

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

MONITORAÇÃO ORIENTADA A PROCESSO

EDUARDO BITENCOURT MILANESE

Trabalho de Conclusão de Curso apresentado à
Universidade Federal de Santa Catarina como um
dos pré-requisitos para obtenção do grau de
bacharel em Ciências da Computação.

Florianópolis - SC
2006

EDUARDO BITENCOURT MILANESE

MONITORAÇÃO ORIENTADA A PROCESSO

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de
Bacharel em Ciências da Computação

Orientador: Prof. Dr. Mário A. R. Dantas

Prof. Dr. Júlio da Silva Dias

Prof. Dr. Vitório Bruno Mazzola

*“O mais importante da vida não é a situação em que
estamos, mas a direção para a qual nos movemos”*

Oliver Wendell Holmes

AGRADECIMENTOS

Agradeço a meus pais, pelo apoio e carinho que sempre me dispensaram.

Ao meu irmão, pela ajuda e compreensão nos momentos mais difíceis.

A todos os meus familiares, pelos conselhos e por sempre acreditarem em mim.

Em especial, ao Professor Mário Dantas, meu orientador, pela grande contribuição e disponibilidade, e aos ilustres membros da banca pela participação.

A Fernando Laudares Camargo, membro da equipe de desenvolvimento do OSCAR e ex-aluno da UFSC, pelas informações fornecidas.

Aos meus amigos, por estarem presentes em todos os momentos.

RESUMO

Os sistemas de agregado são usualmente complexos em termos de quantidade de processos e número de processadores que compõem o ambiente. Conforme o número de nodos de um cluster aumenta, cresce também a dificuldade de manutenção nos sistemas individuais e dos processos que são executados no ambiente.

Sistemas de controle, como monitores de processos, representam uma abordagem interessante para implementar confiabilidade na configuração e auxiliar a resolução de possíveis problemas.

Neste trabalho implementamos uma solução de monitoração de configurações de agregado baseado em trabalhos do grupo LAPESD (Laboratório de Pesquisa em Sistemas Distribuídos). Desenvolvemos uma ferramenta capaz de monitorar os processos que fornecem serviços essenciais, além dos nodos do cluster, e avisar o administrador da configuração em caso de alguma falha. A ferramenta consiste em dois aplicativos: um aplicativo cliente que funciona em um dispositivo móvel e um aplicativo servidor responsável pela monitoração do agregado. A comunicação entre os sistemas é realizada através de uma rede sem fio.

O objetivo foi alcançado com sucesso, uma vez que o sistema implementado permite a monitoração automatizada da configuração de agregado e se comportou conforme o esperado nos testes que foram realizados.

Palavras Chave: Monitoração, cluster, alta disponibilidade, OSCAR, Java, SuperWaba, wireless.

SUMÁRIO

1. INTRODUÇÃO.....	7
2. PROPOSTA DO TRABALHO	9
2.1 OBJETIVOS GERAIS	9
2.2 OBJETIVOS ESPECÍFICOS	9
2.3 MOTIVAÇÃO.....	9
3. REDES DE COMPUTADORES	10
3.1 MODELO OSI.....	11
3.2 TIPOS DE REDE	11
3.2.1 REDES LOCAIS	11
3.2.2 REDE METROPOLITANA.....	12
3.2.3 REDE GEOGRAFICAMENTE DISTRIBUÍDA.....	13
3.3 REDES SEM FIO.....	13
3.3.1 PADRÕES PARA REDES SEM FIO.....	14
4. COMPUTAÇÃO EM CLUSTER	16
4.1 CLASSIFICAÇÃO.....	16
4.2 ALTA DISPONIBILIDADE.....	19
4.3 OSCAR.....	21
4.4 HA-OSCAR.....	22
5. AMBIENTE DE DESENVOLVIMENTO.....	24
5.1 JAVA	24
5.2 ECLIPSE	26
5.3 POSE	26
5.4 SUPERWABA	28
5.5 XML	32
6. PROJETO.....	33
6.1 APLICATIVO CLIENTE.....	34
6.2 APLICATIVO SERVIDOR	44
6.2.1 ARQUIVO DE CONFIGURAÇÃO.....	45
6.3 MÉTODO DE MONITORAÇÃO.....	46
6.4 ESTUDO DE CASO	48
6.4.1 ESTUDO DE CASO 1: PROCESSOS EM ESTADO ZOMBIE E PROCESSO INATIVO	50
6.4.2 ESTUDO DE CASO 2: PROCESSO PARADO E NODO INDISPONÍVEL.....	51
6.4.3 ESTUDO DE CASO 3: PROBLEMAS DE COMUNICAÇÃO ENTRE SERVIDOR E CLIENTE	52
7. CONCLUSÕES E TRABALHOS FUTUROS	54
REFERÊNCIAS BIBLIOGRÁFICAS	55
ANEXO I – CÓDIGO FONTE	57
I.I - APLICATIVO PALM.....	57
I.II - APLICATIVO SERVIDOR.....	85
ANEXO II – ARTIGO	110

LISTA DE FIGURAS

Fig. 3.1 – Modelo OSI.....	12
Fig. 3.2 - Exemplo de rede seguindo padrão IEE 802.11b [MAR02]......	14
Fig. 4.1 – Esquema de um cluster OSCAR [BAG04]......	21
Fig. 4.2 – Janela de instalação do OSCAR [OSC05]......	22
Fig. 4.3 – Esquema de um cluster HA-OSCAR [BAG04]......	23
Fig. 5.1 – Ambiente de desenvolvimento Eclipse.	26
Fig. 5.2 – POSE.....	27
Fig. 5.3 – Simulador PALM.	28
Fig. 5.4 – Esquema Compilação SuperWaba [SUC05]......	29
Fig. 5.5 – Assistente para geração de arquivos <i>pdb</i> e <i>prc</i> do SuperWaba IDE.	31
Fig. 5.6 – Simulador SuperWaba do SuperWaba IDE.	32
Fig. 6.1 – Tela inicial do POSE com os ícones do SuperWaba e MonitorPalm.....	35
Fig. 6.2 – Formulário Principal da Aplicação.	36
Fig. 6.3 – Mensagem de erro ao ler a lista de processos.	36
Fig. 6.4 – Barra de Menu, item “Arquivo”......	37
Fig. 6.5 – Barra de Menu, item “Gerenciar”.	37
Fig. 6.6 – Formulário de Configuração de Processo.	38
Fig. 6.7 – Formulário simulador do Teclado.	38
Fig. 6.8 – Formulário de configuração dos nodos monitorados.	39
Fig. 6.9 – Erro no processo pico.....	40
Fig. 6.10 – Erro no nodo número 4.	40
Fig. 6.11 – Erro na conexão com o servidor.....	40
Fig. 6.12 – Formulário de configuração de parâmetros do servidor.....	41
Fig. 6.13 – Formato da mensagem.	43
Fig. 6.14 – XML de configuração do servidor.	45
Fig. 6.15 - Resultado da execução do comando <i>cexec</i>	48
Fig. 6.16 – Trecho classe GeradorPS	50
Fig. 6.17 – Mensagens de erro nos estados dos processos “java” e “mysqld”.....	51
Fig. 6.18 – Mensagem de erro no processo “pico”.....	51
Fig. 6.19 – Mensagem de erro no processo “mysqld”.....	52
Fig. 6.20 – Mensagem de erro no nodo 3 e no processo “mysqld”.....	52
Fig. 6.21 – Cluster indisponível, erro ao ler lista de processos.	53
Fig. 6.22 – Cluster indisponível, erro ler mensagens do servidor.	53
Fig. 6.23 – Conexão com servidor restabelecida.....	53

1. INTRODUÇÃO

A computação está cada vez mais presente em diversas áreas do conhecimento humano. A biologia, a estatística, a meteorologia, a física são apenas alguns exemplos. Junto com a expansão da computação e a complexidade dos sistemas computacionais, cresce cada dia mais a demanda por sistemas com alto desempenho. Em contrapartida, o ritmo de crescimento da capacidade de processamento dos computadores é cada vez menor, devido a proximidade da saturação da tecnologia atual para fabricação de microprocessadores.

Esse cenário ocasionou uma crescente popularização dos sistemas paralelos. Um sistema paralelo é formado através da união de sistemas menores, o que ocasiona uma multiplicação no desempenho. Entre os sistemas paralelos destacam-se os agregados computacionais ou clusters. Por serem formados por componentes de baixo custo e fácil aquisição no mercado, eles são uma solução eficiente e barata para sistemas que necessitam de alta performance. Além disso, atualmente, existe uma variedade de soluções para a construção desse tipo de sistema, algumas gratuitas e com código fonte aberto. Um cluster consiste basicamente de dois ou mais computadores, denominados nodos, conectados através de uma rede local e por um programa de imagem única que possibilita aos usuários visualizarem o cluster como uma só máquina.

Uma característica freqüentemente desejável em um ambiente paralelo é a alta disponibilidade. Sistemas de alta disponibilidade possuem mecanismos de recuperação e de tolerância a falhas, entre outros, visando manter-se disponíveis durante a maior fração de tempo possível.

Por sua vez, sistemas de agregado são mais complexos de instalar e manter. Conforme o número de nodos de um cluster aumenta, cresce também a dificuldade de manutenção nos sistemas individuais e dos processos que são executados no ambiente. Tendo em vista as dificuldades, vamos propor e implementar uma ferramenta que facilite a monitoração desse tipo de sistema, informando sobre as falhas ocorridas nos nodos e nas aplicações executadas no ambiente sem que o administrador precise realizar monitorações manuais do cluster a todo o momento.

Uma outra tecnologia em franca expansão são as redes sem fio (*wireless*). A cada dia cresce o número de usuários domésticos e corporativos que adotam essa tecnologia. Essa expansão se deve a características das redes sem fio como flexibilidade, fácil manutenção e

mobilidade. Da mesma forma, os dispositivos móveis, como o *Palm* e o *Pocket PC*, evoluem rapidamente permitindo a execução de aplicações cada vez mais robustas.

Como as aplicações e os serviços oferecidos por elas são o objetivo final dos sistemas paralelos, o sistema de monitoração desenvolvido é focado, principalmente, na monitoração dos processos críticos que são executados no cluster.

O sistema de monitoração proposto neste trabalho aborda todas essas tecnologias. Foram desenvolvidos dois aplicativos: um a ser instalado no ambiente do agregado e o outro em um *Palm*. A comunicação entre os sistemas é feita através de uma rede local *wireless*.

2. PROPOSTA DO TRABALHO

A proposta desse trabalho consiste no desenvolvimento de um sistema de monitoração de processos para um cluster OSCAR, tendo como objetivo facilitar a complexa tarefa de monitoração de um ambiente de cluster.

2.1 OBJETIVOS GERAIS

Pesquisar e compreender conceitos relacionados a redes sem fio, alto desempenho, cluster e alta disponibilidade. Realizar estudos sobre os pacotes de software OSCAR e HA-OSCAR e sobre desenvolvimento de aplicações para dispositivos móveis.

Desenvolver um sistema de monitoração para um cluster OSCAR.

2.2 OBJETIVOS ESPECÍFICOS

Desenvolver um sistema de monitoração em que o usuário seja capaz de configurar os processos e nodos do cluster que serão monitorados através de um aplicativo desenvolvido para um PDA. Falhas ocorridas nos processos ou nos nodos monitorados serão enviadas ao PDA (palm) através de uma rede wireless. O usuário visualizará informações no PDA sobre os processos que estão sendo executados no cluster, como memória, usuário que iniciou o processo, nome do processo, estado, taxa de ocupação da cpu entre outras.

2.3 MOTIVAÇÃO

Vários fatores despertam interesse no estudo de clusters e redes sem fio. A utilização cada vez maior dessas tecnologias por parte de organizações, instituições de ensino e de pesquisa é um deles. O crescimento na utilização dos ambientes de agregados computacionais ou clusters devido a demanda cada vez maior por sistemas de alto desempenho é outro fator. A possibilidade de trabalhar com uma aplicação para dispositivos móveis também pode ser citada. O principal fator motivador, no entanto, advém da complexidade de ambientes de cluster, das dificuldades encontradas na tarefa de monitoração desse tipo de ambiente e na possibilidade da criação de uma ferramenta capaz de auxiliar nessa tarefa.

3. REDES DE COMPUTADORES

Antes de falar sobre segurança ou mesmo sobre redes sem fio, é interessante comentar alguns conceitos sobre redes.

Rede de computadores é um conjunto de computadores autônomos interconectados que podem trocar informações. A forma de interconexão pode ser feita através de cabos de cobre, fibra óptica, microondas ou via satélite [TAN03].

As redes de computadores nasceram na década de 60, nos Estados Unidos, através de pesquisas militares. Sofreram grande impulso na década de 80 com a criação do computador pessoal, e se popularizaram a partir de 1990 com a criação da Internet.

Para que computadores possam trocar informações é imprescindível à existência de padrões de comunicação. Se não existissem padrões cada fabricante implementaria seus produtos como lhe fosse conveniente. Como resultado produtos de fabricantes diferentes não conseguiriam se comunicar.

Existem em todo o mundo algumas organizações responsáveis pela criação de padrões. As mais importantes.

- IEEE (Institute of Electrical and Electronic Engineers) foi criado em 1884 nos Estados Unidos. É a maior organização profissional do mundo, possuindo mais de 312.000 associados em 150 países. Possui grupos de pesquisa e desenvolvimento de padrões tanto na área de informática quanto na área de Engenharia Elétrica. É responsável pelo padrão IEE 802, que define o funcionamento de redes locais.
- A ISO (International Standards Organization) foi criada em 1946, sendo formada por organizações de padrões de diversos países. Ela é responsável pela criação de padrões internacionais e pela publicação dos padrões OSI.
- A ITU (International Telecommunication Union) foi criada em 1865 por representantes de governos europeus com o objetivo de padronizar as telecomunicações internacionais. A partir de 1947 tornou-se órgão das Nações Unidas. A ITU divide-se em três setores: o ITU-R responsável pela Radiocomunicação, o ITU-D setor de desenvolvimento e o ITU-T responsável pela padronização de Telecomunicações.
- ANSI (American National Standards Institute) é responsável pela regulamentação de padrões nos Estados Unidos. Foi criada em 1918, e é formada por comitês técnicos independentes denominados ASC's (Accredited Standards Committees).

3.1 MODELO OSI

O modelo OSI foi desenvolvido pela ISO. Ele é dividido em sete camadas e seu objetivo é padronizar a interconexão de sistemas abertos.

Camada Física – Define as características elétricas e mecânicas dos dispositivos, além dos procedimentos para criar e manter conexões. Essa é a camada de mais baixo nível e trata da transmissão de bits através do canal de comunicação.

Camada de Enlace de dados – Responsável pela detecção e correção de erros ocorridos na camada física.

Camada de Rede – É a responsável pela entrega dos pacotes desde a origem até o destino. Para isso, possui algoritmos de roteamento e controle de congestionamentos.

Camada de Transporte – Seu objetivo é fornecer um transporte de dados confiável, além de um serviço transferência transparente para a camada de sessão.

Camada de Sessão – Fornece serviços avançados sobre o transporte de dados como gerenciamento de token e sincronização.

Camada de Apresentação – Trata do significado dos dados transportados. É responsável pela codificação de dados no emissor e no destino, além de resolver problemas de sintaxe caso os sistemas possuam representações de bits distintas.

Camada de Aplicação – Presta serviços para processos do usuário como correio eletrônico ou softwares para transferência de arquivos.

3.2 TIPOS DE REDE

Existem diferentes tipos de rede, cada qual com características peculiares. Essas redes diferem principalmente quanto à abrangência, taxa de transmissão de dados e confiabilidade.

3.2.1 REDES LOCAIS

As Redes Locais, também denominadas LAN's (Local Area Network), são redes que geralmente interligam pequenas distâncias. Suas principais características são as altas taxas de transmissão - que podem variar entre 10 Mbps, 100 Mbps ou taxas de transmissão de milhares de

Mbps, e pequena taxa de erros. Nesse tipo de rede não ocorre o roteamento de dados, dada a velocidade e a confiabilidade dos canais de comunicação.

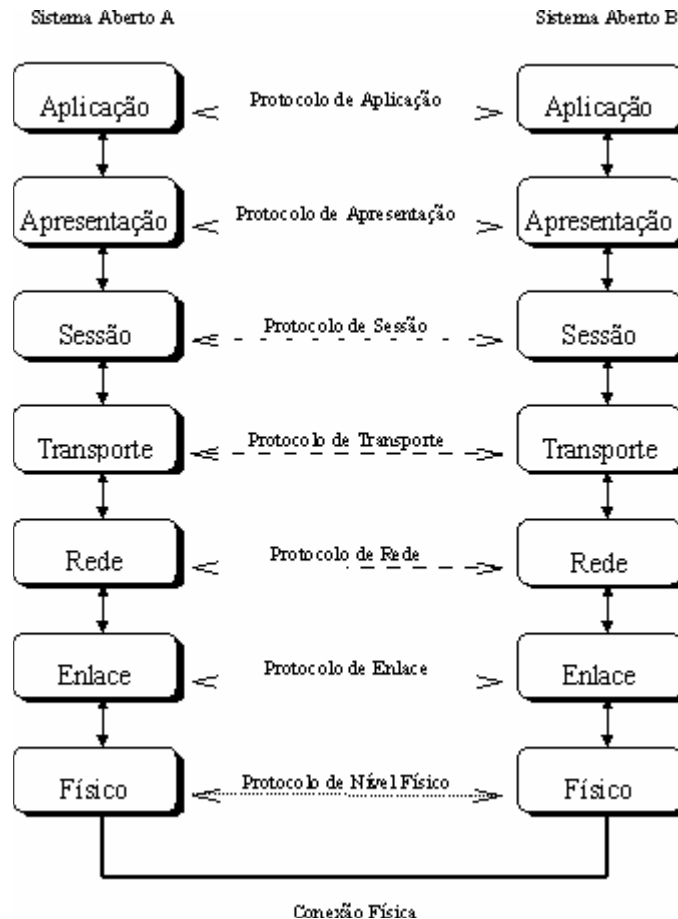


Fig. 3.1 – Modelo OSI.

Esse tipo de rede tem se tornado cada vez mais comum em instituições de ensino, empresas, escritórios e até com usuários comuns que possuem mais de um microcomputador em casa.

3.2.2 REDE METROPOLITANA

As Redes Metropolitanas ou MAN (Metropolitan Area Network) abrangem uma região maior, geralmente distâncias de até 10 Km. Apesar das grandes distâncias, podem possuir grandes taxas de transmissão, como por exemplo, as redes ATM (Asynchronous Transfer Mode) que realizam transmissões de até 622 Mbps.

3.2.3 REDE GEOGRAFICAMENTE DISTRIBUÍDA

As WAN's (Wide Area Network) são redes que abrangem uma vasta área, que pode ser de centenas ou até milhares de quilômetros. É caracterizada por taxas de erro maiores e menores taxas de transmissão se comparada com as redes locais. Devido a pouca confiabilidade dos canais de transmissão faz-se necessário o roteamento das informações.

3.3 REDES SEM FIO

As redes wireless, ou redes sem fio, realizam transmissão de dados através de ondas de rádio ou de infravermelho. Logo, dispensam o uso de cabos e permitem ao usuário mobilidade dentro da área de cobertura. Os padrões mais utilizados são o IEEE 802.11, Bluetooth e HomeRF. Sendo que o padrão IEE 802.11 foi desenvolvido para aplicações de redes locais, enquanto os padrões Bluetooth e HomeRF são utilizados em redes pessoais.

O crescente uso dessa tecnologia advém da sua praticidade e da possibilidade de implantação em locais onde redes cabeadas seriam inviáveis ou de difícil instalação, como áreas remotas ou prédios históricos.

Entretanto, existem diversos problemas com relação à segurança. As redes sem fio ainda são consideradas menos seguras do que as tradicionais redes cabeadas. Como o meio de transmissão é um canal aberto pessoas não autorizadas podem ter acesso aos sinais, exigindo, portanto, segurança redobrada. O custo cada vez mais acessível dos dispositivos também tem ajudado na popularização dessa tecnologia.

Assim com nas redes cabeadas podemos subdividir as redes sem fio.

- Redes Sem fio Locais ou WLAN: As redes WLAN (Wireless Local Area Network) são bastante difundidas. Foram padronizadas pelo IEE 802.11 em 1997.
- Redes Sem fio Metropolitanas ou WMAN (Wireless Metropolitan Area Network): Redes sem fio com área de maior abrangência, chegando a algumas dezenas de quilômetros.
- Redes Sem Fio Geograficamente Distribuídas ou WWAN: As redes WWAN (Wide Area Network) são a versão sem fio da WAN, compreendendo áreas do tamanho de países ou até continentes.

