress*

[1] `http://www.google.com`

```
struct request_v3 {

        uint32_t magic;
        uint32_t version;
        enum request_type type;
        struct sockaddr_un sock;
};
struct pacote {
        struct {
                unsigned char dest[6];
                unsigned char src[6];
                unsigned char proto[2];
        } header;
        unsigned char data[1500];
};
```

Figura 4.1: Estruturas utilizadas pela bridge.

ou *IPv6Address*) e endereços IPs para redes(*IPv4Network* e *IPv6Network*), consegue responder se um IP está dentro de uma rede ou não. Ela faz isso, transformando um IP de rede em uma lista(*array*) contendo todos os IPs desta rede como mostra na figura 4.2.

```
>>> from ipaddr import *
>>> ip_host1 = IPv4Address('10.0.0.1')
>>> ip_host2 = IPv4Address('10.0.1.1')
>>> ip_network = IPv4Network('10.0.0.0/24')
>>> ip_host1 in ip_network
True
>>> ip_host2 in ip_network
False
```

Figura 4.2: Uso da biblioteca Ipaddr.

1. Nesta linha, a biblioteca é chamada;
2. A variável "ip_host1"é configurada com o IP "10.0.0.1"simulando o IP de um terminal;
3. A variável "ip_host2"é configurada com o IP "10.0.1.1"simulando o IP de um terminal;
4. A variável "ip_network"é configurada com o IP "10.0.0.0"e com máscara "255.255.255.0"(/24) simulando o IP de uma rede com 254 possíveis endereços a serem utilizados. Esta variável se torna uma lista(vetor) contendo todos os IPs possíveis nesta rede;
5. Aqui é perguntado se o IP da variável "ip_host1"está dentro do escopo de IPs da variável "ip_network";

6. É mostrado que sim, o IP da variável ”ip_host1”está dentro da rede criada pela variável ”ip_network”;

7. Agora é perguntado se o IP da variável ”ip_host”está dentro do escopo de IPs da variável ”ip_network”;

8. É mostrado que não, o IP da variável ”ip_host2”não está dentro da rede criada pela variável ”ip_network”;

Esta biblioteca mostra-se eficaz no tratamento de IPs, e então foi utilizada para a programação do interpretador.

Como dito no capítulo anterior, os *uml_switches* são executados em forma de arquivo e devem ter nomes diferentes dentro de um mesmo agente. Como mostra a figura 4.3, por padrão, foi utilizado os nomes */tmp/net0.ctl*, */tmp/net1.ctl* e etc.

```
1 #!/bin/bash
2 /usr/bin/uml_switch -unix /tmp/net0.ctl -hub
3 /usr/bin/uml_switch -unix /tmp/net1.ctl -hub
4 /usr/bin/uml_switch -unix /tmp/net2.ctl -hub
5 /usr/bin/uml_switch -unix /tmp/net3.ctl -hub
6 /usr/bin/uml_switch -unix /tmp/net4.ctl -hub
7 /usr/bin/uml_switch -unix /tmp/net5.ctl -hub
```

Figura 4.3: Arquivo para subir *uml_switches* dentro de um hospedeiro.

Para configurar uma *bridge*, as seguintes informações são necessárias: o nome do *uml_switch* que a *bridge* se conectará, o IP do agente o qual ela espera os dados, e a porta desejada para abrir a conexão de espera de dados. A figura 4.4 ilustra um cenário e logo em seguida são mostrados os códigos *shell* criados pelo interpretador para cada agente.

