

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**Um Ambiente de Middleware para Melhoria de
Desempenho para Sistemas Distribuídos e Paralelos**

Tiago Silva Pereira

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

**TÍTULO: Um Ambiente de Middleware para Melhoria
de Desempenho para Sistemas Distribuídos e Paralelos**

Tiago Silva Pereira

Prof. Dr. Mário Antônio Ribeiro Dantas
Orientador

Banca Examinadora:

Prof. Dr. Antônio A. Medeiros Fröhlich

Prof. Dr. José Mazzucco Jr.

Florianópolis, Julho de 2005

Um Ambiente de Middleware para Melhoria de Desempenho para Sistemas Distribuídos e Paralelos

Tiago Silva Pereira

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação(área de concentração Redes e Sistemas Distribuídos), aprovada em sua forma final pelo Programa de Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Prof. Dr. Mário Antônio Ribeiro Dantas
Orientador

Prof. Dr. Antônio A. Medeiros Fröhlich
Departamento de Informática e Estatística
UFSC

Prof. Dr. José Mazzucco Jr.
Departamento de Informática e Estatística
UFSC

*”A mente que se abre a uma nova idéia jamais
voltará ao seu tamanho original.”
Albert Einstein*

Agradecimentos

Primeiro quero agradecer a duas pessoas que são as minhas bases na vida, meu pai, Luiz Alberto Pereira, e minha mãe, Márcia Silva Pereira. Foram eles que me colocaram aqui onde estou.

Depois não posso deixar de lembrar de minha irmã, Luana Silva Pereira, que apesar das brigas sempre estivemos juntos e nos entendemos muito bem, mesmo estando longe(muito longe) me dá a maior força.

Agora é claro tenho que agradecer a uma pessoa que surgiu em minha vida a pouco tempo, mas que já faz a maior diferença em minha vida. Tai, meu amor, devo muito a você e estou adorando crescer ao seu lado.

Agora, só falta agradecer aos meus grandes amigos, amigos que nunca faltaram ao meu lado e que sabem que são minha família também. Um beijo para todos do Fundão e para todos que fizeram parte de minha vida de alguma forma.

Dedico este trabalho a todos vocês!

Sumário

Lista de Figuras	viii
Lista de Tabelas	ix
Resumo	x
Abstract	xii
1 Introdução	1
1.1 Objetivos e motivação	2
1.2 Visão Geral	3
2 Sistemas Distribuídos, Paralelos e Agentes	4
2.1 O Que São Sistemas Distribuídos?	4
2.1.1 Cluster	5
2.1.2 Grids	7
2.1.3 Single System Image	10
2.2 Programação Paralela	11
2.2.1 Modelos de Programação Paralela	12
2.2.2 Paradigmas de Programação Paralela	16
2.2.3 <i>Message Passing Interface</i> (MPI)	18
2.3 Agentes de Software	19
2.3.1 Classificação dos Agentes de Software	21
2.3.2 Plataformas de Agentes Móveis	22

	vii
2.3.3 Agentes e Objetos de software	24
2.3.4 Paradigmas de computação em rede	26
2.3.5 Plataforma Aglets	31
3 Descrição do Problema	41
4 Descrição da Solução	43
5 Resultados obtidos	47
6 Conclusão e trabalhos futuros	49
Referências Bibliográficas	51
Anexo A - Artigo	55
Anexo B - Agente Jota	59
Anexo C - Código Fonte da Aplicação	62

Lista de Figuras

2.1	Aglomerado Beowulf	6
2.2	Paralelismo de Dados(SIMD)	15
2.3	Esquemático da classificação dos agentes	22
2.4	Ilustração de um sistema de agentes móveis	23
2.5	Paradigma Cliente-Servidor	27
2.6	Paradigma de Execução Remota	27
2.7	Paradigma de Código sob Demanda	28
2.8	Paradigma de Agentes Móveis	29
2.9	Modelo Conceitual da plataforma <i>Aglets</i>	32
2.10	Interface gráfica do <i>Aglet Server</i> , Tahiti	35
5.1	Multiplicação de Matrizes 500x500	47
6.1	Janela Inicial do agente Jota	60
6.2	Janela Principal - JotaMaster	60
6.3	Configuração dos <i>Hosts</i>	61
6.4	Tela de opções	61

Lista de Tabelas

2.1	Tabela de Comparação (Grid x Cluster)	9
2.2	Tabela de Comparação (Objetos x Agentes)	25
2.3	Tabela de Comparação (Paradigmas de programação Distribuída)	30
2.4	Eventos relacionado ao ciclo de vida dos agentes	36
2.5	Métodos da interface <i>AgletContext</i>	38

Resumo

Com o avanço, cada vez mais rápido, das tecnologias de ponta novas classes de problemas surgem no mundo computacional, e é nosso dever buscar meios de solucioná-los. Estes problemas detêm uma grande complexidade, e cada vez mais são de extrema importância para a humanidade. Tomemos como exemplo o projeto de mapeamento do genoma humano, projeto de extrema importância pois pode levar a cura de diversas doenças ainda sem cura e que demanda uma quantidade extremamente grande de cálculos e processamentos, sendo impossível utilizar-se de apenas uma máquina para resolvê-lo.

Desta necessidade, de solucionar problemas cada vez mais complexos, surgiram novas maneiras programação e utilização dos recursos computacionais disponíveis. Sistemas que se utilizam desta nova maneira de tratar dados e processamento trazem consigo uma nova gama de problemas, como já era de se esperar.

Assim ambientes distribuídos e paralelos vêm tomando cada vez mais espaço em pesquisas e aplicações comerciais. Sendo aplicado nas mais diversas formas, tais ambientes têm se tornado uma solução barata e de fácil implementação para problemas cada vez mais complexos e custosos¹.

Este trabalho apresenta uma visão geral destes ambientes, para que o leitor possa estar mais ambientado com o problema, além, também, de um software que tem como objetivo melhorar o desempenho dos mesmos, permitindo ao usuário definir requisitos mínimos para que uma máquina possa fazer parte de uma aplicação específica.

¹Problemas que demandam uma grande quantidade de recursos computacionais para se alcançar resultados em tempo viável e realístico.

Vale ressaltar que este trabalho faz uso de dois trabalhos já realizados como tese de mestrado [1] e como trabalho de conclusão de curso [2]. O objetivo deste trabalho é fazer a análise dos dois trabalhos já feitos e juntar os dois em um único software, que tenha as melhores características de cada um.

PALAVRAS CHAVE: Sistemas Distribuídos, Paralelos, Redes, Recursos

Abstract

With the technological advance, each day growing faster, new classes of problems come up in the computer world, and it is our job to solve them. Those problems are very complex and represent a important kind of problems that need answers, because they are crucial to mankind. For example, we could say the Genome project, which try to map the human's DNA.

From that need to solve complex problems new kinds of computation and programming came up. On the other hand new problems started to appears, what was already predicted.

Distributed and parallel environment came to the news. Each day they are the subject of a very large number of research and commercial applications. They became an easy and cheap way to deal with very complex problems. One reason that the became so popular is the fact that it is not possible to solve some problems in a life time using only one computer.

This work will show a short introduction to distributed and parallel environment so the readers can feel comfortable with the problem. Will show a software that will improve the way that problems can be distributed in a distributed and parallel system, giving to the user the power to select which of the computers connected to the network will be part of the process.

For this work we make use of two other works. The first one is a master thesis [1] and the other one is a work done by a undergraduate student for the University of Brasília [2].

KEYWORDS: Parallel,Network,Resources

Capítulo 1

Introdução

No mundo computacional existe um grande número de problemas de naturezas distintas, muitas vezes não existindo nenhuma relação entre os mesmos. Para este trabalho iremos analisar apenas um pequeno grupo, que são problemas que demandam uma grande quantidade de recursos computacionais, quebra de uma chave de criptografia, por exemplo¹.

Problemas desta natureza podem ser resolvidos de diversas formas. Pode-se, por exemplo, fazer uso de supercomputadores. Supercomputadores são computadores que possuem uma grande capacidade de processamento e armazenamento. Tais computadores demandam um investimento altíssimo em sua aquisição e um custo elevado em sua manutenção, fazendo com que seja, muitas vezes, inviável financeiramente, ou até mesmo, dependendo da demanda de recursos, computacionalmente.

Uma maneira de contornar as impossibilidades financeiras e computacionais é a utilização de computadores interconectados, computadores de uso geral, e que muitas vezes passam a maior parte de seu tempo de operação em estado ocioso. Assim sendo, pode-se aproveitar recursos disponíveis nestas máquinas, que de alguma forma já estão conectadas entre si (pela internet ou uma LAN, por exemplo), utilizando-se assim todos os computadores como se fosse apenas um. Esta técnica é chamada de computação distribuída.

¹Aplicativos do tipo: criptoanálise, fatoração, otimização combinatória, problemas de enumeração, sequenciamento molecular, entre outros.

Além da utilização de computação distribuída, para se ter uma melhor desempenho, muitos utilizam também os conceitos de programação paralela, gerando assim um ambiente totalmente distribuído e paralelo, mas que para o usuário comporta-se como um único computador, um supercomputador, com um altíssimo poder de processamento e armazenamento.

Assim temos de um lado a utilização de supercomputadores de custo elevado e processamento específico e do outro lado a utilização de computadores de uso geral de baixo custo e de fácil disponibilidade. É aqui que este trabalho se encaixa. Será proposto aqui uma ferramenta que irá otimizar os sistemas distribuídos e paralelos, melhorando sua interface e facilitando a sua usabilidade.

1.1 Objetivos e motivação

Este trabalho, embora tenha como meta final possibilitar a conclusão do curso de Ciências da Computação na UFSC(Universidade Federal de Santa Catarina), tem alguns objetivos a serem alcançados. Tais objetivos foram traçados segundo uma perspectiva do autor com o trabalho, e compõem parte da motivação para o mesmo. Algumas das principais objetivos a serem atingidas são:

- **Estudar sistemas distribuídos, agentes móveis, Jota;**
- **Melhorar interface e funcionalidades do Jota;**
- **Trabalhar com a linguagem de programação Java;**
- **Produzir um texto como base para trabalhos futuros relacionados.**

Motivos para querer fazer este trabalho não faltaram. Primeiramente surgiu o desafio, trabalhar com agentes e com sistemas distribuídos, coisas que não haviam sido exploradas por mim durante o curso. Depois, com um pouco de estudo, descobri que era uma oportunidade incrível, trabalhar com algo tão inovador e tão comentado no momento. Terceiro unir o útil ao agradável, fazer o trabalho com algo que gosto bastante, redes de computadores.

Como se pode ver motivos para mim não faltaram. E é com esta empolgação que começo a trilhar por este caminho.

1.2 Visão Geral

O trabalho foi dividido em sete capítulos para um melhor entendimento. A estrutura do trabalho segue.

- **Capítulo 2** descreverá de forma resumida os conceitos básicos para o entendimento do restante do trabalho;
- **Capítulo 3** descrição detalhada do problema a ser tratado por este trabalho;
- **Capítulo 4** tratará do solução proposta de forma detalhada e explicativa;
- **Capítulo 5** fará uma análise dos resultados obtidos e uma crítica dos mesmos;
- **Capítulo 6** conclusões obtidas a partir de testes e sugestões para trabalhos futuros.

Capítulo 2

Sistemas Distribuídos, Paralelos e Agentes

Neste capítulo faremos uma varredura em alguns conceitos importantes para um bom entendimento do problema e a solução aqui proposta. Com isso pretende-se deixar o leitor um pouco mais familiarizado e inserido nas questões abordadas neste trabalho.

Seguirá uma descrição de sistemas distribuídos e suas duas "configurações" mais comumente utilizadas, aglomerado de computadores, ou clusters, e grade de computadores, os conhecidos grids. Veremos alguns exemplos e trataremos em detalhe de uma aplicação em especial, o Jota, que será a base para este trabalho.

Por fim traremos uma breve explicação e descrição de sistemas de agentes móveis, tratando de seu funcionamento, características e ambiente de execução.

2.1 O Que São Sistemas Distribuídos?

Como já visto anteriormente, as pessoas esperam cada vez mais de um computador. Empresas compram supercomputadores para funções extremamente específicas como processamento de dados e imagens, bancos de dados gigantescos, sistemas de armazenamento de alta disponibilidade, entre outros. Porém o avanço tecnológico é

tanto que mesmo um supercomputador hoje pode ser apenas mais uma máquina "rápida" amanhã. Cada dia aplicações diferentes surgem e demandam uma grande quantidade de recursos. Isso torna muito difícil para as instituições estarem sempre atualizadas em seus equipamentos, pois, por se tratar de equipamentos específicos e caros, não são preparados para um larga gama de aplicações e sua substituição num período muito curto torna a sua utilização economicamente proibitiva. Assim, dessa necessidade, surgem os ambientes distribuídos. Segundo definição de Anderson em [1], sistemas distribuídos são um conjunto de estações de processamento (EP) que estão de alguma forma interconectados, por meio de uma *Local Area Network* (LAN) ou por uma *Wide Area Network* (WAN) por exemplo.

Quando se faz uso destes computadores interligados, como se fosse apenas uma supermáquina de processamento, pode-se chamar isto de um sistema distribuído. Se além disto cada máquina na rede faça parte do processamento total de forma paralela e dependente, podemos chamar isto de um sistema paralelo e distribuído. Vale lembrar que nem todo sistema paralelo é distribuído. Existe, por exemplo, computadores com n processadores e tais processadores trabalham de forma paralela, sendo assim possível executar uma aplicação paralela em um ambiente único, centralizado, não distribuído.

Veremos a seguir dois conceitos importantes neste mundo de sistemas distribuídos. O conceito de cluster, ou aglomerado de computadores, e os grids, grades computacionais.

2.1.1 Cluster

Segundo [3] um cluster é o conjunto de computadores distintos que estão conectados e são usados como um computador paralelo, ou como uma forma de redundância para alta disponibilidade e alto desempenho. Os computadores em um aglomerado não são computadores especialmente feitos para esse tipo de aplicação e podem ser usados como computadores isolados. Ou seja, os computadores pertencentes a um cluster não são feitos especificamente para cluster assim como a forma de comunicação, que pode ser, por exemplo, Ethernet.

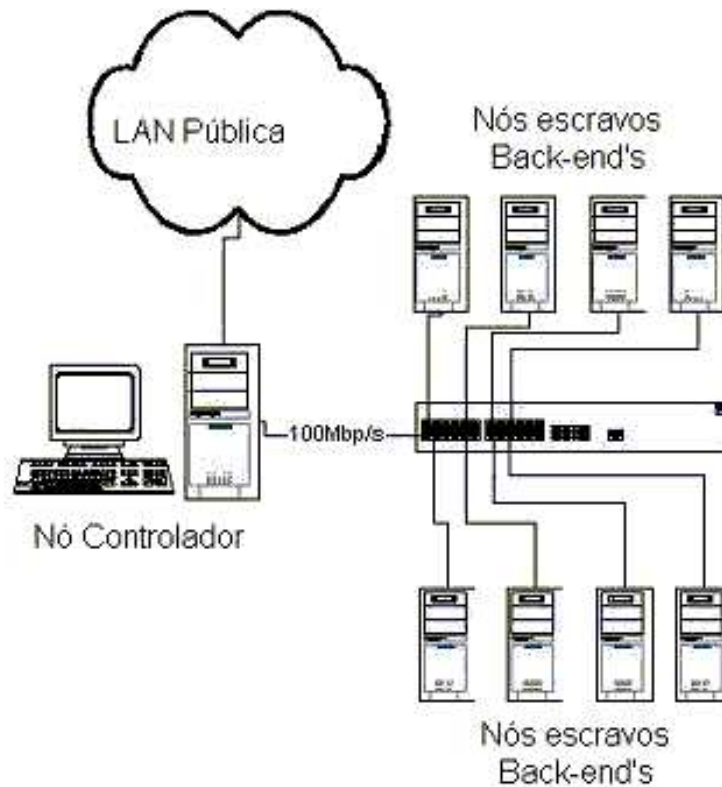


Figura 2.1: Aglomerado Beowulf

Tal definição é bem difundida e bastante utilizada, porém, falta mencionar que normalmente um cluster compreende apenas uma rede administrativa, ou apenas um domínio de rede. Ou seja, as partes que fazem parte do aglomerado estão interligadas em uma pequena área, em uma mesma rede. Existe ainda uma segunda classe de clusters [4], que são clusters, em sua forma mais básica, que são compostos por hardwares específicos, por exemplo, uma forma específica de comunicação. Tais clusters visam ainda mais o aumento de desempenho e para isso especializam um pouco mais o hardware.

Mesmo sendo uma solução para o aproveitamento de hardware e economia, ainda assim os clusters não utilizam os recursos disponíveis sempre, pois as máquinas pertencentes ao cluster são dedicadas, e apesar de ser composto por máquinas workstation, tais máquinas não podem ser utilizadas como tal, já que são de uso exclusivo dos clusters.

Uma solução clássica utilizada por instituições é a utilização de máquinas

rodando linux e software livre com comunicação ethernet, como mostra a figura 2.1. Tal configuração é conhecida como Beowulf Cluster [4]. Com isso empresas conseguem economizar bastante, tendo máquinas aproveitadas e rodando software livre, além de terem o total controle sobre o aglomerado. O Beowulf Cluster, idealizado pela Nasa para processar informações espaciais, é voltado para a computação paralela, assim as aplicações submetidas ao cluster devem ter sido desenvolvidas especialmente para isto. E mais ainda, cada cluster implementa um tipo de comunicação entre os nós e muitas vezes uma aplicação desenvolvida para um cluster A não funcionará no cluster B.

Um outro tipo de cluster, o MOSIX (Multicomputer Operating System for UnIX), também é bastante utilizado. O MOSIX é um cluster desenvolvido para o balanceamento de carga. Uma das suas principais características é a possibilidade de se rodar aplicações feitas para uma máquina normal, sem a necessidade de se modificar nada no código da aplicação. Isto acontece porque os clusters de balanceamento de carga distribuem as aplicações para os nós pertencentes a ele, porém, cada máquina roda uma aplicação por inteiro, e só ela fará o processamento, assim, tarefas não paralelizadas rodam normalmente no MOSIX e sempre retornam os seus resultados a quem submeteu a tarefa.

Em um cluster, todos os nós disponibilizam seus recursos unicamente ao controlador do cluster, que é sempre apenas uma única máquina. Apenas um gerente de recursos é executado no cluster, ou seja, tudo que será alocado a cada nós é responsabilidade do controlador, e esta alocação estará sempre exata, pois este é o gerente de recursos. O que veremos a seguir irá mostrar uma abordagem um pouco diferente a computação distribuída.

2.1.2 Grids

Depois de classificar e detalhar os sistemas distribuídos como cluster, veremos um novo conceito, mais recente e mais evoluído. Uma nova forma de utilização de recursos e computação paralela de alto desempenho e disponibilidade, as grades de computadores, ou grids.

Segundo [5] um tipo de sistema paralelo e distribuído que possibilita

compartilhamento, seleção e alocação de recursos autônomos, geograficamente distribuídos, em tempo de execução dependendo da sua disponibilidade, capacidade, performance, custo e qualidade de serviço requerido pelos usuários é um grid. Ou seja, um sistema paralelo e distribuído, assim como o cluster, que tem a capacidade de dividir recursos com usuários locais.

Com isso, empresas conseguem um alto poder de processamento de dados sem nenhuma alteração em seu conjunto de máquinas. Por exemplo, uma empresa que pretende usar um grid pode utilizar períodos noturnos, momento em que seus funcionários, usuários dos micros da empresa, não estejam utilizando os mesmos, para distribuir tarefas e processar dados em um "supercomputador" recém formado. Diferente dos clusters, os grids não determinam que os recursos devam ser dedicados única e exclusivamente ao compartilhamento. E faz ainda mais, é possível que um usuário determine quanto de cada um de seus recursos ele pretende compartilhar com a grade e em que momentos isto será feito.

Isso diminui ainda mais o custo para se obter um poder de processamento tão grande já que além de serem usadas máquinas normais, estações de trabalho, tais máquinas não precisam de dedicação exclusiva. E os recursos são usados pela grade em momentos que não estão sendo utilizados por seus usuários locais e podendo ser interrompido a qualquer momento pelo mesmo. Porém, isto gera uma infinidade de problemas, que devem ser tratados pelos programadores e administradores dos grids. Garantir a integridade do sistema, garantir a privacidade do usuário, como validar e aceitar os resultados obtidos, o que fazer em caso de perda de controle do recurso, estes são apenas alguns dos problemas gerados pelos grids.

Uma outra característica dos grids é a possibilidade de se utilizar nós em redes completamente diferentes, geograficamente separadas. É assim que muitas das grades atuais funcionam, utilizam a rede da internet para ser o seu canal de comunicação. Iniciativas como a Seti@home[6], distributed.net[7], grid.org[8], InteGrade[9], entre muitos outros. Tais grades funcionam da seguinte forma, o usuário é encorajado a instalar de alguma forma um cliente em sua máquina. Uma vez instalado, o computador fará parte da grade, porém, não necessariamente, o computador estará disponibilizando todos os

Comparativo entre os sistemas		
	Grids	Cluster
Composição dos nós	geograficamente distribuído	rede local
Disponibilidade de recursos	alta variação	baixa variação
Gerenciamento de recursos	descentralizado	centralizado e único
Escalonamento de tarefas	resultado aproximado	resultado preciso
Dedicação dos nós	variável e configurável	total

Tabela 2.1: Tabela de Comparação (Grid x Cluster)

seus recursos. É aí que os grids oportunistas[9], como são chamados, diferem uns dos outros. Alguns disponibilizam, ao usuário, a opção de configurar quanto de seus recursos o usuário pretende compartilhar com a grade. Outros sistemas utilizam os recursos dos usuários somente quando o computador estiver sem uso. Algumas grades inclusive disponibilizam informações sobre o processamento da grade para os usuários em um protetor de tela, por exemplo, assim enquanto o usuário pode ver estatísticas de processamento, a grade utiliza seus recursos, até que o usuário a interrompa.

Assim é possível perceber que grid oportunistas, que utilizam recursos de terceiros e que dependem dos mesmos para aumentar seu poder de processamento, devem tomar cuidado com as suas aplicações clientes. Ou seja, devem ser sempre muito pequenas e transparentes ao usuário. E devem sempre respeitar o poder maior, que é do usuário local. E esse poder de controle se deve ao fato de que, diferentemente aos clusters, cada nó da grade possui seu próprio gerenciador de recursos e o mesmo dá sempre prioridade as aplicações locais.