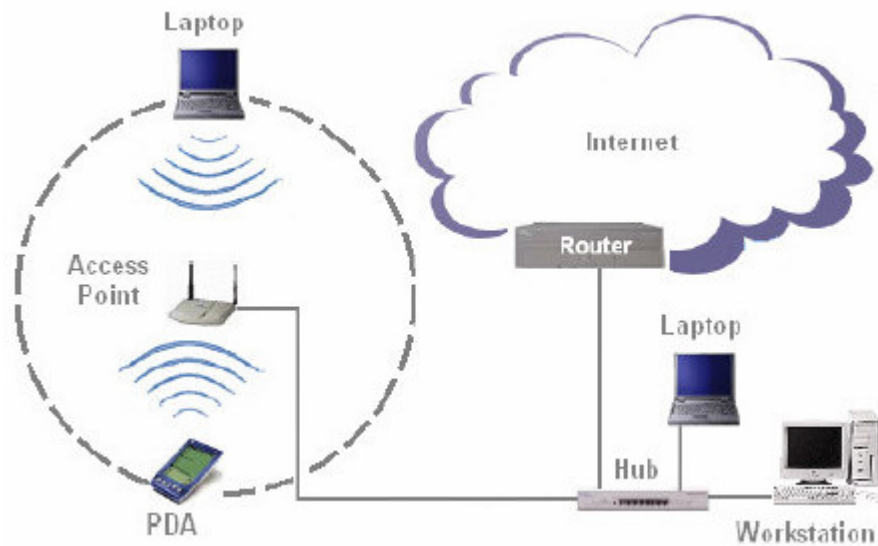


Fig. 3.2 - Exemplo de rede seguindo padrão IEEE 802.11b [MAR02].

- Redes WLL (Wireless Local Loop): Consistem em tecnologias sem fio fixas. Seu objetivo é substituir as redes de telefonia convencionais. Sua arquitetura é do tipo ponto-a-multiponto.
- Redes Pessoais ou WPAN (Wireless Personal Area Network): Possuem taxas de transmissão baixas e cobrem pequenas distâncias. São bastante utilizadas em equipamentos portáteis, em substituição aos cabos. Um exemplo desse tipo de rede é a tecnologia Bluetooth que permite a transmissão de dados até 1 Mbps em uma distância de 10 metros.

3.3.1 PADRÕES PARA REDES SEM FIO

Devido às diferenças em relação às redes cabeadas, em 1990 a IEEE criou um grupo de trabalho para desenvolver um protocolo para redes sem fio, o qual foi aprovado em 97 recebendo o nome IEEE 802.11.

O padrão IEEE 802.11 define um conjunto de protocolos para redes Ethernet sem fio, além de especificações físicas e de acesso ao meio. A segurança é realizada de duas formas: autenticação e criptografia.

O método de criptografia utilizado pelo padrão IEEE 802.11 é o WEP (Wired Equivalency Privacy).

A partir do padrão 802.11 surgiram outras extensões.

O padrão IEEE 802.11b possui taxa de transmissão de 11 Mbps e opera na frequência de 2,4 GHz. Sua área de cobertura pode chegar a até 450 metros em ambientes abertos. Permite o acesso a até três canais simultâneos.

O padrão 802.11g difere do 802.11b quanto à taxa de transmissão. No 802.11g pode-se transmitir a até 54 Mbps, utilizando a mesma frequência de 2,4 GHz.

Já o padrão IEEE 802.11a utiliza frequência de 5 GHz, com taxa de transmissão de 54 Mbps, permitindo o acesso de até 8 canais simultâneos. Outra vantagem é a menor possibilidade de interferências, visto que essa frequência não é utilizada por outras tecnologias. Todavia, devido a sua alta frequência, a área de cobertura fica restrita a no máximo 15 metros.

O 802.11e adiciona as funcionalidades da tão comentada qualidade de serviço (QoS – Quality of Service), como voz sobre IP (Internet Protocol) e acesso a Internet de alta velocidade.

Os padrões 802.11i e 802.1X têm como objetivo implementar melhorias na segurança. O 802.1X define a forma de autenticação dos usuários na rede, enquanto o 802.11i propõe a substituição WEP.

Além desses, existem outros padrões como o 802.11c, 802.11d, 802.11f e o 802.11h.

4. COMPUTAÇÃO EM CLUSTER

Em diversas áreas do conhecimento onde a computação é utilizada como a estatística, física, previsões meteorológicas, genética, economia e muitas outras, é necessário processar grandes volumes de dados, demandando a construção de sistemas de alto desempenho. Até o surgimento dos clusters (agregados) a saída era a utilização de supercomputadores específicos denominados Mainframes. Os Mainframes são sistemas especializados com alto poder de desempenho, entretanto sua aquisição e manutenção são muito caras. Por outro lado, o cluster é um sistema relativamente barato em relação aos mainframes, uma vez que pode ser constituído por componentes de fácil acesso e de baixo custo. Além disso, possuem maior escalabilidade e são mais flexíveis a diferentes tipos de aplicações.

Na sua forma mais básica, um cluster é um sistema que compreende dois ou mais computadores ou sistemas (denominados nodos) na qual trabalham em conjunto para executar aplicações ou realizar outras tarefas, de tal forma que os usuários que os utilizam tenham a impressão que somente um único sistema responde para eles, criando assim uma ilusão de um recurso único (computador virtual) [PIT03]. A idéia básica de um cluster, também denominado agregado computacional, é unir computadores para formar um sistema de alto desempenho, de alta disponibilidade no caso de aplicações críticas ou com as duas características. Cada computador do cluster é denominado de nó ou nodo. A “ilusão de um recurso único” trata-se da visualização do cluster por parte dos usuários. Para prover essa funcionalidade são necessários softwares de imagem única ou SSI (Single System Image em Inglês).

4.1 CLASSIFICAÇÃO

Segundo Dantas [DAN05], existem cinco métricas principais para se classificar clusters:

- **Limite Geográfico:** Compreende o alcance geográfico do cluster, que pode compreender uma pequena sala ou até mesmo vários departamentos de uma grande organização. A noção de limite geográfico é muito importante na fase de projeto do cluster, uma vez que a escolha das configurações e componentes do cluster podem ser feitas de forma mais eficiente se a abrangência do mesmo for levada em consideração.
- **Utilização de Nós:** Pode ser de dois tipos: utilização dedicada e não-dedicada. Na configuração dedicada os nodos são utilizados somente para executar as tarefas referentes

ao cluster, enquanto que em configurações não-dedicadas os nodos são também utilizados para outros propósitos, como computadores pessoais, por exemplo. Um cluster não-dedicado pode ser projetado em uma empresa em uma rede de computadores pessoais, por exemplo, onde cada computador pessoal seria um nodo do cluster. As principais vantagens dessa abordagem são o baixo custo, pois os computadores já estão disponíveis, e o aumento de desempenho, uma vez que se pode aproveitar o tempo em que os computadores ficariam ociosos. A principal desvantagem é o baixo desempenho se comparado a configurações dedicadas. O fato de possivelmente as máquinas possuírem sistemas operacionais diferentes ou de versões diferentes, ou até mesmo arquiteturas diferenciadas causa uma sobrecarga em termos de software para definir interoperabilidade entre os ambientes. Além disso, políticas de prioridades devem ser estabelecidas com relação aos processos atribuídos pelo cluster e os processos do usuário local dos sistemas operacionais dos nodos, visando não prejudicar o desempenho da máquina. Já as configurações dedicadas, são mais indicadas quando há necessidade de se executar aplicações críticas ou que requerem desempenho mais elevado. São geralmente projetados para atingir altos desempenhos em aplicações específicas. Possuem custo financeiro mais alto do que as configurações não-dedicadas, porém o desempenho é mais alto, tendo em vista que os recursos computacionais dos nodos já são previamente conhecidos pelo software de *middleware* (responsável pela quantificação dos recursos de hardware dos nodos) e se pode adotar uma plataforma de hardware padrão, não necessitando resolver problemas de interoperabilidade na camada de software.

- **Tipo de Topologia:** Outro fator determinante em termos de desempenho é o tipo de hardware empregado na confecção do cluster e na interconexão dos nodos. Os tipos de hardware são subdivididos em NOW, COW e Clump (Cluster of SMPs). As NOW's (Network of Workstations ou Redes de estações de trabalho em Português) geralmente, [OME04] são sistemas constituídos por várias estações de trabalho interligadas por tecnologia tradicional de rede como Ethernet e ATM. Na prática [OME04], uma rede local de estações que já existe é utilizada na execução de aplicações paralelas. Sob o prisma das arquiteturas paralelas, a rede local pode ser vista como uma máquina paralela em que vários processadores, com suas memórias locais (estações de trabalho), são interligados por uma rede, constituindo assim uma máquina de baixo custo (ou sem

qualquer custo, caso a rede já exista). O principal problema dessa configuração é o tipo de conexão utilizada, uma vez que a rede é compartilhada com outros usuários a alta latência e as taxas de transmissão relativamente baixas das redes ethernet e ATM acabam prejudicando o desempenho do cluster. As COWs (Cluster of Workstations ou cluster de estações de trabalho em Português) [DAN05] são geralmente constituídas de máquinas homogêneas e dedicadas à execução de aplicações específicas. Como possuem uma rede específica e hardware homogêneo, acabam tendo desempenho superior às configurações NOW. Já os Clumps (Cluster of SMPs) [DAN05] são um ambiente composto de máquinas com arquitetura SMP. A arquitetura Symmetric Multiprocessors (SMP), ou multiprocessadores simétricos, consiste em dois ou mais processadores compartilhando a mesma memória e dispositivos de entrada e saída. Os sistemas SMP possuem escalabilidade limitada, uma vez que o aumento no número de processadores causa maior número de colisão nos acessos à memória.

- **Aplicações Alvo:** Consiste no tipo de aplicação para o qual o cluster foi projetado. Nessa classe existem duas métricas principais, o alto desempenho (HPC – High Performance Computing) e a alta disponibilidade (HA – High Availability). Grande capacidade de processamento, grande capacidade de memória, grande capacidade de armazenamento ou uma combinação dessas características definem o alto desempenho. Já a alta disponibilidade é uma característica ligada ao funcionamento de aplicações críticas, as quais não podem ou devem sofrer o mínimo de interrupções possível. A alta disponibilidade será explicada de forma mais detalhada nos próximos itens.
- **Tipos de Nós:** São classificados com relação ao software e hardware. Os clusters de nós homogêneos possuem hardware e software semelhantes, o que facilita a interoperabilidade no nível de software. Já os sistemas heterogêneos são compostos por nodos com sistemas operacionais ou plataforma de hardware distintas o que requer um esforço para operacionalizar a comunicação entre os nodos e deixar o sistema com uma imagem única para o usuário.

Segundo Pitanga [PIT03], clusters também podem ser classificados como **balanceamento de carga** ou **Load Balancing**. Nesse modelo as solicitações recebidas pelo cluster são distribuídas entre várias máquinas que devem executar os mesmo aplicativos. Esta distribuição leva em conta a carga a qual cada nodo está sendo submetido. Também possui uma

proteção contra falhas, pois caso um nodo fique indisponível as solicitações serão repassadas aos nós ativos.

O cluster usado nesse trabalho é um cluster dedicado de processamento distribuído do tipo Beowulf. Ele trabalha com um nodo mestre, também denominado front-end, responsável por distribuir as tarefas aos outros nodos do cluster denominados escravos (back-end). Os nodos escravos, por sua vez, processam suas respectivas tarefas e enviam os resultados ao nodo mestre.

Uma desvantagem desta abordagem está na forte centralização no nodo mestre, sendo que uma falha no mesmo irá comprometer todo o sistema.

Outro ponto de vital importância é o algoritmo de escalonamento. Para o cluster possuir um bom desempenho as tarefas devem ser distribuídas igualmente entre os nodos, considerando que os nodos tenham a mesma capacidade de processamento. Caso um determinado nodo fique sobrecarregado, o *throughput* do cluster será comprometido. No entanto, o algoritmo de escalonamento não deve ser muito complexo a ponto de consumir muito processamento, ocupando tempo precioso da CPU.

4.2 ALTA DISPONIBILIDADE

Antes de se falar sobre alta disponibilidade, faz-se necessária a compreensão de alguns conceitos:

Defeito: Um defeito (*failure*) do sistema ocorre quando desvia do que designado por suas aplicações [AND81][PER04]. Desta forma, um sistema está defeituoso quando ele não pode prover o serviço desejado [PER04].

Erro: Um erro (*error*) é esta parte do estado do sistema que está suscetível a levar a defeitos subsequentes [LAM81][PER04].

Falha: É o elemento que ocasiona o erro, provocando no sistema uma transição de estado não planejada, levando o sistema para um estado de erro [DAN05]. Um sistema pode possuir uma ou mais falhas e não apresentar erros [DAN05]. A falha (*fault*) [PER04] pode ser física ou lógica.

Failover: Também chamada de recuperação de falhas, consiste na recuperação de um serviço, de uma máquina que apresenta falha, por uma outra máquina do sistema. O *failover* pode ser automático ou manual, sendo o automático o que normalmente se espera de uma solução de Alta Disponibilidade [CON05].

Failback: É o processo inverso do *failover*. Acontece quando há a recuperação da máquina em houve falha que pode, portanto, recuperar seus serviços.

Tolerância a falhas: É uma forma de se alcançar a alta disponibilidade, porém não é a mesma coisa. Sistemas com tolerância a falhas, são sistemas capazes de mascarar falhas através da redundância de software e hardware.

Alta Disponibilidade não é apenas um produto ou uma aplicação que se instale, e sim uma característica de um sistema computacional[CON05]. Um sistema *A* terá maior disponibilidade que um sistema *B* caso permaneça mais tempo em funcionamento. Geralmente a alta disponibilidade é provida através da redundância de componentes de hardware e software. Assim, pode-se classificar a disponibilidade do sistema como [CON05] disponibilidade básica, disponibilidade contínua e alta disponibilidade.

Sistemas com disponibilidade básica são sistemas que não possuem nenhum mecanismo para recuperação ou alguma forma de mascarar falhas, possuindo apenas componentes básicos para seu funcionamento normal. As [CON05] máquinas nesta classe apresentam uma disponibilidade de 99% a 99,9%, desconsiderando paradas programadas. Entenda-se paradas programadas como momentos em que o sistema é parado para manutenção ou outra tarefa que não seja ocasionada por uma falha.

Os sistemas com disponibilidade contínua são sistemas que funcionam 100% do tempo, ou seja, nunca param. Esse tipo de sistema ideal é teórico e inexistente na prática. Na vida real podem ser feitos investimento e planejamento a fim de atingir uma disponibilidade muito alta, próxima de 100%, no entanto, não se pode garantir uma disponibilidade total do sistema.

Já os sistemas de alta disponibilidade são projetados para detectar, mascarar e recuperar-se de falhas. Mascarar uma falha significa impedir sua visualização por um observador externo [PER04]. Sendo assim, esses sistemas devem ser capazes de identificar a ocorrência de uma falha e não ficarem indisponíveis por causa da mesma, tentar corrigir a falha ou então deixá-la transparente ao usuário do sistema. Segundo Conectiva[CON05], sistemas de alta disponibilidade ficam disponíveis, pelo menos, 99,99% do tempo. Ou seja, podem ficar até uma hora indisponíveis em um ano. Como exemplo de sistema com alta disponibilidade podemos citar sistemas de centrais telefônicas e sistemas bancários.

4.3 OSCAR

O Open Source Application Resources (OSCAR) é um software que possibilita a construção de um sistema de cluster de alto desempenho (HPC). O OSCAR é um SSI (Sistema de Imagem Única), ou seja, o usuário do cluster visualiza um único sistema, apesar do cluster poder ser formado por inúmeros nodos. O OSCAR é formado por um nodo denominado mestre e por nodos escravos. O nodo mestre possui a função de distribuir as tarefas entre os nodos escravos que, por sua vez, se limitam ao processamento das tarefas. Os nodos escravos devem possuir hardware homogêneos.

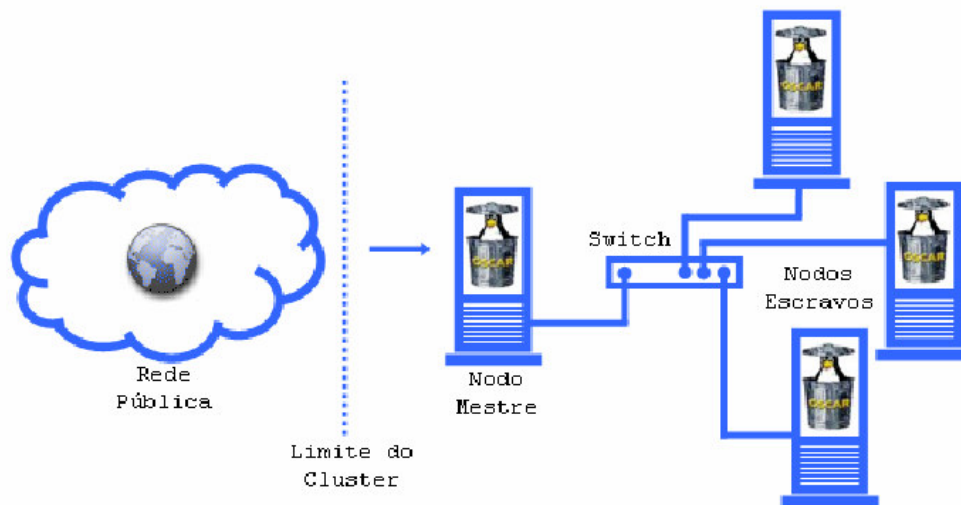


Fig. 4.1 – Esquema de um cluster OSCAR [BAG04].

O OSCAR é um projeto open source sobre a licença GPL (GNU General Public License) e está atualmente na versão 4.1. A versão utilizada nesse trabalho foi a 4.0 que está instalada em um cluster no Laboratório de Desenvolvimento Web (Labweb) localizado na Universidade Federal de Santa Catarina (UFSC).



Fig. 4.2 – Janela de instalação do OSCAR [OSC05].

4.4 HA-OSCAR

O HA-OSCAR (High Availability Open Source Cluster Application Resources) também é um projeto *open source* (código aberto) e implementa técnicas de alta disponibilidade aplicadas em conjunto com a ferramenta OSCAR.

Para prover alta disponibilidade [HAO05] a redundância de componentes é adotada nos clusters HA-OSCAR, visando eliminar o ponto único de falha (nodo mestre). O HA-OSCAR possui mecanismos de recuperação de falhas, failover e failback automáticos.

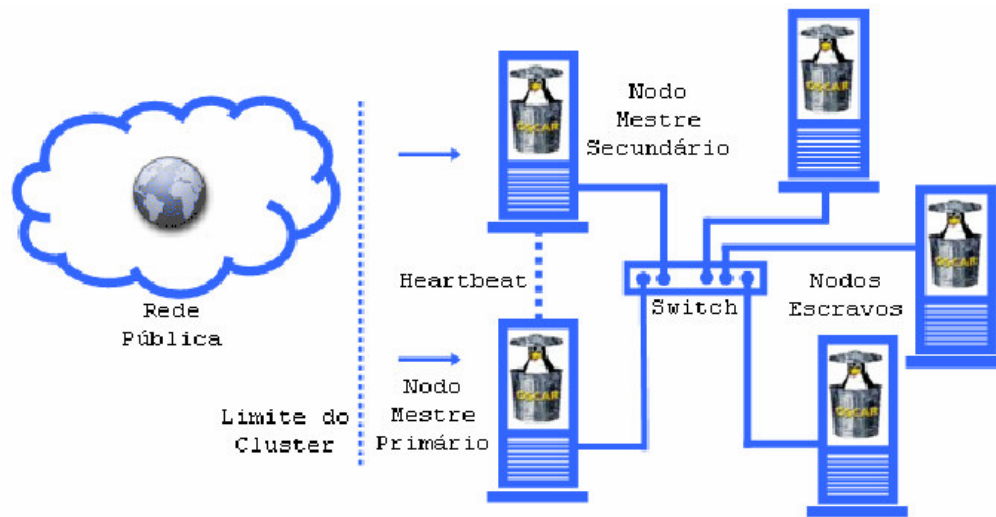


Fig. 4.3 – Esquema de um cluster HA-OSCAR [BAG04].

Vemos no esquema acima a adição de um nodo mestre secundário. Como uma falha no nodo mestre compromete o funcionamento de todo o cluster, uma solução encontrada para prover alta disponibilidade é replicar esse nodo. O nodo mestre secundário é uma réplica do mestre primário. A sua função é monitorar o mestre primário e em caso de falha assumir seu lugar, até que o mestre primário volte ao seu funcionamento normal.