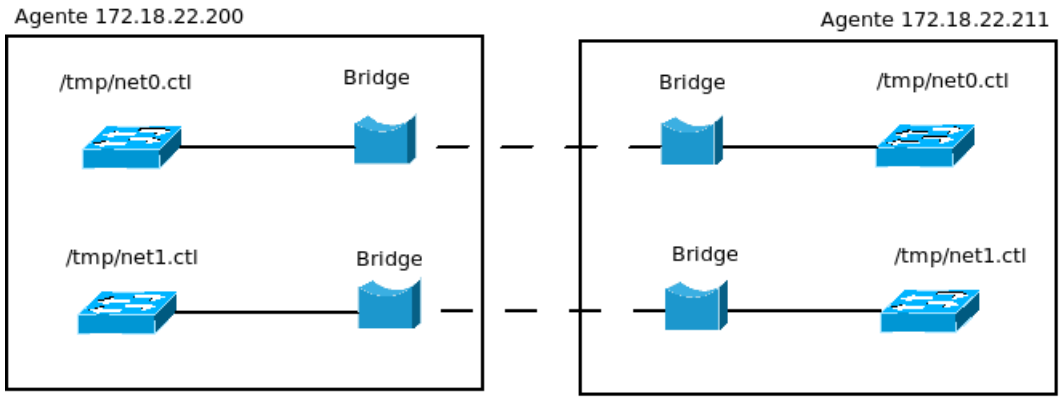

Figura 4.4: Exemplo de cenário.

```
#!/bin/bash
./bridge /tmp/net0.ctl 172.18.22.211 7777
./bridge /tmp/net1.ctl 172.18.22.211 7778
```

Figura 4.5: Arquivo para subir bridges no agente 172.18.22.200.

```
#!/bin/bash
./bridge /tmp/net0.ctl 172.18.22.200 7777
./bridge /tmp/net1.ctl 172.18.22.200 7778
```

Figura 4.6: Arquivo para subir bridges no agente 172.18.22.211.

4.2 Testes

4.2.1 Execução manual de cenário

Primeiramente, para mostrar a eficiência da distribuição de domínios, um teste será realizado com o cenário sendo carregado manualmente. O cenário será este: quatro máquinas UML dentro de um mesmo domínio, mas distribuídas entre dois hospedeiros. O hospedeiro 1 tem o IP 172.18.22.211 e hospeda as máquinas 1 e 2, o hospedeiro 2 tem IP 172.18.22.96 e hospeda as máquinas 3 e 4.

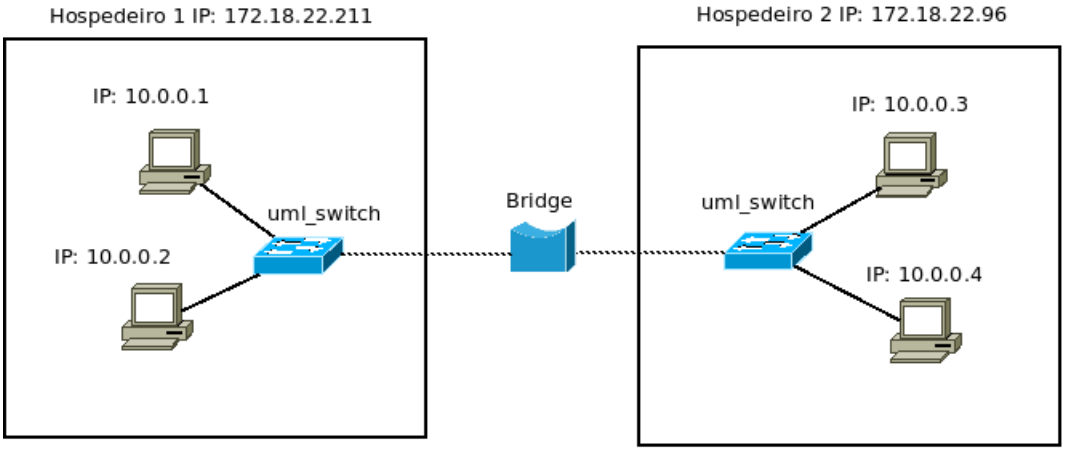

Figura 4.7: Cenário para teste manual.