O Boinc(Berkley Open Infrastructure for Network Computing)[10] é uma plataforma aberta para projetos de processamento distribuído, inicialmente desenvolvido pela Universidade de Berkley, Califórnia. É baseado nesta plataforma que muitas das grades atuais se sustentam, como, por exemplo, a SETI@Home[6], Einstein@home[11], entre outras. Na página oficial¹ do projeto é possível obter informações passo-a-passo de

¹<http://boinc.ssl.berkeley.edu/>

como criar a sua própria grade utilizando o Boinc.

2.1.3 Single System Image

Após pesquisa utilizando a internet algumas definições foram encontradas. Todas, com suas palavras, definem SSI (Single System Image) como sendo a idéia de que os recursos presentes em um Cluster, ou em qualquer outro sistema que agrega recursos, seja apresentado ao usuário com uma única interface. Tornando, assim, transparente ao usuário que se trata de um ambiente distribuído[12]².

O SSI é um sistema que faz com que os recursos aparentem ser da máquina que controla o sistema, assim sendo todas as funcionalidades e variáveis do sistema operacional devem trabalhar de forma a adaptar-se ao novo ambiente. O OpenSSI é um ótimo exemplo de um SSI aberto. Com as palavras do autor³ do projeto OpenSSI iremos defini-lo:

”The OpenSSI project is a comprehensive clustering solution offering a full, highly available SSI environment for Linux. Goals for OpenSSI Clusters include availability, scalability and manageability, built from standard servers. Technology pieces include: membership, single root and single init, cluster file systems and DLM, single process space and process migration, load leveling, single and shared IPC space, device space and networking space, and single management space.”

No caso em questão, várias funcionalidades do sistema operacional, por exemplo, sistema de arquivos distribuídos, apenas um arquivo de inicialização, entre outros estão, implementadas como se fosse apenas uma máquina. Caracterizando-se assim o SSI.

É claro que um cluster com SSI implementado cria muitas vantagens. Toni Cortes[13], membro da Força Tarefa em Computação em Cluster (TFCC) da IEEE

²Não que o usuário não possa ver os recursos separadamente, mas que ele não tenha que procurar em cada máquina esta informação.

³ou conjunto de autores.

(Institute of Electrical and Electronics Engineers)[14] e responsável por SSI, destaca algumas:

- Tira a responsabilidade do usuário de saber onde a aplicação esta rodando.
- Tira a responsabilidade do operador de saber onde os recursos estão fisicamente.
- Simplifica muito a gerência do sistema.
- Reduz o risco de erro por parte do operador.
- Disponibiliza o uso de qualquer recurso transparentemente, escondendo sua localização física.

Toni ainda destaca, ainda, duas funções indispensáveis ao programa para que o mesmo seja caracterizado como SSI. Tais funções são: interface única para o usuário e único gerenciador de tarefas no sistema.

2.2 Programação Paralela

Em sistemas tradicionais cada instrução de um programa é executada de forma seqüencial, uma após a outra, pela Unidade Central de Processamento (CPU). A princípio uma única CPU estava presente nos computadores e não havia suporte a vários processos executando ao mesmo tempo em uma máquina. Porém, sempre existiu a busca por desempenho e poder computacional, para que aplicações mais complexas pudessem ser tratadas com a ajuda dos computadores. Assim surgiu a computação paralela, como uma evolução da antiga computação seqüencial. Tal evolução seguiu o desenvolvimento da sociedade, em que, cada vez mais, as pessoas buscam realizar tarefas diversas simultaneamente.

A idéia básica da computação paralela é dividir o problema em partes menores resolvendo cada parte separadamente, em paralelo, utilizando vários processos distribuídos entre os recursos disponíveis. É importante salientar que, necessariamente, para se obter uma computação paralela real é preciso ter mais de um processador, caso

contrário obtém-se um ambiente pseudo-paralelo, onde o usuário tem a impressão de que seu programa está rodando de forma totalmente paralela, porém, cada processo utiliza a CPU por uma fatia de tempo, e suspende sua execução caso essa fatia acabe, entrando na fila para esperar sua vez de controlar a CPU novamente.

Atualmente a demanda por poder computacional é maior do que o desenvolvimento de hardware, e isso impulsiona o desenvolvimento de novas técnicas de software, como a programação paralela. Neste capítulo, descreveremos modelos, paradigmas⁴ e ambientes da programação paralela.

2.2.1 Modelos de Programação Paralela

Vários modelos de programação paralela podem ser encontrados na literatura, porém, alguns são mais comuns. Tais modelos são uma camada que abstrai a arquitetura do hardware, para que o programador não necessite saber em que tipos de arquitetura seu programa irá rodar. Os modelos mais utilizados são:

- **passagem de mensagem.**
- **threads.**
- **memória compartilhado.**
- **paralelismo de dados.**
- **híbrido.**

Cada um desses modelos podem tratar todos os tipos de problemas que devem ser resolvidos de forma paralela e não existe um melhor que o outro. O que existe são melhores implementações para cada modelo e adequação ao problema. Porém a escolha deve ser feita pelo programador, levando em conta sua intimidade com cada modelo e as suas preferências. Algumas vezes o hardware pode influenciar nesta escolha também assim como as características da aplicação.

⁴Tais paradigmas diferem de outros paradigmas de programação, mas não impede que, por exemplo, um programa escrito sob o paradigma de programação orientada a objetos possa ser também paralelo.

Passagem de Mensagem

Algumas bibliotecas estão disponíveis para este modelo de programação, que consiste em prover primitivas de comunicação como send e receive, transmitindo assim os dados através da rede. Normalmente este modelo de programação é usado em ambientes distribuídos como clusters.

Além das primitivas oferecidas pelas bibliotecas, ainda é disponibilizado ao programador rotinas de inicialização e configuração do ambiente de execução. Cada nó executa um processo, e este processo utiliza apenas memória local da máquina em que está rodando e um processo não tem permissão de escrever na memória de outro processo remoto.

PVM (Parallel Virtual Machine) e MPI (Message Passing Interface) são duas das principais bibliotecas de passagem de mensagem. Veremos um pouco mais a fundo MPI em seções subsequentes.

Threads

No modelo baseado em threads cada processo possui vários caminhos possíveis de execução. Cada thread, ou caminho, será executada de forma concorrente e apesar de terem suas variáveis locais ela compartilha recursos com as outras threads do processo.

Um ambiente multi-thread é criado. Com isso novas preocupações surgem. É necessário garantir que apenas uma thread por vez escreva sobre uma mesma posição de memória, ou seja, apenas uma thread irá alterar uma variável compartilhada, global.

Uma implementação de threads muito usado é o POSIX Threads, conhecidos como Pthreads.

Memória Compartilhada

Este modelo de programação caracteriza a existência de um espaço de memória comum, ou seja, todas as tarefas compartilham o espaço de endereçamento.

Todos os processos podem ler e escrever de um mesmo espaço de memória e isto, assim como no modelo de threads, requer métodos de sincronização, para manter a consistência das variáveis, como semáforos.

Este modelo apresenta vantagens e desvantagens em relação os outros modelos. Uma vantagem é a facilidade em se trabalhar com apenas um espaço de endereçamento e a velocidade de acesso à memória já que estas estão próximas as CPUs. Por outro lado, este modelo tem como principal desvantagem a baixa escalabilidade, pois com o aumento da quantidade de nós no sistema, o barramento de acesso à memória pode congestionar e tornar-se um gargalo do sistema. Outra desvantagem é para o programador, pois este deve, necessariamente, controlar e sincronizar o acesso à memória, para garantir a integridade dos dados lidos. E por ultimo, podemos apontar o alto custo para a fabricação ode máquinas com memória compartilhada.

Paralelismo de Dados

Este modelo é bastante utilizado em aplicações científicas, já que, a maioria destas, manipulam grandes estruturas de dados regulares, o que torna o modelo a melhor opção, sendo o mais eficiente para essa categoria de problema.

A principal característica deste modelo é permitir que tarefas executem, em paralelo, diferentes partes de uma estrutura de dados regular, porem rodando o mesmo código. Por exemplo, é preciso atualizar os valores de um array de cinco mil posições com três vezes o próximo valor no array. Tal tarefa pode ser executada utilizando-se de paralelismo de dados, como mostra a figura 2.2, onde cinco processos são executados em paralelo, todos rodando o mesmo código porém com parâmetros diferentes. Ao final do processamento teremos o novo array, atualizado de forma paralela. A essa característica da-se o nome de SIMD(Single Instruction Multiple Data), já que uma única instrução será executada uma série de valores independentes.

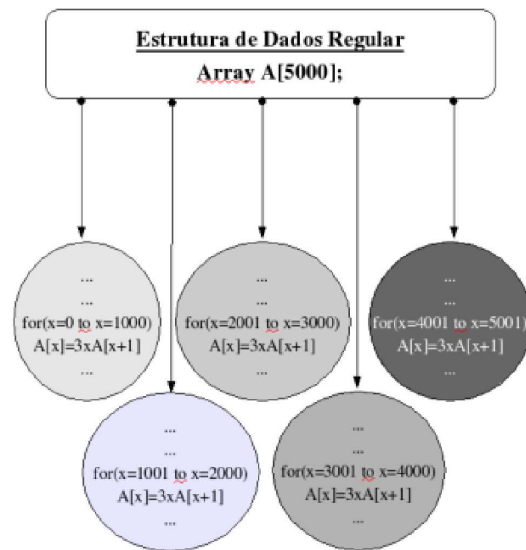


Figura 2.2: Paralelismo de Dados(SIMD)

Modelo Híbrido

Como o nome já sugere este modelo é o agrupamento de dois ou mais dos modelos vistos acima. Algumas vezes um modelo isolado não é poderoso o suficiente para o problema, então os programadores combinam modelos. Um modelo comumente utilizado é a combinação do modelo de passagem de mensagem (MPI) e o de threads, o que possibilita um grande poder de execução paralela ao programador.

Como vimos, brevemente, todos os modelos são soluções possíveis e independentes e a decisão de qual modelo usar cabe quase que exclusivamente ao programador, podendo este utilizar um ou mais modelos ao mesmo tempo. Veremos agora, de forma resumida, três dos paradigmas de programação paralela que os programadores podem usar para estruturar melhor os seus programas.

Memória Distribuída

Este não é exatamente um modelo, porém todos os modelos acima utilizam memória distribuída, exceto o modelo de memória compartilhada. Ao se utilizar o esquema de memória distribuída, em contraposição a memória compartilhada, o pro-

gramador goza de certas vantagens e sofre com algumas desvantagens. As principais vantagens são:

- Alta escalabilidade. O tamanho da memória aumenta conforme novas CPUs são adicionadas ao sistema;
- Ausência de necessidade de sincronização na escrita e leitura de memória;
- Cada processador tem sua memória local e não sofre interferência de outras CPUs .

E algumas diferenças que podem ser apontadas são:

- Tempo de acesso a memória não uniforme;
- A comunicação entre os processadores fica a cargo do programador;
- Dificuldade em mapear estrutura de dados para todas as CPUS.

2.2.2 Paradigmas de Programação Paralela

Assim como modelos, existem vários paradigmas de programação, alguns mais usados, outros nem tanto. É sabendo disto que iremos nos aprofundar um pouco mais em três dos paradigmas, Mestre-Escravo, Dividir para conquistar e Híbrido. Mais uma vez, o programador deverá escolher o paradigma que mais lhe agrada, que melhor representa o problema⁵ e que atenda as restrições do hardware disponível.

Mestre-Escravo

Este paradigma se concentra na idéia de que existem dois tipos de entidades fundamentais no sistema: primeiro o mestre e segundo o escravo. Ao mestre cabe as funções de gerenciamento e coordenação das tarefas, enquanto ao escravo destina-se a função de executor das tarefas, quando ”ordenados”pelos mestres.

Habitualmente o sistema é composto por um mestre e alguns escravos. Assim o mestre recebe a tarefa, passada pelo usuário, e a divide em sub-tarefas, as quais

⁵tipo de paralelismo encontrado.

cada escravo executará uma sub-tarefa. Após isto, o escravo envia os resultados obtidos ao mestre que tem a incumbência de juntar os dados, fazer alguma computação necessária⁶ e produzir o resultado final. Toda a comunicação é feita entre mestre e escravo, nunca entre escravos e escravos ou mestres com mestres, se existi uma comunicação é porque as entidades obedecem a uma hierarquia entre si.

Em se tratando de um sistema distribuído, onde um mestre controla escravos espalhados em máquinas na rede, onde são submetidos a diferentes cargas de trabalho, é vantajoso utilizar-se algum tipo de balanceamento de carga, aumentando assim o desempenho. Pois não é interessante quem sempre alguns escravos recebam tarefas, enquanto outros não recebem tarefa alguma por um período longo de tempo. Dois tipos de balanceamento podem ser aplicados, o estático ou o dinâmico.

O balanceamento estático, as tarefas são divididas e distribuídas no início da computação e não mudam no decorrer do processo. Já no balanceamento dinâmico, as tarefas podem ser redivididas e reorganizadas durante o processamento, caso um escravo tenha saturado seus recursos, ou não é conhecido no início da computação a quantidade de tarefas que são desempenhadas pelo sistema.

As vantagens de se utilizar este paradigmas são: o alto grau de escalabilidade, é muito fácil acrescentar nós ao sistema, e a melhoria de desempenho, ou seja, o *speedup* na aplicação. Em contra partida, o controle é centralizado no mestre, podendo se tornar um gargalo no sistema.

Dividir para Conquistar

Este paradigma já é muito utilizado na programação seqüencial, onde o programador quebra o problema em subproblemas menores e independentes. Com isso o programador ganha modularidade no sistema.

Quando portado para sistemas distribuídos, esta técnica é muito interessante e proveitosa. Cada subproblema é resolvido por um nó da rede, em paralelo, e como os problemas são auto contidos e independentes, não há a necessidade de comunicação

⁶se existir.

entre os nós.

Híbrido

Quando se mistura dois ou mais paradigmas temos o que é chamado de paradigma híbrido. Tal ferramenta é utilizada quando o problema é complexo o suficiente e permite identificar diversos tipos de paralelismo. É uma solução quando o problema é grande e diversificado, já que se utiliza paradigmas diferentes em partes diferentes do problema.

2.2.3 *Message Passing Interface (MPI)*

Veremos agora um pouco mais a fundo este paradigma de programação paralela, pois é de especial interesse, já que é este paradigma que utilizamos na solução do problema tratado por este trabalho.

Tanto o interesse comercial quanto o acadêmico e científico para este paradigma é bastante antigo. Os primeiros computadores paralelos utilizavam este método no desenvolvimento de aplicações. E isso fez com que muitos fabricantes, de hardware, desenvolvessem bibliotecas específicas para seus produtos, dificultando a interoperabilidade.

Assim surgiu a necessidade de se criar um padrão para passagem de mensagens em programas paralelos que fosse independente de hardware. O MPI surgiu desta padronização, definido as especificações das bibliotecas de passagem de mensagem. O MPI prove um ambiente prático, portátil, flexível e eficiente para a programação paralela de alta performance [15]. Por estes motivos o MPI tem se tornado um padrão para a implementação de programas paralelos em ambientes de memória distribuída.

A criação do padrão MPI foi feita com a interação de mais de quarenta organizações⁷ e sua última versão foi disponibilizada em maio de 1994⁸. O MPI foi pensado para ser um mecanismo para a transferência de mensagens tanto ponto-a-ponto

⁷incluindo fabricantes de hardware, programadores e pesquisadores.

⁸www.mcs.anl.gov/Projects/mpi/standard.html

quanto coletivas, tornando-a bastante flexível [16].

Várias implementações do padrão foram surgindo e se tornando conhecidas e disponíveis. Apesar de existir várias implementações todas apresentam um desempenho semelhante, já que implementam o mesmo conceito, o mesmo padrão.

A troca de mensagens serve tanto para transferir dados quanto para sincronizar tarefas. Duas operações básicas para a troca de mensagens são: *send* e *receive*. É necessário também, ser capaz de identificar unicamente cada nó e cada tarefa executada no sistema.

Barney [15] destaca algumas vantagens em se usar MPI em programação paralela, que são:

- padronização: o MPI é um padrão, e não uma tecnologia;
- desempenho: os fabricantes de hardware podem explorar as características do próprio produto para melhorar desempenho;
- portabilidade: não é necessário alterar o código para migrar entre plataformas;
- funcionalidade: existe mais de cento e quinze rotinas definidas no padrão;
- disponibilidade: várias implementações disponíveis, seja domínio público ou proprietárias.

Originalmente o MPI tinha sido projetado para ser usado em ambientes de memória distribuída, porém, atualmente, alguns problemas têm sido resolvidos com este modelo em ambientes de memória compartilhada, como é o caso de problemas envolvendo paralelismo de dados.

2.3 Agentes de Software

Com a utilização cada vez maior de computadores interligados e com o esgotamento de hardware, surge a necessidade de se melhorar o software. Novos conceitos, novas técnicas e novas práticas de programação surgiram, entre elas está agentes de software.

O termo agente é bastante difundido entre diversas áreas do conhecimento, como, psicologia, sociologia, biologia, entre outros. Em cada campo o agente assume uma definição diferente, um pouco particular para a área em questão. Porém a maioria concorda que agente é algo que interage com o sistema, que sente e reage a estímulos. Assim agentes móveis, segundo este conceito, é um agente capaz de se mover em seu ambiente. E é neste caminho que a ciência da computação descreve seus agentes de software. Porém, mesmo dentro deste escopo, algumas diferenças podem ser apontadas, por exemplo, para a inteligência artificial, agentes necessitam de "inteligência", ou seja, são capazes de aprender com seus atos e reagir de formas diferentes a mesmos estímulos dependendo do ambiente. Já para a computação distribuída, um agente precisa ser móvel, e isso os torna reativos ao sistema.

Já que o consenso sobre o que é agente de software esta longe de se realizar, a melhor abordagem ao assunto é o reconhecimento de algumas características dos agentes:

- **Reatividade** - Capacidade de perceber o seu ambiente e responder a mudanças no mesmo.
- **Autonomia** - Habilidade do agente de agir por "conta própria", sem a necessidade de intervenção ou supervisão do usuário. Ser pró ativo e não somente reativo.
- **Colaboratividade** - Trabalhar em conjunto, com outros agentes, para atingir um objetivo.
- **Capacidade de aprender** - Capacidade de melhorar e evoluir.
- **Mobilidade** - Habilidade do agente de se mover na rede que se encontra, de forma objetiva e controlada, ocupando recursos e nós diversos durante o seu ciclo de vida.
- **Racionalidade** - Identificar objetivos impossíveis ou muito custosos e recusar-se a aceita-los.
- **Benevolência** - Adotar objetivos de outros agentes, desde que estes não conflitem com seus próprios objetivos.

Algumas das características apresentadas podem não estar presentes nos agentes, como a mobilidade e a racionalidade. Já outras são obrigatórias, são características indispensáveis a um agente, como a reatividade. Segue uma classificação dos agentes segundo suas características, proposto por Nwana[17].

2.3.1 Classificação dos Agentes de Software

Um agente pode ser classificado segundo suas características. Essa classificação é mais uma ferramenta para definirmos agentes. Segundo Nwana[17] temos as seguintes classes de agentes:

- **Grau de reatividade** - Tal classe diz respeito ao grau de reatividade do agente. Um agente deliberativo tem uma estrutura interna de "pensamento". Ele age segundo uma programação e planos pré-definidos, sempre buscando o seu objetivo. Agentes ditos reativos não possuem um plano ou projeto interno, apenas respondem diferentemente a diferentes estímulos.
- **Mobilidade** - Segundo a habilidade de se mover, um agente pode ser móvel ou estático.
- **Aprendizado e cooperação** - Um agente pode ser classificado como esperto, ou inteligente, se tem a capacidade de aprender e colaborar com outros agentes no sistema. Já agentes que conseguem aprender, são classificados como aprendizes e os que têm a capacidade de colaborar são os colaborativos. Já os agentes com a capacidade de aprender e com autonomia são classificados como agentes de interface. A figura 2.3 mostra de forma gráfica estes conceitos.

Ainda segundo Nwana[17], os agentes podem ser classificados segundo a função que desempenham no sistema, como por exemplo, agentes de informação. Estes agentes são comumente utilizados em aplicações onde é necessário fazer uma busca de informações para ajudar o usuário, como sites de busca na internet⁹, comércio eletrônico, entre outras.

⁹Cadê, Google, entre outros

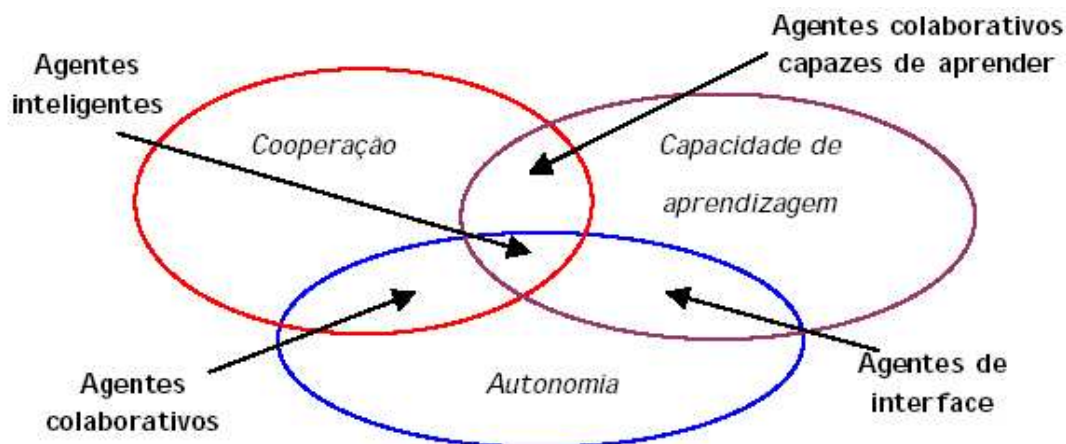


Figura 2.3: Esquemático da classificação dos agentes

E por ultimo podemos classificar um mesmo agente em mais de uma categoria vista, isso o torna um agente híbrido, por exemplo, um agente móvel colaborativo deliberativo é um agente híbrido. Qualquer agente que apresente duas ou mais características apresentadas acima é considerado um agente híbrido.

2.3.2 Plataformas de Agentes Móveis

Para se ter um sistema de agentes móveis completo, duas coisas são necessárias, os agentes, que executam as tarefas, e a plataforma, no qual o agente posso operar, como ilustra a figura 2.4. As plataformas rodam em cada nó da rede que se pretende executar algum agente. Assim é possível descrever um sistema de agentes móveis a partir das características dos agentes em sua plataforma de operação.