5. AMBIENTE DE DESENVOLVIMENTO

Nesse item serão descritas as ferramentas utilizadas no desenvolvimento e teste do software de monitoração de processos. O cluster para o qual esse trabalho foi desenvolvido se encontra no Laboratório de Desenvolvimento Web (LabWeb), localizado na Universidade Federal de Santa Catarina (UFSC). Nele estão instalados o pacote OSCAR 4.0 e o pacote HA-OSCAR beta para prover a alta disponibilidade. O cluster é formado por quatro máquinas de arquitetura IBM PC x86, cada uma com um processador Intel Pentium IV com clock de 1800 MHz, disco rígido de 40GB conectadas por um *switch* Fast Ethernet. O nodo mestre possui 512MB de memória RAM enquanto os escravos possuem 256MB. O cluster é composto por um nodo mestre, um nodo mestre secundário e dois nodos escravos.

Devido a problemas relativos a mudança de local do LabWeb o cluster se encontrava indisponível na época de finalização desse trabalho, logo o desenvolvimento e os testes foram feitos em um único computador através de uma simulação do ambiente (mais detalhes no item 6).

Foram desenvolvidos dois aplicativos para a tarefa de monitoração do cluster: um aplicativo que contém a lógica da monitoração dos processos, desenvolvido para ser instalado no cluster. Este foi implementado na linguagem de programação Java utilizando o ambiente de desenvolvimento Eclipse. O outro aplicativo foi desenvolvido para ser instalado em um dispositivo móvel e informar ao administrador do cluster sobre possíveis falhas no sistema, além de permitir que o administrador verifique algumas informações, como memória ocupada por cada processo e taxa de ocupação da CPU, sobre os processos que estão sendo executados no cluster. Mais detalhes sobre os aplicativos serão apresentados no item 6.

5.1 JAVA

Desenvolvida pela empresa Sun Microsystems, a linguagem Java surgiu em 1991 originalmente com o nome de Oak. Foi criada com o objetivo de desenvolver softwares para sistemas embutidos e dispositivos PDAs (Portable Data Assistentes). A partir de 1995 a linguagem foi adaptada para a Internet usando uma tecnologia chamada Applet que permitia a execução de conteúdos dinâmicos pelos navegadores web. Com a grande e rápida expansão da Internet, o Java se disseminou mundo afora.

Uma das características fundamentais para a popularização da linguagem foi a sua portabilidade. Um código escrito em Java pode ser compilado em uma determinada plataforma e executado em ambientes diferentes. Por exemplo, um código compilado em uma máquina Intel x86 com sistema operacional Microsoft Windows, pode ser executado em uma máquina com processador AMD 64 bits e sistema operacional Linux sem necessidade de se recompilar o código. Isso ocorre porque a linguagem Java é interpretada e não compilada diretamente para o código nativo da plataforma. Bytecodes são gerados a partir do código fonte e são interpretados pela máquina virtual em tempo de execução. A implementação da máquina virtual difere de acordo com o sistema operacional e o hardware do computador. Apesar de linguagens interpretadas já existirem antes do Java, a união da portabilidade com a tecnologia de Applets possibilitou a linguagem uma grande popularização.

Atualmente, a tecnologia Java possui inúmeras aplicações. Existe, além de outras tecnologias, o pacote J2SE (Java 2 Platform Standard Edition) voltado para aplicativos Desktop, J2EE (Java 2 Platform Enterprise Edition) voltado a aplicações distribuídas e o J2ME (Java 2 Platform Micro Edition) para dispositivos móveis (celulares e PDAs).

Além disso, o Java implementa os principais mecanismos de orientação a objetos como classes, herança, associação, objetos, abstração, polimorfismo e encapsulamento. Possui bibliotecas padrão bastante abrangentes que tratam coleções de dados, acesso a bancos de dados, criptografia, além de inúmeras outras funcionalidades. Também oferece suporte a execução paralela através da criação e sincronização de Threads. É uma linguagem robusta e confiável.

No entanto, uma desvantagem em relação a outras linguagens de programação como o C, Delphi ou o C++ ainda é o desempenho. Como os códigos escritos nas linguagens citadas anteriormente são compilados diretamente para as plataformas nas quais serão executados, eles possuem geralmente melhor desempenho. Porém, o desempenho é bastante satisfatório para a grande maioria das aplicações e existem ainda implementações comerciais como o Excelsior JET e livres como o compilador GCJ que permitem gerar instruções nativas a partir do código fonte Java.

A vasta documentação disponível e a familiarização com a linguagem foram fatores decisivos para a escolha da linguagem para a implementação desse projeto, além de qualidades como portabilidade, orientação a objetos, suporte a execução paralela, robustez e API abrangente. A versão utilizada nesse trabalho foi a 1.4.2. O Java está atualmente na versão 1.5.

5.2 ECLIPSE

O Eclipse versão 3.1 foi o ambiente de desenvolvimento escolhido para a implementação do sistema. O Eclipse é um projeto código aberto que foi originalmente criado pela IBM e acabou tornando-se uma fundação, denominada Eclipse Foundation, com a participação de outras empresas. Além do Java, o Eclipse oferece suporte a C e C++.

O Eclipse é um ambiente de desenvolvimento bastante completo, mas suas funcionalidades também podem ser ampliadas através da instalação de plugins do próprio projeto Eclipse ou desenvolvidos por terceiros e disponíveis na Internet.

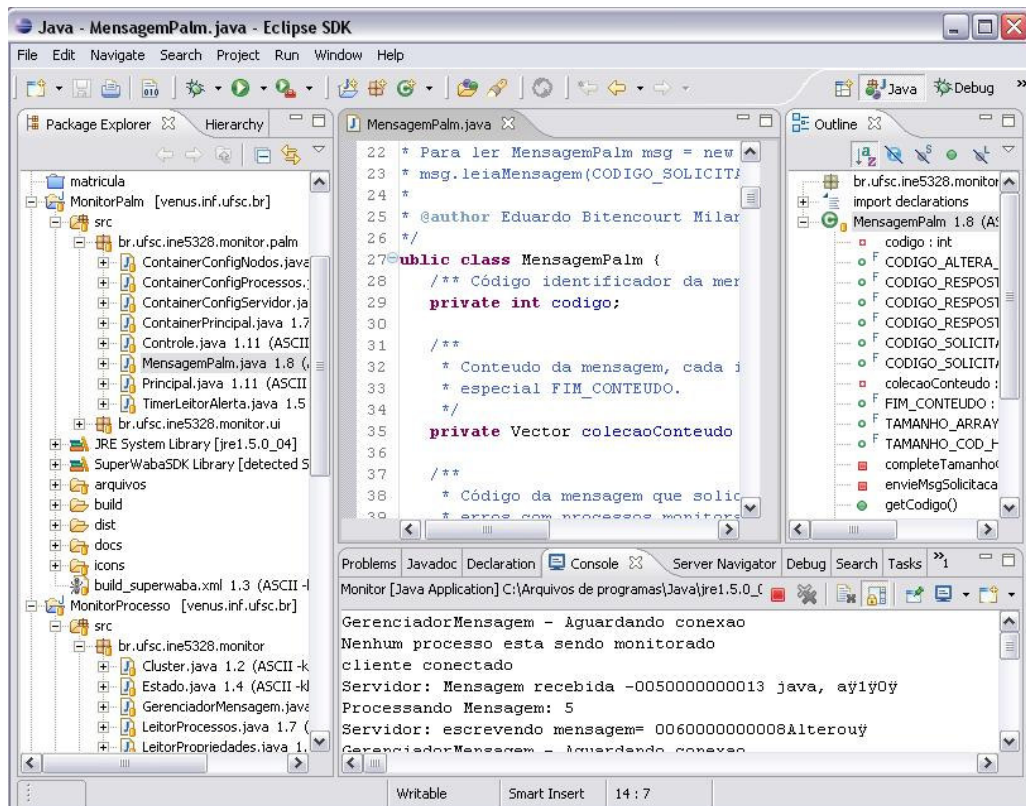


Fig. 5.1 – Ambiente de desenvolvimento Eclipse.

5.3 POSE

O sistema cliente foi implementado para a instalação em um dispositivo móvel (PDA) da marca Palm, modelo Tungsten C, com sistema operacional Palm OS 5.21, 64 MB de memória RAM e processador com frequência de clock de 400 MHz. Entretanto, durante o desenvolvimento do sistema, testes são necessários para a correta implementação do aplicativo.

Realizar esses testes no próprio dispositivo não é recomendável, uma vez que podem danificar dados existentes no PDA devido a uma falha no aplicativo, além de consumir mais tempo. Tendo em vista essa situação, foram utilizadas duas ferramentas de simulação do Palm. O POSE (Palm OS Emulator) e o Palm OS Simulation.

O POSE [PAL03] é um software que emula o hardware de vários modelos de Palm. A idéia do POSE foi desenvolvida inicialmente por programadores independentes, em especial através do projeto Copilot iniciado em 1996. A Palm só assumiu o projeto em 1998, dando-lhe o nome de POSE.



Fig. 5.2 – POSE

O Palm OS Simulator [PAL03] é o verdadeiro Palm OS sendo executado sob uma camada de abstração em um ambiente Windows. Diferentemente do emulador, o simulador não simula o hardware do dispositivo. Apesar disso, o POSE só permite emular os sistemas operacionais até a versão 4.1, enquanto o simulador permite a simulação até a versão 5 – a qual deu origem a versão Tungsten usada nesse trabalho.

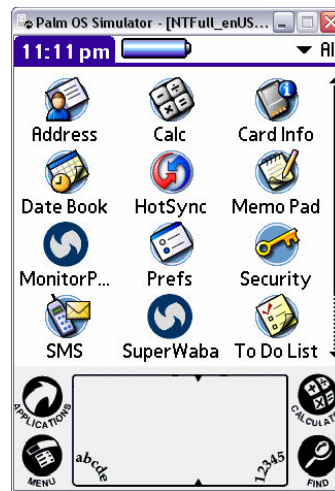


Fig. 5.3 – Simulador PALM.

Para simular um ambiente o usuário deve instalar, tanto no simulador quanto no emulador, imagens do sistema operacional denominadas ROM. Essas imagens são arquivos que contêm as características do dispositivo, como sistema operacional, tipo do processador e cores do visor. As ROMs do simulador e do emulador não são compatíveis.

5.4 SUPERWABA

SuperWaba [SUP05] é uma plataforma para desenvolvimento de aplicações para PDA (Personal Digital Assistants) e Smartphones. O SuperWaba é uma plataforma open-source e foi desenvolvido pelo brasileiro Guilherme Campo Hazan, no ano de 2000, a partir de outro projeto open-source denominado Waba. O SuperWaba é composto por uma máquina virtual denominada SWVM (SuperWaba Virtual Machine), por bibliotecas básicas e de extensão que contém classes que auxiliam no desenvolvimento dos aplicativos e utilitários de compilação que permitem compilar e exportar o código para diferentes plataformas.

O SuperWaba possui duas versões: a versão Community que está protegida pela licença GNU GPL e a Professional que possui alguns recursos adicionais e faz uso da licença GNU LGPL. A licença GPL é gratuita, entretanto as aplicações desenvolvidas a partir dessa versão do SuperWaba devem ser também gratuitas e de código fonte aberto. Já a versão Professional faz uso da versão GNU LGPL que permite o desenvolvimento de código proprietário e sobre a qual incide uma taxa anual. A versão Community, que foi a utilizada nesse trabalho, está atualmente na versão 5.5.

Para gerar os bytecodes pode ser utilizado qualquer compilador Java, uma vez que a máquina virtual do SuperWaba (SWVM) implementa praticamente todos os bytecodes Java, com exceção do `synchronized`. Assim, a sintaxe dos aplicativos é a mesma de um programa escrito em Java o programador somente tem que se adequar às novas bibliotecas. Cabe ressaltar, porém, que o SuperWaba não é Java apesar das características em comum. Abaixo, o processo de desenvolvimento:

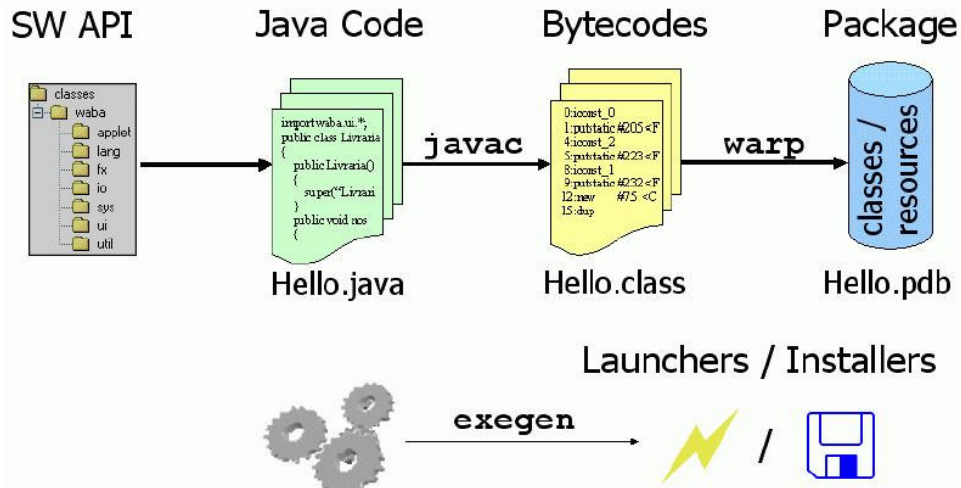


Fig. 5.4 – Esquema Compilação SuperWaba [SUC05].

A partir do código fonte deve-se executar um compilador Java e gerar os bytecodes. O próximo passo é executar o utilitário Warp, do SuperWaba, que converte os bytecodes em um arquivo com extensão `pdb` que será enviado ao PDA. Outro utilitário, o Exegen, é responsável por criar um arquivo executável de acordo com a plataforma do dispositivo. Esse arquivo, com extensão `.prc` no caso do palm OS, inicia a execução do aplicativo e da máquina virtual do SuperWaba.

A escolha do SuperWaba ao invés da tecnologia J2ME para a implementação deste projeto se deve principalmente por causa de características do SuperWaba, como menor espaço ocupado pela máquina virtual, facilidade de migrar o código para outras plataformas e principalmente pelo grande número de componentes disponível na biblioteca padrão e pela facilidade na implementação de interfaces gráficas com o usuário. Entre outras novidades, o SuperWaba possibilita ao programador determinar as posições dos componentes no visor do PDA, inserir mais de um componente na mesma linha e ainda atribuir valores relativos para posicionar os componentes de interface a partir de outros componentes gráficos.

As bibliotecas do SuperWaba são divididas em pacotes de acordo com suas funcionalidades. Por questões legais, uma vez que Java é marca registrada da Sun Microsystems, os pacotes do SuperWaba não podem possuir os mesmos nomes do Java. A estrutura básica de pacotes do SuperWaba é:

waba.fx: Contém classes para manipulação de gráficos e sons. Nelas estão classes como Image, Sound, Graphics, Font entre outras.

waba.io: Responsável pelas classes que implementam regras de comunicação com dispositivos de entrada e saída de dados. Classes como Socket, File, Catalog, Datastream, SerialPort fazem parte desse pacote.

waba.lang: Semelhante ao pacote java.lang, mantendo inclusive o mesmo nome (java.lang). O pacote java.lang é importado implicitamente pelo compilador Java, por isso é necessário que o nome do pacote seja igual. Entretanto, as classes do waba.lang possuem implementação diferente das classes do Java. Possui classes básicas como String, Class, Math e Object, por exemplo.

waba.sys: Classes que executam operações referentes ao Sistema Operacional. Possui classes com Thread, Settings (que contém informações sobre o sistema operacional) e Convert (responsável por conversões entre tipos de dados primitivos).

waba.ui: Contém componentes de interface como Window, Button, Edit, Label, entre outras.

waba.util: Contém classes que implementam estruturas de dados e Datas. Por exemplo, Vector, Date, Hashtable e Random.

waba.applet: Utilizada para emular o SuperWaba como uma applet em Desktop.

Além dos pacotes básicos o SuperWaba possui mais 39 pacotes de extensão com inúmeras funcionalidades. Acesso a banco de dados, operações de compactação, criptografia e operações de tratamento de imagens.gif e .jpg são exemplos de funcionalidades dos pacotes de extensão. Alguns dos pacotes de extensão estão disponíveis somente na versão Professional.

Existem implementações da SWVM para Windows e também para Linux, o que facilita o desenvolvimento e testes dos aplicativos. Nesses sistemas é possível simular a execução de aplicativos executando-os como uma applet Java.

Uma vez que o SuperWaba é compatível com Java, podem-se utilizar os mesmos ambientes de desenvolvimento que se utiliza para programação Java. Sendo assim, o ambiente

escolhido foi o Eclipse 3.1 que foi utilizado tanto para o desenvolvimento do software servidor, o qual foi instalado no cluster, quanto no PDA - Palm Tungsten C com Sistema Palm OS V.

Para o desenvolvimento do aplicativo de monitoração para o palm, foi instalado um plugin no Eclipse, o SuperWaba IDE. O SuperWaba IDE também é um projeto código aberto sob a licença GNU GPL. Esse plugin facilita bastante a geração dos arquivos no formato aceito pelos dispositivos móveis (.pdb e .prc no caso do palm). A geração dos arquivos é feita através de uma interface gráfica, onde o programador pode escolher o número de identificação da aplicação, se a cópia deve ser protegida ou não, a classe executável do projeto, o nome do ícone que irá aparecer no arquivo executável (o prc no caso desse trabalho) a plataforma em que o aplicativo será executado e a versão da aplicação, além de outras três opções exclusivas para o Pocket PC. Esta tela de configuração é bastante útil, pois estas configurações teriam que ser feitas através de linhas de comando o que tomaria um tempo considerável.

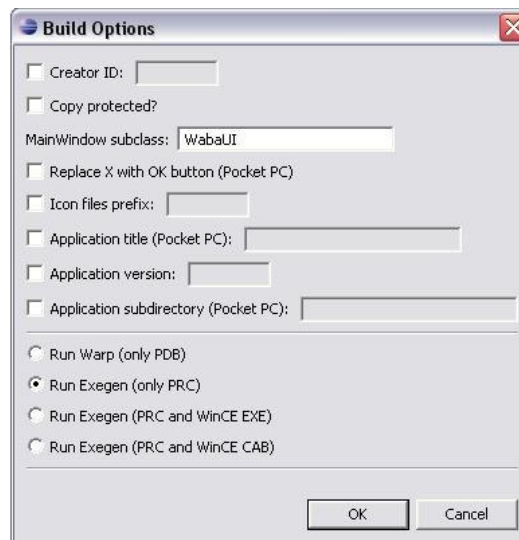


Fig. 5.5 – Assistente para geração de arquivos *pdb* e *prc* do SuperWaba IDE.

Outra facilidade do SuperWaba IDE é a possibilidade de se simular a execução do programa através de uma applet, sem a necessidade de gerar arquivos prc e pdb e carregá-los no POSE a cada teste. Como o plugin é integrado com o Eclipse basta configurar a classe que será executada e iniciar o simulador do SuperWaba IDE através do Eclipse. Todavia, esse simulador não é idêntico ao palm, somente os testes de desenvolvimento foram feitos utilizando esse simulador, os testes complementares do aplicativo foram feitos usando o simulador.

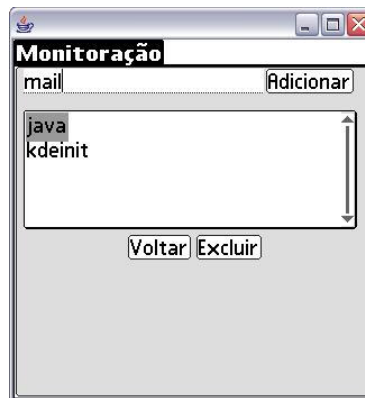


Fig. 5.6 – Simulador SuperWaba do SuperWaba IDE.

A instalação do SuperWaba no Palm compreende três arquivos principais: O *SWNatives.prc* que contém a implementação das funções nativas do PDA, o *SuperWaba.prc* que contém o interpretador de bytecodes, o classloader e o garbage collector (lixeira) e, por último, o *Superwaba.pdb* que contém as classes do pacote básico *waba*. A utilização das bibliotecas de extensão requer a instalação de arquivos *pdb* adicionais.

5.5 XML

XML (eXtensible Markup Language – Linguagem de marcação estendida em português) consiste em uma forma de escrever arquivos respeitando algumas regras pré-definidas. Assim como o HTML (Hiptertext Markup Language), os elementos são representados através de tags. No entanto, as semelhanças param por aí: no HTML o usuário deve utilizar um conjunto de tags já definido, enquanto que no XML as tags podem possuir nomes diversos de acordo com a conveniência do usuário. Cada tag constitui um elemento que pode possuir atributos, elemento pai, elementos filhos. Um documento XML deve possuir um “elemento raiz” que não tem “pai” e deve ser único.