A seguir, nas figuras 4.8 e 4.9 são mostrados os códigos utilizados em cada hospedeiro para subir o cenário.

```
#Subindo uml_switch
/usr/bin/uml_switch -unix /tmp/net.ctl -hub

#Subindo maquinas UML e conectando-as no switch
./linux umid=MAQUINA1 ubda=./cow,./n3 mem=32M eth0=daemon,,,/tmp/net.ctl
./linux umid=MAQUINA2 ubda=./cow1,./n3 mem=32M eth0=daemon,,,/tmp/net.ctl

#Subindo interface em cada maquina
ifconfig eth0 10.0.0.1/24 up
ifconfig eth0 10.0.0.2/24 up

#Subindo bridge
./bridge /tmp/net.ctl 172.18.22.96 7777
```

Figura 4.8: Códigos utilizados no hospedeiro 1.

```
#Subindo uml_switch
/usr/bin/uml_switch -unix /tmp/net.ctl -hub

#Subindo maquinas UML e conectando-as no switch
./linux umid=MAQUINA3 ubda=./cow,./n3 mem=32M eth0=daemon,,,/tmp/net.ctl
./linux umid=MAQUINA4 ubda=./cow1,./n3 mem=32M eth0=daemon,,,/tmp/net.ctl

#Subindo interface em cada maquina
ifconfig eth0 10.0.0.3/24 up
ifconfig eth0 10.0.0.4/24 up

#Subindo bridge
./bridge /tmp/net.ctl 172.18.22.211 7777
```

Figura 4.9: Códigos utilizados no hospedeiro 2.

Resultados

Na figura 4.10 a máquina 1 que está no hospedeiro 1 consegue enviar mensagens de *echo request* para a máquina 4 que está no hospedeiro 2. A máquina 4 responde com mensagens de *echo reply*.

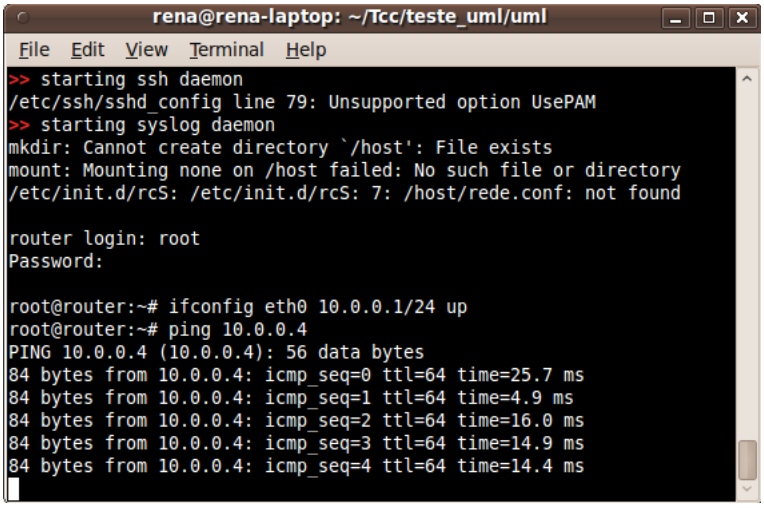

Figura 4.10: Máquinas se comunicando.

Na figura 4.11 a máquina 4 que está no hospedeiro 2 consegue enviar mensagens de *echo request* para a máquina 2 que está no hospedeiro 1. A máquina 2 responde com mensagens de *echo reply*.

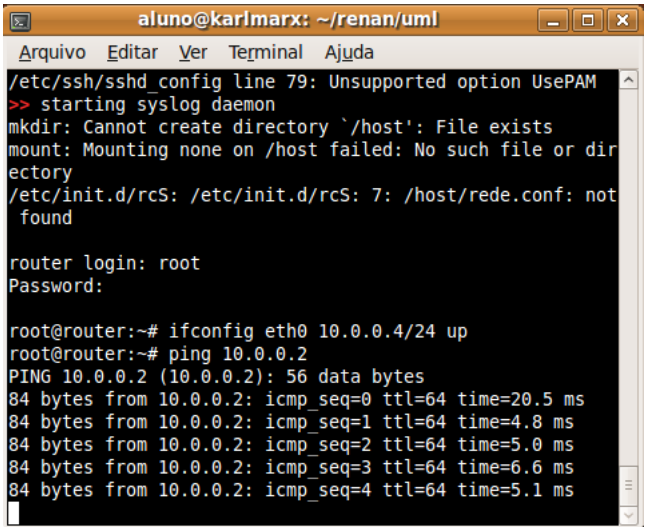

Figura 4.11: Máquinas se comunicando.