Muitas plataformas existem, várias são amplamente utilizadas outras nem tanto. Algumas são extensões de outras feitas em linguagens diferentes. Descreveremos alguns desses sistemas mais utilizados.

- **Aglets:** sistema criado pela IBM do Japão. Tem como principal característica a utilização combinada dos agentes de software com applets Java. Tal plataforma será detalhada mais a frente, já que esta é especialmente importante por ter sido a plataforma escolhida no trabalho.

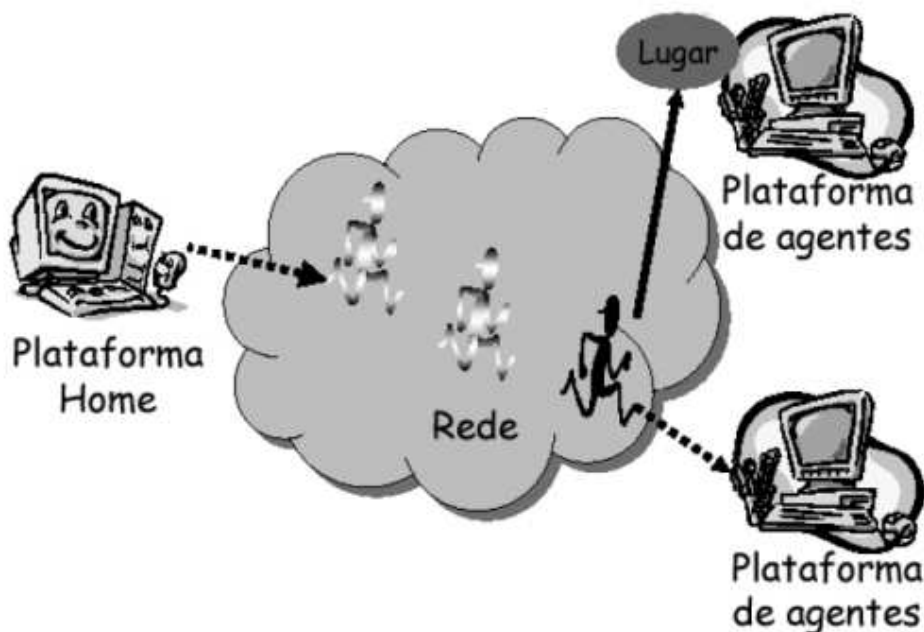


Figura 2.4: Ilustração de um sistema de agentes móveis

- **Agent TCL:** desenvolvido pelo Dartmouth College em linguagem Tcl, daí o nome. Esta plataforma implementa diversos serviços, como comunicação, navegação, segurança, debugging, entre outros. Nesta plataforma é possível migrar todo o estado de execução de um agente, inclusive variáveis locais e um ponteiro para próxima instrução que será executada. Quando um agente deseja se mover na rede ele faz uma chamada à função `agent_jump`, que faz o salvamento de contexto e envia-o ao destino. Ao chegar ao destino o agente consegue se reiniciar a partir do ponto de parada. Este tipo de migração é denominado migração forte.
- **ARA:** também escrito em Tcl pela Universidade de Kaiserslautern, prioriza a segurança e portabilidade da plataforma, possibilitando a execução em uma rede heterogênea.
- **Concordia:** desenvolvido pela Mitsubishi para o gerenciamento e desenvolvimento de aplicações de agentes móveis. É formado por múltiplos componentes Java, que juntos fornecem um ambiente completo de desenvolvimento de aplicação distribuída.

- **Odissey-Telescript:** a empresa General Magic desenvolveu o Telescript, primeiro sistema comercial de agentes móveis, em uma linguagem proprietária, assim como a arquitetura de rede para qual ele foi projetado. Com o advento da Internet e a linguagem Java a General Magic reimplementou seu sistema, adaptado as novas tecnologias, dando origem ao Odissey.
- **Tacoma:** desenvolvido em C¹⁰, utilizando a arquitetura TCP e o ambiente Unix como base. O Tacoma possibilita a utilização de agentes escritos nas linguagens C, Tcl, Perl, Python e Scheme. Geralmente utilizado na resolução de problemas tradicionalmente referentes aos sistemas operacionais.
- **Voyager:** criado pela ObjectSpace, fornece um conjunto de funções para programação distribuída. Possibilita que objetos movam-se pela rede como agentes através da utilização de Object Request Broker (ORB) de Java. Assim é possível que programadores misturem técnicas de programação tradicional e programação distribuída baseada em agentes.

Os sistemas aqui, sumariamente, descritos têm diferenças expressivas em suas arquiteturas e implementações, o que dificulta¹¹ a integração dos sistemas, limitando a solução para apenas um sistema. Algumas iniciativas para a padronização da tecnologia de agentes móveis vêm ganhando atenção. Segundo Ruy Lopes e José Oliveira[18], duas organizações já estabeleceram alguns padrões, são elas: a *Mobile Agent System Interoperability Facility* (MASIF) e a *Foundation for Intelligent Physical Agents* (FIPA).

2.3.3 Agentes e Objetos de software

Várias diferenças estruturais entre objetos e agentes são discutidas por diversos pesquisadores, como mostra a tabela 2.2¹². Para Bigus[19] a diferença fundamental diz respeito à capacidade de agir sozinho, ou seja, autonomia. Não basta ao objeto ser capaz de se mover no sistema para se considerado agentes móveis.

¹⁰linguagem de programação bastante utilizada para problemas relacionados ao hardware

¹¹quase que impossibilita, já que o trabalho para interoperação entre os sistemas é muito grande

¹²Tabela baseada na perspectiva de Bigus [19]

Comparativo	
Objetos	Agentes
definidos pela terminologia de orientação a objetos	Conjunto de entidades autônomas
objetos que são utilizados por outros	Lógica interna complexa e própria
papéis definidos, inflexíveis	papéis dinâmicos, muito flexíveis
em objetos distribuídos, cada objeto contribui de forma independente	agentes cooperam entre si para um objetivo comum
invocação de métodos	comunicação entre agentes
espera por eventos específicos	decisão ativa de quais eventos acionar

Tabela 2.2: Tabela de Comparação (Objetos x Agentes)

Como visto na tabela 2.2 embora um objeto tenha estados e comportamentos, os objetos não possuem capacidade para escolher a ação a ser executada. Eles apenas reagem segundo pedidos de outros objetos, de forma pre-programada. Segundo esta ótica, os agentes desenvolvidos na plataforma *Aglets*¹³ não são agentes de software autênticos, já que não apresentam este tipo de autonomia, são apenas objetos. Porém uma diferença prática é considerada para este trabalho, assumindo assim que objetos criados utilizando o *Aglets* são agentes.

2.3.3.1 Diferença prática

Estas diferenças práticas são apontadas por Danny Lange e Mitsuru Oshima[20], criadores da plataforma *Aglets*[21]. Segundo eles, se em todos os nós existirem servidores que criam o ambiente e existir uma API que definem os métodos, os objetos executados sobre estas características são considerados agentes.

Estes servidores provêm configurações de acesso, segurança, comunicação entre outros serviços do sistema. Logo se não for necessário, ao sistema, nenhum servidor ou API, e não implementa, na prática, características de autonomia, tal sistema é consid-

¹³Plataforma utilizada neste trabalho

erado apenas um sistema distribuído orientado a objetos.

Seguindo estes conceitos, um agente de software é um objeto que:

- situa-se em ambiente de execução;
- propriedades reativas;
- podem apresentar características de cooperação, mobilidade e aprendizado.

2.3.4 Paradigmas de computação em rede

Para se entender melhor as vantagens da utilização de agentes móveis para aplicações distribuídas, devemos compará-la aos atuais paradigmas de computação em rede. Assim podemos traçar um paralelo entre os paradigmas tradicionais e as baseadas em código móvel.

Veremos três paradigmas clássico de programação em rede baseando-se na perspectiva de Fuggetta [22].

Paradigma Cliente-Servidor

Amplamente utilizado, este paradigma, ilustrado pela figura 2.5, contém um componente computacional, chamado servidor, que oferece serviços a rede. Todos os recursos e código necessário para execução do servidor encontram-se na mesma máquina em que o servidor está "rodando". Um outro componente de software, o cliente, localizado em outro computador, que não do servidor¹⁴, solicita tarefas a serem executadas pelo servidor. Como resposta a solicitação, o servidor processa o pedido com seus recursos próprios e envia o resultado do processamento de volta ao cliente. Alguns exemplos de aplicações que utilizam este paradigma: páginas dinâmicas¹⁵, RMI, entre outros.

¹⁴via de regra. Porém existem sistemas cliente-servidor em uma mesma máquina.

¹⁵script interpretados no servidor

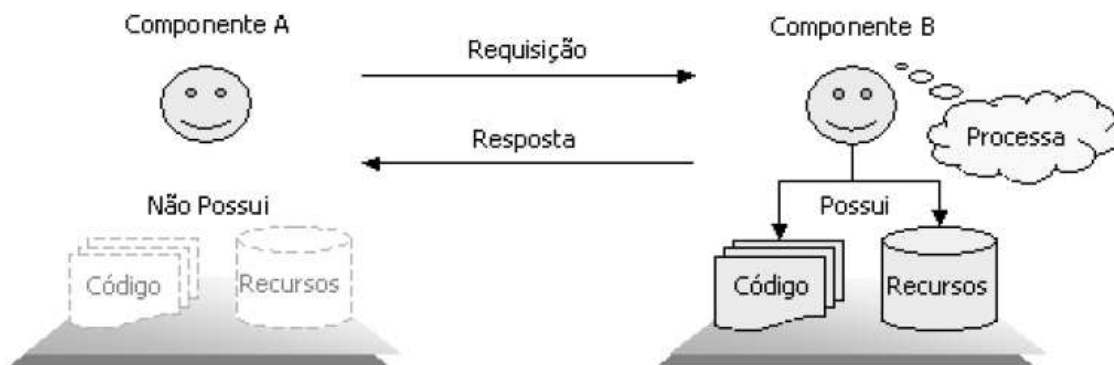


Figura 2.5: Paradigma Cliente-Servidor

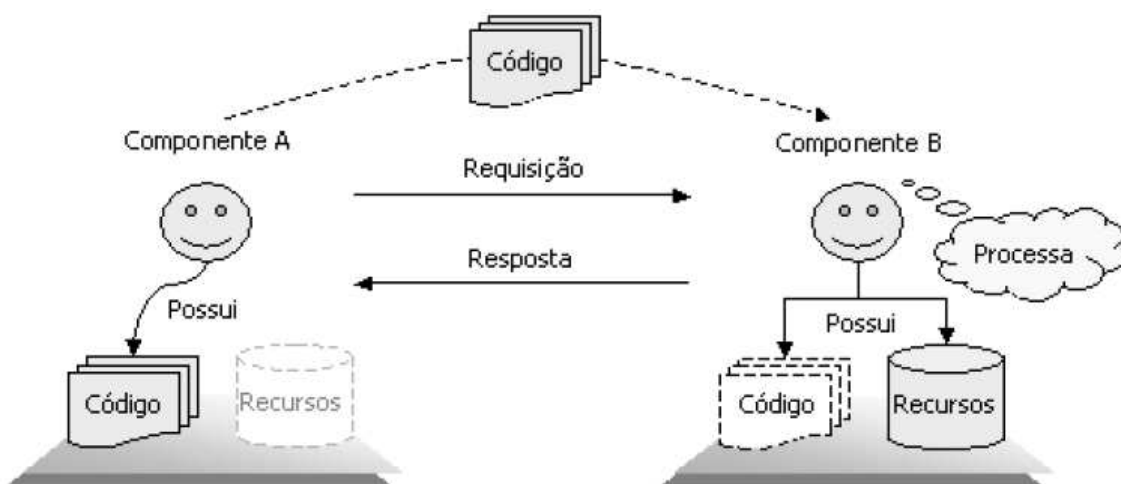


Figura 2.6: Paradigma de Execução Remota

Paradigma de Execução Remota

Neste paradigma, um componente X possui o código a ser executado, mas faltam recursos mínimos à execução da mesma. Então, o componente X envia o código, a ser executado, para o componente B, situado em um outro local, onde existe recurso suficiente para executar a tarefa. O componente B efetua o processo e retorna os resultados alcançados ao componente A. Aplicações como ssh (*Secure Shell*), rsh (*Remote Shell*), entre outras, utilizam este paradigma. A figura 2.6 demonstra graficamente o conceito.

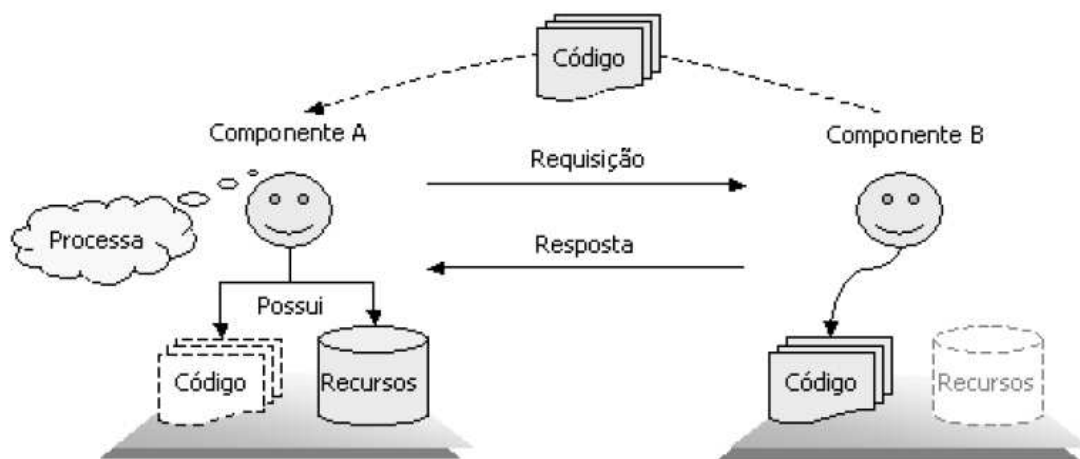


Figura 2.7: Paradigma de Código sob Demanda

Paradigma de Código Sob Demanda

Ao contrário do paradigma anterior, execução remota, aqui o componente A possui o recurso, porém não possui o código a ser executado. Assim o componente A requisita do componente B o código a ser executado.

Exemplificando, digamos que Tais pretende fazer um bolo. Então ela sai ao mercado e compra vários ingredientes que ela sabe que são necessários ao bolo. Ao chegar em casa, Tais verifica que não possui a receita para o preparo. Neste momento Tais tem todos os recursos necessários à preparação do bolo, mas não possui a receita, ou seja, o código para se fazer o bolo. Então ela liga para Lara e solicita a receita do bolo. Lara passa para Tais a receita e Tais usa agora a receita e seus ingredientes para executar a ação. Applets Java utilizam este paradigma. A figura 2.7 ilustra o paradigma.

Paradigma de Agentes Móveis

Um componente computacional A (o agente), inicialmente hospedado em X, possui seu código a ser processado. Porém a máquina X, que o hospeda, não possui os recursos necessário ao processamento e existe na rede uma máquina Y que os possui. Então, o agente A move-se através da rede até o local Y, levando consigo o código a ser processado, e executa a sua tarefa. Assim o agente A consome os recursos de Y e

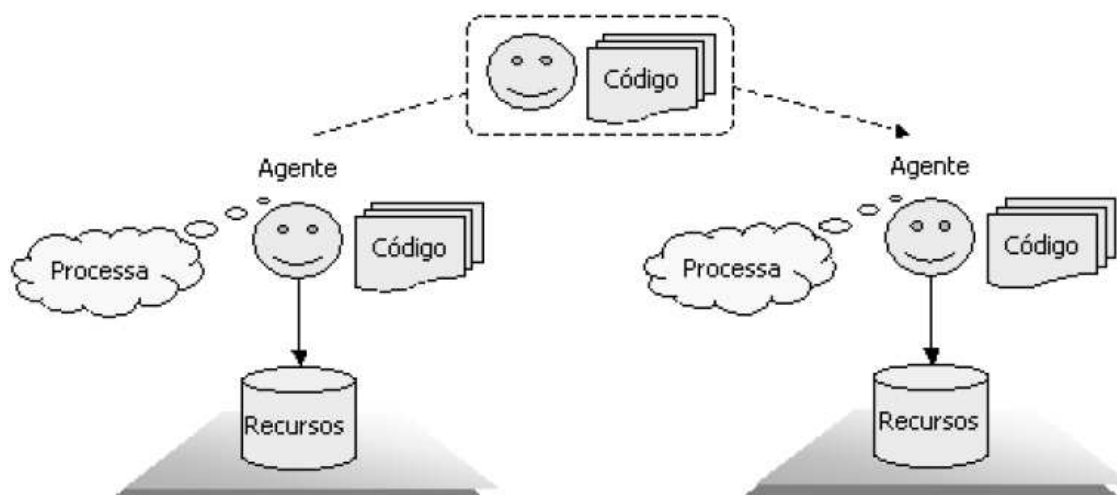


Figura 2.8: Paradigma de Agentes Móveis

termina sua tarefa, completando seu objetivo. A figura 2.8 mostra graficamente o que foi dito. Algumas aplicações que já utilizam este paradigma são: gerência de redes, correios eletrônicos, busca de informações¹⁶, entre outras.

A tabela 2.3 mostra, em um resumo, a diferença entre todos os quatro paradigmas apresentados.

Como exposto podemos concluir que a principal diferença entre o paradigma de agentes móveis e os demais, é a capacidade de se mover componentes inteiros de um lugar para outro na rede, para que o mesmo possa utilizar recursos disponíveis no hospedeiro e realizar sua função. Os agentes, ou componentes do paradigma de agentes móveis, podem migrar para outro hospedeiro levando consigo o seu estado, seu código e até mesmo alguns recursos necessário à execução da tarefa em questão. Enquanto, nos outros paradigmas apenas o código é transferido para um local remoto.

Segundo Lange [23] as vantagens em se utilizar o modelo de agentes móveis são:

- **Redução do tráfego da rede:** transmitir dados pela rede em uma aplicação distribuída é, muitas vezes, custoso demais. Por isso com a utilização de agentes

¹⁶sistemas de buscas na internet, por exemplo

Cliente-Servidor	Execução Remota	Sob Demanda	Agentes Móveis
Cliente solicita serviços ao servidor	Todo código é transferido ao servidor	Todo código carregado do servidor	Alto grau de flexibilidade
Servidor possui o conhecimento	Conhecimento passado pelo cliente	Conhecimento passado pelo servidor	O conhecimento esta espalhado
Servidor possui os recursos para executar a ação	O recurso só esta presente no servidor	Cliente tem todos os recursos necessários ao processamento	Agentes migram para onde estão os recursos
Servidor controla o processamento	Servidor controla o processamento	cliente controla o processamento	Hospedeiro do agente comanda
Exemplo: Interpretação de página web	Exemplo: <i>Secure Shell</i> (ssh)	Exemplo: Java Applets	Exemplo: Gerência de redes

Tabela 2.3: Tabela de Comparação (Paradigmas de programação Distribuída)

pode-se encapsular as solicitações de dados e enviar o agente para onde estão os dados. O agente então processa a requisição localmente e retorna com o resultado, diminuindo assim, significativamente, o tráfego na rede.

- **Diminuição da latência de rede:** Em sistemas críticos, a necessidade de resposta em tempo real pode não ser obedecida se o controle for centralizado. Assim agentes podem ser enviados para controlar cada parte do sistema localmente, reduzindo assim a latência da rede e o tempo de resposta.
- **Encapsulamento de protocolos:** Em um sistema distribuído todos os nós da rede devem implementar o protocolo para o recebimento de dado e para o envio do mesmo. Com o aumento do número de nós da rede pode se tornar muito difícil a atualização dos protocolos, gerando assim um problema. Com a utilização de agentes, o agente é enviado ao nó remoto para estabelecer um canal de comunicação baseado no protocolo definido.

- **Execução autônoma e assíncrona:** Com a maior utilização de computação móvel, o problema da desconexão de dispositivos portáteis vêm sendo muito estudado. Manter uma conexão aberta, por muito tempo, é caro ou tecnicamente impossível. Assim as tarefas podem ser encapsuladas no agente que será enviado até o dispositivo. Após este envio o agente pode se desconectar do processo que o criou, processar as tarefas localmente e algum tempo depois reconectar ao processo "pai" e voltar à base com os resultados.
- **Adaptação dinâmica:** Agentes móveis podem ser implementados com a capacidade de aprender, assim adaptando-se dinamicamente às mudanças em seu ambiente de execução.
- **Integração de sistemas heterogêneos:** Como os agentes são independentes de arquitetura de hardware, dependem apenas do seu ambiente de execução, são uma forma de abstrair as diferenças e facilitar a integração de diversos sistemas heterogêneos.
- **Tolerância à falhas:** Reagir dinamicamente as mas diversas situações é uma das características que fazem com que um sistema de agentes móveis seja mais robusto e tolerante a falhas. Os agentes podem ser desenvolvidos para que, se encontrarem situações adversas, migrem para um outro local menos hostil para executar sua tarefa, deixando sistema mais seguro e tolerante a falhas.

2.3.5 Plataforma Aglets

O ASDK (*Aglets Software Development Kit*) é uma plataforma para desenvolvimento de agentes móveis totalmente escrita em Java [24], que tem como características principais a portabilidade, segurança e ser multitarefa. O nome *Aglets* surgiu da contração das palavras *agent* e *applet*. Inicialmente desenvolvida na IBM Japão pelos pesquisadores Danny Lange e Mitsuru Oshima em 1996. Esta versão não é compatível com as novas versões da linguagem Java. O *Aglets 2.0* é uma versão de código-livre do ASDK [21] licenciada sob os termos da *IBM Public License*.



Figura 2.9: Modelo Conceitual da plataforma *Aglets*

A estrutura conceitual do *Aglets* é ilustrada pela figura 2.9. A plataforma *Aglets* fornece uma API com funções básicas para o processos de migração, comunicação e gerência dos agentes e o servidor *Aglets* é *multithread*. Detalharemos este modelo no decorrer desta seção.

Agentes *Aglets*

Os agentes *aglets* são objetos Java com a capacidade de se mover entre os nós da rede e são caracterizados por possuírem a própria *thread* de controle, serem dirigidos a eventos e se comunicarem através de passagem de mensagens, segundo a definição do autor da plataforma Danny Lange [23]. Esta definição combina agentes com o código móvel dos *applets* Java.

- **Autonomia**

Característica fundamental para um agente, em um agente *aglet* a autonomia é garantida pelo fato de cada agente possuir a sua própria *thread* de controle. Para isso as classes do agente implementam a interface *Runnable* do pacote *java.lang*.