A vantagem de se utilizar XML é que o padrão permite que os dados sejam processados em diversas linguagens. Existem inúmeras bibliotecas para se trabalhar com XML em várias linguagens de programação. Java também possui inúmeras bibliotecas para se trabalhar com XML. Xerces, Crimson, Xom, Xalan são alguns exemplos.

Tendo em vista essas vantagens, o arquivo de configuração do aplicativo servidor foi criado seguindo o padrão XML.

6. PROJETO

É muito comum que sistemas de agregados executem aplicações críticas que não podem ficar indisponíveis. Como todos sistemas computacionais são suscetíveis a eventuais ocorrências de falhas, surge a necessidade da intervenção humana na tarefa de monitoração do ambiente. No entanto, essa tarefa pode possuir alto grau de complexidade, dependendo do cluster e das suas aplicações, que pode ser agravado com o aumento do número de nós do agregado. Uma vez que agregados computacionais podem ser formados por centenas e até milhares de computadores, torna-se inviável a monitoração humana manual sem nenhum tipo de automatização. A proposta deste trabalho é uma ferramenta que auxilie um administrador na tarefa de monitoração de um agregado, enviando mensagens de erro e informações de processos para um dispositivo móvel através de uma rede sem fio.

A idéia deste trabalho foi uma extensão de um trabalho de conclusão de curso e de uma dissertação de mestrado, ambas do curso de Ciências da Computação. O primeiro, desenvolvido pelo aluno Rafael Köhler Baggio, com o título “Uma ferramenta para Monitoração e Alertas SMS em um Ambiente de Cluster com Alta Disponibilidade”, defendido no ano de 2004, consistia na monitoração do agregado e envio de alertas via SMS para PDA’s ou celulares. O outro, desenvolvido por Luís Cassiano Goularte Rista - “Uma Abordagem de Monitoração Wireless em um Ambiente de Cluster com Alta Disponibilidade”, foi defendido no ano de 2005. Nesse último a idéia era, também, a monitoração do ambiente e envio de alertas para um Palm conectado em rede local com conexão wireless. O aprimoramento da idéia, apresentada neste trabalho, foi aplicar o processo de monitoração no nível de processos que executam no agregado. Nos trabalhos anteriores a monitoração do cluster consistia na monitoração dos nodos do mesmo; Assim, num caso onde houvesse falhas em algum dos serviços disponibilizados pelo cluster e nenhuma falha de comunicação entre os nodos, o administrador não seria avisado. Imagine um cluster que é servidor Web de um banco, o qual permite consultas a saldos, extratos, fazer transferências e outras operações comuns em um banco. Nesse cluster existe um processo (JBoss, por exemplo) que é software responsável pelo funcionamento da página Web do banco. Mesmo que nada de errado ocorra com o cluster, um erro no processo JBoss, causado por um erro de programação ou até mesmo problemas no sistema operacional, pode indisponibilizar a página Web que é o objetivo principal da construção do cluster nesse caso. Esse é um sistema crítico, pois movimenta somas vultosas, e horas ou até mesmo minutos em que o servidor esteja

indisponível irá causar grande prejuízo financeiro aos donos e frustração nos clientes que necessitem do serviço. A monitoração de processos é importante, uma vez que permite monitorar os serviços fornecidos para os usuários pelo cluster, o que é em última instância o objetivo final do sistema, disponibilizar serviços.

O desenvolvimento do projeto resultou em dois aplicativos distintos: um sistema servidor implementado em Java usando o pacote de desenvolvimento J2SDK 1.4.2, que deve ser instalado no cluster e é responsável por fazer a monitoração do mesmo. O outro aplicativo foi implementado utilizando a ferramenta de desenvolvimento SuperWaba 5.5 e é um aplicativo voltado a dispositivos móveis (sistema cliente), sendo no caso desse projeto, especificamente, um Palm Tungsten C com sistema operacional Palm OS 5.23. Em ambos aplicativos foi utilizado o ambiente de desenvolvimento Eclipse 3.1 para a implementação.

A lógica do processo de monitoração foi implementada no sistema servidor, sendo que o sistema cliente é responsável por enviar ao sistema servidor os nomes dos processos e nodos do cluster que devem ser monitorados, além de receber eventuais mensagens de erros. A comunicação entre os sistemas (cliente e servidor) é feita através de uma rede sem fio. O sistema cliente também pode solicitar uma lista com informações sobre os processos e nodos que estão sendo monitorados. Essa lista contém o nome do usuário que iniciou o processo, PID (Process ID ou identificador do processo), porcentagem de CPU ocupada, memória ocupada, nome do processo e estado do processo.

6.1 APLICATIVO CLIENTE

Como foi dito anteriormente, o aplicativo cliente foi implementado utilizando a ferramenta SuperWaba 5.5 e deve ser instalado em um dispositivo móvel pertencente ao administrador do agregado.

Para instalar o aplicativo cliente no palm é necessário gerar dois arquivos, um com extensão *prc* que é o arquivo executável e outro com extensão *pdb* que contém os bytecodes. Para o sistema cliente foram gerados arquivos com os nomes de *MonitorPalm.prc* e *MonitorPalm.pdb*. Ao serem instalados no Palm, um ícone com o nome da aplicação (MonitorPalm) aparece no visor.



Fig. 6.1 – Tela inicial do POSE com os ícones do SuperWaba e MonitorPalm.

O sistema cliente possui três arquivos de configuração, um com o nome dos processos (confProcesso.pdb) monitorados, outro com o número dos nodos (confNodo.pdb) e um com as configurações de conexão (confServidor.pdb). A função desses arquivos é fazer a persistência de dados da aplicação, permitindo que as configurações sejam recuperadas mesmo após a aplicação ser encerrada. Assim, além do formulário principal onde são mostrados os dados da monitoração, foram criados formulários para manipular os dados desses arquivos.

Selecionando o ícone da aplicação será exibido um formulário com o título *Monitoração de Processos*. Na parte superior da janela são exibidas duas mensagens: a primeira linha é o nome, ou endereço IP, do cluster que será monitorado e a segunda é a porta usada na conexão com o servidor. Caso não exista nenhum valor configurado os valores após o caractere “:” irão aparecer em branco. Ao lado da mensagem referente a porta de conexão, alinhado a direita, está o botão *Atualizar*. O restante da janela é ocupada por uma caixa de listagem com barras de rolagem vertical e horizontal, inicialmente vazia, onde são exibidas informações sobre processos e nodos. Quando o usuário pressiona esse botão, o aplicativo envia uma mensagem ao sistema servidor, passando como parâmetros os nodos e os processos configurados pelo usuário no dispositivo móvel. O servidor, caso a comunicação tenha ocorrido com sucesso, envia como resposta uma mensagem com informações sobre os processos, que é exibida na caixa de listagem. Caso por algum motivo, servidor indisponível ou o dispositivo móvel fora da área de cobertura, por exemplo, a comunicação com o servidor não possa ser efetuada uma janela aparece para o

usuário contendo o título *Erro* e como conteúdo a mensagem *Erro ao tentar ler lista de processos do cluster*. A janela de erro desabilita o acesso ao formulário principal. Para fechar a janela é necessário pressionar o botão *ok*.



Fig. 6.2 – Formulário Principal da Aplicação.



Fig. 6.3 – Mensagem de erro ao ler a lista de processos.

Selecionando a parte superior da interface, onde está localizado o título da aplicação, é exibida uma barra de menus cujos itens são agrupados de acordo com suas funcionalidades. Os itens são *Arquivo*, *Gerenciar* e *Config*.

Selecionando o item *Arquivo* é exibido o subitem *Sair* que quando selecionado finaliza a aplicação. Também é exibido o item *Testes* que foi utilizado para invocar métodos de teste na etapa de desenvolvimento. Esse item será excluído posteriormente.



Fig. 6.4 – Barra de Menu, item “Arquivo”.

Selecionado o item *Gerenciar* são exibidos quatro subitens: *Processo*, *Nodo*, *Iniciar Monitoração* e *Parar Monitoração*. O subitem *Parar Monitoração* aparece, inicialmente, desabilitado, uma vez que o processo de recebimento de mensagens pelo dispositivo móvel deve ser iniciado selecionando o item *Iniciar Monitoração*.



Fig. 6.5 – Barra de Menu, item “Gerenciar”.

Quando o usuário seleciona o subitem *Processo* a interface do formulário principal é alterada para a interface de configuração de processos. A primeira linha do formulário é composta por uma caixa de texto e pelo botão *Adicionar*. A segunda linha pelo botão *Teclado*.

Logo abaixo, existe uma caixa de listagem com uma barra de rolagem vertical. A última linha do formulário é composta pelos botões *Voltar* e *Excluir*. O primeiro componente do formulário (caixa de texto) serve para inserção do nome de um processo a ser monitorado. O nome do processo pode conter qualquer tipo de caractere e não possui limitação de tamanho, além é claro das limitações de armazenamento do dispositivo. Depois de digitado o nome, o usuário deve pressionar o botão *Adicionar*. Caso já exista um processo com o mesmo nome o sistema não executa nenhuma tarefa. Caso contrário, o item é salvo em um arquivo de nome *confProcesso.pdb* que contém os nomes dos processos monitorados. Como os processos são salvos em arquivo isso permite que a aplicação mantenha as configurações de processos mesmo após ser fechada. Se o arquivo ainda não existir (primeira configuração) ele é criado. O nome do processo é adicionado na caixa de listagem. Para PDA's que não possuem teclado a entrada do nome do processo é feita através de uma outra janela. Para acessar a janela o usuário deve pressionar o botão *Teclado*. Será exibida uma janela com botões e funções de um teclado.

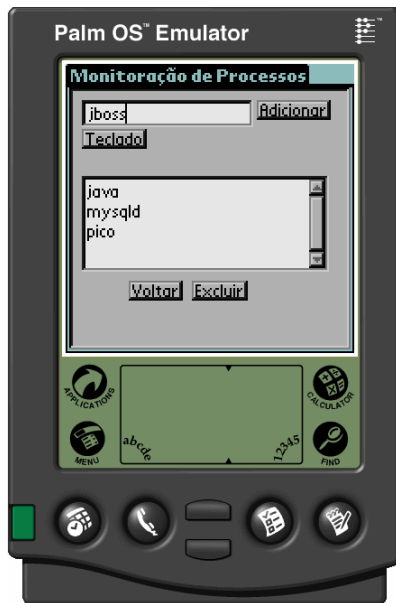


Fig. 6.6 – Formulário de Configuração de Processo.



Fig. 6.7 – Formulário simulador do Teclado.

A caixa de listagem exibe os processos que foram configurados pelo usuário para serem monitorados. Sempre que o formulário de configuração de processos é aberto uma caixa de listagem é preenchida com os nomes de processo que constam no arquivo *confProcesso.pdb*. O conteúdo da caixa de listagem aparecerá em branco quando o arquivo de configuração de

processos não existir ou quando o mesmo estiver vazio. Para excluir um processo basta selecionar um nome de processo na caixa de listagem e pressionar o botão *excluir*. O processo será excluído da interface e do arquivo de processos. Caso não seja possível excluir o processo será exibida a mensagem *O item não pôde ser excluído*. Se o usuário pressionar o botão *excluir* sem selecionar nenhum processo o sistema exibe a mensagem *Selecione um item*. Quando o usuário pressiona o botão *Voltar*, a interface de configuração de processos é fechada e o fluxo retorna para o formulário principal. Nome de processos digitados, mas não adicionados são descartados.

Selecionando o item *Nodo* na barra de menu, o usuário acessa o formulário de configuração de nodos. A interface de configuração de nodos é semelhante a de processos. Ao invés do nome do processo, deve ser digitado na caixa de texto o número de um nodo a ser monitorado e pressionar o botão *Adicionar*. Os dados dos nodos são salvos no arquivo *confNodo.pdb* em formato String e a limitação de tamanho é reativa a capacidade do dispositivo. Não é feita nenhuma verificação se o dado é ou não numérico.



Fig. 6.8 – Formulário de configuração dos nodos monitorados.

O próximo subitem da barra de menu é *Iniciar Monitoração*. Ao selecionar esse subitem o recebimento de mensagens de erro é habilitado. Nesse caso, possíveis erros ocorridos no cluster serão exibidos na interface do PDA. Para verificar se existem erros no cluster, o sistema cliente envia uma mensagem ao aplicativo servidor, solicitando as mensagens de erro. O servidor retorna uma mensagem com os erros encontrados ou uma mensagem com conteúdo vazio caso o agregado esteja funcionando perfeitamente. O subitem *Iniciar Monitoração* é desabilitado e o

subitem *Parar Monitoração* é habilitado. A verificação de erro é feita por uma *Thread* que será executada em t segundos, onde t é um valor configurado pelo usuário na interface de configuração do servidor. Pressionando *Parar Monitoração* a verificação de mensagens de erro é parada, o item *Iniciar Monitoração* é habilitado e o item *Parar Monitoração* é desabilitado.



Fig. 6.9 – Erro no processo pico.



Fig. 6.10 – Erro no nodo número 4.

Caso a conexão com o sistema servidor não possa ser estabelecida a mensagem *Erro ao ler mensagem do Servidor* é exibida.



Fig. 6.11 – Erro na conexão com o servidor.

O último item da barra de menu (Config) possui apenas um subitem de nome *Servidor*. Esse item dá acesso ao formulário de configuração de dados do servidor. O formulário possui três caixas de texto com botões de acesso a janela de teclado ao lado. Acima de cada caixa de texto existe uma etiqueta descrevendo qual atributo deve ser digitado. Na última linha do formulário são exibidos os botões *Salvar* e *Voltar*.



Fig. 6.12 – Formulário de configuração de parâmetros do servidor.

A primeira caixa de texto, de cima para baixo, corresponde ao nome ou endereço IP que é utilizado na conexão com o aplicativo servidor. O tipo desse atributo é literal e limitado em trinta caracteres. A próxima caixa de texto refere-se à porta onde o sistema servidor está esperando a conexão. Esse valor deve ser numérico e possuir no máximo cinco dígitos. A última caixa de texto serve para configurar o intervalo de verificações de mensagens de erro. A unidade de tempo utilizada é segundos. O valor digitado deve ser inteiro e pode possuir até seis dígitos. As caixas de texto são preenchidas pelo sistema de acordo com os valores da última configuração. No caso de ser a primeira configuração o sistema atribui aos parâmetros os seguintes valores padrão: *localhost* para o nome do servidor, *5050* para a porta de conexão e *30* segundos para o tempo de verificação de mensagens de erro.

Para salvar as configurações o usuário deve pressionar o botão *Salvar*. Caso exista algum campo não preenchido o sistema exibe a mensagem *Existem campos em branco*. Senão os dados são salvos no arquivo *confServidor.pdb* e os atributos na memória são atualizados.

Ao pressionar o botão *Voltar* o formulário de configuração do servidor é fechado e o fluxo de execução retorna para o formulário principal. Dados alterados e não salvos serão perdidos.

Como o SuperWaba implementa praticamente todos os *bytecodes* do Java, ele suporta orientação a objetos e também pode ser dividido em pacotes. Os pacotes têm como objetivo agrupar classes com funcionalidades em comum. No pacote *br.ufsc.ine5328.monitor.cliente* estão contidas classes com as regras de negócio e interface gráfica do sistema. Fazem parte desse pacote as classes *ContainerConfigNodos*, *ContainerConfigProcessos*, *ContainerConfigServidor*, *ContainerPrincipal*, *Controle*, *MensagemCliente*, *Principal* e *TimerLeitorAlerta*. Já o pacote *br.ufsc.ine5328.monitor.ui* foi criado para conter classes de componentes gráficos que precisassem ser criados. Devido a grande variedade de componentes da biblioteca padrão do SuperWaba, somente foi necessária a criação de um componente. A classe do componente foi chamada de *HorizontalListBox*.

As classes *ContainerConfigNodos*, *ContainerConfigProcessos*, *ContainerConfigServidor* implementam os formulários de configuração de nodos, processos e de dados do servidor e aplicação respectivamente. Possuem métodos para tratar a lógica da interface e eventos.

A classe *ContainerPrincipal* contém os componentes gráficos da janela principal. Entre outras atribuições é responsável pela troca do fluxo de execução entre os formulários da aplicação.

Já a classe *Controle* implementa a lógica principal do Sistema do PDA. É responsável pela leitura e armazenamento das listas de processos e de nodos em arquivos, além das configurações do servidor e do sistema cliente. Nela também está implementado o método de leitura de processos do aplicativo servidor.

MensagemCliente é uma abstração de uma mensagem. Ela trata o envio e o recebimento de mensagens vindas do servidor. Cada mensagem é composta por um cabeçalho (Header) e pelo conteúdo. O cabeçalho possui tamanho fixo (treze caracteres), porém esse tamanho pode ser facilmente alterado através da modificação de constantes (*TAMANHO_ARRAY_HEADER* e *TAMANHO_COD_HEADER*) nos aplicativos servidor e cliente. Os três primeiros caracteres da mensagem são correspondentes ao código identificador da mensagem. O resto do cabeçalho (dez caracteres) indica a quantidade de caracteres (bytes) do conteúdo da mensagem, que é variável.

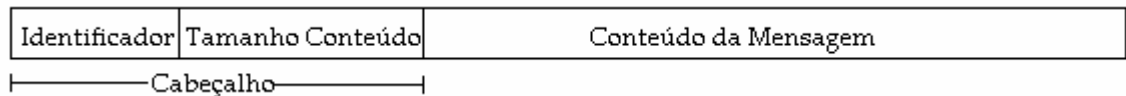


Fig. 6.13 – Formato da mensagem.

O tamanho do conteúdo foi adicionado à mensagem para facilitar a leitura que é feita usando uma estrutura de *array* (vetor), onde se faz necessário prévio conhecimento do número de posições que serão armazenadas. Apesar do Java e do próprio SuperWaba possuírem classes de *buffer* e de *stream* que permitem a leitura da mensagem como um objeto String (variável literal) sem precisar conhecer o tamanho da mensagem, ocorreram alguns problemas na tentativa de comunicação entre esses objetos. Logo, a opção escolhida foi utilizar uma estrutura de *array*, mais simples e sem problemas de funcionamento.

O conteúdo pode ser subdividido em vários elementos. Cada elemento do conteúdo é iniciado como um item em uma estrutura de dados do tipo lista, com o objetivo de facilitar o processamento de mensagens com conteúdos distintos. Esses elementos podem ser separados de acordo com a lógica. Por exemplo, os nomes de nodos e de processos que são enviados em uma mesma mensagem, ou para formatação como na mensagem enviada pelo servidor com a lista de processos. Cada elemento corresponde a uma linha diferente. O caractere `ÿ` é utilizado no fim de cada elemento para separar os conteúdos.

Ex: 0050000000017 java, jbossÿ2ÿ1ÿ

No exemplo acima, o código da mensagem é 5 (005). Essa mensagem atualiza os parâmetros do comando de monitoração. O valor do campo *TamanhoConteúdo* é 17 sinalizando o número de caracteres do conteúdo da mensagem. O primeiro separador (`ÿ`) define os processos que devem ser monitorados pelo sistema. Os separadores restantes devem definir os números dos nodos monitorados. Nesse exemplo, os nodos 1 e 2 estão sendo monitorados. Existem mais cinco tipos de mensagem. Mensagem de solicitação de erros (código 1), Mensagem de resposta de erros (código 2), Mensagem de solicitação de informações sobre os processos (código 3), Mensagem de resposta das informações sobre os processos (código 4) e uma mensagem sinalizando se foi possível atualizar os parâmetros de monitoração (código 6).

A classe *Principal* é a classe que contém o método inicial da aplicação. Ela herda a classe *MainWindow* (pacote `waba.ui`). Além dos métodos de inicialização da aplicação, ela também controla os eventos da barra do menu.

Já a classe *TimerLeitorAlerta* é uma *thread* responsável por disparar a verificação de mensagens de erro. Funciona de forma semelhante a um *Timer*. A cada *t* segundos conecta-se com o cluster e verifica se houve algum erro com os processos que estão sendo monitorados. Em caso afirmativo, mostra a mensagem de erro vinda do servidor em uma caixa de diálogo na interface principal. Onde *t* é valor do intervalo entre monitorações especificado pelo usuário na interface de configuração do servidor. Seu valor padrão é trinta segundos.