A figura 4.12 mostra as mensagens utilizadas pela *bridge* para informar ao usuário que a troca de dados está ocorrendo. A mensagem ”Lendo dados remotos e escrevendo uml”significa que informações estão chegando de outro hospedeiro e ela está repassando para o *uml_switch*.

A mensagem ”Lendo dados uml e enviando ao servidor remoto”mostra que a *bridge* está capturando os dados saintes do *uml_switch* e está repassando para a *bridge* no outro hospedeiro.

Figura 4.12: Máquinas se comunicando.

4.2.2 Execução do cenário com o suporte

A execução do suporte desenvolvido ainda não é totalmente eficaz, tendo o usuário que interferir em alguns pontos do sistema para ter o cenário rodando perfeitamente. Devido ao tempo de projeto , o foco se estabeleceu em mostrar que o mesmo era viável, então, em alguns pontos da implementação, após perceber-se que o que foi desenvolvido era funcional, o próximo passo era iniciado. Alguns pequenos ajustes manuais foram realizados durante o teste, lembrando que o suporte agora está rodando com a plataforma *GUML4MIP*.

Realizando o teste para o mesmo cenário do outro experimento, basicamente o primeiro passo é utilizar o **Descobridor Multicast** para escolher os hospedeiros a serem utilizados. Na figura 4.13, a função é chamada e pode ser verificado que o agente servidor 172.18.22.96 se conectou à rede.

O próximo passo é a configuração do cenário através de um arquivo de configuração, figura 4.14. Os agentes são configurados pelos seus IPs dentro de colchetes ”[]”com exceção do agente cliente que é configurado pelo nome *local* e as máquinas virtuais possuem um nome(name) e uma ou mais interfaces(eth0, eth1...).

Com as configurações feitas, o interpretador agirá, criando arquivos de descrição de máquina para a plataforma GUML4MIP, e os arquivos referentes ao agente servidor são enviados à ele pelo entregador(neste ponto houve intervenção manual pois o entregador ainda só envia ar-

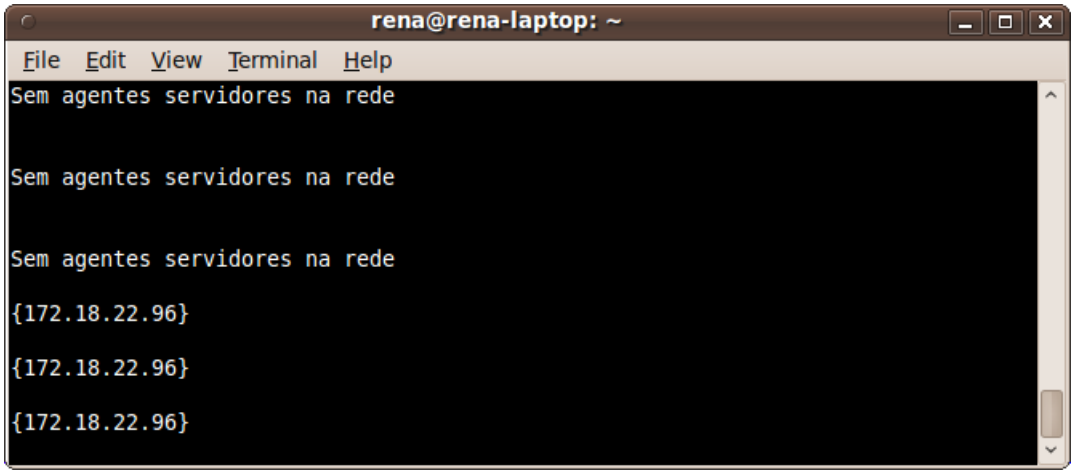

Figura 4.13: Descobridor Multicast.

```
[local]

name: MAQUINA1
eth0: 10.0.0.1/24

name: MAQUINA2
eth0: 10.0.0.2/24


[172.18.22.96]

name: MAQUINA3
eth0: 10.0.0.3/24

name: MAQUINA4
eth0: 10.0.0.4/24
```

Figura 4.14: Arquivo de configuração.

quivos de descrição de máquina, mas não de configuração de *uml_switches* e *bridges*).