- **Reatividade**

Esta característica é garantida na API *Aglet* que implementa diversas funções que executam em resposta a estímulos recebidos. Tais funções são os sensores e atuadores do agente.

- Persistência

A manutenção do seu estado interno é muito importante para um agente. Esta persistência pode ser verificada com o ciclo de vida do agente que inclui cinco etapas.

- **criação:** estado inicial. O agente recebe um identificador único e todas as configurações iniciais são executadas, ficando então pronto para receber instruções. Esta etapa só é executada uma vez no ciclo de vida do agente e é chamada através do método *createAglet*;
- **clonagem:** segunda etapa do processo, onde o agente se copia e é executada através do método *clone*. Todo o estado interno e o código do agente antigo é copiado exatamente igual ao original, com exceção do identificador;
- **desativação:** interrompe a execução do agente, porém todo seu estado interno e seus resultados intermediários são salvos em disco, e é utilizada através do método *deactivate*;
- **ativação:** quarta etapa do processo executada através do método *activate*. Esta etapa restaura um agente, previamente desativado, com informações contidas no disco;
- **destruição:** a última etapa do ciclo de vida do agente. Aqui o agente libera todos os recursos do ambiente que estava utilizando e finaliza sua atividade não podendo mais ser ativado. O método *dispose* é utilizado para destruir definitivamente um agente do sistema.

- Mobilidade

Através dos métodos *dispatch* e *retract* os agentes podem se mover entre os nós da rede. Sempre, antes de se mover, o objeto é serializado, mantendo assim todos os dados da execução atual. Pelas características do Java apenas migrações do tipo fraca são possíveis, ou seja, o estado interno e os dados são mantidos, porém, ao chegar ao novo nó o agente inicia uma nova *thread*. Uma outra migração possível é a migração forte, não presente nas implementações em Java, onde todo o estado

do agente é mantido, inclusive a sua execução, ou seja, ao chegar ao novo nó, o agente executa de onde havia parado no nó anterior. Estes tipos de migração são bem explorados por Damir [25].

- **Habilidade Social**

O modelo de comunicação da plataforma ASDK foi desenvolvido de maneira que fosse possível que os agentes colaborassem entre si para completar uma tarefa. O mecanismo de passagem de mensagens permite que os agentes se comuniquem, dando assim uma habilidade social ao agente.

No caso do ASDK os agentes não se comunicam diretamente entre si. Cada agente tem o seu *proxy* que se comunicará com o *proxy* de outros agentes.

A API *Aglet* disponibiliza o envio de mensagens síncronas e assíncronas, tanto para agentes no mesmo contexto como agente em contextos remotos.

- **Ambiente de Execução**

Como visto acima, um servidor de agentes é um elemento indispensável a um sistema de agentes móveis [20]. E por isso o *Aglets* não poderia deixar de dispor de um ambiente de execução para os seus agentes.

Os agentes *aglets* são executados num ambiente chamado *Aglet Server*. Este ambiente fornece ao usuário uma interface gráfica¹⁷ que possibilita a criação, destruição, transferência e configuração dos agentes. Trata-se de um gerenciador de serviços prestados pelo *Aglet Server*. A figure 2.10 mostra a interface gráfica, Tahiti, para a manipulação dos agentes.

API *Aglet*

A API *Aglet*, ou simplesmente AAPI (*Aglet Application Programming Interface*), é um padrão para a programação de agentes *aglet* e seu ambiente de execução, ou seja é o padrão da interface entre as duas entidades. É nesta API que estão definidos os métodos necessários para a comunicação dos agentes e seus ambientes.

¹⁷Denominada Tahiti

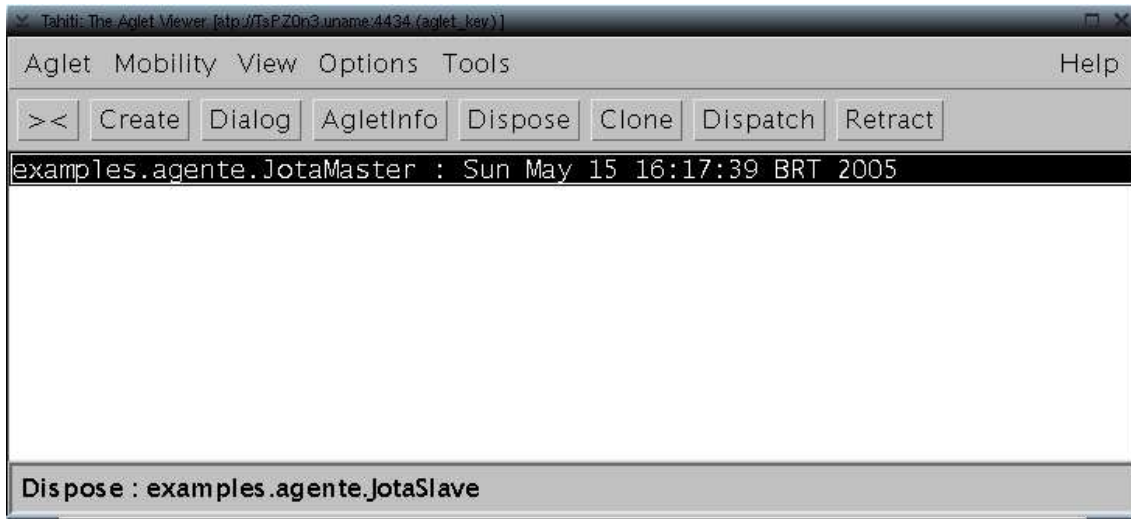


Figura 2.10: Interface gráfica do *Aglet Server*, Tahiti

Segundo Lange [23], criador da plataforma, a interface de programação dos agentes é caracterizada pela simplicidade e flexibilidade. A API foi projetada para tirar o maior proveito possível da linguagem Java e é totalmente construída em cima da Máquina Virtual Java, não necessitando de integração nenhuma com código nativo. A maior vantagem da utilização da AAPI é a portabilidade do código dos agentes. Uma vez escrito o código de um agente ele poderá rodar em qualquer máquina que tiver o Aglet Server sem se preocupar com o *hardware*.

Veremos, nesta seção, a definição da AAPI através de suas classes fundamentais e do modelo de mensagens que é implementado pela API.

Classes Fundamentais

Algumas abstrações do mundo real são indispensáveis neste ambiente e são representadas pelas classes e interfaces: *Aglet*, *Identificador*, *Contexto*, *Proxy* e *Mensagem*.

- **Classe *Aglet***

Um agente é implementado através da classe abstrata *Aglet* da AAPI. Trata-se da "classe base" para todos os agentes *aglets*, como define o próprio criador [23].

Eventos	Métodos finais	Métodos Modificáveis	
		No momento dos eventos	Após os eventos
Criação	<i>createAglet()</i>		<i>onCreation()</i>
Clonagem	<i>clone()</i>	<i>onCloning()</i>	<i>onClone()</i>
Envio	<i>dispatch()</i>	<i>onDispatching()</i>	<i>onArrival()</i>
Retorno	<i>retractAglet()</i>	<i>onReverting()</i>	<i>onArrival()</i>
Destruição	<i>dispose()</i>	<i>onDisposing()</i>	
Desativação	<i>deactivate()</i>	<i>onDeactivating()</i>	
Ativação	<i>activate()</i>		<i>onActivation()</i>
Mensagens	<i>handleMessage()</i>		

Tabela 2.4: Eventos relacionado ao ciclo de vida dos agentes

Conforme a API a classe *Aglet* deve conter os métodos para a manipulação do ciclo de vida dos agentes, como por exemplo, o método *clone()*. Os métodos desta classe que possuem o modificador "final" em suas assinaturas não podem ser modificados por agentes que implementam esta classe. Porém, alguns outros métodos podem ser sobrescritos a fim de particularizar o comportamento de cada agente. Estes métodos são chamados automaticamente pelo sistema quando alguns eventos ocorrem no ciclo de vida do agente. O método *onCreation()*, por exemplo, será chamado imediatamente após a criação do agente e pode ser particularizado da forma que o programador desejar.

Um método muito importante para os agentes, e que deve ser particularizado, é o *handleMessage(Message)*, que tem a função de tratar as mensagens recebidas pelo agente e definir as ações a serem tomadas.

Um outro método que deve ser, necessariamente, sobrescrito é o método *run()*, que é executado após um agente ser criado, ativado, ou existir alguma mudança de contexto. Sobrescrever o método *run()*, de um agente, significa definir um comportamento autônoma para ele, segundo Lange [23].

- **Classe *AgletID***

Os agentes, assim como os objetos, possuem um valor associado a eles que permite a identificação única de cada agente. Este identificador é imutável e independente de contexto, como apontado por Lange [23]. Este valor é atribuído ao agente no momento de sua criação e é válido até a sua destruição e todos os nós da rede o "conhecem" por este identificador.

Este identificador é implementado pela classe *AgletID*. Geralmente esta classe não precisa de modificações por parte dos programadores e é possível obter dados desta classe através do método *getAgletID()* da classe *Aglet*, ou através do método *getProxy().getApletID()* implementado pela interface *AgletProxy*.

- **Interface *AgletContext***

"Assim como um objeto tradicional, um *aglet* possui estado, comportamento e identidade. Porém, contrariamente a um objeto tradicional, um *aglet* também possui uma localização.", afirma Bill Venners [26].

De acordo com Lange [23], o contexto fornece os meios para gerenciar um agente em seu ambiente de execução. O contexto de um agente limita o seu escopo de atuação no ambiente em que se encontra, dando assim mais segurança ao nó que hospeda o agente, evitando que agentes mal intencionados afetem o funcionamento da máquina. Desde o momento em que o agente é criado ele se associa a um contexto e a maior parte de sua vida será associada a este contexto. Mesmo após migrar para outro nó, o contexto do agente permanece o mesmo, abstraindo assim a heterogeneidade da rede.

Através do método *getAgletProxies()* é possível obter uma lista com todos os *proxies* de *algets* disponíveis em um contexto. Assim um agente sabe quais são os outros agentes que estão em execução no mesmo contexto. E através do método *getHostingURL()* é possível identificar a localização do contexto atual. A tabela 2.5 resume os principais métodos desta interface.

Uma outra forma de gerenciar os contextos é utilizando a interface gráfica Tahiti.

Métodos	Descrição
<i>creatAglet()</i>	Cria o agente no contexto
<i>retractAglet()</i>	Retorna o agente ao nó de origem
<i>getAgletProxies()</i>	Obtém um lista dos proxies dos agentes deste contexto
<i>getHostingURL()</i>	Obtém a localização do servidor do contexto
<i>getName()</i>	Retorna o nome do contexto
<i>getProperty()</i>	Retorna o valor de uma propriedade definida pelo usuário
<i>setProperty()</i>	Determina um valor para uma propriedade definida pelo usuário

Tabela 2.5: Métodos da interface *AgletContext*

Através da interface é possível criar, ativar, clonar e destruir agentes sem ter que fazer nenhum tipo de programação.

- **Interface *AgletProxy***

Acessar métodos públicos dos agentes é possível, porém, não aconselhável, pois desta forma o agente perde autonomia além de violar a propriedade de encapsulamento dos agentes. Além disso, o acesso direto a métodos dos agentes não garante a proteção de suas referências, como destaca Grimshaw [20].

O *proxy* é então um representante do agente para tratar comunicação com outros agentes. Além disto o *Proxy* esconde a localização real do agente de forma transparente aos outros agentes, assim se um agente migra para outro nó a comunicação existente anteriormente continua sendo possível. Esta camada intermediária de comunicação é implementada pela interface *AgletProxy*.

- **Classe *Message***

Como agentes, os objetos *aglets* devem implementar a habilidade social. Deve ser possível haver cooperação entre os agentes, deve haver comunicação entre eles. Na plataforma *Aglets* isto é implementado por troca de mensagens, pela classe *Message*.

Segundo Lange [23], na plataforma *Aglets*, uma mensagem consiste em um objeto

trocado entre agentes. As mensagens trocadas entre os agentes permitem interação e troca de informações entre os mesmos.

Para a criação de um objeto da classe *Message* dois parâmetros são passados, sendo o primeiro obrigatório e o segundo opcional. O primeiro indica o tipo de mensagem e o segundo parâmetro são argumentos passados como valor associados à mensagem, podendo ser dos tipos primitivos Java ou ser instância da classe *Object* ou de suas subclasses.

Quando a mensagem é trocada por objetos remotos o segundo parâmetro da mensagem deve obedecer a uma restrição: o objeto deve ser serializável. Recomenda-se usar, nestes casos, *String*, *Vectors* ou *Arrays* que são classes de Java serializáveis por padrão da linguagem.

Os métodos *setArg()* e *getArg()*, definidos nesta classe, permitem a manipulação dos argumentos associados a uma mensagem. Esta classe ainda implementa o método *sendReply()* que é a forma mais fácil de responder a uma mensagem.

Envio de Mensagens

Na plataforma *Aglets* três tipos de mensagens são suportadas. Mensagens sem respostas, mensagens com respostas e mensagens assíncronas. O método *sendOnewayMessage()* tem em sua assinatura *void*, o que significa que não excite retorno para este método, e é usado quando o emissor da mensagem não espera por respostas. Já o método *sendMessage()* recebe como retorno um objeto da classe *Object* e por isso é usado quando o emissor espera uma resposta. E o método *sendAsynncMessage()* recebe como retorno um objeto da classe *FutureReply* o que caracteriza o envio de forma assíncrona.

Modelo de Eventos

A programação orientada a eventos é uma das características do *Aglets*, segundo o seu criador [23]. A AAPI utiliza modelo de delegação de eventos muito parecido ao implementado na linguagem Java, no qual os objetos geram eventos ao executar certas

ações no sistema. Para responder a estes eventos deve-se implementar uma interface do tipo *Listener*.

Como objetos Java, os *aglets* pode ser implementados de forma a responder a qualquer evento já existente na linguagem, no entanto, em um sistema de agentes móveis, outros eventos acontecem e não fazem parte da linguagem Java. Estes eventos devem ser implementados pelos programadores, afim de que se possa responder a estes eventos relacionados ao agente. Três receptores de eventos estão definidos na AAPI, que são: *CloneListener*, *PersistencyListener* e *MobilityListener*.

Para o tratamento de eventos relacionado com a clonagem do agente, o receptor *CloneListener* dispõem de três métodos.

- *OnCloning()* - que determina a ação a ser tomada imediatamente antes de um clone ser criado;
- *OnClone()* - que dita o que fazer no momento da criação do clone;
- *OnCloned()* - que executa a ação logo após a clonagem.

O receptor de eventos *PersistencyListener* é responsável por determinar as ações a serem tomadas em resposta a eventos de persistência.

- *OnDeactivating()* - antes do desativamento de um agente;
- *OnActivation()* - logo após a ativação do mesmo.

O terceiro receptor *MobilityListener* é responsável pelos eventos que dizem respeito à mobilidade dos agentes.

- *OnDispatching()* - antes que um agente seja enviado a outro nó;
- *OnArrival()* - exatamente no momento de sua chegada ao novo nó;
- *OnReverting()* - antes que ele volte ao local de origem.

Capítulo 3

Descrição do Problema

Com o avanço da ciência os problemas se tornaram cada vez mais específicos e complexos. Mesmo com o acelerado processo de renovação tecnológica e melhorias na área de processamento e armazenamento de dados ainda assim é virtualmente impossível resolvermos alguns problemas utilizando-se de uma supermáquina, mesmo desconsiderando o altíssimo custo destas máquinas, ainda assim seria muito difícil conseguir resolver alguns problemas.

Tais problemas são considerados computáveis, porém levariam anos, ou até mesmo décadas ou séculos, para serem resolvidos, o que os tornaram, na prática, problemas "impossíveis", pois depois de tanto tempo os resultados já não seriam mais importantes ou significativos, pois o avanço da ciência já os teria sobreposto.

Assim, estudiosos buscaram formas de se construir uma supermáquina, que pudesse processar um grande número de dados em alta velocidade e que tivesse uma capacidade de armazenamento suficientemente grande para garantir a integridade dos dados do problema. Assim surge as soluções de processamento distribuído, mais que isso, distribuído e paralelo. Os chamados Clusters ou Grids computacionais.

O problema é que estes ambientes são, geralmente, complexos e a escolha das máquinas que farão parte do sistema normalmente é feita sem nenhum critério e as tarefas são distribuídas aleatoriamente, o que não gera o melhor resultado possível. As configurações das máquinas devem ser levadas em consideração quando se estiver

distribuindo as tarefas já que o atraso de uma máquina significa o atraso de todo o processamento, pois pode se tratar de um ambiente paralelo, onde os computadores mais rápidos terminam suas tarefas e ficam ociosos, "esperando", os mais lentos finalizarem sua parte no processamento.

Outro problema encontrado nos Clusters ou Grids é a dificuldade em se trabalhar com um ambiente heterogêneo. Seria necessário que para cada arquitetura, existente na rede, um novo programa fosse escrito e uma nova configuração fosse feita.

Como um programador poderá saber em quantas máquinas, ou até mesmo em quais máquinas da rede seu programa esta sendo executado? Como o programador pode escolher, explicitamente, de um conjunto de máquinas, quais ele pretende utilizar em seu processamento? Como estimar um tempo de execução sem saber estas informações? Como o programador pode verificar os recursos totais disponíveis em seu aglomerado ou grade de computadores? Como ter uma visão geral da sua "supermáquina"?

Estas perguntas fazem parte do problema que este trabalho trata, e é objetivo do trabalho responde-las apresentando soluções práticas e satisfatórias.

Capítulo 4

Descrição da Solução

O problema visto no capítulo anterior é dividido em partes por este trabalho para que uma melhor solução, mais específica, fosse desenvolvida. Primeiramente devemos efetuar algumas escolhas que tocam o tipo de sistema distribuído. Depois devemos escolher uma linguagem de programação para trabalhar, levando-se em conta diversas variáveis para esta escolha. Devemos ainda escolher um modelo de programação em rede e, por último, refinar o sistema e escolher uma forma de apresentá-lo ao usuário.

A primeira girava em torno de Clusters ou Grids, e devido ao tempo previsto para este trabalho e o seu grau de aprofundamento, a opção foi pela utilização de um ambiente de Cluster. Tal ambiente necessita de uma quantidade inferior de preocupações para o programador, já que os recursos são de total dedicação ao sistema, evitando assim a possibilidade de um ambiente hostil. Um outro ponto importante para a escolha foi a falta de recursos para se trabalhar com grids, levando-se em consideração que grids são formados por redes geograficamente distribuídas, dificultando-se assim os testes do protótipo.

Java foi a linguagem escolhida para desenvolvermos a solução deste trabalho. Para se chegar a essa decisão, alguns pontos fortes da linguagem foram considerados como:

- Independência de plataforma

Java foi projetada para ser executada em uma grande variedade de plataformas.

Isto torna o sistema portátil e facilita no desenvolvimento das aplicações, pois características específicas de *hardware* não são levadas em consideração. Assim um código escrito em uma plataforma rodará em uma outra plataforma diferente sem nenhuma alteração no código fonte da aplicação. E isso torna a linguagem perfeita para e trabalhar em ambientes heterogêneos.

E essa independência de arquitetura torna o Java ideal para se projetar sistemas que não se sabe em qual, ou quais, arquiteturas irá rodar.

- Execução segura

Desde o início, Java foi pensada como uma linguagem capaz de ter boas "relações" com a Internet. Assim um *software* Java, ou melhor, um *applet*, pode ser baixado por um cliente e ser executado na máquina local. Para isso, a linguagem foi projetada com uma série de funções que garantissem a segurança e integridade do usuário. Restrições como, não permissão de escrita ou leitura de arquivos no sistema de arquivos do usuário ou execução de arquivos binários, são algumas atitudes tomadas por Java para garantir esta segurança.

Assim quando se tratando de agentes móveis, essas restrições de linguagem podem garantir que um agente mal intencionado não terá sucesso algum na tentativa de danificar o sistema do usuário. Dando mais tranquilidade ao programador ao desenvolver seus agentes.

- Carregamento dinâmico de classes

Nos sistemas de agentes, delimitar um espaço de endereçamento das classes permite o agente executar-se de maneira segura e independente dos outros e isso é garantido por Java uma vez que as classes Java são carregadas dinamicamente em tempo de execução.

- *Multithread*

Java não só permite programação *multithread* como implementa uma série de primitivas de sincronização. Isso facilita ao programador na implementação do ciclo de vida dos agentes, assim como em sua comunicação.

- Serialização dos objetos

Objetos Java podem ser serializados e salvos em disco ou enviados pela rede. Independentemente da plataforma, um objeto pode ser serializado no *Windows* e enviado para uma máquina *Linux* onde será salva em disco. Posteriormente esse objeto pode ser recuperado e restaurado na máquina origem sem problemas. Isso é de extrema importância para sistemas de agentes no tocante a persistência dos agentes.

- Vasta documentação

Java é uma linguagem que apesar de ser bastante nova, se comparado a outras, já é bastante utilizada. E isso torna mais fácil a troca de informações entre programadores. Além disso a documentação disponível é bastante grande e a descrição de sua API é bem detalhada.

Além dos pontos positivos da linguagem também foram levados em consideração os pontos negativos, porém estes não superaram as vantagens. As principais desvantagens são:

- Falta de suporte para controle de recursos

Um objeto Java pode consumir um tempo muito grande de CPU ou uma quantidade muito grande de memória sem que o usuário possa limitar os esses recursos. O que, no contexto deste trabalho não é tão decisivo já que se trata de um ambiente dedicado.

- Referência a objetos

É possível quem um objeto acesse diretamente os métodos públicos de uma classe Java. Porém isso numa ambiente de agentes móveis não é favorável pois não existe forma de controlar a comunicação entre objetos se forem feitos acessos direto aos métodos.

- Impossível recuperar totalmente o estado interno dos objetos

Pelas características da linguagem, principalmente por ser uma linguagem que tem como objetivo abstrair o *hardware*, é impossível salvar e recuperar totalmente o estado interno de um objeto, como destaca Lange [23]. Mesmo sendo possível,

através de *software*, implementar esta funcionalidade o gasto computacional para isso é muito grande e não é compensador.

A terceira escolha foi por se trabalhar com agentes móveis. Esta decisão tem uma base pessoal, pois curiosidade e motivação, foram os principais fatores para a escolha de se trabalhar com uma tecnologia nova e em expansão. Além de trabalhar com agentes móveis, algumas escolhas extras tiveram que ser feitas. A utilização da plataforma *Aglets* reforça a utilização da linguagem de programação Java. Por se tratar de um ambiente de código-livre e gratuito e que tem como principal objetivo incentivar a pesquisa e desenvolvimento da tecnologia, esta plataforma de agentes móveis contempla os objetivos deste trabalho.