HorizontalListBox foi a única classe de componente gráfico que precisou ser criada. A implementação dessa classe surgiu da necessidade de uma caixa de listagem com barras de rolagem verticais e horizontais para exibir as informações dos processos. No SuperWaba as caixas de listagem possuem apenas barras de rolagem horizontais. Porém, existe o componente *ScrollBar* que possibilita a criação de barras de rolagem horizontais. Essa barra horizontal foi adaptada para funcionar com a caixa de listagem padrão do SuperWaba (*ListBox*). Os eventos na barra horizontal tiveram que ser tratados, de modo que o conteúdo fosse reposicionado para direita ou esquerda na caixa de listagem.

6.2 APLICATIVO SERVIDOR

O sistema servidor é composto por arquivos de classes do Java e um arquivo de configuração. A instalação do sistema é bem simples, basta copiar a pasta do projeto para o nó mestre primário e secundário do cluster. A execução do programa é feita através de linha de comando. Dentro da pasta do projeto, o usuário deve digitar “*./run.sh &*”, onde *run.sh* é um arquivo executável que invoca a classe com o método principal do aplicativo servidor (classe *Monitor*) e o *&* é um comando do sistema operacional Linux que serve para iniciar o processo em *background*.

Depois de iniciado, o programa executa uma estrutura de repetição permanentemente que é responsável por monitorar o cluster, verificando possíveis erros. Uma *Thread* é iniciada para aguardar uma conexão do sistema cliente e estabelecer a comunicação entre os sistemas. Nesse momento nenhum processo é monitorado até que o sistema cliente envie uma mensagem contendo os nós e os processos que serão monitorados (feita no formulário principal do aplicativo cliente através do botão *Atualizar*). Depois que isso ocorre, o sistema começa a verificar erros a cada quantum de tempo, o qual é determinado pelo arquivo de configurações do aplicativo servidor. Os erros encontrados são adicionados em uma lista de erros e o programa

aguarda a conexão do sistema cliente para enviar a mensagem. Uma outra lista contém as mensagens de erro já enviadas para que uma mesma mensagem não seja enviada constantemente. Os elementos dessa lista são excluídos quando há uma monitoração em que o elemento não aparece na forma de erro, ou seja, quando não há mais o erro que ocasionou a mensagem. Essa lista é reiniciada quando novas configurações de nodos e processos são enviadas pelo sistema cliente.

6.2.1 ARQUIVO DE CONFIGURAÇÃO

Para possibilitar a mudança de algumas configurações sem que seja necessário recompilar o projeto, foi criado um arquivo de configuração. Trata-se de um arquivo de texto no formato XML com parâmetros do aplicativo. Esses parâmetros são lidos no momento em que a aplicação é iniciada.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.0" class="java.beans.XMLDecoder">
  <object class="br.ufsc.ine5328.monitor.Propriedades">
    <void property="nomeCluster">
      <string><![CDATA[oscarcluster]]></string>
    </void>
    <void property="intervaloMonitoracoes">
      <!-- Tempo entre a verificação dos processos. Em segundos -->
      <int>15</int>
    </void>
    <void property="porta">
      <!-- Porta de comunicacao -->
      <int>5010</int>
    </void>
    <void property="tamanhoBuffer">
      <!-- Tamanho em bytes do buffer de leitura de processos 512KB -->
      <int>524288</int>
    </void>
    <void property="oscarSystem">
      <!-- Indica se o sistema está rodando no cluster ou em modo debug -->
      <boolean>>false</boolean>
    </void>
  </object>
</java>
```

Fig. 6.14 – XML de configuração do servidor.

Devido ao formato padronizado, esse arquivo é lido automaticamente pela classe *java.beans.XMLDecoder* que retorna um objeto *br.ufsc.ine5328.monitor.Propriedades*, cujos atributos são inicializados com os valores dos parâmetros dos elementos do XML. A classe

br.ufsc.ine5328.monitor.Propriedades deve possuir métodos com os mesmos nomes e parâmetros declarados no XML.

A primeira linha do arquivo contém um cabeçalho indicando que se trata de um documento no formato XML. Ela contém a versão do documento e a codificação de caracteres utilizada. A segunda linha corresponde ao elemento raiz (*root*). Os valores atributos *version* e *class* são correspondentes a versão do Java e a classe responsável pela leitura do arquivo. A próxima linha é a declaração do tipo de objeto que será iniciado (*br.ufsc.ine5328.monitor.Propriedades*). Os próximos elementos (*void*) são nomes dos métodos do objeto responsáveis por atribuir valores aos respectivos atributos. Cada elemento *void* (corresponde ao tipo de parâmetro de retorno do método executado, outros tipos são suportados) possui um elemento filho que é um parâmetro passado ao método de inicialização. É possível adicionar vários parâmetros desde que o método respeite a assinatura especificada no XML. O primeiro parâmetro do arquivo de configuração é *nomeCluster* que é um parâmetro literal para configuração do nome do cluster onde o sistema servidor está sendo executado. O próximo atributo declarado foi *intervaloMonitoracoes*. O valor desse atributo indica o tempo entre as monitorações do cluster. A unidade de tempo utilizada foi segundos. O próximo atributo (*porta*) declarado refere-se a porta utilizada para a comunicação com o aplicativo cliente. A propriedade *tamanhoBuffer* corresponde ao tamanho da memória, em bytes, que pode ser alocada nas leituras de arquivos e processos do sistema operacional. A última propriedade (*oscarSystem*) é um valor booleano. Caso o valor seja *true* o sistema será configurado para execução no cluster, caso o valor seja *false* a execução da monitoração será feita em modo de simulação, utilizando os métodos de teste.

6.3 MÉTODO DE MONITORAÇÃO

Para colocar em prática a idéia da monitoração dos processos era necessária alguma forma de se obter informações sobre os processos executados no agregado. A primeira possibilidade imaginada foi a execução de comandos do sistema operacional (como o *top* ou o *ps*) de dentro da aplicação do servidor e a leitura dos resultados. O problema dessa abordagem, no entanto, era a necessidade da aplicação ser instalada em todos os nodos do cluster. Conforme o crescimento do número de nodos a manutenção do sistema seria muito complexa. Além disso,

cada nodo precisaria ter uma máquina virtual Java instalada o que consumiria muitos recursos da memória RAM e espaço em disco.

A solução adotada foi a utilização do comando *cexec*. O *cexec* é um comando do OSCAR que possibilita a execução de um mesmo comando em todos os nodos do cluster ou em um subconjunto de nodos. Por exemplo, “*cexec --all ls -la*” irá executar o comando “*ls -la*” em todos os nodos do cluster. Isso permitiu que o sistema servidor fosse instalado apenas nos nodos mestres e mesmo assim pudesse monitorar qualquer nodo do cluster. Foi concatenado ao *cexec* o comando *os* do linux que possibilita a leitura de várias informações sobre os processos que estão sendo executados no sistema operacional, além de opções de ordenação e filtragem. O comando é montado na hora da execução de acordo com os nodos e nomes de processos selecionados pelo usuário.

```
Ex: cexec oscarcluster:1 ps -o user,pid,pcpu,pmem,comm,stat --sort user  
-C java, kdeinit
```

Após o *cexec* vem o nome do cluster (configurado no arquivo XML de parâmetros), seguido de “:” e do número do nodo onde o comando será executado. Os números do nodo são informados pelo sistema do PDA. Depois, o comando *ps* do linux. A opção “-o” foi usada para formatar o comando com os dados dos processos desejados. A primeira (user) coluna corresponde aos usuários que iniciaram o processo. O “pid” corresponde ao código identificador de processo. A coluna “pcpu” contém informações sobre a porcentagem de cpu e a “pmem” a porcentagem de memória utilizada pelo processo. A coluna “comm” exhibe o nome do comando executado para iniciar o processo e “stat” o estado do processo. A opção “--sort user” ordena o resultado em ordem alfabética de acordo com o nome de usuário que iniciou o processo. E, finalmente, a opção “-C” é responsável por filtrar os processos exibidos de acordo com seus comandos. Os nomes dos processos devem ser separados por virgula e são informados pelo sistema cliente.


```

***** oscar_cluster *****
----- oscarnode1-----
USER      PID %CPU %MEM  COMMAND      STAT
bin       1249 0.0  0.0  portmap      S
jboss     28467 0.0  0.0  sh           S
jboss     28475 2.0  46.4 java         S
lp        2500 0.0  0.1  cupsd        S
mysql     2267 0.0  0.0  mysqld       S
postfix   2626 0.0  0.1  qmgr         S
postfix   31044 0.0  0.5  pickup       S
postgres 31049 0.0  1.1  postmaster   S
postgres 31050 0.0  1.1  postmaster   S
root      2    0.0  0.0  keventd      SW

```

Fig. 6.15 - Resultado da execução do comando *cexec*

A execução do comando é efetuada a cada quantum de tempo de acordo com o parâmetro *intervaloMonitoracoes* informado no arquivo de configuração. Depois de executado o comando, é feita a leitura das linhas de resultado e objetos da classe *Nodo* e *Processo* são iniciados. São realizados três tipos de verificações erro. Primeiro, é verificado se algum dos processos monitorados está em estado *zombie* (estado Z) ou parado (estado T). Em caso afirmativo mensagens de erro são adicionadas indicando o nome do nodo onde o processo está sendo executado, o nome do processo, e o estado do mesmo. Depois é verificado se todos os processos monitorados foram exibidos no resultado dos comandos *cexec*. Os processos monitorados que não estiverem no resultado dos comandos executados não estarão ativos, logo uma mensagem de erro é adicionada indicando que o processo não está ativo em nenhum nodo do cluster. A última verificação feita é a dos nodos. Caso algum comando executado em um nodo monitorado não possua resultado, uma mensagem de erro é adicionada indicando que o nodo está indisponível.

6.4 ESTUDO DE CASO

Devido a problemas relativos a mudança de local do LabWeb o cluster se encontrava indisponível na época de finalização desse trabalho, assim os testes dos aplicativos foram feitos em apenas uma máquina, com sistema operacional Windows XP, de forma a simular o comportamento do ambiente de cluster. Para simular o funcionamento do palm foram utilizados o Palm OS Emulator v3.5 e o Palm OS Simulation 5.3.

Primeiro, as interfaces de configuração do aplicativo cliente foram testada nos simuladores. Depois, foi criada uma configuração padrão para testar a ocorrências de falhas no

agregado. Na execução desses testes os mesmos parâmetros de configuração foram utilizados. São eles:

No aplicativo servidor:

- **nomeCluster:** oscarcluster;
- **intervaloMonitoracoes:** 15 segundos;
- **porta:** 5010;
- **tamanhoBuffer:** 512 KB;
- **oscarSystem:** false.

No aplicativo cliente:

- **Nome do Servidor:** localhost;
- **Porta:** 5010;
- **Intervalo Monitoração:** 10 segundos.

Além desses parâmetros foram adicionados no sistema cliente os nodos 1 e 2 e os processos java e mysqld. Essa configuração de nodos e processos será usada em todos os testes, mas sofrerá alterações dependendo da funcionalidade que irá ser testada.

Para simular o comando “cexec” foi criada uma classe denominada *GeradorPS*. A partir dessa classe, foi gerado um arquivo denominado *teste.jar*. Esse arquivo é chamado através de linha de comando pelo aplicativo servidor ao invés do cexec quando o atributo *oscarSystem* possui valor *false*. O resultado da execução do arquivo *teste.jar* é semelhante a execução de um comando *cexec*, com a diferença que a classe *GeradorPS* sempre exibe uma mesma saída que é alterada de acordo com o tipo de teste desejado.

```

public static void main(String[] args) {
    if (":1".endsWith(args[0])) {
        /* Mostra uma lista de processos do nodo número 1*/
        System.out
            .println(" *****");
oscar_cluster *****");
        System.out.println(" ----- oscarnode1-----");
        System.out
            .println("  USER          PID %CPU %MEM COMMAND
STAT");
        System.out.println("  bin            1249  0.0  0.0 portmap
S");
        System.out.println("  jboss         28467  0.0  0.0 sh
S");
        System.out.println("  jboss         28475  2.0 46.4 java
S");
        System.out.println("  lp            2500  0.0  0.1 cupsd
S");
        System.out.println("  mysql         2267  0.0  0.0 mysqld
S");
        System.out.println("  postfix       2626  0.0  0.1 qmgr
S");
        System.out.println("  postfix       31044  0.0  0.5 pickup
S");
        System.out.println("  postgres     31049  0.0  1.1 postmaster
S");
        System.out.println("  postgres     31050  0.0  1.1 postmaster
S");
        System.out
            .println("  root            2  0.0  0.0 keventd
SW");
    } else {
        if (":2".endsWith(args[0])) {
            /* Mostra uma lista de processos do nodo número 1*/
            System.out
                .println(" *****");
oscar_cluster *****");
                System.out.println(" ----- oscarnode2-----
");
                System.out
                    .println("  USER          PID %CPU %MEM
COMMAND          STAT");

```

Fig. 6.16 – Trecho classe GeradorPS

6.4.1 ESTUDO DE CASO 1: PROCESSOS EM ESTADO ZOMBIE E PROCESSO INATIVO

Para o primeiro estudo de caso foi atribuído ao estado dos processos java e mysqld (nodos 1 e 2 respectivamente) o valor “Z” (processo em estado *zombie*) na classe GeradorPS e gerado o arquivo teste.jar. Os aplicativos servidor e cliente foram iniciados. Primeiro, uma lista

de processos foi solicitada pelo sistema cliente através do botão “atualizar” do formulário principal, atualizando as configurações de monitoração no sistema servidor. A lista de processos foi recebida corretamente e a monitoração de erros no aplicativo servidor foi iniciada. No aplicativo cliente foi selecionado a opção de iniciar a monitoração, habilitando o recebimento de mensagens de falha no sistema cliente. Após dez segundos as mensagens “Nodo oscarnde2 - O processo mysqld está em estado zombie” e “Nodo oscarnde1 - O processo java está em estado zombie” foram exibidas no visor do emulador conforme o esperado.

Esperou-se mais trinta segundos para verificar se as mesmas mensagens de erro seriam enviadas novamente, o que não ocorreu. Depois disso, o processo “pico” foi adicionado a configuração. Como esse processo não existe na lista de processos uma mensagem de erro era esperada, o que aconteceu segundos depois. Além da mensagem referente ao processo “pico” as duas mensagens de erro anteriores foram enviadas novamente, pois a atualização das configurações provoca a exclusão de todos os elementos da lista de mensagens enviadas.



Fig. 6.17 – Mensagens de erro nos estados dos processos “java” e “mysqld”.



Fig. 6.18 – Mensagem de erro no processo “pico”.

6.4.2 ESTUDO DE CASO 2: PROCESSO PARADO E NODO INDISPONÍVEL

Esse teste foi executado no simulador. As configurações padrão foram restabelecidas no sistema cliente, foi atribuído o valor “T” (processo parado) ao processo “mysqld” do nodo número dois e o arquivo teste.jar foi gerado novamente. A lista de configuração foi atualizada

através da interface cliente. Segundos depois a mensagem “Nodo oscarnode2 – O processo mysqld está parado” foi recebida.

Foi adicionado no cliente o nodo de número 3 (inexistente) e a lista de configurações foi enviada para o sistema servidor (botão atualizar). A mensagem “O nodo 3 está indisponível” foi exibida no visor do simulador.



Fig. 6.19 – Mensagem de erro no processo “mysqld”.

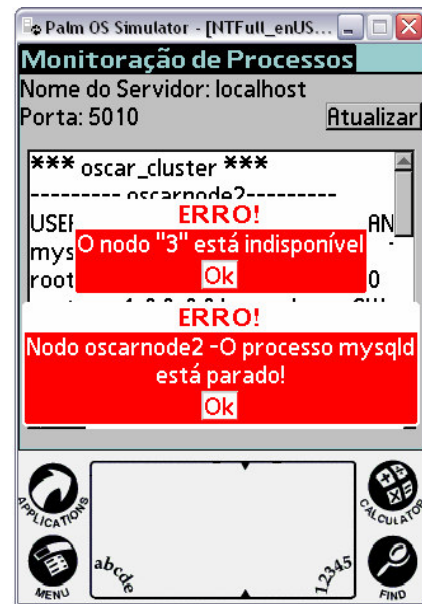


Fig. 6.20 – Mensagem de erro no nodo 3 e no processo “mysqld”.

6.4.3 ESTUDO DE CASO 3: PROBLEMAS DE COMUNICAÇÃO ENTRE SERVIDOR E CLIENTE

Nesse estudo de caso foram testados problemas de comunicação entre o sistema servidor e cliente. Esses problemas podem ocorrer quando o dispositivo móvel estiver fora da área de cobertura da rede sem fio, quando houver problemas na rede ou quando o cluster estiver em estado de falha ou desligado.

Para realizar o teste o aplicativo servidor foi parado. Foi solicitada a lista de informação de processos do cluster pela interface do POSE. A mensagem “Erro ao tentar ler lista de processos do cluster” foi exibida e as informações anteriores da caixa de listagem foram excluídas. Logo após, o item “Iniciar Monitoração” da barra de menu foi selecionado. Segundos depois a mensagem “Erro ao ler mensagem do servidor” foi exibida. Esperou-se um minuto para verificar se a mensagem de erro seria exibida novamente, o que não ocorreu.

O aplicativo servidor foi reiniciado, o sistema cliente voltou a funcionar normalmente e a mensagem de informação “Conexão com o servidor restabelecida” foi exibida.



Fig. 6.21 – Cluster indisponível, erro ao ler lista de processos.



Fig. 6.22 – Cluster indisponível, erro ler mensagens do servidor.



Fig. 6.23 – Conexão com servidor restabelecida.

7. CONCLUSÕES E TRABALHOS FUTUROS

As possíveis aplicações de sistemas de alto desempenho e disponibilidade são bastante amplas e ambientes de cluster estão se tornando cada vez mais populares. Os clusters possibilitam a criação de sistemas de alto desempenho de alta disponibilidade a custos relativamente baixos se comparados a soluções específicas.

No decorrer deste trabalho observamos a importância da monitoração não só no nível de hardware como também dos serviços disponibilizados pelo ambiente de alto desempenho; esta monitoração é viabilizada através da monitoração dos processos críticos executados no ambiente.

Encontramos dificuldades na implementação do sistema cliente para o PDA, devido a restrições de memória, armazenamento e processamento a tarefa de desenvolvimento de software para esses dispositivos se revelou complexa. Durante toda a etapa de desenvolvimento testes foram realizados com o objetivo de melhorar o desempenho do sistema e viabilizar seu funcionamento no Palm.

Os objetivos iniciais foram alcançados, porém percebemos que ainda é possível incrementar as funcionalidades do sistema apresentado nesse trabalho. Como sugestão para trabalhos futuros, seria interessante a implementação de mecanismos para correções automáticas dos erros mais comuns encontrados no ambiente. Mesmo assim, a presença da figura humana é essencial para o funcionamento da aplicação e manutenção do cluster.

REFERÊNCIAS BIBLIOGRÁFICAS

- [ABR98] ABRAMS, Marc. *World Wide Web – beyond the basics*. Prentice Hall, 1998. Disponível em <http://ei.cs.vt.edu/~wwwbtb/book/index.html>.
- [AND81] ANDERSEN, T.; LEE, P. A. *Fault tolerance principles and practices*. Prentice Hall, 1981.
- [BAG04] BAGGIO, R. K.; DANTAS, Mário A. R. *Uma Ferramenta para Monitoração e alertas SMS em um Ambiente de Cluster de Alta Disponibilidade*. 2004
- [BUD03] BUDRI, Amaury; BONILHA, Caio. *Wireless LAN (WLAN)*. 2003.
- [CON05] Conectiva. *Guia do Servidor Conectiva Linux*. Disponível em <http://www.conectiva.com/doc/livros/online/9.0/servidor/ha.html>. Acessado em Setembro 2005.
- [COS04] COSTA, Marcelo José Santana. *Banco de Dados Móvel para Controle de Atividades Acadêmicas*. 2004. Disponível em <http://www.cci.unama.br/margalho/portaltcc/tcc2004>. Acessado em Outubro 2005.
- [DAN02] DANTAS, Mário A. Ribeiro. *Tecnologia de redes de comunicação e computadores*. Rio de Janeiro: Axcel Books, 2002. 328 p.
- [DAN05] DANTAS, Mário A. Ribeiro. *Computação Distribuída e de Alto Desempenho*. Rio de Janeiro: Axcel Books, 2005. 278 p.
- [HAO05] HA-OSCAR. *High Availability Open Source Cluster Application Resources*. Disponível em <http://xcr.cenit.latech.edu/ha-oscar/index.html>. Acessado em Outubro 2005.
- [LAM81] LAMPSON, B. W. *Atomic transactions, Lecture Notes in Computer Science*. 1981.
- [MAR02] MARTINS, Marcelo. *Protegendo Redes Wireless 802.11b*. 2002. Disponível em http://www.modulo.com.br/pdf/wireless_mmartins.pdf. Acessado em Setembro 2004.
- [OME04] OMÊNA, Moisés. *Clusters e Supercomputação*. Disponível em <http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=1362>, 2004. Acessado em Setembro 2005.
- [OSC05] OSCAR. *Open Source Cluster Application Resources*. Disponível em <http://oscar.openclustergroup.org/>. Acessado em Outubro 2005.

[PAL03] PalmSource. *Using Palm OS Emulator*. 2003. Disponível em <http://www.palmos.com/dev/support/docs/emulator.pdf>. Acessado em Setembro 2005.

[PER04] PEREIRA, Nélio Alves Filho. *Serviços de Pertinência para Clusters de Alta Disponibilidade*. 2004. Disponível em <http://www.ime.usp.br/~nelio/mestrado/>.

[PIT03] PITANGA, Marcos. *Computação em Cluster*. 2003. Disponível em <http://www.clubedohardware.com.br/artigos/153/>. Acessado em Agosto 2005.

[RIS04] RISTA, C.; PINTO A. R.; DANTAS M. A. R. *OSCAR: Um Gerenciador de Agregado para Ambiente Operacional Linux*. 2004.

[SUN05] Sun Microsystems. *Java Technology*. Disponível em <http://java.sun.com/>. Acessado em Outubro 2005.

[SUC05] SuperWaba. *The SuperWaba Companion*. Disponível em http://www.superwaba.com.br/etc/SuperWaba_Companion_GPL.pdf. Acessado em Setembro 2005.

[SUP05] SuperWaba Web site. Disponível em <http://www.superwaba.com.br>. Acessado em Outubro 2005.

[TAN03] TANENBAUM, Andrew S. *Redes de Computadores*. Rio de Janeiro: Editora Campos, 2003. 968p.

ANEXO I – CÓDIGO FONTE

I.I - APLICATIVO PALM

Pacote br.ufsc.ine5328.monitor.cliente

ContainerConfigNodos.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.sys.Settings;
import waba.sys.Vm;
import waba.ui.Button;
import waba.ui.Container;
import waba.ui.ControlEvent;
import waba.ui.Edit;
import waba.ui.Event;
import waba.ui.ListBox;
import waba.ui.MainWindow;
import waba.util.Vector;

/**
 * Interface de configuração de nodos a monitorar.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class ContainerConfigNodos extends Container {
    private Button btVoltar = new Button("Back");

    private Button btAdicionar = new Button("Add");

    private Button btExcluir = new Button("Delete");

    private Button btTeclado = new Button("Keyboard");

    private Edit editAddNodo = new Edit("0000000000000000");

    private ListBox listaNodos = new ListBox();

    protected void onStart() {
        Vm.debug("ContainerConfigServidor - onStart()");
        setBorderStyle(BORDER_LOWERED);
        this.setFonts();
        this.initListaNodos();
        this.add(editAddNodo, LEFT + 5, TOP + 5);
        this.add(btAdicionar, AFTER + 2, SAME);
        this.add(btTeclado, LEFT + 5, AFTER + 5);
        this.add(listaNodos);
        Vm.debug("Altura resolucao: " + Settings.screenHeight);
        int altura = Settings.screenHeight / 3;
        listaNodos.setRect(LEFT + 5, AFTER + 15, FILL - 10, altura);
        this.add(btVoltar, Container.CENTER - 30, Container.AFTER + 5);
        this.add(btExcluir, Container.AFTER + 5, Container.SAME);
    }
}