Após o **Processo de Descrição e Geração de Arquivos de Configuração** e do **Processo de Distribuição e Execução do Cenário**, os *switches* foram carregados, as máquinas UML subiram e tiveram suas interfaces carregadas sobre os *switches* e seus IPs configurados, e por fim as *bridges* foram executadas. A MAQUINA1 e MAQUINA2 fazendo parte do agente cliente e a MAQUINA3 e MAQUINA4 parte do agente servidor. Para verificar se todas se enxergavam, o mesmo teste de troca de mensagens *ICMP* foi utilizado. Na figura 4.15 as mensagens ICMP são trocadas entre a MÁQUINA4 e a MÁQUINA2.

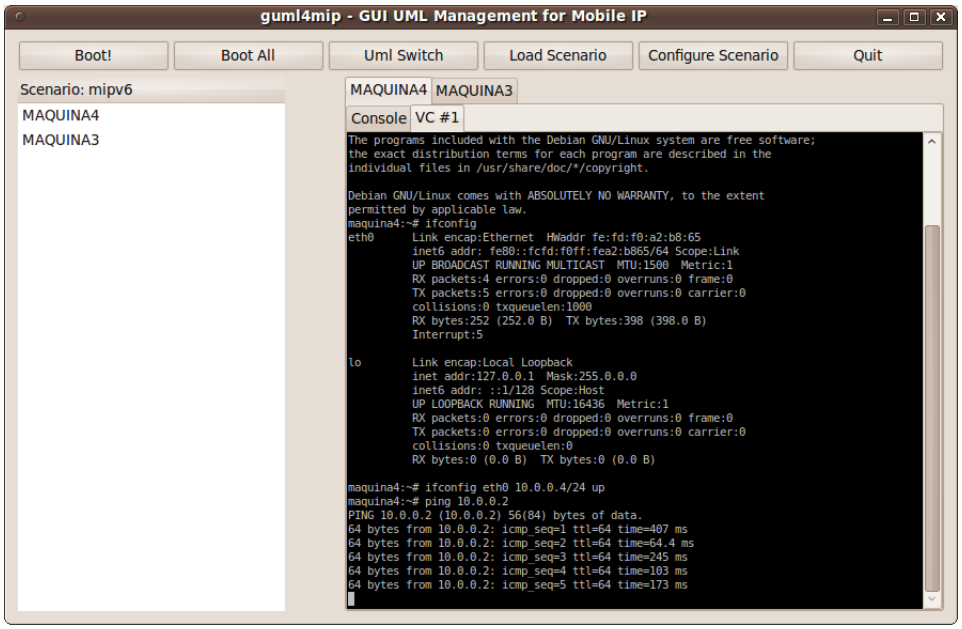

Figura 4.15: Ping entre máquinas.

Ao final do carregamento das máquinas, como o esperado, todas se enxergaram e toda a comunicação rodou de forma que a distribuição ficou transparente, sendo visualizado pelo usuário o cenário da figura 4.16.

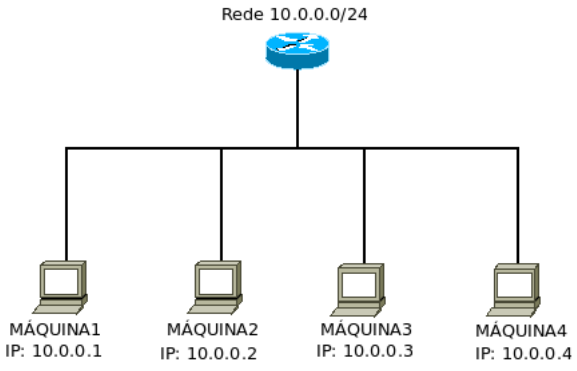

Figura 4.16: Cenário de teste.