Para que o programador possa ter uma noção do sistema como um todo, e ter a capacidade de estimar o tempo necessário para a execução de uma tarefa. Ou ainda para que ele seja capaz de escolher, entre as máquinas disponíveis no sistema, qual deverá participar na execução da tarefa, levando-se em conta os recursos disponíveis em cada máquina, este trabalho propõe a criação de um SSI (*Single System Image*), utilizando-se agentes móveis para colher informações do sistema e disponibilizar um ambiente propício à execução de tarefas distribuídas, tanto paralelas ou não.

Este trabalho baseia-se na solução descrita por Tânia Gomes Ramos, José Geraldo Lopes e Mario Antônio Dantas [27] e em seu agente Jota, detalhado no **Anexo A**. O agente Jota foi modificado para este trabalho para obter maiores informações e melhorar a sua interface gráfica para o usuário, além de implementado novas funções de configuração.

Capítulo 5

Resultados obtidos

Utilizando-se das ferramentas descritas acima, foi possível disponibilizar para o usuário de Clusters uma opção no gerenciamento de seus recursos. Além disso, uma solução barata e fácil de utilizar, foi criada para o emprego em ambientes distribuídos que melhora o desempenho da computação executada.

Dados retirados dos experimentos com o Jota [27], feitos por Tânia Ramos, comprovam a melhoria no desempenho do cluster, quando feita a utilização do agente, como demonstra a figura 5.1. A tarefa submetida ao sistema foi uma multiplicação de matrizes quadradas de ordem 500 e à medida que o número de processo cresce, os resultados se aproximam já que o número de processos vai se igualando ao número de nós na rede, como se pode perceber. Isto acontece porque o balanceamento de carga e a es-

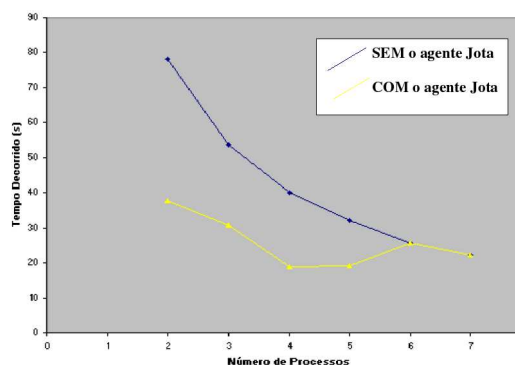


Figura 5.1: Multiplicação de Matrizes 500x500

colha de nós tornam-se inúteis pois todos os nós da rede deveram participar da execução. No experimento em questão oito máquinas foram utilizadas.

Além dos resultados concretos obtidos, outros objetivos foram atingidos. Pesquisar e interagir com sistemas distribuídos, entendendo seus conceitos e paradigmas foi o primeiro a ser alcançado. Porém, o mais inspirador dos objetivos, entender e programar utilizando agentes móveis foi o elemento motivante para todo este trabalho e atingi-lo, ao final de tudo, e obter uma aplicação prática e funcional foi o maior resultado alcançado por este trabalho.

Capítulo 6

Conclusão e trabalhos futuros

O avanço tecnológico acelerado demanda uma atualização de maquinário muito rápida, o que na maioria das situações se torna insustentável. Instituições que dependem de um alto poder de processamento estão sempre à procura de novas tecnologias, de meios para reaproveitar o seu *hardware*. É aí que sistemas distribuídos ganham força e se tornam uma opção para as instituições.

Este trabalho, de graduação, apresentou conceitos mínimos ao entendimento do funcionamento de um sistema distribuído e também expos modelos distribuídos e paralelos. Foi utilizado o que há de mais novo em termos de computação em rede, o paradigma de agentes móveis, que vem se tornando um importante aliado em ambientes distribuídos, desempenhando uma infinidade de funções nos sistemas. Foi desenvolvido ainda neste trabalho, uma implementação de um ambiente SSI configurável baseado-se no agente Jota e utilizando-se da plataforma de agentes *Aglets*. Para tal, foi utilizada a linguagem Java por suas características específicas no que dizem respeito a objetos e comunicação.

Sistemas únicos de imagem (SSI), permitem que ambientes, outrora, difíceis de programar e de obter informações relativas ao sistema como um todo, transformem-se em ambientes amigáveis ao programador e usuário final. Tornando, desta forma, a execução de uma tarefa paralela e distribuída tão trivial quanto de uma tarefa seqüencial e executada localmente.

Por se tratar de tecnologias novas ainda há muito que ser feito nesta área. Imagine o poder de processamento de um grid oportunístico¹ rodando no ambiente de Internet, ou ainda a enorme quantidade de aplicações de agentes móveis possíveis em um ambiente de rede. Neste trabalho em especial, um futuro desenvolvimento pode ser uma implementação do agente Jota em uma outra linguagem de programação que pudesse disponibilizar, por exemplo, suporte a migração "forte". Isto geraria um grande avanço neste projeto.

A utilização do agente Jota em um ambiente de grid oportunístico é um bom campo de pesquisa, para trabalhos futuros, podendo gerar resultados práticos, tendo em mente que para isso, necessariamente, o agente Jota teria que ser reescrito em outra linguagem, pois como visto anteriormente, Java não permite que o usuário determine quanto de recurso deseja compartilhar, o que é uma característica de ambientes oportunísticos.

¹grids como Seti/@home [6]

Referências Bibliográficas

- [1] ALVARENGA, A. T. H. *Um ambiente para processamento paralelo oportunístico na internet*. Dissertação (Mestrado) — Universidade de Brasília, 2003.
- [2] DANTAS, M.; LOPES, J.; RAMOS, T. *An Enhanced Scheduling Approach in a Distributed Parallel Environment Using Mobile Agents*. Brasília, 2002.
- [3] GLOSSARY: Parallel Computing. Fevereiro 2005. Online. Disponível em: <<http://www.cise.ufl.edu/research/ParallelPatterns/glossary.htm>>.
- [4] BEOWULF Cluster. Março 2005. Online. Disponível em: <<http://www.beowulf.org/>>.
- [5] BUYYA, R. *Grid Computing Info Center*. Dezembro 2004. Online. Disponível em: <www.gridcomputing.com>.
- [6] SETI@HOME. maio 2005. Online. Disponível em: <<http://setiathome.ssl.berkeley.edu>>.
- [7] DISTRIBUTED.NET. dezembro 2005. Online. Disponível em: <<http://www.distributed.net>>.
- [8] GRID.ORG. janeiro 2005. Online. Disponível em: <<http://www.grid.org>>.
- [9] GOLDCHLEGER, A. *InteGrade: Um Sistema de Middleware para Computação em Grade Oportunista*. Dissertação (Mestrado) — IME-USP, dezembro 2004.
- [10] BERKELEY Open Infrastructure for Network Computing. Fevereiro 2005. Online. Disponível em: <<http://boinc.ssl.berkeley.edu/>>.

- [11] EINSTEIN@HOME. Maio 2005. Online. Disponível em: <<http://einstein.phys.uwm.edu/>>.
- [12] WHATIS.COM - IT Search. Maio 2005. Online. Disponível em: <<http://whatis.techtarget.com>>.
- [13] CORTES, T. *Task Force on Cluster Computing - SSI*. Maio 2005. Online. Disponível em: <<http://people.ac.upc.edu/toni/CCTaskForce/SSI.html>>.
- [14] IEEE. Abril 2005. Online. Disponível em: <<http://www.ieee.org>>.
- [15] BARNEY, B. M. *Message Passing Interface (MPI)*. [S.l.], 2005. Disponível em: <<http://www.llnl.gov/computing/tutorials/mpi/>>.
- [16] AL-TAWIL, K.; MORITZ, C. A. Performance modeling and evaluation of mpi. *Parallel Distrib. Comput.*, v. 61, n. 2, p. 202–223, 2001.
- [17] NWANA, H. S. Software agents: An overview. *Knowledge Engineering Review*, v. 11, n. 2, 1996. Disponível em: <Hyacinth S. Nwana>.
- [18] LOPES, R. P.; OLIVEIRA, J. L. Descrição e implementação de uma mib para sistemas masif. *3.a. Conferência Sobre Redes de Computadores (CRC2000)*, 2000. Disponível em: <http://www.fccn.pt/crc2000/CRC2000_2.3.pdf>.
- [19] BIGUS, J. P.; BIGUS, J. *Constructing Intelligent Agents Using Java*. 2. ed. [S.l.]: Professional Developer's Guide, 2001.
- [20] GRIMSHAW, D. *Artificial Intelligence Topics with Agents*. Maio 2001. Online. Disponível em: <<http://www.ryerson.ca/dgrimsha/courses/cps720/index.html>>.
- [21] AGLETS. maio 2005. Online. Disponível em: <<http://aglets.sourceforge.net/>>.
- [22] FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding code mobility. *IEEE Transactions on Software Engineering*, v. 24, n. 5, p. 342–361, 1998. Disponível em: <<https://www.cis.strath.ac.uk/teaching/ug/classes/52.477/e0342.pdf>>.

- [23] LANGE, D. Mobile objects and mobile agents: The future of distributed computing? *PROCEEDINGS OF THE EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING '98*, 1998. Disponível em: <<http://www.moe-lange.com/danny/ecoop98.pdf>>.
- [24] SUN. *Java*. Online. Disponível em: <<http://java.sun.com>>.
- [25] HORVAT, D. Mobile agents and java mobile agents toolkit. *PROCEEDINGS OF THE HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*, 2000. Disponível em: <<http://csdl.computer.org/comp/proceedings/hicss/2000/0493/08/04938029.pdf>>.
- [26] VENNERS, B. The significance of mobile agents. *JavaWorld*, 1997. Disponível em: <<http://www.artima.com/underthehood/pointP.html>>.
- [27] DANTAS, M. A. R.; LOPES, J. G. R. C.; RAMOS, T. G. Agentes móveis - seu uso no auxílio a um melhor escalonamento em um ambiente paralelo e distribuído. *Geleria.org*. Disponível em: <<http://geleira.org/pdf/agentesprocdistribuido.pdf>>.
- [28] FOSTER, I. What is grid? a three point checklist. Julho 2002. Disponível em: <www-fp.mcs.anl.gov/foster/Articles/WhatIsTheGrid.pdf>.
- [29] JINGYUE, L. *Code Mobility Overview*. [S.l.]. Disponível em: <<http://www.idi.ntnu.no/emner/dif8914/essays/Jingyue-essay2002.pdf>>.
- [30] TANEMBAUM, A. S. *Sistemas Operacionais Modernos*. Rio de Janeiro: Livros técnicos e Científicos Editora, 1999. Traduzido por Nery Machado Filho. Título Original: *Modern Operating Systems*.
- [31] THOMAS, R. M. *Introdução às Redes Locais*. São Paulo: Makron Books, 1997. Traduzido por Altair D. C. de Moraes. Título Original: *Introduction to Local Area Networks*.
- [32] OPENSSI: Single System Image. Maio 2005. Online. Disponível em: <<http://openssi.org/cgi-bin/view?page=openssi.html>>.

- [33] GLOBUS. maio 2005. Online. Disponível em: <<http://www.globus.org>>.
- [34] SETI@HOME Brasil. Maio 2005. Online. Disponível em: <<http://www.setiathome.com.br/>>.
- [35] POPCORN: Global Distributed Computation Over Internet in Java. Online. Disponível em: <<http://www.cs.huji.ac.il/labs/popcorn/>>.

Anexo A - Artigo

Middleware SSI utilizando Agentes Móveis

Tiago Silva Pereira

tiagosp@inf.ufsc.br

Universidade Federal de Santa Catarina

17 de maio de 2005

Abstract

Whit the technological advance each day growing faster, new classes of problems come up in the computer world, and it is our job to solve them. And some times it is not possible to solve some problems, in a life time, using only on computer, even if it is a very powerfull computer.

It is part os this work to show some concepts to distributed, parallel systems and mobile agents. It will present a implementation to *Algets*, based on a agent called Jota [1].

1 Introdução

Sistemas distribuídos estão cada vez mais presentes em nosso meio. Ambientes de clusters deixam de ser privilégio de poucos para se tornar necessidade para muitos.

As aplicações demandam, cada vez mais, um poder de processamento crescente. E as instituições necessitam de respostas imediatas. Este trabalho descreverá brevemente uma solução para se trabalhar com ambiente distribuídos de forma prática, e fácil.

2 Motivação

Este trabalho tem como principal motivação a oportunidade de pesquisar e trabalhar com tecnologias inovadoras e de especial interesse para o autor, a área de redes.

3 Sistemas Distribuídos

Atualmente dois tipos de ambientes distribuídos vem sendo pesquisados e utilizados. Cluster e Grids diferem em alguns pontos, porém, todos os dois tem como objetivo final aumentar o poder de processamento disponível.

Cluster tem como principal característica a dedicação total de todos os nós da rede e são compostos por computadores próximos, que fazem parte de uma LAN (Local Area Network), por exemplo.

Os Grids se caracterizam por englobar nós geograficamente distribuídos, além de permitir que o usuário determine quanto de seu poder computacional pretende compartilhar para a grade [2].

4 Single System Image

Esta nova abordagem permite que o usuário de cluster possa vizualizar de forma unificada os recursos do sistema. Isto facilita a utilização do cluster, uma vez que o usuário trabalha como se estivesse executando tarefas localmente e trata todos os recursos como se fossem recursos locais [3].

5 Agentes Móveis

Um ambiente de agentes móveis é composto por dois elementos, os servidores e os agentes. Os servidores criam um ambiente de execução para o agente, sendo assim os agentes independem do *hardware* das máquinas.



Figura 1: Ilustração do modelo conceitual do *Aglets*

O *Aglets* [4] é uma plataforma desenvolvida pela IBM Japão com o objetivo de pesquisa e desenvolvimento de agentes móveis. O *Aglet* foi totalmente desenvolvido em Java [5], sem alteração alguma na máquina virtual Java, o que o torna portátil. A figura 1 demonstra graficamente o modelo da plataforma.

6 Problema e solução

Os problemas a serem solucionados estão ficando cada vez mais complexos. Isto demanda um poder computacional muito grande, e necessita de soluções específicas.

A utilização de um ambiente distribuído e paralelo é, atualmente, uma das melhores soluções para lidar com este problema. Este trabalho desenvolveu uma implementação do Agente Jota, para que pudesse ser disponibilizado ao programador uma interface única do seu sistema.

A linguagem de programação utilizada no trabalho foi Java, considerando suas vantagens de portabilidade, o que torna fácil no desenvolvimento de agentes quando não se sabe em qual arquitetura irá rodar. Assim um agente *aglet* desenvolvido em Java pode, sem nenhuma alteração, rodar num ambiente *Windows* ou *Linux*.

7 Conclusão

Sistemas distribuídos estão cada vez mais presentes nas instituições, o que transforma este assunto num ponto importante de estudos. Porém, a dificuldade de gerenciar estes sistemas tem se tornado um bloqueio para a sua popularização.

Este trabalho apresentou uma solução no gerenciamento de clusters, utilizando SSI (*Single System Image*) e agentes móveis. A plataforma *Aglets* e o agente Jota foram bases para o desenvolvimento, que gerou um ambiente SSI configurável e fácil, para que o usuário pudesse utilizar sistemas distribuídos com a mesma naturalidade que utiliza uma máquina local.

Referências

- [1] DANTAS, M. A. R.; LOPES, J. G. R. C.; RAMOS, T. G. Agentes móveis - seu uso no auxílio a um melhor escalonamento em um ambiente paralelo e distribuído. *Geleria.org*. Disponível em: <<http://geleira.org/pdf/agentesprocdistribuido.pdf>>.
- [2] BUYYA, R. *Grid Computing Info Center*. Dezembro 2004. Online. Disponível em: <www.gridcomputing.com>.
- [3] OPENSSI: Single System Image. Maio 2005. Online. Disponível em: <<http://openssi.org/cgi-bin/view?page=openssi.html>>.
- [4] AGLETS. maio 2005. Online. Disponível em: <<http://aglets.sourceforge.net/>>.
- [5] SUN. *Java*. Online. Disponível em: <<http://java.sun.com>>.
- [6] ALVARENGA, A. T. H. *Um ambiente para processamento paralelo oportunístico na internet*. Dissertação (Mestrado) — Universidade de Brasília, 2003.
- [7] BIGUS, J. P.; BIGUS, J. *Constructing Intelligent Agents Using Java*. 2. ed. [S.l.]: Professional Developer's Guide, 2001.
- [8] DANTAS, M.; LOPES, J.; RAMOS, T. *An Enhanced Scheduling Approach in a Distributed Parallel Environment Using Mobile Agents*. Brasília, 2002.
- [9] LANGE, D. Mobile objects and mobile agents: The future of distributed computing? *PROCEEDINGS OF THE EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING*

- '98, 1998. Disponível em: <<http://www.moe-lange.com/danny/ecoop98.pdf>>.
- [10] GRID.ORG. janeiro 2005. Online. Disponível em: <<http://www.grid.org>>.
- [11] SETI@HOME Brasil. Maio 2005. Online. Disponível em: <<http://www.setiathome.com.br/>>.
- [12] BEOWULF Cluster. Março 2005. Online. Disponível em: <<http://www.beowulf.org/>>.
- [13] BERKELEY Open Infrastructure for Network Computing. Fevereiro 2005. Online. Disponível em: <<http://boinc.ssl.berkeley.edu/>>.
- [14] GLOBUS. maio 2005. Online. Disponível em: <<http://www.globus.org>>.

Anexo B - Agente Jota

Na figura 6.1 é mostrado a tela inicial do agente Jota, onde algumas configurações preliminares sobre o sistema devem ser preenchidas.

A tela principal do Jota é ilustrada pela figura 6.2. Nesta figura podemos observar oito espaços em branco, onde serão preenchidos com as informações dos nós. Inicialmente nenhuma informação é mostrada até que seja clicado no botão obter informações. Este botão irá criar um agente JotaSlave, que será enviado ao destinatário e recolherá informações do mesmo. Após esta coleta, retornará ao seu ponto de partida com as informações dos *hosts*.

Podemos verificar na figura 6.2 a tela de configuração dos nós. É possível especificar o endereço ip do *host* e a porta na qual o servidor *Algets* está configurado. Desta forma é possível executar diversas instâncias do servidor *Aglets* numa mesma máquina e cada servidor irá ser executado em uma porta. Isso possibilita a simulação de um ambiente distribuído, por exemplo.

Na tela de opções mostrada na figura 6.4 é possível selecionar uma série de atributos que se deseja recolher dos nós do sistema. É ainda possível configurar onde estão as classes que serão utilizadas pelo servidor caso algum agente seja criado.