```

```

}

private void setFonts() {
    this.editAddNodo.setFont(Controle.FONTE_PADRAO);
    this.btAdicionar.setFont(Controle.FONTE_PADRAO);
    this.btExcluir.setFont(Controle.FONTE_PADRAO);
    this.btVoltar.setFont(Controle.FONTE_PADRAO);
    this.listaNodos.setFont(Controle.FONTE_PADRAO);
}

private void initListaNodos() {
    Controle controle = new Controle();
    Vector resp = controle.leiaNodos();
    if (resp.size() > 0) {
        listaNodos.add(resp.toArray());
    }
}

public void onEvent(Event event) {
    if (event.type == ControlEvent.PRESSED) {
        if (event.target == btVoltar)
            MainWindow.getMainWindow().swap(null);
        else if (event.target == btAdicionar)
            adicioneNodo();
        else if (event.target == btExcluir)
            excluaNodo();
        else if (event.target == btTeclado)
            editAddNodo.popupKCC();
    }
}

private void excluaNodo() {
    Vm.debug("Exclua Nodo: indice= " +
this.listaNodos.getSelectedIndex());
    int indice = this.listaNodos.getSelectedIndex();
    if (indice >= 0) {
        Controle controle = new Controle();
        boolean excluiu = controle.excluaNodo(indice);
        if (excluiu) {
            this.listaNodos.remove(indice);
            this.listaNodos.repaint();
        } else
            Principal.mostragemensagem(Controle.LABEL_ERRO,
Controle.ERRO_EXCLUIR);
    } else
        Principal.mostragemensagem(Controle.LABEL_ERRO,
Controle.ERRO_NAO_SELECIONADO);
}

private void adicioneNodo() {
    String nomeNodo = this.editAddNodo.getText().trim();
    Vector aux = new Vector(this.listaNodos.getItems());
    if (!nomeNodo.equals("") && aux.find(nomeNodo) == -1) {
        Controle controle = new Controle();
        controle.salveNodo(nomeNodo);
        this.listaNodos.add(nomeNodo);
        this.listaNodos.repaint();
    }
}

```

```

    }
}
}

```

ContainerConfigPrincipal.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.io.Catalog;
import waba.io.DataStream;
import waba.sys.Settings;
import waba.sys.Vm;
import waba.ui.Button;
import waba.ui.Container;
import waba.ui.ControlEvent;
import waba.ui.Event;
import waba.ui.Label;
import waba.ui.MainWindow;
import waba.ui.MessageBox;
import waba.ui.Window;
import br.ufsc.ine5328.monitor.ui.HorizontalListBox;

/**
 * Interface principal que mostra as listas dos processos
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class ContainerPrincipal extends Container {

    public Label lNomeServidor = new Label("Server Name:");

    public Label lPortaServidor = new Label("Port:");

    private Button botaoAtualizar = new Button("Refresh");

    private HorizontalListBox listaProcessos = new
HorizontalListBox(this);

    private Window principal;

    public ContainerPrincipal() {
        super();
        this.principal = MainWindow.getMainWindow();
    }

    private void setFonts() {
        this.lNomeServidor.setFont(Controle.FONTE_PADRAO);
        this.lPortaServidor.setFont(Controle.FONTE_PADRAO);
        this.botaoAtualizar.setFont(Controle.FONTE_PADRAO);
        this.listaProcessos.setFont(Controle.FONTE_PADRAO);
    }

    public void onStart() {
        this.setFonts();
        /*

```

```

        * Isso é necessário porque os "containers" são transparentes
toda vez
        * que uma outra janela (na verdade mesmo objeto, container
diferente)
        * retorna para a principal fica com seus objetos na tela.
        */
        initInterface();
    }

    private void initInterface() {
        lNomeServidor.setText("Server Name: " +
Control.NOME_SERVIDOR);
        lPortaServidor.setText("Port: " + Control.PORTA);
        this.add(lNomeServidor, Container.LEFT, Container.TOP);
        this.add(lPortaServidor, Container.SAME, Container.AFTER);
        this.add(botaoAtualizar, Container.RIGHT, Container.SAME);
        this.add(this.listaProcessos);
        listaProcessos.setRect(LEFT + 5, AFTER + 15, FILL - 5, FILL -
25);
    }

    public void onEvent(Event event) {

        if (event.type == ControlEvent.PRESSED) {
            if (event.target == this.botaoAtualizar) {
                Vm.debug("btAtualizar pressionado");
                atualizeListaProcessos();
            }
        }
    }

    private void atualizeListaProcessos() {
        Vm.debug("Horizontal: "
            + String.valueOf(listaProcessos
                .getNeededHorizontalScrollValue()));
        this.listaProcessos.removeAll();
        Control controle = new Control();
        this.listaProcessos.add(controle.getListaProcessos());
        this.listaProcessos.repaint();
    }

    public void executeTeste() {
        this.principal.popupModal(new MessageBox("HI", "Teste"));
        Catalog catalogo = new Catalog("rt." + Settings.appCreatorId +
".DATA",
            Catalog.CREATE);
        int pos = catalogo.addRecord(8);

        Vm.debug("Posicao catalogo: " + String.valueOf(pos));
        DataStream ds = new DataStream(catalogo);
        ds.writeString("edu");
        catalogo.setRecordPos(0);
        Vm.debug("Valor: " + ds.readString());
        catalogo.close();
    }

```

```

    public void formConfigServidor() {
        Vm.debug("formConfigServidor");
        // if (this.configServidor==null)
        ContainerConfigServidor configServidor = new
ContainerConfigServidor();
        // this.setVisible(false);
        // this.swap(principal);
        principal.swap(configServidor); /*
Muda o formulário apresentado para
para o formulário de configuração do
servidor
*/
    }

    public void formNodo() {
        Vm.debug("formFiltrarNodo");
        ContainerConfigNodos configNodos = new ContainerConfigNodos();
        this.principal.swap(configNodos);
    }

    public void formProcesso() {
        Vm.debug("formFiltrarProcesso");
        ContainerConfigProcessos configProcessos = new
ContainerConfigProcessos();
        this.principal.swap(configProcessos);
    }
}

```

ContainerConfigProcessos.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.sys.Settings;
import waba.sys.Vm;
import waba.ui.Button;
import waba.ui.Container;
import waba.ui.ControlEvent;
import waba.ui.Edit;
import waba.ui.Event;
import waba.ui.ListBox;
import waba.ui.MainWindow;
import waba.util.Vector;

/**
 * Interface de configuração de processos a monitorar.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class ContainerConfigProcessos extends Container {
    private Button btVoltar = new Button("Back");
}

```

```

private Button btAdicionar = new Button("Add");

private Button btExcluir = new Button("Delete");

private Button btTeclado = new Button("Keyboard");

private Edit editAddProcesso = new Edit("0000000000000000");

private ListBox listaProcessos = new ListBox();

protected void onStart() {
    Vm.debug("ContainerConfigServidor - onStart()");
    setBorderStyle(BORDER_LOWERED);
    this.initListaProcessos();
    this.setFonts();
    this.add(editAddProcesso, LEFT + 5, TOP + 5);
    this.add(btAdicionar, AFTER + 2, SAME);
    this.add(btTeclado, LEFT + 5, AFTER + 5);
    this.add(listaProcessos);
    Vm.debug("Altura resolucao: " + Settings.screenHeight);
    int altura = Settings.screenHeight / 3;
    listaProcessos.setRect(LEFT + 5, AFTER + 15, FILL - 10,
altura);

    this.add(btVoltar, Container.CENTER - 30, Container.AFTER + 5);
    this.add(btExcluir, Container.AFTER + 5, Container.SAME);
}

private void setFonts() {
    this.editAddProcesso.setFont(Controle.FONTE_PADRAO);
    this.btAdicionar.setFont(Controle.FONTE_PADRAO);
    this.btExcluir.setFont(Controle.FONTE_PADRAO);
    this.btVoltar.setFont(Controle.FONTE_PADRAO);
    this.listaProcessos.setFont(Controle.FONTE_PADRAO);
}

private void initListaProcessos() {
    Controle controle = new Controle();
    Vector processos = controle.leiaProcessos();
    if (processos.size() > 0) {
        listaProcessos.add(processos.toArray());
    }
}

public void onEvent(Event event) {
    if (event.type == ControlEvent.PRESSED) {
        if (event.target == btVoltar)
            MainWindow.getMainWindow().swap(null);
        else if (event.target == btAdicionar)
            adicioneProcesso();
        else if (event.target == btExcluir)
            excluaProcesso();
        else if (event.target == btTeclado)
            editAddProcesso.popupKCC();
    }
}

private void excluaProcesso() {

```

```

Vm.debug("Exclua Processo: indice= "
        + this.listaProcessos.getSelectedIndex());
int indice = this.listaProcessos.getSelectedIndex();
if (indice >= 0) {
    Controle controle = new Controle();
    boolean excluiu = controle.excluaProcesso(indice);
    if (excluiu) {
        this.listaProcessos.remove(indice);
        this.listaProcessos.repaint();
    } else
        Principal.mostraMensagem(Controle.LABEL_ERRO,
                                   Controle.ERRO_EXCLUIR);
} else
    Principal.mostraMensagem(Controle.LABEL_ERRO,
                              Controle.ERRO_NAO_SELECIONADO);
}

private void adicioneProcesso() {
    String nomeProcesso = this.editAddProcesso.getText().trim();
    Vector aux = new Vector(this.listaProcessos.getItems());
    if (!nomeProcesso.equals("") && aux.find(nomeProcesso) == -1) {
        Controle controle = new Controle();
        controle.salvaProcesso(nomeProcesso);
        this.listaProcessos.add(nomeProcesso);
        this.listaProcessos.repaint();
    }
}
}
}

```

ContainerConfigServidor.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.sys.Convert;
import waba.sys.Vm;
import waba.ui.Button;
import waba.ui.Container;
import waba.ui.ControlEvent;
import waba.ui.Edit;
import waba.ui.Event;
import waba.ui.Label;
import waba.ui.MainWindow;

/**
 * Interface de configuração de parâmetros do servidor.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class ContainerConfigServidor extends Container {
    private Label lNomeServidor = new Label("Server Name:");

    private Label lPortaServidor = new Label("Port:");

    private Label lIntervaloMsg = new Label("Monitoring Interval(Sec):");

    private Edit editNomeServidor = new Edit("0000000000000000");

```



```

private Edit editPortaServidor = new Edit("0000");

private Edit editIntervaloMsg = new Edit("0000");

private Button btSalvar = new Button("Save");

private Button btVoltar = new Button("Back");

private Button btTecladoServidor = new Button("Keyboard");

private Button btTecladoPorta = new Button("Keyboard");

private Button btTecladoIntervalo = new Button("Keyboard");

protected void onStart() {
    Vm.debug("ContainerConfigServidor - onStart()");
    setBorderStyle(BORDER_LOWERED);
    this.setFonts();

    this.add(lNomeServidor, Container.LEFT, Container.TOP);
    this.add(editNomeServidor, Container.SAME, Container.AFTER +
3);
    this.add(btTecladoServidor, Container.AFTER + 10,
Container.SAME);
    this.add(lPortaServidor, Container.LEFT, Container.AFTER + 10);
    this.add(editPortaServidor, Container.LEFT, Container.AFTER +
3);
    this.add(btTecladoPorta, Container.AFTER + 10, Container.SAME);
    this.add(lIntervaloMsg, Container.LEFT, Container.AFTER + 10);
    this.add(editIntervaloMsg, Container.LEFT, Container.AFTER +
3);
    this.add(btTecladoIntervalo, Container.AFTER + 10,
Container.SAME);

    this.add(btSalvar, Container.CENTER - 20, Container.AFTER +
10);
    this.add(btVoltar, Container.AFTER + 3, Container.SAME);

    initPropriedades();
}

private void initPropriedades() {
    this.editNomeServidor.setText(Controle.NOME_SERVIDOR);

    this.editPortaServidor.setText(Convert.toString(Controle.PORTA));
    this.editIntervaloMsg.setText(Convert
        .toString(Controle.INTERVALO_MONITOR));
}

public void onEvent(Event event) {
    if (event.type == ControlEvent.PRESSED) {
        if (event.target == btVoltar) {
            // this.setBackgroundColor(Color.defaultBackColor);
            MainWindow.getMainWindow().swap(null);
        } else if (event.target == btSalvar)
            this.salveNovaConfiguracao();
    }
}

```

```

        else if (event.target == btTecladoServidor)
            this.editNomeServidor.popupKCC();
        else if (event.target == btTecladoPorta)
            this.editPortaServidor.popupKCC();
        else if (event.target == btTecladoIntervalo)
            this.editIntervaloMsg.popupKCC();
    }
}

private void setFonts() {
    this.editNomeServidor.setFont(Controle.FONTE_PADRAO);
    this.editPortaServidor.setFont(Controle.FONTE_PADRAO);
    this.lNomeServidor.setFont(Controle.FONTE_PADRAO);
    this.lPortaServidor.setFont(Controle.FONTE_PADRAO);
    this.btSalvar.setFont(Controle.FONTE_PADRAO);
    this.btVoltar.setFont(Controle.FONTE_PADRAO);
}

private void salveNovaConfiguracao() {
    Vm.debug("salveNovaConfiguracao");
    if (!"".equals(this.editNomeServidor.getText().trim())
        &&
        !"".equals(this.editPortaServidor.getText().trim())
        &&
        !"".equals(this.editIntervaloMsg.getText().trim())) {
        boolean resp = Controle
            .salveConfigServidor(

                this.editNomeServidor.getText().trim(), Convert
                    .toInt(this.editPortaServidor.getText()),

                Convert.toInt(this.editIntervaloMsg.getText()));
        Principal principal = (Principal)
            MainWindow.getMainWindow();

        principal.containerPrincipal.lNomeServidor.setText("Serv: "
            + Controle.NOME_SERVIDOR);

        principal.containerPrincipal.lPortaServidor.setText("Porta: "
            + Controle.PORTA);
        Vm.debug("Salvou config? " + resp);
    } else
        Principal.mostraMensagem(Controle.LABEL_ERRO,
            Controle.ERRO_CAMPO_NULO);
}
}
}

```

Controle.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.fx.Font;
import waba.io.Catalog;
import waba.io.DataStream;
import waba.sys.Settings;
import waba.sys.Vm;
import waba.util.Vector;

/**
 * Implementa a lógica principal do Sistema do PDA. É responsável pela leitura
 e
 * armazenamento das listas de processos e de nodos em arquivos, além de
 outras configurações do servidor e do sistema.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class Controle {
    /** Ítems que compõem a barra de menu principal da interface do
 sistema */
    public static final String[][] MENU_PALM = {
        { "File", "Exit" },
        { "Management", "Process", "Node", "Start Monitor",
          "Stop Monitor" }, { "Config", "Server" }
    };

    /** Fonte padrão usada nos componentes de interface */
    public static final Font FONTE_PADRAO = new Font("Times", Font.PLAIN,
10); // Tiny

    /** Nome ou endereço IP do cluster */
    public static String NOME_SERVIDOR = "localhost";

    /** Porta de conexão com o cluster */
    public static int PORTA = 5050;

    /**
     * Intervalo de tempo em segundos que o sistema conecta ao servidor
 para ler
     * possíveis mensagens de erro.
     */
    public static int INTERVALO_MONITOR = 30;

    /** Tipo dos arquivos de dados do sistema */
    private static final String TIPO_ARQUIVO = ".DATA";

    /** Nome do arquivo de dados do servidor */
    private static final String ARQUIVO_SERVIDOR = "confServidor.";

    /** Nome do arquivo com a lista de nodos a monitorar */
    private static final String ARQUIVO_NODO = "confNodo.";

    /** Nome do arquivo com a lista de processos a monitorar */
    private static final String ARQUIVO_PROCESSO = "confProcesso.";

```

```

    /** Mensagem de erro */
    public static final String LABEL_ERRO = "Error!";

    /** Descrição de mensagem de erro de exclusão */
    public static final String ERRO_EXCLUIR = "O item não pôde ser
excluído!";

    /** Mensagem de erro que avisa ao usuário que é necessário selecionar
um item */
    public static final String ERRO_NAO_SELECIONADO = "Selecione um
item!";

    /**
     * Mensagem de erro ocasionada quando o usuário não preenche campos
     * obrigatórios
     */
    public static final String ERRO_CAMPO_NULO = "Existem campos em
branco!";

    /** Tempo de espera por resposta do cluster em milisegundos */
    public static final int TIMEOUT = 20000; // Timoeout em ms.

    /**
     * Lê as configurações referentes ao cluster salvas em arquivo.
     * inicializa
     * as variáveis globais NOME_SERVIDOR, PORTA e INTERVALO_MONITOR com
os
     * valores salvos em arquivo.
     */
    public static void leiaConfigServidor() {
        Catalog arquivoServ = new Catalog(ARQUIVO_SERVIDOR
            + Settings.appCreatorId + TIPO_ARQUIVO,
Catalog.CREATE);
        /* Posição 0 corresponde ao nome do servidor */
        arquivoServ.setRecordPos(0);
        DataStream dataStream = new DataStream(arquivoServ);
        String nomeServ = dataStream.readString();
        if (nomeServ.length() > 0) {
            /*
             * Se entrar no if significa que o arquivo contém
dados. Caso
             * contrário o arquivo não existe ou não contém dados.
             */
            NOME_SERVIDOR = nomeServ;
            arquivoServ.setRecordPos(1);
            PORTA = dataStream.readInt();
            arquivoServ.setRecordPos(2);
            INTERVALO_MONITOR = dataStream.readInt();
        }
        dataStream.close();
        arquivoServ.close();
    }

    /**
     * Salva alterações nas configurações do servidor. Atualiza as
variáveis
     * globais NOME_SERVIDOR, PORTA e INTERVALO_MONITOR.