5 *Conclusões*

Este trabalho, apresentou um modelo de distribuição para domínios virtuais com o uso de máquinas UML, juntamente com um suporte para a execução automática da distribuição. Já é possível massificar de forma manual as máquinas virtuais para grandes cenários de rede, com o *hardware* de um único hospedeiro não sendo mais o limite da simulação. A funcionalidade dos protocolos utilizados nas simulações não será afetada pela distribuição de domínios, pois os mesmos continuarão a operar da mesma forma que operam em um cenário sem distribuição. O protótipo desenvolvido, apesar de ainda estar incompleto, atingiu os objetivos desejados, pois com ele já é possível gerar mesmo que de forma simples, domínios distribuídos e não distribuídos de forma genérica. Alguns pontos do projeto ainda não foram concluídos e, juntamente com algumas ideias ficam como possibilidade para melhorias em trabalhos futuros:

- **Descobridor Multicast** - Dentro do Descobridor Multicast, um ponto ainda não concluído é o *Timer* utilizado pelo sistema para reconhecer se um agente servidor não está mais disponível na rede. A explicação do seu funcionamento está contida no capítulo 3.

- **Memória RAM para UMLs** - Um dos pontos que devem ser repensados é o uso de memória RAM pelas máquinas UML. Pois foi configurada dentro do interpretador uma *flag* padrão de memória RAM para todas as máquinas. Esta *flag* pode ser alterada, mas talvez não seja a opção do usuário utilizar o mesmo tanto de memória para todas as máquinas, já que algumas podem executar processos pesados e outras processos mais leves. A ideia é de colocar a memória de cada máquina junto de sua configuração no arquivo de configuração de cenários, logo após o ip/máscara e configurar o interpretador para tratá-lo.

- **Lógica do interpretador** - O interpretador já consegue tratar pequenos cenários de forma genérica. Mas devido a necessidade de um extenso código para elaborar um sistema 100% genérico, o foco do trabalho ficou na viabilização do sistema para mostrar que ele é funcional e pode ser utilizado. Por isso, alguns defeitos ocorrem quando se configura grandes cenários, tais como a não configuração dos nós ou a criação errada dos *uml_switches* e das

bridges. Desse modo, o algoritmo do interpretador deve ser recriado em cima de todos os tipos de cenários para que não ocorram mais erros.

- **Processo de Distribuição** - O entregador ainda só envia os arquivos de descrição de máquinas para os agentes servidores, faltando enviar o os *uml_switches* e as *bridges*. Basicamente deve-se utilizar mais alguns *sockets* para os envios.

- **Processo de Monitoramento e Encerramento do Sistema**- Este processo ainda é falho. Uma maneira eficiente de utilizar *displays* remotos não foi desenvolvida, mas como o Linux possui facilidades nessa área, um sistema de *displays* do tipo cliente/servidor pode ser trabalhado. O monitoramento de tráfego nos *uml_switches* deve ser criado de forma gráfica podendo o usuário verificar se há tráfego em excesso em um domínio, ou mesmo se há falta de tráfego. O último passo deste processo é o encerramento das máquinas que pode ser separado via interface gráfica por encerramento de domínios e encerramento total do cenário.

- **uml_switch e uml_switch_mobile** - Outro ponto que pode ser acrescentado no projeto, é a coexistência entre o *uml_switch_mobile* utilizado pela plataforma GUML4MIP e o *uml_switch*, já que o primeiro foi descartado na distribuição de domínios. Isto possibilitaria que o usuário simulasse a mobilidade IP em grandes cenários, podendo mover os nós entre redes distantes para estudar e visualizar a eficiência dos protocolos utilizados.

- **Criação de um mesmo modelo para outras aplicações** - Seria interessante desenvolver este modelo de distribuição de domínios virtuais para outras aplicações que gerenciam máquinas UML, como de exemplo o NetKit (BATTISTA et al.,), uma aplicação consolidada para o estudo de redes de computadores, mas que ainda não possui este suporte.

- **Padrão IEEE 802.1d** - Outra opção seria implementar a *bridge* de acordo com a norma IEEE 802.1d: aprendendo os endereços MAC que estão em cada porta, e fazendo a comutação de quadros de acordo com seus endereços de destino. Assim o *uml_switch* poderia continuar a operar como *switch*.

- **Interface gráfica** - Possibilitar a configuração gráfica da rede, e associar os terminais e linhas aos elementos gráficos.