Figura 6.1: Janela Inicial do agente Jota

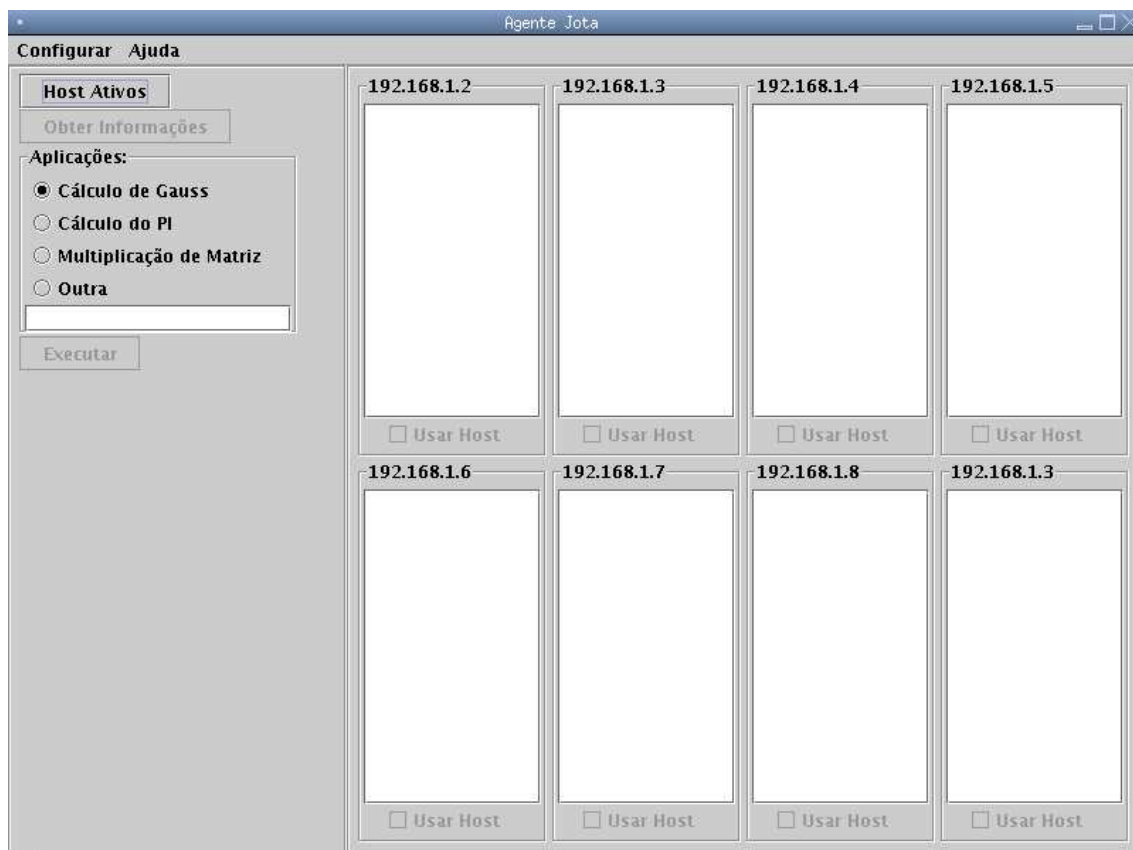


Figura 6.2: Janela Principal - JotaMaster

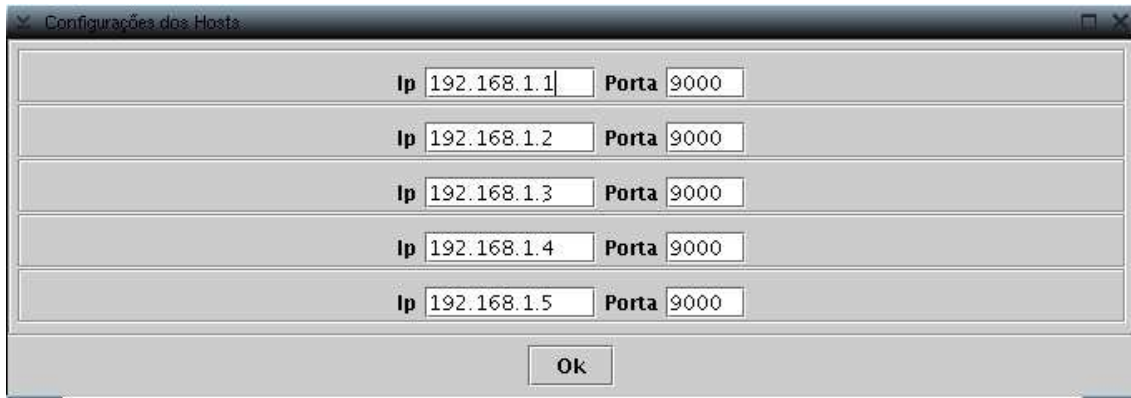


Figura 6.3: Configuração dos *Hosts*

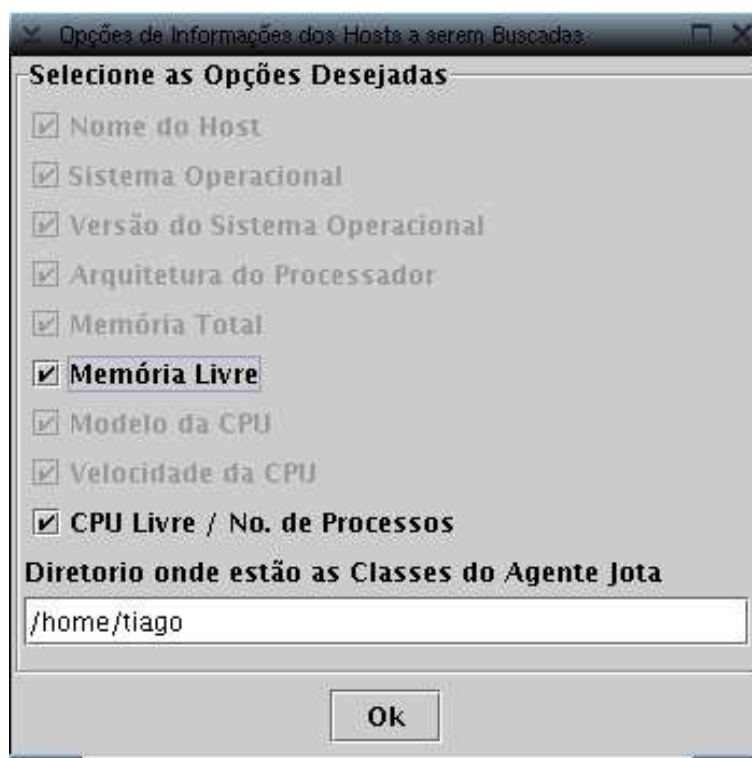


Figura 6.4: Tela de opções

Anexo C - Código Fonte da Aplicação

Classe Endereço

```
1 package examples.agente;
2
3 import java.lang.*;
4 import java.io.*;
5
6 public class Endereco {
7     private String ip;
8     public Endereco(String ip){
9         this.ip = ip;
10    }
11    public void setIp(String ip){
12        this.ip = ip;
13    }
14
15    public String getIp(){
16        return(ip);
17    }
18    public String getAtp(){
19        String atp;
20        atp="atp://" + ip + ":4434";
21        return(atp);
22    }
23
24    public String toAtp(String ip){
25        String atp="atp://" + ip + ":4434";
26        return(atp);
27    }
28 }
29 }
```

Classe Host2

```
1 package examples.agente;
2 import java.lang.*;
3 import java.io.*;
```

```

4
5
6
7
8 public class Host2 {
9     public static void main(String args [])
10    {
11
12
13        Process p=null;
14        Process prUname=null;
15
16        try{
17            System.out.println("executando _a_thread_do_host");
18
19            p=Runtime.getRuntime().exec("/bin/hostname");
20            prUname=Runtime.getRuntime().exec("uname_-mrs");
21
22            InputStream fora= p.getInputStream();
23            InputStream SaidaUname= prUname.getInputStream();
24
25            int i=0;
26            String s="";
27            String SO="";
28            String VersaoSO="";
29            String ArquiteturaProcessador="";
30
31            try{
32
33                byte b[]= new byte[280];
34                byte VetorSaidaUname [] = new byte[255];
35
36                SaidaUname.read(VetorSaidaUname);
37                fora.read(b);
38
39                while (i<255){
40                    if (java.lang.String.valueOf((char)b[i])=="\n"){
41                        break;
42                    }//if
43                    s=s+java.lang.String.valueOf((char)b[i]);
44                    i++;
45                }//while
46
47                i=0;
48                while (i<255){
49                    /* while (java.lang.String.valueOf((char)
VetorSaidaUname[i])!="\n"){
50                        SO = SO + java.lang.String.valueOf((char)
VetorSaidaUname[i]);
51                        i++;
52                    }//while SO
53
54                        i++;

```

```

55         while (java.lang.String.valueOf((char)
VetorSaidaUname[i])!=" "){
56             VersaoSO = VersaoSO + java.lang.String.valueOf
((char)VetorSaidaUname[i]);
57             i++;
58             }//Versao do SO
59
60             i++;
61         while (java.lang.String.valueOf((char)
VetorSaidaUname[i])!=" "){
62             ArquiteturaProcessador= ArquiteturaProcessador
+ java.lang.String.valueOf((char)VetorSaidaUname[i]);
63             i++;
64             }//Arquitetura do Processador
65     */
66         //if (java.lang.String.valueOf((char)
VetorSaidaUname[i])==" "){
67             // break;
68             }//if
69
70         SO = SO + java.lang.String.valueOf((char)
VetorSaidaUname[i]);
71         i++;
72         /* while (SO.substring(i-1,1)!=" "){
73             SO = SO + java.lang.String.valueOf((char)
VetorSaidaUname[i]);
74             i++;
75             }//while SO
76
77
78     /* VersaoSO = VersaoSO + java.lang.String.valueOf
((char)VetorSaidaUname[i]);
79         i++;
80         while (VersaoSO.substring(i-1,1)!=" "){
81             VersaoSO = VersaoSO + java.lang.String.valueOf
((char)VetorSaidaUname[i]);
82             i++;
83             }//while VersaoSO
84
85         ArquiteturaProcessador =
ArquiteturaProcessador + java.lang.String.valueOf((char)VetorSaidaUname
[i]);
86         i++;
87         while (ArquiteturaProcessador.substring(i-1,1)!="
") {
88             ArquiteturaProcessador = ArquiteturaProcessador
+ java.lang.String.valueOf((char)VetorSaidaUname[i]);
89             i++;
90             }//while ArquiteturaProcessador
91     */
92         break;
93
94     }//while Uname
95

```

```

96         System.out.println("O_Hostname_é:" + s);
97         System.out.println("O_SO_é:" + SO);
98         System.out.println("A_ãVero_do_SO:" + VersaoSO);
99         System.out.println("A_Arquitetura_do_Processador_é:"
+ ArquiteturaProcessador);
100     }catch (Exception e){
101         System.out.println("erro_na_string");
102     }//catch
103 }catch (Exception e){
104     System.out.println("erro_no_Comando");
105 }//catch
106
107 }
108 }

```

Classe HostPanel

```

1 package examples.agente;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6 import javax.swing.border.*;
7
8
9 import java.lang.*;
10 import java.io.*;
11
12 public class HostPanel extends JPanel{
13
14     ImageIcon imagem;
15     ImageIcon imagem2;
16     ImageIcon imagemNaoDisponivel;
17     JLabel imagemLabel;
18
19     JCheckBox selecionarBox;
20     String diretorio = "/home/tiago/facu/projeto/aglets/public/examples
/agente/";
21
22     boolean estaAtivo;
23     String so="";
24     String versaoSo="";
25     String nomeHost="";
26     String MemTotal="";
27     String MemLivre="";
28     String arquiteturaProcessador="";
29     float memTotal;
30     float memLivre;
31     String CPULivre="10.45";
32     String CPUModelo="";
33     String CPUMHz="";

```

```

34     String numeroProcessos="";
35     JTextArea text = new JTextArea(10,50);
36     JScrollPane scroll = new JScrollPane(text);
37
38     PainelNaoDisponivel painelNaoDisponivel = new PainelNaoDisponivel(
diretorio);
39
40     Endereco endereco = new Endereco("192.168.1.2");
41
42     //Criando o Painel para os Hosts com suas çõinformaes
43     public HostPanel(String title){
44         endereco.setIp(title);
45
46         setBorder(BorderFactory.createTitledBorder
47             (BorderFactory.createEtchedBorder(),title));
48         setLayout(new BorderLayout(this,BoxLayout.Y_AXIS));
49
50         //ImageIcon imagem = new ImageIcon(diretorio+" gifs/
computador_pb.png");
51         imagemLabel = new JLabel("");
52         imagemLabel.setOpaque(true);
53         //imagemLabel.setIcon(imagem);
54         add(imagemLabel);
55
56         //add(text);
57         add(scroll);
58         text.setLineWrap(true);
59         text.setEditable(false);
60
61         selecionarBox = new JCheckBox("Usar Host");
62         //selecionarBox.addActionListener(this);
63         selecionarBox.setEnabled(false);
64         //selecionarBox.setEnabled(true);
65         add(selecionarBox);
66     }
67
68     public final void setDisponivel(boolean disponivel){
69         if (!disponivel){
70             remove(scroll);
71             remove(selecionarBox);
72             add(painelNaoDisponivel);
73             selecionarBox.setEnabled(false);
74             add(selecionarBox);
75         }
76         else{
77             remove(painelNaoDisponivel);
78             add(scroll);
79             add(selecionarBox);
80         }
81     }
82
83     public final void setAtivo(boolean host_ativo){
84

```

```

85         if (!host_ativo) {
86             //ImageIcon imagem2 = new ImageIcon(diretorio+" gifs/
computador_verde.png");
87             imagemLabel.setOpaque(false);
88             //imagemLabel.setIcon(imagem2);
89             selecionarBox.setEnabled(true);
90             selecionarBox.setSelected(false);
91             setDisponivel(true);
92             revalidate();
93             estaAtivo=true;
94         }
95         else{
96             //ImageIcon imagem2 = new ImageIcon(diretorio+" gifs/
computador_red.png");
97             imagemLabel.setOpaque(true);
98             //imagemLabel.setIcon(imagem2);
99             selecionarBox.setEnabled(false);
100            selecionarBox.setSelected(false);
101            setDisponivel(true);
102            revalidate();
103            estaAtivo=false;
104        }
105    }
106
107    public final void setTitulo(String titulo){
108        setBorder(BorderFactory.createTitledBorder
109            (BorderFactory.createEtchedBorder(), titulo));
110    }
111
112    public final void setInformacoes(String bufferInformacoes){
113
114        int indicePipe = 0;
115        int pipeAnterior = 0;
116
117        System.out.println(bufferInformacoes);
118        bufferInformacoes.trim();
119        indicePipe=bufferInformacoes.indexOf("|");
120        System.out.println("_Indice_do_Pipe_:" +indicePipe);
121
122        nomeHost=bufferInformacoes.substring(0,indicePipe);
123
124        System.out.println("Nome_do_Host_:" +nomeHost);
125
126        pipeAnterior=++indicePipe;
127        indicePipe=bufferInformacoes.indexOf("|", indicePipe);
128        so = bufferInformacoes.substring(pipeAnterior, indicePipe++);
129
130        pipeAnterior=++indicePipe;
131        indicePipe=bufferInformacoes.indexOf("|", indicePipe);
132        versaoSo = bufferInformacoes.substring(--pipeAnterior,
indicePipe++);
133
134        pipeAnterior=++indicePipe;

```



```

135         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
136         arquiteturaProcessador = bufferInformacoes.substring(--
pipeAnterior , indicePipe++);
137
138         pipeAnterior=++indicePipe;
139         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
140         MemTotal = bufferInformacoes.substring(--pipeAnterior ,
indicePipe++);
141         memTotal = ConverteEmMB(MemTotal);
142
143         pipeAnterior=++indicePipe;
144         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
145         MemLivre = bufferInformacoes.substring(--pipeAnterior ,
indicePipe++);
146         memLivre = ConverteEmMB(MemLivre);
147
148         pipeAnterior=++indicePipe;
149         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
150         CPUMHz = bufferInformacoes.substring(--pipeAnterior , indicePipe
++);
151
152         pipeAnterior=++indicePipe;
153         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
154         CPUModelo = bufferInformacoes.substring(--pipeAnterior ,
indicePipe++);
155
156         pipeAnterior=++indicePipe;
157         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
158         CPULivre = bufferInformacoes.substring(--pipeAnterior ,
indicePipe++);
159
160         pipeAnterior=++indicePipe;
161         indicePipe=bufferInformacoes.indexOf("|",indicePipe);
162         numeroProcessos= bufferInformacoes.substring(--pipeAnterior ,
bufferInformacoes.length());
163         setTitulo(nomeHost);
164         mostraInformacoes();
165     }
166     public final void mostraInformacoes(){
167         limpaPainel();
168
169         text.append(" Host: ͇"+nomeHost+"\n");
170         text.append(" SO: ͇"+so+"\n");
171         text.append(" ͇Verso: ͇"+versaoSo+"\n");
172         text.append(" Arquitetura: ͇"+arquiteturaProcessador+"\n");
173         text.append(" Mem. ͇Total: ͇"+memTotal+" MB"+" \n");
174         text.append(" Mem. ͇Livre: ͇"+memLivre+" MB"+" \n");
175         text.append(" CPU: ͇"+CPUMHz+" MHz"+" \n");
176         text.append(" Modelo: ͇"+CPUModelo+"\n");
177         text.append(" CPU ͇Livre: ͇"+CPULivre+"%\n");
178         text.append(" Processos: ͇"+numeroProcessos+"\n");
179     }
180
181     public final void limpaPainel(){

```

```

182         text.selectAll();
183         text.replaceSelection("");
184     }
185
186     public float ConverteEmMB(String mem){
187         return(Integer.parseInt(mem)/1024);
188     }
189
190 }
191
192 class PainelNaoDisponivel extends JPanel{
193     JLabel imagemLabelNaoDisponivel;
194
195     public PainelNaoDisponivel(String diretorio){
196         //ImageIcon imagemNaoDisponivel = new ImageIcon(diretorio+" gifs
/nd.png");
197         imagemLabelNaoDisponivel = new JLabel("");
198         imagemLabelNaoDisponivel.setOpaque(true);
199         //imagemLabelNaoDisponivel.setIcon(imagemNaoDisponivel);
200
201         add(imagemLabelNaoDisponivel);
202     }
203 }// class PainelNaoDisponivel

```

Classe InformaçãoHost

```

1 package examples.agente;
2
3 import java.lang.*;
4 import java.io.*;
5 import java.net.*;
6 /*
7 *import de tese
8 */
9
10 //import java.text.*;
11
12 public class InformacaoHost implements Serializable{
13
14     public static String nomeHost="";
15     public static String buffer="";
16     public static String so="";
17     public static String versaoSo="";
18     public static String arquiteturaProcessador="";
19     public static int painel;
20     public static String CPULivre="";
21     public static String CPUModelo="";
22     public static String CPUMHz="";
23     public static String MemTotal="";
24     public static String MemLivre="";
25     public static String numeroProcessos="";

```

```

26     InetAddress maquinaLocal;
27     int i=0;
28
29     Process p = null;
30     String bufferMem="";
31     String bufferCPU="";
32
33     public InformacaoHost(int painel){
34         this.painel=painel;
35         so = System.getProperty("os.name", "Unknown");
36         versaoSo = System.getProperty("os.version", "Unknown");
37         arquiteturaProcessador = System.getProperty("os.arch", "Unknown
38     ");
39         verificaMemoriaTotal();
40         verificaMemoriaLivre();
41         verificaCPUMHz();
42         verificaCPUModelo();
43         verificaCPULivreProcessos();
44         buffer= ""+painel+" | "+nomeHost+" | "+so+" | "+versaoSo;
45         buffer= buffer+" | "+arquiteturaProcessador+" | "+MemTotal+" | "+
MemLivre;
46         buffer = buffer+" | "+CPUMHz+" | "+CPUModelo+" | "+CPULivre+" | "+
numeroProcessos;
47     }
48
49     public final void MostraResultado() {
50         System.out.println("Host: \"+nomeHost+"\n"+"SO: \"+so+"\n"+"
Versao do SO: \"+versaoSo+"\n"+
51         "Arquitetura do Processador: "+arquiteturaProcessador+"\n"
52         +" Painel: \"+painel+"\n"+" Processos: \"+numeroProcessos);
53     }
54
55     public final String verificaMemoriaTotal() {
56         MemTotal="";
57         p=null;
58         try {
59             p=Runtime.getRuntime().exec("/bin/grep \"+MemTotal+"/proc/
meminfo");
60             InputStream fora= p.getInputStream();
61             try {
62
63                 byte b[]= new byte[30];
64                 fora.read(b);
65                 i = 9;
66                 while (i<30){
67                     MemTotal=MemTotal+java.lang.String.valueOf((char)b[
i]);
68                     i++;
69                 }//while
70                 MemTotal = MemTotal.trim();
71                 MemTotal = MemTotal.substring(0,MemTotal.length()-3);
72

```

```

73         }catch (Exception e){
74             System.out.println("erro_nao_string");
75         }//catch
76         p.destroy();
77     }catch (Exception e){
78         System.out.println("erro_no_Comando");
79     }
80     return MemTotal;
81 }
82
83 public final String verificaMemoriaLivre(){
84     MemLivre = "";
85     p=null;
86     try{
87 meminfo");
88         p=Runtime.getRuntime().exec("/bin/grep_MemFree_/proc/
89         InputStream fora= p.getInputStream();
90         try{
91             byte b[]= new byte[30];
92             fora.read(b);
93             i = 8;
94             while (i<30){
95                 MemLivre=MemLivre+java.lang.String.valueOf((char)b[
96                 i]);
97                 i++;
98             }//while
99             MemLivre = MemLivre.trim();
100            MemLivre = MemLivre.substring(0,MemLivre.length()-3);
101        }catch (Exception e){
102            System.out.println("erro_nao_string");
103        }//catch
104        p.destroy();
105    }catch (Exception e){
106        System.out.println("erro_no_Comando_da_óMemria_Livre");
107    }
108    return MemLivre;
109 }
110
111 public final String verificaCPUMHz(){
112     CPUMHz = "";
113     p=null;
114     try{
115         p=Runtime.getRuntime().exec("/bin/grep_MHz_/proc/cpuinfo");
116         InputStream fora= p.getInputStream();
117         try{
118             byte b[]= new byte[30];
119             fora.read(b);
120             i = 7;
121             while (i<30){
122                 CPUMHz=CPUMHz+java.lang.String.valueOf((char)b[i]);
123                 i++;
124             }//while
125             CPUMHz = CPUMHz.trim();

```

```

124         CPUMHz = CPUMHz.substring(2,CPUMHz.length());
125         CPUMHz = CPUMHz.substring(0, CPUMHz.indexOf("."));
126     }catch (Exception e){
127         System.out.println("erro_na_string");
128     }//catch
129     p.destroy();
130 }catch (Exception e){
131     System.out.println("erro_no_Comando_da_CPU_MHz");
132 }
133     return CPUMHz;
134 }
135
136 public final String verificaCPUModelo(){
137     CPUModelo = "";
138     p=null;
139     try{
140         p=Runtime.getRuntime().exec("/bin/grep _name_/proc/cpuinfo")
;
141         InputStream fora= p.getInputStream();
142         try{
143             byte b[]= new byte[40];
144             fora.read(b);
145             i = 10;
146             while (i<40){
147                 CPUModelo=CPUModelo+java.lang.String.valueOf((char)
b[i]);
148                 i++;
149             }//while
150             CPUModelo = CPUModelo.trim();
151             CPUModelo = CPUModelo.substring(2,CPUModelo.length());
152             CPUModelo = CPUModelo.trim();
153         }catch (Exception e){
154             System.out.println("erro_na_string");
155         }//catch
156         p.destroy();
157     }catch (Exception e){
158         System.out.println("erro_no_Comando_da_CPU_Modelo");
159     }
160     return CPUModelo;
161 }
162
163 public final void verificaCPULivreProcessos(){
164     numeroProcessos = "_";
165     CPULivre = "_";
166     p=null;
167     try{
168         p=Runtime.getRuntime().exec("/bin/sh_/aglets1.2.0/public/
examples/agente/dadosdinhost.sh");
169         InputStream fora= p.getInputStream();
170
171         try{
172             byte b[]= new byte[20];
173             fora.read(b);

```

```

174         i = 0;
175         while (i<10){
176             if (isNumber(java.lang.String.valueOf((char)b[i])))
177             {
178                 numeroProcessos=numeroProcessos+java.lang.
String.valueOf((char)b[i]);
179             }
180             else if (java.lang.String.valueOf((char)b[i]).
equals(".")){
181                 numeroProcessos=numeroProcessos+java.lang.
String.valueOf((char)b[i]);
182             }
183             else if (java.lang.String.valueOf((char)b[i]).
equals("\n")){
184                 numeroProcessos=numeroProcessos+java.lang.
String.valueOf((char)b[i]);
185             }
186             i++;
187         }//while
188         numeroProcessos.trim();
189         CPULivre = numeroProcessos.substring(0,numeroProcessos.
indexOf("\n"));
190         CPULivre.trim();
191         i= numeroProcessos.indexOf("\n");
192         numeroProcessos = numeroProcessos.substring(++i,
numeroProcessos.length());
193         numeroProcessos.trim();
194         System.out.println("CPU_Livre: "+CPULivre+"%");
195         System.out.println("úNmero_de_Processos: "+
numeroProcessos);
196     }catch (Exception e){
197         System.out.println("erro_na_string");
198     }//catch
199     p.destroy();
200 }catch (Exception e){
201     System.out.println("erro_no_Comando_do_úmero_de_processos"
);
202 }
203
204 public final boolean isNumber(String str){
205     try{
206         Integer.parseInt(str);
207         return true;
208     }catch (NumberFormatException e){
209         return false;
210     }
211 }
212
213 }
214