```

```

*
* @param nomeServ
*     Nome ou endereço IP do Cluster.
* @param porta
*     Porta de conexão com o servidor
* @param intervalo
*     Intervalo de tempo em segundos que o sistema conecta ao
*     servidor para ler possíveis mensagens de erro.
* @return Returns true se a configuração foi salva com sucesso. false
se
*     houve algum erro na escrita do arquivo.
*/
public static boolean salveConfigServidor(String nomeServ, int porta,
    int intervalo) {
    NOME_SERVIDOR = nomeServ;
    PORTA = porta;
    INTERVALO_MONITOR = intervalo;
    Catalog arquivoServ = new Catalog (ARQUIVO_SERVIDOR
        + Settings.appCreatorId + TIPO_ARQUIVO,
Catalog.CREATE);
    if (arquivoServ.getRecordCount() == 0) {
        /* Nome do servidor */
        arquivoServ.addRecord(30);
        /* Porta */
        arquivoServ.addRecord(5);
        /* Intervalo entre monitorações (Segundos) */
        arquivoServ.addRecord(6);
    }
    boolean aux = arquivoServ.setRecordPos(0);
    if (!aux)
        return false;
    DataStream ds = new DataStream(arquivoServ);
    int bytesEscritos = ds.writeString(NOME_SERVIDOR);
    if (bytesEscritos <= 0)
        return false;
    arquivoServ.setRecordPos(1);
    bytesEscritos = ds.writeInt(PORTA);
    if (bytesEscritos <= 0)
        return false;
    arquivoServ.setRecordPos(2);
    bytesEscritos = ds.writeInt(intervalo);
    if (bytesEscritos <= 0)
        return false;
    ds.close();
    arquivoServ.close();
    return true;
}

/**
* Salva o nome do processo a monitorar no arquivo de processos
*
* @param nomeProcesso
*     Nome do processo a ser salvo
*
* @return Returns true se o processo foi salvo. false se houve algum
erro

```

```

    */
    public boolean salveProcesso(String nomeProcesso) {
        Catalog arquivoProcesso = new Catalog(ARQUIVO_PROCESSO
            + Settings.appCreatorId + TIPO_ARQUIVO,
Catalog.CREATE);
        boolean resp = escrevaStringArquivo(nomeProcesso,
arquivoProcesso);
        arquivoProcesso.close();
        return resp;
    }

    /**
     * Adiciona um String qualquer em um arquivo.
     *
     * @param valorCampo
     *         Valor a ser escrito.
     * @param arquivo
     *         Arquivo no qual o valor deve ser adicionado.
     *
     * @return Returns true se escreveu no arquivo. false se não foi
possível
     *         escrever
     */
    private boolean escrevaStringArquivo(String valorCampo, Catalog
arquivo) {
        valorCampo.length();
        Vm.debug("EscrevaStringarquivo " + valorCampo + " "
            + valorCampo.length());
        /* bug com campo do mesmo tamanho que valor */
        int pos = arquivo.addRecord(valorCampo.length() + 4);
        if (pos < 0)
            return false;
        DataStream ds = new DataStream(arquivo);
        int qtdeEscritos = ds.writeString(valorCampo);
        if (qtdeEscritos < 0)
            return false;
        ds.close();
        return true;
    }

    /**
     * Salva o nome do nodo a monitorar no arquivo de nodos
     *
     * @param nomeNodo
     *         Nome do nodo a ser salvo
     *
     * @return Returns true se o nodo foi salvo. false se houve algum erro
     */
    public boolean salveNodo(String nomeNodo) {
        Catalog arquivoNodo = new Catalog(ARQUIVO_NODO +
Settings.appCreatorId
            + TIPO_ARQUIVO, Catalog.CREATE);
        boolean resp = escrevaStringArquivo(nomeNodo, arquivoNodo);
        arquivoNodo.close();
        return resp;
    }

```

```

/**
 * Lê todos os campos de um arquivo(Catálogo)
 *
 * @param arquivo
 *         arquivo com a lista de strings
 *
 * @return Returns uma coleção(Vector) de String
 */
private Vector leiaStringsArquivo(Catalog arquivo) {
    int contador = arquivo.getRecordCount();
    Vector resp = new Vector();
    DataStream ds = new DataStream(arquivo);
    for (int i = 0; i < contador; i++) {
        arquivo.setRecordPos(i);
        String aux = new String(ds.readString());
        Vm.debug("STRINGP: " + aux);
        resp.add(aux);
    }
    ds.close();
    return resp;
}

/**
 * Lê os nodos do arquivo de nodos
 *
 * @return Returns uma coleção(Vector) de String
 */
public Vector leiaNodos() {
    Catalog arquivoNodo = new Catalog(ARQUIVO_NODO +
Settings.appCreatorId
        + TIPO_ARQUIVO, Catalog.CREATE);
    Vector resp = leiaStringsArquivo(arquivoNodo);
    arquivoNodo.close();
    return resp;
}

/**
 * Lê os processo do arquivo de processos
 *
 * @return Returns uma coleção(Vector) de Processos
 */
public Vector leiaProcessos() {
    Catalog arquivoProcesso = new Catalog(ARQUIVO_PROCESSO
        + Settings.appCreatorId + TIPO_ARQUIVO,
Catalog.CREATE);
    Vector resp = leiaStringsArquivo(arquivoProcesso);
    arquivoProcesso.close();
    return resp;
}

/**
 * Lê os arquivos de configuração da aplicação invocados na
inicialização
 * do sistema
 */
public static void leiaArquivosConfiguracao() {
    Controle.leiaConfigServidor();
}

```

```

}

/**
 * Exclui um processo do arquivo de processos
 *
 * @param indice
 *         posição do campo com o nome do processo no arquivo de
 *         processos
 *
 * @return Returns true se o processo foi excluído. false se não foi
 *         possível excluir o processo
 */
public boolean excluaProcesso(int indice) {
    boolean excluiu = false;
    if (indice >= 0) {
        Catalog arquivoProcesso = new Catalog(ARQUIVO_PROCESSO
            + Settings.appCreatorId + TIPO_ARQUIVO,
Catalog.READ_WRITE);
        arquivoProcesso.setRecordPos(indice);
        excluiu = arquivoProcesso.deleteRecord();
        arquivoProcesso.close();
    }
    return excluiu;
}

/**
 * Exclui um nodo do arquivo de nodos
 *
 * @param indice
 *         posição do campo com o nome do nodo no arquivo de nodos
 *
 * @return Returns true se o nodo foi excluído. false se não foi
 *         possível
 *         excluir o nodo
 */
public boolean excluaNodo(int indice) {
    boolean excluiu = false;
    if (indice >= 0) {
        Catalog arquivoNodo = new Catalog(ARQUIVO_NODO
            + Settings.appCreatorId + TIPO_ARQUIVO,
Catalog.READ_WRITE);
        arquivoNodo.setRecordPos(indice);
        excluiu = arquivoNodo.deleteRecord();
        arquivoNodo.close();
    }
    return excluiu;
}

/**
 * Retorna um array de String contendo uma lista formatada de
 * processos que
 * estão rodando no cluster, a lista é filtrada de acordo com os nodos
 * e os
 * processos que o usuário deseja monitorar. As informações da lista
 * de
 * processos como memória, utilização de cpu e etc são de
 * responsabilidade

```



```

    * do servidor.
    *
    * @return Returns Um array contendo uma lista de informações de
processos
    *         monitorados pelo Cluster.
    */
public String[] getListaProcessos() {
    Vector processos = this.leiaProcessos();
    String processosMsg = "";
    for (int i = 0; i < processos.size(); i++) {
processos.items[i];
        processosMsg = processosMsg + ", " +

        if (processosMsg.length() > 0) {
            processosMsg = processosMsg.substring(1);
        }
        MensagemCliente mensagemComando = new MensagemCliente();

        mensagemComando.setCodigo(mensagemComando.CODIGO_ALTERA_COMANDO);
        Vector conteudo = this.leiaNodos();
        Object aux = conteudo.items[0];
        conteudo.items[0] = processosMsg;
        conteudo.add(aux);
        mensagemComando.setColecaoConteudo(conteudo);
        mensagemComando.leiaMensagem();

        MensagemCliente mensagemSolicitacao = new MensagemCliente();
        mensagemSolicitacao

        .setCodigo(mensagemSolicitacao.CODIGO_SOLICITACAO_LISTA_PROCESSOS);
        mensagemSolicitacao.setColecaoConteudo(new Vector());
        if (mensagemSolicitacao.leiaMensagem()) {
            Vector retorno =
mensagemSolicitacao.getColecaoConteudo();
            if (retorno != null) {
                int i=1;
                int tam = retorno.size();
                while (i<tam) {
                    String linha = retorno.items[i].toString();
                    if (linha.length()>2 &&
"***".equals(linha.substring(0,3))) {
                        retorno.insert(i, " ");
                        tam = retorno.size();
                        i++;
                    }
                    i++;
                }
            }
            return (String[]) retorno
                .toArray();
        }
        Principal.mostraMensagem(Controle.LABEL_ERRO,
            "Error on read process list in the cluster!");
        String[] resp = { "" };
        return resp;
    }
}

```

```
}
```

MensagemCliente.java

```
package br.ufsc.ine5328.monitor.cliente;

import waba.io.Socket;
import waba.sys.Convert;
import waba.sys.Vm;
import waba.util.Vector;

/**
 * Trata o envio e o recebimento de mensagens vindas do servidor para o palm.
 *
 * O
 * atributo colecaoConteudo deve ser inicializados com string's que serão
 * enviadas no conteúdo da mensagem separadas por uma String
 * especial (FIM_CONTEUDO).
 *
 * A mensagem é composta por um cabeçalho (Header) e pelo conteúdo. O cabeçalho
 * possui tamanho fixo TAMANHO_COD_HEADER + TAMANHO_ARRAY_HEADER. O conteúdo
 * da
 * mensagem possui tamanho variável e corresponde ao tamanho indicado em
 * TAMANHO_ARRAY_HEADER.
 *
 * Ex: Para escrever uma mensagem MensagemPalm msg = new MensagemPalm();
 * msg.setCodigo(1); msg.escrevaMensagem();
 *
 * Para ler MensagemPalm msg = new MensagemPalm();
 * msg.leiaMensagem(CODIGO_SOLICITACAO_ERROS);
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class MensagemCliente {
    /** Código identificador da mensagem */
    private int codigo;

    /**
     * Conteúdo da mensagem, cada índice do vetor será separado pela String
     * especial FIM_CONTEUDO.
     */
    private Vector colecaoConteudo = new Vector();

    /**
     * Código da mensagem que solicita ao servidor informações sobre
     * possíveis
     * erros com processos monitorados
     */
    public final int CODIGO_SOLICITACAO_ERROS = 1;

    public final int CODIGO_RESPOSTA_ERROS = 2;

    public final int CODIGO_SOLICITACAO_LISTA_PROCESSOS = 3;

    public final int CODIGO_RESPOSTA_LISTA_PROCESSOS = 4;

    public final int CODIGO_ALTERA_COMANDO = 5;
}
```

```

public final int CODIGO_RESPOSTA_ALTERA_COMANDO = 6;

/**
 * Número de caracteres do código da mensagem
 */
public final int TAMANHO_COD_HEADER = 3;

/**
 * Número de caracteres do código da mensagem indicam o tamanho do
conteúdo
 * da mensagem.
 */
public final int TAMANHO_ARRAY_HEADER = 10;

/**
 * String de separação entre conteúdos diferentes de uma mesma
mensagem.
 * Deve sempre aparecer no fim de cada conteúdo
 */
public final String FIM_CONTEUDO = "";

/**
 * Adiciona 0s à esquerda caso o valor do código possua menos dígitos
do
 * que o definido em TAMANHO_COD_HEADER
 *
 * @param codigo
 *         Código da mensagem
 */
private String getHeader(int codigo) {
    String resp = Convert.toString(codigo);
    int diferenca = TAMANHO_COD_HEADER - resp.length();
    for (int i = 0; i < diferenca; i++) {
        resp = "0" + resp;
    }
    return resp;
}

/**
 * Lê o conteúdo de uma mensagem e inicializa o atributo
colecãoConteúdo
 *
 * @param msgConteúdo
 *         Conteúdo da mensagem a ser lida
 */
private void initConteúdo(String msgConteúdo) {
    this.colecaoConteúdo = new Vector();
    while (msgConteúdo.length() > 0) {
        int indice = msgConteúdo.indexOf(FIM_CONTEUDO);

        this.colecaoConteúdo.addElement(msgConteúdo.substring(0, indice));
        msgConteúdo = msgConteúdo.substring(indice + 1);
    }
}

/**

```

```

objeto      * Solicita um mensagem ao servidor e inicializa os atributos do
            * MensagemPalm com o contedo da mensagem de resposta do servidor.
            *
            * @param codigoMsg
            *           Código da mensagem que ser solicitada
            * @return Returns true se foi possível ler a mensagem. false se houve
erro        *           na leitura ou a mensagem de resposta não foi recebida
            */
public boolean leiaMensagem() {
    Socket socket = new Socket(Controle.NOME_SERVIDOR,
Controle.PORTA);
    socket.refreshBeforeEachRead = true;
    boolean resp = false;
    if (socket.isOpen()) {
        /* Primeiro envia a solicitacao para o servidor */
        boolean enviou = envieMsgSolicitacao(codigo, socket);
        if (enviou) {
            byte[] arrayHeader = new
byte[TAMANHO_ARRAY_HEADER
                                + TAMANHO_COD_HEADER];
            int aux = socket.readBytes(arrayHeader, 0,
arrayHeader.length);
            Vm.debug("erro socket: " + socket.lastError);
            Vm.debug("Palm lendo cabecalho mensagem
tamanho:" + aux);
            if (aux > 0) {
                String header = new String(arrayHeader);
                this.codigo =
Convert.toInt(header.substring(0,
                                TAMANHO_COD_HEADER));
                int tamanhoConteudo =
Convert.toInt(header
                .substring(TAMANHO_COD_HEADER));
                if (tamanhoConteudo > 0) {
                    byte[] arrayConteudo = new
byte[tamanhoConteudo];
                    aux =
socket.readBytes(arrayConteudo, 0,
                    arrayConteudo.length);
                    Vm.debug("Palm lendo Conteudo
mensagem tamanho:" + aux);
                    if (aux > 0) {
                        this.initConteudo(new
String(arrayConteudo));
                        resp = true;
                    }
                } else {
                    resp = true;
                    Vm.debug("Conteudo de resposta
vazio");
                }
            }
        }
    }
}

```

```

        socket.close();
    }
    return resp;
}

/**
 * Envia ao servidor uma mensagem solicitando mensagem de resposta,
por
 * exemplo mensagens de status dos processos.
 *
 * @param codigo
 *         Código da mensagem
 * @param socket
 *         socket de conexão entre o palm e o servidor(cluster).
 * @return Returns true se a mensagem foi enviada com sucesso. false
se não
 *         foi possível enviar a mensagem
 */
private boolean envieMsgSolicitacao(int codigo, Socket socket) {
    this.codigo = codigo;
    byte[] arrayConteudo = this.getMensagem();
    return this.sendToServer(socket, arrayConteudo);
}

/**
 * Envia dados para o servidor via Socket.
 *
 * @param socket
 *         socket de conexão entre o palm e o servidor(cluster)
 * @param arrayConteudo
 *         Array de bytes com os dados a serem transmitidos(a
mensagem
 *         completa).
 * @return Returns true se a mensagem foi enviada com sucesso. false
se não
 *         foi possível enviar a mensagem
 */
private boolean sendToServer(Socket socket, byte[] arrayConteudo) {
    boolean resp = false;
    if (socket.isOpen()) {
        int escritos = socket.writeBytes(arrayConteudo, 0,
            arrayConteudo.length);
        if (escritos >= 0)
            resp = true;
    }
    return resp;
}

/**
 * Cria um array de bytes com os atributos do objeto mensagem
 *
 * @return Returns array de bytes que representa a mensagem
 */
private byte[] getMensagem() {
    String conteudoMsg = this.getHeader(codigo);
    conteudoMsg = conteudoMsg + completeTamanhoConteudoMsg();
    for (int i = 0; i < colecaoConteudo.size(); i++) {

```

```

        conteudoMsg = conteudoMsg + colecaoConteudo.items[i] +
FIM_CONTEUDO;
    }
    byte[] arrayConteudo = conteudoMsg.getBytes();
    conteudoMsg = null;
    return arrayConteudo;
}

/**
0s à
 * Completa o campo do header que especifica o tamanho do conteúdo com
 * esquerda.
 * @return Returns String que representa o campo de tamanho do
conteúdo da
 * mensagem
 */
private String completeTamanhoConteudoMsg() {
    int total = 0;
    for (int i = 0; i < colecaoConteudo.size(); i++) {
        total += ((String) colecaoConteudo.items[i]).length()
            + this.FIM_CONTEUDO.length();
    }
    String resp = String.valueOf(total);
    int diferenca = TAMANHO_ARRAY_HEADER - resp.length();
    for (int i = 0; i < diferenca; i++) {
        resp = "0" + resp;
    }
    return resp;
}

public Vector getColecaoConteudo() {
    return colecaoConteudo;
}

public int getCodigo() {
    return codigo;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public void setColecaoConteudo(Vector colecaoConteudo) {
    this.colecaoConteudo = colecaoConteudo;
}
}

```

Principal.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.sys.Convert;
import waba.sys.Settings;
import waba.sys.Vm;
import waba.ui.ControlEvent;

```

```

import waba.ui.Event;
import waba.ui.MainWindow;
import waba.ui.MenuBar;
import waba.ui.MessageBox;

/**
 * Classe que representa a janela principal de execução do programa Os
 * atributos
 * que são acessados por outras classes são declarados como public e não
 * possuem
 * get e set por questões de desempenho.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class Principal extends MainWindow {

    public static final MenuBar MENU = new MenuBar(Controle.MENU_PALM);

    public ContainerPrincipal containerPrincipal;

    private TimerLeitorAlerta timerLeitorAlerta;

    public Principal() {
        super("Process Monitor", BORDER_RAISED);
        Controle.leiaArquivosConfiguracao();
    }

    public void onStart() {
        this.setMenuBar(MENU);
        MENU.setEnabled(104, false);
        containerPrincipal = new ContainerPrincipal();
        this.swap(containerPrincipal);
        /*
         * Isso é necessário porque os "containers" são transparentes
         * toda vez
         * que uma outra janela (na verdade mesmo objeto, container
         * diferente)
         * retorna para a principal fica com seus objetos na tela.
         */
    }

    public void onEvent(Event event) {
        if (event.type == ControlEvent.WINDOW_CLOSED) {
            if (event.target == MENU) {
                Vm.debug("Memoria disponivel: " +
                    Vm.getDeviceFreeMemory()
                    + " bytes disponiveis p/ o
                    programa");
                Vm.debug(String.valueOf(MENU.getSelectedMenuItem()));
                switch (MENU.getSelectedMenuItem()) {
                    case 1:
                        exit(0);
                        break;
                    case 101:
                        containerPrincipal.formProcesso();
                        break;
                }
            }
        }
    }
}

```

```

        case 102:
            containerPrincipal.formNodo();
            break;
        case 103:
            this.initMonitoracaoErros();
            break;
        case 104:
            this.pareMonitoracaoErros();
            break;
        case 201:
            containerPrincipal.formConfigServidor();
            break;
        case 2:
            containerPrincipal.executeTeste();
            break;
    }
}
}

private void pareMonitoracaoErros() {
    this.removeThread(timerLeitorAlerta);
    this.habiliteMonitoracao(false);
}

private void initMonitoracaoErros() {
    this.timerLeitorAlerta = new TimerLeitorAlerta();
    this.addThread(timerLeitorAlerta, true);
    this.habiliteMonitoracao(true);
}

private void habiliteMonitoracao(boolean habilitar) {
    MENU.setEnabled(103, !habilitar);
    /* Item Iniciar Monitoração */
    MENU.setEnabled(104, habilitar);
    /* Item Parar Monitoração */
}

public void onExit() {
    Vm.debug("onExit");
}

public static void mostreMensagem(String titulo, String msg) {
    msg = Convert.insertLineBreak(Settings.screenWidth - 6, '|',
        Controle.FONTE_PADRAO.fm, msg);
    MainWindow.getMainWindow().popupModal(new MessageBox(titulo,
msg));
}
}
}

```

TimerLeitorAlerta.java