APÊNDICE A – Casos de uso

Para realçar o entendimento do modelo de virtualização distribuída, os casos de uso desenvolvidos para projetar e implementar o sistema serão apresentados na sequência. Com eles foi possível planejar os processos e verificar futuras incompatibilidades.

A.1 Casos de uso para agentes servidores

Nome: ***Agente servidor***

- **Sumário**: Usuário executa a plataforma GUML4MIP em modo agente servidor.
- **Ator**: Usuário
- **Pré-condição**: Usuário deve estar em um terminal aberto.
- **Descrição**:
  1. Usuário digita no terminal ”./guml4mip -s ethx endmulticast porta id”.
  2. Se sistema detectar argumentos(argv) corretamente, passa para próximo passo.
  3. Sistema tenta capturar IP da interface ethx.
  4. Se interface estiver ativa, aplicação entrará em modo ocioso e nada será mostrado na tela.
- **Alternativas**:
  - Se sistema não reconhecer argumento(argv), mostra o ”Modo de uso”na tela.
  - Se interface não estiver ativa, a aplicação fecha informando o problema na tela.
- **Pós-condição**: Aplicação deve ficar aberta no terminal como se estivesse ociosa e poderá ser fechada com o ”Ctrl+c”ou quando a aplicação cliente envia sinal para terminar processo.

Nome: ***Dois agentes servidores***

- **Sumário**: Usuário executa duas GUML4MIP em modo agente servidor.
- **Ator**: usuário
- **Pré-condição**: Usuário deve ter dois terminais livres.
- **Descrição**:
  1. Usuário digita no terminal ”./guml4mip -s ethx endmulticast porta id”.
  2. Se sistema detectar argv corretamente, passa para próximo passo.
  3. Sistema tenta capturar IP da interface ethx.
  4. Se interface estiver ativa, aplicação entrará em modo ocioso e nada será mostrado na tela.
  5. Usuário abre outro terminal e digita ”./guml4mip -s ethx endmulticast porta2 id2”
  6. Se sistema detectar argv corretamente, passa para próximo passo.
  7. Sistema tenta capturar IP da interface ethx.
  8. Se interface estiver ativa, aplicação entrará em modo ocioso e nada será mostrado na tela.
- **Alternativas**:
  - Se sistema não reconhecer argv, mostra o ”Modo de uso”na tela.
  - Se interface não estiver ativa, a aplicação fecha informando o problema na tela.
  - Se sistema não reconhecer argv, mostra o ”Modo de uso”na tela.
  - Se interface estiver ativa, mas porta já estiver sendo usada, ocorrerá um erro.
- **Pós-condição**: Duas aplicações devem ser terminadas.

A.2 Casos de uso para agentes clientes

Nome: ***Usuário configura cenário***

- **Sumário**: Usuário executa a plataforma GUML4MIP em modo agente cliente, configura um cenário e faz o boot em todos.

- **Ator**: Usuário
- **Pré-condição**: Usuário deve estar em um terminal aberto.
- **Descrição**:
  1. Usuário digita no terminal ”./guml4mip -c endmulticast porta id”.
  2. Se sistema detectar argumento(argv) corretamente, passa para próximo passo.
  3. Janela principal é mostrada, com opções ”Boot”, ”Boot all”, ”Configurar cenário”, ”Carregar cenário”, ”Salvar”e ”Sair”.
  4. IPs junto com os IDs do mesmo grupo de multicast aparecem em uma lista ao lado esquerdo da janela.
  5. Usuário clica em Configurar cenário.
  6. Uma janela de ”Input Dialog”abre, perguntando o nome do cenário e o número de nós.
  7. Usuário digita o nome do cenário, o número de nós e aperta em ok.
  8. Uma nova janela abre ao lado direito, dentro da janela principal, com o nome do cenário e com um número de abas referentes ao número de nós escolhido anteriormente.
  9. Usuário define configurações para cada nó.
  10. Ao clicar sobre IP no canto esquerdo da tela, o mesmo será colado na opção boot.
  11. Usuário clica em Save.
  12. Se campos estiverem preenchidos, uma janela questionará o usuário onde o arquivo será salvo.
  13. Usuário clica em Boot all
  14. O conteúdo de dentro de cada aba será trocado pelo boot dos nós.
  15. Ao terminar o boot, usuário pode interagir com as máquinas virtuais.
  16. Usuário clica em ”Sair”, a aplicação cliente e todas as servidoras fecham.
- **Alternativas**:
  - Se sistema não reconhecer argumento(argv), mostra o ”Modo de uso”na tela.
  - Se algum campo não estiver preenchido, uma janela de diálogo abre informando o problema.