```

Classe JotaMaster

```

1 package examples.agente;
2
3 import com.ibm.aglet.*;
4
5
6 import java.io.Externalizable;
7 import java.io.ObjectInput;
8 import java.io.ObjectOutput;
9 import java.io.IOException;
10 import java.net.URL;
11
12 public class JotaMaster extends Aglet {
13     transient AgletProxy [] remoteProxy = new AgletProxy [10];
14     AgletProxy [] proxy = new AgletProxy [10];
15
16     String name = "Master";
17     Tela_Jota window = null;
18     Tela_Inicial tela;
19     String [] destino = new String [10];
20     int i=1;
21     int j=1;
22     int k=1;
23
24     Process ping = null;
25
26     public void separaAtp(String buffer){
27         int pipeAnterior=0;
28         i =1;
29         int indicePipe=buffer.indexOf("|");
30         destino [0]="atp://192.168.1.2:4434";
31         while (indicePipe < buffer.lastIndexOf("|")){
32             destino [i]=buffer.substring(pipeAnterior ,indicePipe);
33             System.out.println(destino [i]);
34             System.out.println(i);
35             i++;
36             System.out.println(i);
37             pipeAnterior=++indicePipe;
38             indicePipe=buffer.indexOf("|",indicePipe);
39         }
40         System.out.println(i);
41         destino [i]=buffer.substring(pipeAnterior ,indicePipe);
42         System.out.println(destino [i]);
43     }
44
45     public void separaPainel(String buffer){
46         int painel;
47         buffer.trim();
48         int indicePipe=buffer.indexOf("|");
49         String buffer2="";
50
51         System.out.println(buffer.substring(0,1));

```

```

52     System.out.println(buffer.substring(0, indicePipe));
53
54     painel=Integer.parseInt(buffer.substring(0, indicePipe));
55
56     buffer2=buffer.substring(++indicePipe, buffer.length());
57     System.out.println(buffer2);
58     window.painelLaico[painel].setInformacoes(buffer2);
59 }
60
61 public void setMemLivre(String buffer){
62     int painel;
63     buffer.trim();
64     int indicePipe=buffer.indexOf("|");
65     String buffer2="";
66
67     System.out.println(buffer.substring(0,1));
68     System.out.println(buffer.substring(0, indicePipe));
69
70     painel=Integer.parseInt(buffer.substring(0, indicePipe));
71
72     buffer2=buffer.substring(++indicePipe, buffer.length());
73     System.out.println(buffer2);
74     window.painelLaico[painel].memLivre=
75         window.painelLaico[painel].ConverteEmMB(buffer2);
76     window.painelLaico[painel].mostraInformacoes();
77 }
78
79 public void setCPULivre(String buffer){
80     int painel;
81     buffer.trim();
82     int indicePipe=buffer.indexOf("|");
83     String buffer2="";
84
85     System.out.println(buffer.substring(0,1));
86     System.out.println(buffer.substring(0, indicePipe));
87
88     painel=Integer.parseInt(buffer.substring(0, indicePipe));
89
90     buffer2=buffer.substring(++indicePipe, buffer.length());
91     System.out.println(buffer2);
92     window.painelLaico[painel].CPULivre=buffer2;
93     window.painelLaico[painel].mostraInformacoes();
94 }
95 public void setProcessos(String buffer){
96     int painel;
97     buffer.trim();
98     int indicePipe=buffer.indexOf("|");
99     String buffer2="";
100
101     System.out.println(buffer.substring(0,1));
102     System.out.println(buffer.substring(0, indicePipe));
103
104     painel=Integer.parseInt(buffer.substring(0, indicePipe));

```



```

105
106     buffer2=buffer.substring(++indicePipe , buffer.length());
107     System.out.println(buffer2);
108     window.painelLaico [painel].numeroProcessos=buffer2;
109     window.painelLaico [painel].mostraInformacoes();
110 }
111
112 public void dispatchSlave(String dest) {
113
114     try{
115         for (j=1;j<=i;j++){
116             try{
117                 if (remoteProxy[j] != null)
118                     remoteProxy[j].sendMessage(new Message("bye"));
119             }catch (InvalidAgletException ex) {
120                 ex.printStackTrace();
121             }catch (Exception ex) {
122                 ex.printStackTrace();
123             }
124         }
125
126         separaAtp(dest);
127         AgletContext context = getAgletContext();
128         for (j=1;j<=i;j++) {
129             try{
130                 proxy[j] = context.createAglet(null,"examples.
131 agente.JotaSlave",getProxy());
132                 remoteProxy[j] = proxy[j].dispatch(new URL(destino [
133 j]));
134                 for (k=1;k<=8;k++){
135                     try{
136                         if (destino [j].equals(window.painelLaico [k].
137 endereco.getAtp())){
138                             window.painelLaico [k].setDisponivel(true);
139                             window.painelLaico [k].revalidate();
140                             System.out.println("destino_"+j+"painel_"+k
141 );
142                             remoteProxy[j].sendMessage(new Message("
143 painel", ""+k));
144                         }
145                     }catch (Exception ex){
146                         ex.printStackTrace();
147                     }
148                 }
149             }catch (InvalidAgletException ex) {
150                 ex.printStackTrace();
151                 System.out.println("Erro_no_Slave_
152 InvalidAgletExcepiton"+j);
153             }catch (Exception ex) {
154                 ex.printStackTrace();
155             }
156         }
157     }

```

```

152         for (k=1;k<=8;k++){
153             try{
154                 if (destino [j].equals(window.painelLaico [k].
endereco.getAtp())){
155                     window.painelLaico [k].setDisponivel ( false );
156                     window.painelLaico [k].revalidate ();
157                 }
158             } catch (Exception ex2){
159                 ex2.printStackTrace ();
160             }
161         }
162
163         proxy [j].dispose ();
164         System.out.println ("####_Erro_no_Slave_Exception"+j
);
165     }
166
167     }
168     } catch (Exception ex) {
169         ex.printStackTrace ();
170     }
171 }
172
173
174 public boolean handleMessage(Message msg) {
175     if (msg.sameKind(" dialog")) {
176         window.show ();
177     }
178     else if (msg.sameKind(" info")) {
179         if (window.isVisible () == false) {
180             window.show ();
181         }
182         window.appendText ((String)msg.getArg ());
183         separaPainel ((String)msg.getArg ());
184         return true;
185     }
186     else if (msg.sameKind(" memLivre")) {
187         if (window.isVisible () == false) {
188             window.show ();
189         }
190         window.appendText ((String)msg.getArg ());
191         setMemLivre ((String)msg.getArg ());
192         return true;
193     }
194     else if (msg.sameKind(" CPULivre")) {
195         if (window.isVisible () == false) {
196             window.show ();
197         }
198         window.appendText ((String)msg.getArg ());
199         setCPULivre ((String)msg.getArg ());
200         return true;
201     }
202     else if (msg.sameKind(" processos")) {

```

```

203         if (window.isVisible() == false) {
204             window.show();
205         }
206         window.appendText((String)msg.getArg());
207         setProcessos((String)msg.getArg());
208         return true;
209     }
210     return false;
211 }
212
213 public void onCreate(Object o) {
214
215     //window = new Tela_Jota(this);
216     //window.setSize(800,600);
217     tela = new Tela_Inicial(this);
218     tela.setSize(400,200);
219     tela.show();
220     //window.show();
221 }
222
223 public void onDisposing() {
224     if (window != null) {
225         tela.dispose();
226         tela = null;
227         window.dispose();
228         window = null;
229     }
230     System.out.println("Entrou_no_sendRequest_info");
231     try {
232         for (j=1;j<=i;j++) {
233             if (remoteProxy[j] != null)
234                 remoteProxy[j].sendMessage(new Message("bye"));
235         }
236     } catch (Exception ex) {
237         ex.printStackTrace();
238     }
239 }
240
241 void sendText(String text) {
242     try {
243         for (j=1;j<=i;j++) {
244             if (remoteProxy[j] != null)
245                 remoteProxy[j].sendMessage(new Message("text",name
+ " : " + text));
246         }
247     } catch (Exception ex) {
248         ex.printStackTrace();
249     }
250 }
251
252 void sendRequest() {
253     System.out.println("Entrou_no_sendRequest_info");
254     try {

```

```

255         for (j=1;j<=i;j++) {
256             if (remoteProxy[j] != null)
257                 remoteProxy[j].sendMessage(new Message(" info"));
258         }
259     } catch (Exception ex) {
260         ex.printStackTrace();
261     }
262 }
263
264 void sendRequestMemLivre() {
265     System.out.println(" Entrou no sendRequest memLivre");
266     try {
267         for (j=1;j<=i;j++) {
268             if (remoteProxy[j] != null)
269                 remoteProxy[j].sendMessage(new Message(" memLivre"))
;
270         }
271     } catch (Exception ex) {
272         ex.printStackTrace();
273     }
274 }
275
276 void sendRequestCPULivreProcessos() {
277     System.out.println(" Entrou no sendRequest CPULivreProcessos");
278     try {
279         for (j=1;j<=i;j++) {
280             if (remoteProxy[j] != null)
281                 remoteProxy[j].sendMessage(new Message("
CPULivreProcessos"));
282         }
283     } catch (Exception ex) {
284         ex.printStackTrace();
285     }
286 }
287
288 void sendPainel() {
289     System.out.println(" Entrou no sendPainel painel");
290     try {
291         for (j=1;j<=i;j++) {
292             for (k=1;k<=8;k++){
293                 try{
294                     if (destino[j].equals(window.painelLaico[k].
endereco.getAtp())){
295                         System.out.println(" destino "+j+" painel "+k
);
296                         remoteProxy[j].sendMessage(new Message("
painel", ""+k));
297                     }
298                 } catch (Exception ex){
299                     ex.printStackTrace();
300                 }
301             }
302         }

```

```

303         }catch (Exception ex) {
304             ex.printStackTrace();
305         }
306     }
307
308     public final boolean verificarHostAtivo(String ip){
309         try{
310             ping = Runtime.getRuntime().exec("/bin/ping -c 1 "+ip);
311             ping.waitFor();
312             try{
313                 if (ping.exitValue() == 1) {
314                     System.out.println("Nao_Recebeu_o_Pacote_do_Host" +
ip);
315                         return(true);
316                     }
317                     else if (ping.exitValue() == 0) {
318                         System.out.println("Host_Ativo");
319                         return(false);
320                     }
321                 }catch (Exception x){
322                     System.out.println("Erro_nos_prints_no_
VerificarHostAtivo");
323                 }
324             }catch (Exception e){
325                 System.out.println("Erro_no_Ping");
326             }
327             return(false);
328         }
329     }
330 }

```

Classe JotaSlave

```

1  package examples.agente;
2
3
4  import com.ibm.aglet.*;
5  import com.ibm.aglet.event.*;
6
7
8  import java.io.ObjectInput;
9  import java.io.ObjectOutput;
10 import java.io.IOException;
11 import java.net.URL;
12 import java.lang.Object;
13
14 public class JotaSlave extends Aglet {
15
16     transient Tela_Jota window = null;
17     transient String nomeHost= "Desconhecido";
18     AgletProxy masterProxy = null;

```

```

19 InformacaoHost informacao_host;
20 int pertencePainel;
21 AgletContext ac;
22
23 public JotaSlave() {}
24
25 public boolean handleMessage(Message msg) {
26     if (msg.sameKind("dialog")) {
27         window.show();
28     }
29     else if (msg.sameKind("text")) {
30         String str = (String)msg.getArg();
31         //getInfo();
32         if (window.isVisible() == false) {
33             window.show();
34         }
35         window.appendText(str);
36         return true;
37     }
38     else if (msg.sameKind("bye")) {
39         window.appendText("Bye_Bye..");
40         try {
41
42             }catch (Exception ex) {}
43         msg.sendReply();
44         dispose();
45     }
46     else if (msg.sameKind("painel")) {
47         String str = (String)msg.getArg();
48         pertencePainel=Integer.parseInt(str);
49
50         if (window.isVisible() == false) {
51             window.show();
52         }
53
54         window.appendText("Painel_deste_Slave_"+pertencePainel);
55         return true;
56     }
57     else if (msg.sameKind("info")) {
58
59         window.appendText("Buscando_çõInformaes...");
60         try{
61             informacao_host.nomeHost=ac.getHostingURL().getHost().
toString();
62             getInfo();
63         } catch (Exception ex1) {
64             System.out.println("Erro");
65         }
66
67         if (window.isVisible() == false) {
68             window.show();
69         }
70         return true;

```

```

71     }
72
73     else if (msg.sameKind("CPULivreProcessos")) {
74
75         window.appendText("Buscando_CPU_Livre_e_Numero_de_Processos
76         ...");
77         try{
78             informacao_host.verificaCPULivreProcessos();
79             sendInfo("CPULivre",informacao_host.painel+"|"+
informacao_host.CPULivre);
80             sendInfo("processos",informacao_host.painel+"|"+
informacao_host.numeroProcessos);
81         } catch (Exception ex1) {
82             System.out.println("Erro");
83         }
84
85         if (window.isVisible() == false) {
86             window.show();
87         }
88         return true;
89     }
90
91     else if (msg.sameKind("memLivre")) {
92
93         window.appendText("Buscando_memLivre...");
94         try{
95             sendInfo("memLivre",informacao_host.painel+"|"+
informacao_host.verificaMemoriaLivre());
96         } catch (Exception ex1) {
97             System.out.println("Erro");
98         }
99
100        if (window.isVisible() == false) {
101            window.show();
102        }
103        return true;
104    }
105
106    return false;
107
108    public void onCreate(Object o) {
109        masterProxy = (AgletProxy)o;
110        addMobilityListener(new MobilityAdapter() {
111            public void onArrival(MobilityEvent ev) {
112                window = new Tela_Jota(JotaSlave.this);
113                window.show();
114                ac = getAgletContext();
115
116                try {
117                    if (ac.getHostingURL() == null) {
118                        nomeHost = "Desconhecido";
119                    }

```

```

120         else {
121             nomeHost = ac.getHostingURL().getHost().
toString();
122         }
123
124         }catch (Exception ex) {
125             ex.printStackTrace();
126         }
127     }
128 });
129 }
130
131 public void onDisposing() {
132     if (window != null) {
133         window.dispose();
134         window = null;
135     }
136 }
137
138 private void print(String m) {
139     System.out.println("Receiver: " + m);
140 }
141
142 public void sendInfo(String tipo, String text) {
143     try {
144         if (masterProxy == null) {
145             return;
146         }
147         masterProxy.sendMessage(new Message(tipo, text));
148     }catch (Exception ex) {
149         ex.printStackTrace();
150     }
151 }
152
153 public void getInfo(){
154     try {
155         System.out.println("Criando a thread de Saida");
156         informacao_host = new InformacaoHost(pertencePainel);
157         informacao_host.MostraResultado();
158         sendInfo("info",informacao_host.buffer);
159     }catch (Exception ex){
160         System.out.println("Erro na chamada da Thread!!!!!!!!!!");
161         ex.printStackTrace();
162     }
163 }
164 }

```

Classe Tela_Configuracoes

```

1 package examples.agente;
2

```



```

3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6 import javax.swing.border.*;
7
8
9 import java.lang.*;
10 import java.io.*;
11
12 public class Tela_Configuracoes extends JDialog{
13     public painelIpPorta [] painel;
14     int i = 0;
15     boolean ok;
16     JButton botaoOk;
17     JPanel painelConfiguracaoHosts = new JPanel();
18     String directorio = "/home/tiago/aglets/conf/";
19     Configuracoes conf = new Configuracoes("");
20
21     public Tela_Configuracoes(JFrame parent){
22
23         super(parent, "çõConfiguraes dos Hosts", true);
24
25         painelConfiguracaoHosts.setBorder(BorderFactory.
26 createTitledBorder
27         (BorderFactory.createEtchedBorder(), ""));
28         painelConfiguracaoHosts.setLayout(new GridLayout(6,2));
29         painel = new painelIpPorta[10];
30
31         for (i=1; i<=9;i++) {
32             painel[i] = new painelIpPorta();
33             painelConfiguracaoHosts.add(painel[i]);
34         }
35
36         setSize(680,200);
37
38         JPanel p2 = new JPanel();
39         JButton botaoOk = new JButton("Ok");
40         p2.add(botaoOk);
41
42         Container contentPane = getContentPane();
43         contentPane.add(painelConfiguracaoHosts, "Center");
44         contentPane.add(p2, "South");
45
46         botaoOk.addActionListener(new ActionListener(){
47             public void actionPerformed(ActionEvent evt){
48                 ok =true;
49                 setVisible(false);
50             }
51         });
52     } // public Tela_Configuracoes ()
53
54     public Tela_Configuracoes(JFrame parent, Configuracoes conf2){

```

```

55         super(parent, "çõConfiguraes_dos_Hosts", true);
56
57         painelConfiguracaoHosts.setBorder(BorderFactory.
createTitledBorder
58         (BorderFactory.createEtchedBorder(), ""));
59         painelConfiguracaoHosts.setLayout(new GridLayout(5, 2));
60
61         conf = conf2;
62
63         painel = new painelIpPorta[conf.getNumNo()];
64
65         for (i=1; i<=conf.getNumNo()-1; i++) {
66             painel[i] = new painelIpPorta();
67             painelConfiguracaoHosts.add(painel[i]);
68         }
69
70
71         if (conf.getNumNo() < 10) {
72             setSize(680, ((conf.getNumNo()) * 40));
73         } else {
74             setSize(680, 400);
75         }
76
77         JPanel p2 = new JPanel();
78         JButton botaoOk = new JButton("Ok");
79         p2.add(botaoOk);
80
81         Container contentPane = getContentPane();
82         contentPane.add(painelConfiguracaoHosts, "Center");
83         contentPane.add(p2, "South");
84
85         botaoOk.addActionListener(new ActionListener() {
86             public void actionPerformed(ActionEvent evt) {
87                 ok = true;
88                 setVisible(false);
89             }
90         });
91     } // public Tela_Configuracoes()
92     public final void escreveArquivoIp() {
93         try {
94             PrintWriter out = new PrintWriter(new FileWriter(diretorio+"
IpHost"));
95             for (i=1; i<=9; i++) {
96                 if (!(painel[i].ip.getText().equals(""))){
97                     out.println(painel[i].ip.getText());
98                 } // if
99             } // for
100             out.close();
101         } catch (Exception ex) {
102             System.out.println("###Erro###_Na_çãGravao_do_Arquivo_IP");
103         }
104     } // public final void

```

```

105
106     public boolean showDialog(HostPanel[] painelNo, String diretorio){
107
108         this.diretorio=diretorio;
109
110         for (i=1; i<=conf.getNumNo()-1; i++){
111             painel[i].ip.setText(painelNo[i].endereco.getIp());
112             painel[i].porta.setText("9000");
113         }
114
115         ok = false;
116         show();
117
118         if (ok==true){
119             for (i=1; i<=conf.getNumNo()-1; i++){
120                 painelNo[i].endereco.setIp(painel[i].ip.getText());
121                 painelNo[i].setTitulo(painel[i].ip.getText());
122             }
123             //escreveArquivoIP();
124             try{
125                 PrintWriter out= new PrintWriter(new FileWriter(
126                 diretorio+"IpHost"));
127                 for (i=1;i<=conf.getNumNo()-1;i++){
128                     if (!(painel[i].ip.getText().equals(""))){
129                         out.println(painel[i].ip.getText());
130                     } // if
131                 }// for
132                 out.close();
133             }catch(Exception ex){
134                 System.out.println("###Erro###NaGravãodoArquivo
135                 IP");
136             }
137         }
138     }// class Tela_Configuracoes
139
140     class painelIpPorta extends JPanel{
141         JTextField ip = new JTextField(9);
142         JTextField porta = new JTextField(4);
143         public painelIpPorta(){
144             setBorder(BorderFactory.createTitledBorder
145             (BorderFactory.createEtchedBorder(),""));
146             add(new JLabel("Ip"));
147             add(ip);
148             add(new JLabel("Porta"));
149             add(porta);
150         }
151     }// class painelIpPorta

```

Classe Tela_Inicial

```

1  /*
2  * Created on 16/05/2005
3  *
4  * TODO To change the template for this generated file go to
5  * Window - Preferences - Java - Code Style - Code Templates
6  */
7  package examples.agente;
8
9  /**
10 * @author tiago
11 *
12 * TODO To change the template for this generated type comment go to
13 * Window - Preferences - Java - Code Style - Code Templates
14 */
15
16 import java.awt.*;
17 import java.awt.event.*;
18 import javax.swing.*;
19 import javax.swing.border.*;
20 import java.lang.*;
21
22 public class Tela_Inicial extends JDialog{
23
24     String directorio = "/home/tiago/facu/projeto/aglets/public/examples
25 /agente/";
26     Tela_Jota telaJota;
27     JotaMaster master;
28     Configuracoes conf = new Configuracoes("");
29     Box b = Box.createVerticalBox();
30     JTextField numNo = new JTextField(5);
31     JTextField dir = new JTextField(15);
32     JTextField ip = new JTextField(15);
33     JTextField iplocal = new JTextField(15);
34
35     public Tela_Inicial(JotaMaster master2){
36         this.setTitle("çõConfiguraes_Iniciais");
37
38         master = master2;
39         //telaJota = new Tela_Jota(master);
40         JPanel p1 = new JPanel();
41         JPanel p2 = new JPanel();
42         JPanel p4 = new JPanel();
43         JPanel p5 = new JPanel();
44         p1.add(new JLabel("Quantos_õns_ãh_na_rede?"));
45
46         p1.add(numNo);
47         b.add(p1);
48         p2.add(new JLabel("óDiretorio_de_çãconfigurao:"));
49
50         p2.add(dir);
51         b.add(Box.createGlue());

```

```

51         b.add(Box.createGlue());
52         b.add(p2);
53         p4.add(new JLabel("Ip_inicial_da_rede:"));
54
55         p4.add(ip);
56         b.add(Box.createGlue());
57         b.add(Box.createGlue());
58         b.add(p4);
59         p5.add(new JLabel("Ip_local:"));
60
61         p5.add(iplocal);
62         b.add(Box.createGlue());
63         b.add(Box.createGlue());
64         b.add(p5);
65         getContentPane().add(b,"North");
66         JPanel p3 = new JPanel();
67         JButton ok = new JButton("Ok");
68         p3.add(ok);
69         getContentPane().add(p3,"South");
70
71         ok.addActionListener(new ActionListener(){
72             public void actionPerformed(ActionEvent evt){
73                 setVisible(false);
74                 conf.setDiretorio(dir.getText());
75                 conf.setIplocal(iplocal.getText());
76                 conf.setIprede(ip.getText());
77                 conf.setNumNo((new Integer(numNo.
78                     getText()).intValue()));
79                 telaJota = new Tela_Jota(master,conf);
80                 telaJota.show();
81             }
82         });
83         setSize(480,200);
84     }
85 }
86 }
87 }

```

Classe Tela_Jota

```

1 package examples.agente;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6 import javax.swing.border.*;
7 import java.lang.*;
8 import java.io.*;
9
10 // imports do Tela_Jota

```

```

11 import com.ibm.aglet.util.*;
12
13 public class Tela_Jota extends JFrame implements ActionListener {
14
15     private JButton botaoInformacao = new JButton("Obter_
16     çõInformaes");
17
18     private JButton botaoHost = new JButton("Host_Ativos");
19
20     private JButton botaoExecutar = new JButton("Executar");
21
22     private JRadioButton aplicacao_1 = new JRadioButton("áClculo_de
23     _Gauss");
24
25     private JRadioButton aplicacao_2 = new JRadioButton("áClculo_do
26     _PI");
27
28     private JRadioButton aplicacao_3 = new JRadioButton(
29         "çãMultiplicao_de_Matriz");
30
31     private JRadioButton outraAplicacao = new JRadioButton("Outra")
32     ;
33
34     JTextField textOutraAplicacao = new JTextField(8);
35
36     HostPanel[] painelLaico;
37
38     Configuracoes configuracoes = new Configuracoes(
39         "/home/tiago/facu/projeto/aglets/public/
40     examples/agente/");
41
42     JotaMaster master = null;
43
44     JotaSlave slave = null;
45
46     boolean jaClicado = false;
47
48     private Tela_Orientador mario;
49
50     private Tela_Configuracoes telaConfiguracoes;
51
52     private Tela_Opcoes telaOpcoes;
53
54     private JMenu menuAjuda = new JMenu("Ajuda");
55
56     private JMenuItem itemSobre = new JMenuItem("Sobre_os_Autores
57     ...");
58
59     private JMenuItem itemHelp = new JMenuItem("Ajuda_do_Programa
60     ...");
61
62     private JMenuItem itemOrientador = new JMenuItem("Sobre_o_
63     Orientador...");
64
65     private JMenu menuConfiguracao = new JMenu("Configurar");

```

```

58
59     private JMenuItem itemInformacoes = new JMenuItem("çõInformaes
60     ...");
61     private JMenuItem itemHosts = new JMenuItem("Hosts...");
62
63     // ávariveis no Tela_Jota
64
65     //////////////////////////////////////
66
67     private int i;
68
69     public int numeroMaquinas = 0;
70
71     private String nomeArquivo = "";
72
73     private String bufferAtp = "";
74
75     private JComboBox comboNumeroMaquinas = new JComboBox();
76
77     private JCheckBox sugestaoBox = new JCheckBox("ãSugesto_de_
78     áMquinas");
79
80     private JTextArea text = new JTextArea();
81
82     public Tela_Jota(JotaSlave slave) {
83         super("Jota_Slave");
84         this.slave = slave;
85
86         Container contentPane = getContentPane();
87         contentPane.add(text, "Center");
88
89         text.setEditable(false);
90         setSize(400, 200);
91     }
92
93     public Tela_Jota(JotaMaster master) {
94         super("Jota_Master");
95         this.master = master;
96
97         setTitle("Agente_Jota");
98         setSize(800, 600);
99         addWindowListener(new WindowAdapter() {
100             public void windowClosing(WindowEvent e) {
101                 //System.exit(0);
102                 setVisible(false);
103             }
104         }); //addWindowListener
105
106         desenhaTela();
107     } // public Teste_de_Tela2(JotaMaster master)
108
109     public Tela_Jota(JotaMaster master, Configuracoes conf) {
110         super("Jota_Master");

```

```

110         this.master = master;
111
112         configuracoes = conf;
113         painelLaico = new HostPanel[conf.getNumNo()];
114
115         setTitle("Agente_Jota");
116         setSize(800, 600);
117         addWindowListener(new WindowAdapter() {
118             public void windowClosing(WindowEvent e) {
119                 //System.exit(0);
120                 setVisible(false);
121             }
122         }); //addWindowListener
123
124         desenhaTela();
125     }
126
127     private void desenhaTela() {
128
129         //criando o painel com as Maquinas do Laico
130         painelLaico[0] = new HostPanel(configuracoes.getIplocal
131     ());
132         for(int x=1; x<=configuracoes.getNumNo()-1;x++){
133             painelLaico[x] = new HostPanel("192.168.1."+
134     new Integer(x).toString());
135         }
136         /*painelLaico[0] = new HostPanel("192.168.1.2");
137         painelLaico[1] = new HostPanel("192.168.1.3");
138         painelLaico[2] = new HostPanel("192.168.1.4");
139         painelLaico[3] = new HostPanel("192.168.1.5");
140         painelLaico[4] = new HostPanel("192.168.1.6");
141         painelLaico[5] = new HostPanel("192.168.1.7");
142         painelLaico[6] = new HostPanel("192.168.1.8");
143         painelLaico[7] = new HostPanel("192.168.1.9");
144         painelLaico[8] = new HostPanel("192.168.1.10");
145         painelLaico[9] = new HostPanel("192.168.1.11");
146         */
147         /*
148         * Abre o Arquivo de Configuracoes IpHost e pelos Ips e
149         * números de Porta
150         * á gravados, define-se as máquinas que o agente JOTA
151         * á utilizar.
152         */
153         try {
154             String s;
155             i = 1;
156             BufferedReader in = new BufferedReader(new
157     FileReader(configuracoes
158         .getDiretorio()
159         + "IpHost"));
160             while ((s = in.readLine()) != null) {
161                 painelLaico[i].endereco.setIp(s);

```



```

158                 painelLaico[i].setTitulo(s);
159                 i++;
160             }
161         } catch (Exception ex) {
162             System.out.println("###_Erro_Na_Abretura_do_
Arquivo_IpHost_###");
163         }
164
165         /*
166         a esquerda da
167         * PaineL com os Botoes e o paineL com os RadioButtons
168         * Tela_Jota (West).
169         */
170         JPanel painelBotoes = new JPanel();
171         painelBotoes.setBorder(BorderFactory.createTitledBorder
(BorderFactory
172             .createEtchedBorder(), ""));
173         painelBotoes.setLayout(new BorderLayout(painelBotoes,
BoxLayout.Y_AXIS));
174
175         painelBotoes.add(botaoHost);
176         painelBotoes.add(botaoInformacao);
177         /*
178         * Esses dois ãsero acrescentados no final.
179         * painelBotoes.add(painelRadio); painelBotoes.add(
botaoExecutar);
180         */
181
182         /*
183         éatravs de Ping
184         * Botoes que o painelBotoes áter: botaoHost; verifica
quais ámquin
185         * JotaSlave, caso ão tenha sido criado, ou o
botaoHost tenha sido
186         * clicado anteriormente, e Busca as e Despacha eles
para seus
187         * respectivos destinos. Caso estes agentes áj existam,
ou seja áj
188         * tenham sido despachados, é feita somente uma
çãsolicitao para ele
189         * mandar as çõinformaes dinamicas da ámquina onde eles
ãesto.
190         * botaoExecutar; verifica qual çãaplicao foi
selecionada, verifica se
191         * pelo menos duas ámquin
192         * de configuracao com os nomes de hosts dessas
ámquin
193         * chama o
194         * mpirun com seus parametros adequados.
195         */
196         botaoInformacao.setEnabled(false);

```

```

197         botaoInformacao.addActionListener(this);
198         botaoInformacao.setActionCommand("botaoInformacao");
199
200         botaoExecutar.setEnabled(false);
201         botaoExecutar.addActionListener(this);
202         botaoExecutar.setActionCommand("botaoExecutar");
203
204         botaoHost.addActionListener(new ActionListener() {
205             public void actionPerformed(ActionEvent ae) {
206                 bufferAtp = "";
207                 for (i = 1; i <= configuracoes.getNumNo
208                     (-1; i++) {
209                     painelLaico[i].limpaPainel();
210                     painelLaico[i].revalidate();
211                     painelLaico[i]
212                         .setAtivo(
213                             master
214                                 .verificarHostAtivo(painelLaico[i].endereco
215                                     .getIp()));
216                     if (painelLaico[i].estaAtivo ==
217                         true)
218                             bufferAtp = painelLaico
219                                 [i].endereco.getAtp() + "|"
220                                 +
221                                 bufferAtp;
222                     }
223                     System.out.println(bufferAtp);
224                     master.separaAtp(bufferAtp);
225                     botaoInformacao.setEnabled(true);
226                     botaoExecutar.setEnabled(false);
227                     jaClicado = false;
228                 }
229             });
230
231             /*
232             * Painel com os JRadioButtons que o painelBotoes áter.
233             */
234             JPanel painelRadio = new JPanel();
235             painelRadio.setBorder(BorderFactory.createTitledBorder(
236                 BorderFactory
237                     .createEtchedBorder(), "çõAplicaes:");
238             painelRadio.setLayout(new BorderLayout(painelRadio,
239                 BorderLayout.Y_AXIS));
240
241             ButtonGroup grupo = new ButtonGroup();
242
243             aplicacao_1.addActionListener(this);
244             aplicacao_1.setActionCommand("aplicacao_1");
245
246             aplicacao_2.addActionListener(this);

```

```

242     aplicacao_2.setActionCommand(" aplicacao_2");
243
244     aplicacao_3.addActionListener(this);
245     aplicacao_3.setActionCommand(" aplicacao_3");
246
247     outraAplicacao.addActionListener(this);
248     outraAplicacao.setActionCommand(" outraAplicacao");
249
250     painelRadio.add( aplicacao_1);
251     painelRadio.add( aplicacao_2);
252     painelRadio.add( aplicacao_3);
253     painelRadio.add( outraAplicacao);
254     Dimension tamanho = new Dimension(200, 135);
255     painelRadio.add( textOutraAplicacao);
256
257     painelRadio.setMaximumSize(tamanho);
258
259     grupo.add( aplicacao_1);
260     grupo.add( aplicacao_2);
261     grupo.add( aplicacao_3);
262     grupo.add( outraAplicacao);
263
264     aplicacao_1.setSelected(0 == 0);
265     aplicacao_2.setSelected(1 == 0);
266     aplicacao_3.setSelected(2 == 0);
267     outraAplicacao.setSelected(3 == 0);
268
269     painelBotoes.add(painelRadio);
270     painelBotoes.add(botaoExecutar);
271
272     /**
273     * //Painel com o Gif //JPanel painelGif = new JPanel()
; ImageIcon
274     * imagem = new
275     * ImageIcon( configuracoes.getDiretorio()+" gifs/
smile_pisca.gif");
276     * JLabel imagemLabel = new JLabel(""); imagemLabel.
setOpaque(true);
277     * imagemLabel.setIcon(imagem); painelBotoes.add(new
278     * JLabel("\n")); painelBotoes.add(new JLabel("\n"));
279     * painelBotoes.add(new JLabel("\n")); painelBotoes.add(
new
280     * JLabel("\n")); painelBotoes.add(new
281     * JLabel("\n")); painelBotoes.add(new JLabel("\n"));
282     * painelBotoes.add(new JLabel("\n")); painelBotoes.add(
new
283     * JLabel("\n")); //painelGif.add(imagemLabel);
284     * //painelBotoes.add(painelGif); painelBotoes.add(
imagemLabel);
285     *
286     */
287
288     /*

```

```

289         * Paineis com os HostPanel representando as máquinas da
Rede que são
290         * monitoradas
291         */
292
293 JPanel painelHosts = new JPanel();
294 painelHosts.setBorder(BorderFactory.createTitledBorder(
BorderFactory
295         .createEtchedBorder(), ""));
296 if (configuracoes.getNumNo() <= 10) {
297     painelHosts.setLayout(new GridLayout(2, 5));
298 } else if (configuracoes.getNumNo() <= 15) {
299     painelHosts.setLayout(new GridLayout(3, 5));
300 } else {
301     painelHosts.setLayout(new GridLayout(4, 5));
302 }
303
304 /*
305  * Adiciona-se os 8 painelHosts criados
306  */
307 for (i = 1; i <= configuracoes.getNumNo() - 1; i++)
308     painelHosts.add(painelLaico[i]);
309
310 Container contentPane = getContentPane();
311 contentPane.add(painelBotoes, "West");
312 contentPane.add(painelHosts, "Center");
313
314 /*
315  * Criação dos Menus - menuAjuda - opção Opes
316  */
317
318 JMenuBar mbar = new JMenuBar();
319 mbar.add(menuConfiguracao);
320 mbar.add(menuAjuda);
321 setJMenuBar(mbar);
322 show();
323
324 menuConfiguracao.add(itemInformacoes);
325 menuConfiguracao.add(itemHosts);
326
327 itemHosts.setActionCommand("itemHosts");
328 itemHosts.addActionListener(this);
329
330 menuAjuda.add(itemHelp);
331 menuAjuda.addSeparator();
332 menuAjuda.add(itemSobre);
333 menuAjuda.add(itemOrientador);
334
335 itemOrientador.addActionListener(this);
336 itemOrientador.setActionCommand("itemOrientador");
337
338 itemInformacoes.addActionListener(this);
339 itemInformacoes.setActionCommand("itemInformacoes");

```

```

340     }
341 }
342
343     public void actionPerformed(ActionEvent evt) {
344
345         Object source = evt.getSource();
346
347         if (source == itemOrientador) {
348             if (mario == null) { // primeira vez
349                 mario = new Tela_Orientador(this);
350                 mario.show();
351             } else
352                 mario.show();
353         }
354
355         else if (evt.getActionCommand().equals("itemHosts")) {
356             if (telaConfiguracoes == null) { // primeira
357                 vez
358                     telaConfiguracoes = new
359                     Tela_Configuracoes(this, configuracoes);
360                     telaConfiguracoes.showDialog(
361                     painelLaico, configuracoes
362                                     .getDiretorio());
363             } else
364                 telaConfiguracoes.showDialog(
365                 painelLaico, configuracoes
366                                     .getDiretorio());
367         }
368
369         else if (evt.getActionCommand().equals("itemInformacoes
370 ")) {
371             if (telaOpcoes == null) { // primeira vez
372                 telaOpcoes = new Tela_Opcoes(this);
373                 telaOpcoes.showDialog(configuracoes);
374             }
375             else
376                 telaOpcoes.showDialog(configuracoes);
377         }
378
379         else if (source == botaoInformacao) {
380             if (jaClicado == false) {
381                 master.dispatchSlave(bufferAtp);
382                 //master.sendPainel();
383                 master.sendRequest();
384                 master.sendText("sendRequest!!!!");
385                 jaClicado = true;
386             }
387             //master.sendRequest();
388             master.sendRequestMemLivre();
389             master.sendRequestCPULivreProcessos();
390             master.sendText("sendRequest*");

```



```

430                                     System.out.println(" O numero de
de_maq_é:" + numeroMaquinas);
431                                     }
432                                     }
433                                     out.close();
434     } catch (Exception ex) {
435         System.out.println("erro!!");
436     }
437     System.out.println(" Saiu do comando1");
438 }
439
440     public final void executaMpi() {
441         Process p = null;
442         String Saida = "";
443         String comandoMpi = "/mpich-1.2.2.3/bin/mpirun -np " +
numeroMaquinas
444                                     + " " + nomeArquivo;
445         System.out.println("np=" + numeroMaquinas);
446         System.out.println("arquivo=" + nomeArquivo);
447         System.out.println(comandoMpi);
448         if (numeroMaquinas != 0) {
449             try {
450                 p = Runtime.getRuntime().exec(
comandoMpi);
451                 p.waitFor();
452
453                 InputStream fora = p.getInputStream();
454                 try {
455                     byte b[] = new byte[255];
456                     fora.read(b);
457                     int i = 0;
458                     while (i < 255) {
459                         Saida = Saida + java.
lang.String.valueOf((char) b[i]);
460                                     i++;
461                                     }//while
462                                     System.out.println("A saída é:"
" + Saida);
463                                     } catch (Exception e) {
464                                         System.out.println("erro na
string");
465                                         }//catch
466                                         //p.destroy();
467
468                                     } catch (Exception ex) {
469                                         System.out.println("Nao foi possível
executar o MPI");
470                                     }
471                                     System.out.println(" Saiu do comando2");
472     } else {
473         JOptionPane.showConfirmDialog(null,
474                                     "O usuário deve selecionar pelo
menos um host", "Erro",

```

```
475                                     JOptionPane.DEFAULT.OPTION,  
476                                     }  
477                                     }  
478 }  
479  
480 class Configuracoes {  
481     private String diretorio = "/home/tiago/aglets/conf/";  
482     private String iplocal = "";  
483     private String iprede ="";  
484     private int numNo;  
485  
486     public Configuracoes(String dir) {  
487         diretorio = dir;  
488     }  
489  
490     public String getIplocal(){  
491         return iplocal;  
492     }  
493  
494     public String getIprede(){  
495         return iprede;  
496     }  
497  
498     public int getNumNo(){  
499         return numNo;  
500     }  
501  
502     public void setIplocal(String ip){  
503         iplocal = ip;  
504     }  
505  
506     public void setIprede(String ip){  
507         iprede = ip;  
508     }  
509  
510     public void setNumNo(int no){  
511         numNo = no;  
512     }  
513  
514     public void setDiretorio(String dir) {  
515         diretorio = dir;  
516     }  
517  
518     public String getDiretorio() {  
519         return diretorio;  
520     }  
521 }
```


Classe Tela_Opcoes

```

1 package examples.agente;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6 import javax.swing.border.*;
7
8
9 import java.lang.*;
10 import java.io.*;
11
12 public class Tela_Opcoes extends JDialog{
13
14     int i = 0;
15     boolean ok;
16     JButton botaoOk;
17     JPanel painelOpcoesHosts = new JPanel();
18     JCheckBox modeloCPU = new JCheckBox("Modelo da CPU", true);
19     JCheckBox versaoSo = new JCheckBox("ãVerso do Sistema Operacional",
20     true);
21     JCheckBox velocidadeCPU = new JCheckBox("Velocidade da CPU", true);
22     JCheckBox memLivre = new JCheckBox("óMemria Livre", true);
23     JCheckBox memTotal = new JCheckBox("óMemria Total", true);
24     JCheckBox arqProcessador = new JCheckBox("Arquitetura do
25     Processador", true);
26     JCheckBox so = new JCheckBox("Sistema Operacional", true);
27     JCheckBox host = new JCheckBox("Nome do Host", true);
28     JCheckBox processosCPU = new JCheckBox("CPU Livre / No. de
29     Processos", true);
30     JTextField textFieldDiretorio = new JTextField(20);
31
32     public Tela_Opcoes(JFrame parent){
33
34         super(parent, "çõOpes de çõInformaes dos Hosts a serem Buscadas",
35         true);
36
37         painelOpcoesHosts.setBorder(BorderFactory.createTitledBorder
38         (BorderFactory.createEtchedBorder(), "Selecione as çõOpes
39         Desejadas"));
40         painelOpcoesHosts.setLayout(new GridLayout(11,1));
41
42         setSize(350,350);
43         host.setEnabled(false);
44         painelOpcoesHosts.add(host);
45         so.setEnabled(false);
46         painelOpcoesHosts.add(so);
47         versaoSo.setEnabled(false);
48         painelOpcoesHosts.add(versaoSo);
49         arqProcessador.setEnabled(false);
50         painelOpcoesHosts.add(arqProcessador);

```

```

47     memTotal.setEnabled(false);
48     painelOpcoesHosts.add(memTotal);
49     memLivre.setEnabled(true);
50     painelOpcoesHosts.add(memLivre);
51     modeloCPU.setEnabled(false);
52     painelOpcoesHosts.add(modeloCPU);
53     velocidadeCPU.setEnabled(false);
54     painelOpcoesHosts.add(velocidadeCPU);
55     processossCPU.setEnabled(true);
56     painelOpcoesHosts.add(processossCPU);
57     painelOpcoesHosts.add(new JLabel("Diretorio onde ãesto as
Classes do Agente Jota"));
58     painelOpcoesHosts.add(textFieldDiretorio);
59
60     JPanel p2 = new JPanel();
61     JButton botaoOk = new JButton("Ok");
62     p2.add(botaoOk);
63
64     Container contentPane = getContentPane();
65     contentPane.add(painelOpcoesHosts, "Center");
66     contentPane.add(p2, "South");
67
68     botaoOk.addActionListener(new ActionListener() {
69         public void actionPerformed(ActionEvent evt) {
70             ok = true;
71             setVisible(false);
72         }
73     });
74 } // public Tela_Configuracoes()
75
76
77 public boolean showDialog(Configuracoes configuracoes) {
78
79     textFieldDiretorio.setText(configuracoes.getDiretorio());
80
81     ok = false;
82     show();
83
84
85     if (ok == true) {
86         configuracoes.setDiretorio(textFieldDiretorio.getText());
87     }
88     return ok;
89 }
90 } // class Tela_Configuracoes

```

Classe Tela_Orientador

```

1 package examples.agente;
2
3 import java.awt.*;

```

```

4 import java.awt.event.*;
5 import javax.swing.*;
6 import javax.swing.border.*;
7 import java.lang.*;
8
9 public class Tela_Orientador extends JDialog{
10
11     String diretorio = "/home/tiago/facu/projeto/aglets/public/examples
12     /agente/";
13
14     public Tela_Orientador(JFrame parent){
15         super(parent, "Nosso_Querido_Orientador", true);
16
17         Box b = Box.createVerticalBox();
18
19         JPanel p1 = new JPanel();
20
21         //ImageIcon imagem = new ImageIcon(diretorio+" gifs/
22     mario.gif");
23         JLabel imagemLabel = new JLabel("");
24         imagemLabel.setOpaque(true);
25         //imagemLabel.setIcon(imagem);
26         p1.add(imagemLabel);
27
28         getContentPane().add(p1, "West");
29
30         b.add(Box.createGlue());
31         b.add(Box.createGlue());
32         b.add(new JLabel("M.A.R. Dantas"));
33         b.add(new JLabel("Associate Professor"));
34         b.add(new JLabel("Ph.D. em Cincia da Computao"));
35         b.add(new JLabel("University of Southampton (
36     UK, 1997)"));
37         b.add(new JLabel("Analista de Sistemas - PUC/RJ (Brasil
38     , 1986)"));
39         b.add(new JLabel("Engenheiro Eletricista"));
40         b.add(new JLabel("Universidade Federal de
41     Santa Catarina (Brasil, 1984)"));
42         b.add(new JLabel("www.inf.ufsc.br/~mario"));
43
44         getContentPane().add(b, "Center");
45
46         JPanel p2 = new JPanel();
47         JButton ok = new JButton("Ok");
48         p2.add(ok);
49         getContentPane().add(p2, "South");
50
51         ok.addActionListener(new ActionListener(){
52             public void actionPerformed(ActionEvent evt){
53                 setVisible(false);
54             }
55         });

```

```
52         });  
53  
54         setSize(480,200);  
55  
56     }  
57  
58 }
```