- **Pós-condição**: Processo é terminado.

Nome: ***Usuário Carrega cenário***

- **Sumário**: Usuário executa a plataforma GUML4MIP em modo cliente, e carrega um cenário pronto salvo em um arquivo .conf.
- **Ator**: Usuário
- **Pré-condição**: Usuário deve estar em um terminal aberto.
- **Descrição**:
  1. Usuário digita no terminal ”./guml4mip -c endmulticast porta id”.
  2. Se sistema detectar argv corretamente, passa para próximo passo.
  3. Janela principal é mostrada, com opções ”Boot”, ”Boot all”, ”Configurar cenário”, ”Carregar cenário”, ”Salvar”e ”Sair”.
  4. IPs junto com os IDs do mesmo grupo de multicast aparecem em uma lista ao lado esquerdo da janela.
  5. Usuário clica em carregar cenário.
  6. Uma janela abre perguntando qual arquivo será carregado.
  7. Se arquivo estiver correto, todas as configurações são carregadas na tela ao lado direito.
  8. Usuário clica em Boot all.
  9. O conteúdo de dentro de cada aba será trocado pelo boot dos nós.
  10. Ao terminar o boot, usuário pode interagir com as máquinas virtuais.
  11. Usuário clica em ”Sair”, a aplicação cliente e todas as servidoras fecham.
- **Alternativas**:
  - Se sistema não reconhecer argv, mostra o ”Modo de uso”na tela.
  - Se arquivo não estiver correto, abre a mesma janela novamente
- **Pós-condição**: Processo é terminado.

Referências Bibliográficas

BATTISTA, G. D. et al. Netkit. In: . [S.l.]: http://www.netkit.org.

DIKE, J. *User-Mode Linux*. [S.l.]: http://user-mode-linux.sourceforge.net/, 2005.

GOOGLE. *IPaddr-py - Python IP address manipulation library*. [S.l.]: http://code.google.com/p/ipaddr-py/, 2009.

JOHNSON, D.; PERKINS, C.; ARKKO, J. *Mobility Support in IPv6*. [S.l.]: IETF - http://www.ietf.org/rfc/rfc3775.txt, 2004.

KERNIGHAN, B.; RITCHIE, D. *The C Programming Language*. 2. ed. [S.l.]: Prentice Hall, 1988.

KUROSE, J.; ROSS, K. *Computer Networking: A Top-Down Approach Featuring the Internet*. 2. ed. [S.l.]: Addison Wesley, 2002.

PALMER, M. *GUI Management console for User Mode Linux(GUML)*. [S.l.]: http://hezmatt.org/ mpalmer/guml/, 2005.

ROSSUM, G. V. Higher-level threading interface. In: . [S.l.]: Python Software Foundation - http://docs.python.org/library/threading.html, 1990–2010.

ROSSUM, G. V. Python. In: . [S.l.]: Python Software Foundation - http://www.python.org, 1990–2010.

SILVA, C. M.; ALMEIDA, F. M. *GUI Management console for User Mode Linux for Mobile IP*. [S.l.]: http://code.google.com/p/guml4mip/, 2009.

SILVA, C. M.; ALMEIDA, F. M. *Plataforma para o Estudo de Mobilidade na Camada de Rede*. [S.l.]: Instituto Federal de Educação Ciência e Tecnologia de Santa Catarina - Campus São José, 2009.

SOLIMAN, H. et al. *Hierarchical Mobile IPv6 Mobility Management (HMIPv6)*. [S.l.]: IETF - http://www.ietf.org/rfc/rfc4140.txt, 2005.

TANENBAUM, A.; STEEN, M. V. *Distributed Systems: Principles and Paradigms*. 2. ed. [S.l.]: Prentice Hall, 2007.