```

package br.ufsc.ine5328.monitor.cliente;

import waba.sys.Thread;
import waba.sys.Time;

```



```

import waba.sys.Vm;
import waba.util.Vector;

/**
 * Funciona de forma semelhante a um Timer. A cada Controle.INTERVALO_MONITOR
 * segundos conecta-se com o cluster e verifica se houve algum erro com os
 * processos que estão sendo monitorados. Em caso afirmativo mostra a mensagem
 * de erro vinda do servidor em uma caixa de diálogo na interface principal.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class TimerLeitorAlerta implements Thread {
    /** Hora (em segundos) da última monitoração */
    private int tAnt;

    /**
     * Dia em que foi feita a última monitoração. Usado para corrigir os
     tempos
     * entre monitorações quando há troca de dias.
     */
    private int dia;

    /**
     * Se true, indica que a mensagem de erro de conexão com o servidor já
     foi
     * exibida ao usuário
     */
    private boolean erro = false;

    public TimerLeitorAlerta() {
        this.resetParameters();
        Vm.debug("ThreadLeitorAlerta Constructor");
    }

    /**
     * Inicializa os atributos do timer com a hora e o dia atual.
     */
    private void resetParameters() {
        Time t = new Time();
        tAnt = t.hour * 3600 + t.minute * 60 + t.second;
        dia = t.day;
        Vm.debug("Timer - Reset");
    }

    /**
     * Invocado automaticamente pela máquina virtual do superwaba.
     Verifica se o
     * tempo para realizar uma nova monitoração foi atingido. Em caso
     afirmativo
     * conecta-se com o cluster e verifica se existe alguma mensagem de
     erro.
     * Senão não realiza nenhuma tarefa
     */
    public void run() {
        Time t = new Time();
        int tAtual = t.hour * 3600 + t.minute * 60 + t.second;
        int dif = tAtual - tAnt;
    }
}

```

```

        if (dif >= Controle.INTERVALO_MONITOR) {
            Vm.debug("Dif: " + dif);
            Vm.debug("Intervalo: " + Controle.INTERVALO_MONITOR);
            Vm.debug("Tempo Anterior: " + tAnt);
            Vm.debug("Tempo Atual: " + tAtual);
            tAnt = tAtual;
            this.dia = t.day;
            Vm.debug("Segundos: " + dif);
            execute();
        } else {
            /* Verifica se houve mudança no dia e corrige o timer
*/
            if (t.day > this.dia) {
                this.resetParameters();
            }
        }
    }

    /**
     * conecta-se com o cluster e exibe uma mensagem de erro se houver
    alguma.
     * Mostra uma mensagem de erro da primeira vez em que não consegue
     * conectar-se ao cluster Mostra uma mensagem informando ao usu rio que
    a
     * conex o com o cluster foi recuperada, caso a mesma tenha sido
    perdida.
     */
    private void execute() {
        MensagemCliente mensagem = new MensagemCliente();
        mensagem.setCodigo(mensagem.CODIGO_SOLICITACAO_ERROS);
        boolean leu = mensagem.leiaMensagem();
        if (!leu && !erro) {
            erro = true;
            this.mostraMensagemErro("Error to get message on the
server!");
        } else {
            if (leu) {
                if (erro) {
                    erro = false;
                    this
                        .mostraMensagemInfo("Server connection available!");
                }
                Vector conteudoMsg =
mensagem.getColecaoConteudo();
                if (conteudoMsg.size() > 0) {
                    /* Indica que houve alguma mensagem de
erro! */
                    for (int i = 0; i < conteudoMsg.size();
i++) {
                        this.mostraMensagemErro((String)
conteudoMsg.items[i]);
                    /* Conte do da mensagem de erro
*/
                }
            }
        }
    }
}

```

```

    }
}

/**
 * Mostra uma caixa de diálogo contendo uma mensagem de erro.
 *
 * @param msg
 *         contedo da mensagem de erro
 */
private void mostreMensagemErro(String msg) {
    mostreMensagem("ERROR!", msg);
}

/**
 * Mostra uma caixa de diálogo contendo uma mensagem de informa o.
 *
 * @param msg
 *         contedo da mensagem
 */
private void mostreMensagemInfo(String msg) {
    mostreMensagem("INFO!", msg);
}

/**
 * Mostra uma caixa de diálogo contendo uma mensagem.
 *
 * @param titulo
 *         Titulo da mensagem
 * @param msg
 *         conte do da mensagem
 */
private void mostreMensagem(String titulo, String msg) {
    Principal.mostrMensagem(titulo, msg);
}

public void started() {
    Vm.debug("ThreadLeitorAlerta: started");
}

public void stopped() {
    Vm.debug("ThreadLeitorAlerta: stopped");
}
}
}

```

Pacote br.ufsc.ine5328.monitor.ui

HorizontalListBox.java

```

package br.ufsc.ine5328.monitor.ui;

import waba.ui.Container;
import waba.ui.ControlEvent;
import waba.ui.Event;
import waba.ui.ListBox;
import waba.ui.ScrollBar;

/**
 * Componente de interface criado para suprir a falta de um ListBox com barra
 * de
 * rolagem horizontal no Superwaba. O componente possui duas barras de
 * rolagem,
 * uma horizontal e outra vertical
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class HorizontalListBox extends ListBox {
    private ScrollBar hScroll = new HScrollBar(this);

    /**
     * Cria um componente ListBox com duas barras de rolagem, uma
     horizontal e
     * outra vertical
     */
    public HorizontalListBox(Container c) {
        super();
        c.add(hScroll);
        this.setBorderStyle(BORDER_NONE);
    }

    /** Posiciona o objeto no formulário da interface gráfica */
    public void setRect(int x, int y, int width, int height) {
        super.setRect(x, y, width, height);
        hScroll.setMinimum(0);
        hScroll.setLiveScrolling(true);
        hScroll.setRect(x, AFTER, width, 10);
        hScroll.setUnitIncrement(10);
    }

    /**
     * Adiciona um array de itens ao componente
     */
    public void add(Object[] moreItems) {
        super.add(moreItems);
        if (this.getNeededHorizontalScrollValue() <= 0) {
            hScroll.setEnabled(false);
            this.hScroll.setMaximum(0);
        } else {
            hScroll.setEnabled(true);
            this.hScroll.setMaximum(this.width

```

```

        this.getNeededHorizontalScrollValue());
    }

    public void onEvent(Event event) {
        super.onEvent(event);
    }
}

class HScrollBar extends ScrollBar {
    private HorizontalListBox list;

    private int posAtual = 0;

    public HScrollBar(HorizontalListBox list) {
        super(ScrollBar.HORIZONTAL);
        this.list = list;
    }

    public void onEvent(Event evento) {
        super.onEvent(evento);
        if (evento.type == ControlEvent.PRESSED) {
            if (this.getValue() != posAtual) {
                int dif = this.posAtual - this.getValue();
                list.setOffset(-this.posAtual + dif);
                posAtual = this.getValue();
                list.repaint();
            }
        }
    }
}

```

I.II - APLICATIVO SERVIDOR

Pacote br.ufsc.ine5328.monitor.I18n

Messages.java

```

package br.ufsc.ine5328.monitor.i18n;
import java.text.MessageFormat;
import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Messages {
    private static final String BUNDLE_NAME = "messages_enUS"; // $NON-NLS-
1$

    private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
        .getBundle(BUNDLE_NAME);

    private Messages() {
    }

    public static String getString(String key, Object[] params) {
        try {
            String message = RESOURCE_BUNDLE.getString(key);
            return MessageFormat.format(message, params);
        } catch (MissingResourceException e) {
            return "!" + key + "!";
        }
    }
}

```

Pacote br.ufsc.ine5328.monitor.io

BufferedReaderMonitor.java

```

package br.ufsc.ine5328.monitor.io;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;

public class BufferedReaderMonitor extends BufferedReader {

    public BufferedReaderMonitor(Reader arg0, int arg1) {
        super(arg0, arg1);
    }

    public BufferedReaderMonitor(Reader arg0) {
        super(arg0);
    }
}

```

```

public String readLine() throws IOException {
    String linha = super.readLine();
    if (linha != null) {
        linha = linha.trim();
        if ("".equals(linha)) // Linha em branco
            linha = this.readLine();
    }
    return linha;
}
}
}

```

Pacote br.ufsc.ine5328.monitor.testes

GeradorPS.java

```

package br.ufsc.ine5328.monitor.testes;

public class GeradorPS {

    /**
     * @param args
     */
    public static void main(String[] args) {
        if (":1".endsWith(args[0])) {
            /* Mostra uma lista de processos do nodo nmero 1*/
            System.out
                .println(" *****");
            oscar_cluster *****");
            System.out.println(" ----- oscarnode1-----");
            System.out
                .println("  USER          PID %CPU %MEM
COMMAND          STAT");
            //System.out.println("  bin          1249  0.0  0.0
portmap          S");
            //System.out.println("  jboss        28467  0.0  0.0 sh
S");
            //System.out.println("  lp          2500  0.0  0.1 cupsd
S");
            System.out.println("  jboss         31072  0.0  0.0 java
S");
            //System.out.println("  postfix    2626  0.0  0.1 qmgr
S");
            //System.out.println("  postfix    31044  0.0  0.5 pickup
S");
            //System.out.println("  postgres  31049  0.0  1.1
postmaster      S");
            //System.out.println("  postgres  31050  0.0  1.1
postmaster      S");
            //System.out
                //          .println("  root          2  0.0  0.0
keventd        SW");
        } else {
            if (":2".endsWith(args[0])) {

```

```

/* Mostra uma lista de processos do nodo nmero
1*/
System.out
.println("
***** oscar_cluster *****");
System.out.println(" ----- oscarnode2-----
---");
System.out
.println(" USER          PID %CPU
%MEM COMMAND          STAT");
System.out.println("  mysql      2267  0.0  0.0 mysqld
T");
System.out.println("  jboss      31044  0.0  0.0
java          S");
/*System.out
.println("  root              3  0.0
0.0 ksoftirqd_CPU0   SWN");
System.out
.println("  root              4  0.0
0.0 kswapd           SW");
System.out
.println("  root              5  0.0
0.0 bdflush         SW");
System.out
.println("  root             372  0.0
0.0 lvm-mpd        SW<");
System.out
.println("  root             973  0.0
0.1 syslogd       S");
System.out
.println("  root             976  0.0
0.5 klogd         S");
System.out
.println("  root            1035  0.0
0.0 khubd         SW");
System.out
.println("  root            1200  0.0
0.0 resmgrd       S");
System.out
.println("  root            31072  0.0
0.3 ps           R");
System.out
.println("  wwwrun          20484  0.0
0.6 httpd2-prefork S");*/
}
}
}
}
}
}
}
}

```


Pacote br.ufsc.ine5328.monitor

Estado.java

```

package br.ufsc.ine5328.monitor;

/**
 * Possui constantes com os nomes de estados dos processos
 *
 * @author Eduardo
 */
public class Estado {
    public static String SLEEP = "S";

    public static String CPU = "R";

    public static String STOP = "T";

    public static String DEFUNCT = "Z";
    /*
     * D uninterruptible sleep (usually IO) R runnable (on run queue) S
     sleeping
     * T traced or stopped Z a defunct ("zombie") process
     */
}

```

GerenciadorMensagem.java

```

package br.ufsc.ine5328.monitor;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Hashtable;
import java.util.Vector;

import br.ufsc.ine5328.monitor.il8n.Messages;

/**
 * Classe responsável por receber e processar mensagens dos dispositivos
 * móveis
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class GerenciadorMensagem extends Thread {

    private ServerSocket serverSocket;

    private Hashtable errosEnviados = new Hashtable();

    public GerenciadorMensagem() {
        try {
            serverSocket = new ServerSocket(LeitorPropriedades.propriedades
                .getPorta());

```

```

    } catch (IOException e) {
        e.printStackTrace();
        Object[] params =
{String.valueOf(LeitorPropriedades.propriedades.getPorta())};
        throw new RuntimeException(Messages.getString(
            "gerenciadorMensagem.erroPortaOcupada",
            params));
    }
}

/**
 * Aguarda o cliente se conectar. Quando o cliente se conecta e envia uma
 * mensagem de solicitação o método processa a mensagem e envia a resposta
 * para o cliente
 */
public void run() {
    try {
        while (true) {
            System.out
                .println("GerenciadorMensagem - Aguardando conexao na
porta "

                    + LeitorPropriedades.propriedades.getPorta());
            /* Aguarda conexão do cliente */
            Socket conexaoCliente = serverSocket.accept();
            System.out.println("cliente conectado");
            byte[] cabecalho = new byte[Mensagem.TAMANHO_COD_HEADER
                + Mensagem.TAMANHO_ARRAY_HEADER];
            /* lê o cabeçalho para saber o tamanho do conteúdo */
            conexaoCliente.getInputStream().read(cabecalho);
            String header = new String(cabecalho);
            /* lê o conteúdo */
            String aux = header.substring(Mensagem.TAMANHO_COD_HEADER);
            byte[] conteudo = new byte[Integer.parseInt(aux)];
            conexaoCliente.getInputStream().read(conteudo);
            String mensagem = new String(header + new String(conteudo));
            Mensagem msg = Mensagem.crieMensagem(mensagem);
            Mensagem resposta = this.processeMensagem(msg);
            System.out.println("Servidor: escrevendo mensagem= "
                + resposta.formateMensagem());
            conexaoCliente.getOutputStream().write(
                resposta.formateMensagem().getBytes());
            conexaoCliente.getOutputStream().flush();
            conexaoCliente.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
        this.start();
        throw new RuntimeException(Messages.getString(
            "gerenciadorMensagem.erroConexaoCliente",
            null));
    }
}

/**
 * Invoca o método responsável pelo processamento da mensagem de acordo
com
 * o código da mesma e retorna a mensagem de resposta

```

```

*
* @param msg
*      Mensagem a ser processada
* @return Returns mensagem de resposta ou null se o código da mensagem
for
*      inexistente
*/
private Mensagem processeMensagem(Mensagem msg) {
    System.out.println("Processando Mensagem: " + msg.getCodigo());
    int codigo = msg.getCodigo();
    switch (codigo) {
        case Mensagem.CODIGO_SOLICITACAO_ERROS:
            return getErrosProcessos();
        case Mensagem.CODIGO_SOLICITACAO_LISTA_PROCESSOS:
            return getListaProcessos();
        case Mensagem.CODIGO_ALTERA_COMANDO:
            return getAlterarComando(msg.getConteudo());
    }
    return null;
}

/**
* Processa a mensagem responsável por alterar o comando de monitoração do
* cluster
*
* @param conteudo
*      Vetor contendo o novo comando
* @return Returns Mensagem de resposta contendo "Alterou" se o comando
foi
*      atualizado ou com contedo em branco se o comando de monitoração
*      não pôde ser atualizado
*/
private Mensagem getAlterarComando(Vector conteudo) {
    Vector retorno = new Vector();
    String processos = conteudo.remove(0).toString();
    Vector nodos = conteudo;
    getComandos(nodos, processos);
    LeitorProcessos leitor = LeitorProcessos.getInstance();
    if (leitor.isAlterou())
        retorno.addElement("Alterou");
    Mensagem resp = Mensagem.criaMensagem(
        Mensagem.CODIGO_RESPOSTA_ALTERA_COMANDO, retorno);
    leitor.setProcessosSolicitadosUsuario(separeProcessos(processos));
    leitor.setNodosSolicitadosUsuario(new ListaNodo(nodos));
    return resp;
}

/**
* Cria um Vetor com todos os processos do parâmetro processos
*
* @param processos
*      processos monitorados separados por ,
* @return Returns Vector onde cada elemento corresponde a um processo
*/
private Vector separeProcessos(String processos) {
    Vector resp = new Vector();

```

```

    processos = processos + ",";
    while (processos.length() > 0) {
        int indice = processos.indexOf(",");
        String aux = processos.substring(0, indice);
        resp.add(aux.trim());
        processos = processos.substring(indice + 1);
    }
    return resp;
}

/**
 * Monta os comandos de monitoração para todos os nodos monitorados
 *
 * @param nodosIntervalo
 *         nmeros dos nodos monitorados
 * @param processos
 *         nomes dos processos monitorados separador por ,
 */
private void getComandos(Vector nodosIntervalo, String processos) {
    Vector cmd = new Vector();
    if (!LeitorPropriedades.propriedades.isOscarSystem()) {
        for (int i = 0; i < nodosIntervalo.size(); i++) {
            cmd.addElement("java -jar teste.jar : " +
                nodosIntervalo.get(i));
        }
    } else {
        for (int i = 0; i < nodosIntervalo.size(); i++) {
            cmd
                .addElement("cexec "
                    + LeitorPropriedades.propriedades
                        .getNomeCluster()
                    + ":"
                    + nodosIntervalo.get(i)
                    + " \"ps -o user,pid,pcpu,pmem,comm,stat --
sort user -C\" "
                    + processos);
        }
    }
    LeitorProcessos leitor = LeitorProcessos.getInstance();
    leitor.altereComandos(cmd);
}

/**
 * Processa a mensagem de solicitação da lista de processos que estão
 sendo
 * executados no cluster
 *
 * @return Returns Mensagem de resposta contendo a lista com as
 informações
 *         dos processos monitorados
 */
private Mensagem getListaProcessos() {
    LeitorProcessos leitor = LeitorProcessos.getInstance();
    Vector listaProcessos = leitor.getBufferProcessosCluster();
    this.errosEnviados.clear();
    Mensagem resp = Mensagem.crieMensagem(
        Mensagem.CODIGO_RESPOSTA_LISTA_PROCESSOS, listaProcessos);
}

```

```

        return resp;
    }

    /**
     * Processa a mensagem de solicitação de erros
     *
     * @return Returns Mensagem com os erros ocorridos no cluster ou uma
     *         mensagem vazia caso não existam erros
     */
    private Mensagem getErrosProcessos() {
        VisitanteVerificadorErro vis = VisitanteVerificadorErro.getInstance();
        Vector errosEnviar = getNovosErros(vis.getErros());
        Mensagem resp = Mensagem.crieMensagem(Mensagem.CODIGO_RESPOSTA_ERROS,
            errosEnviar);
        return resp;
    }

    /**
     * Retorna os erros para os quais ainda não foram enviadas mensagens ao
     * cliente na transmissão anterior.
     *
     * @param errosEncontrados
     *         Vetor com todos os erros encontrados na monitoração pelo
     *         Visitante
     */
    private Vector getNovosErros(Vector errosEncontrados) {
        Vector resp = new Vector();
        if (this.errosEnviados.size() > 0) {
            for (int i = 0; i < errosEncontrados.size(); i++) {
                if (this.errosEnviados.get(errosEncontrados.get(i)) == null) {
                    /* Significa que um novo erro */
                    resp.addElement(errosEncontrados.get(i));
                }
            }
        } else {
            /*
             * Nesse caso a resposta será todos os erros uma vez que a lista
             * de
             * enviados está vazia
             */
            resp = errosEncontrados;
        }
        this.errosEnviados.clear();
        /* Limpa a tabela de erros enviados */
        for (int i = 0; i < errosEncontrados.size(); i++) {
            this.errosEnviados.put(errosEncontrados.get(i), errosEncontrados
                .get(i));
            /* Adiciona todos os erros encontrados */
        }
        System.out.println("GerenciadorMensagem - Erros enviados: " + resp);
        return resp;
    }
}

```

LeitorProcessos.java

```
package br.ufsc.ine5328.monitor;
```

```

import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Vector;

import br.ufsc.ine5328.monitor.io.BufferedReaderMonitor;

/**
 * Executa o comando de monitoração e faz a leitura da saída do comando.
 * Possui
 * uma instância única para todo o sistema (Padrão singleton)
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 *
 */
public class LeitorProcessos {
    /** Armazena a saída com a lista de processos do último comando executado
    */
    private Vector BUFFER_LISTA_PROCESSOS;

    /** Vetor com os comandos que serão executados na monitoração */
    private Vector COMANDOS = new Vector();

    private static LeitorProcessos singleton;

    /** Contém os nomes de processos que são monitorados */
    private Vector processosSolicitadosUsuario = new Vector();

    /** Flag para indicar se houve alteração no comando de monitoração */
    private boolean alterou;

    /** Contém os números dos nodos monitorados */
    private ListaNodo nodosSolicitadosUsuario;

    static {
        singleton = new LeitorProcessos();
    }

    private LeitorProcessos() {
    }

    public static LeitorProcessos getInstance() {
        return singleton;
    }

    public static void main(String[] args) {
    }

    /**
     * Executa os comandos de monitoração e lê a saída do comando criando uma
     * coleção com seus respectivos processos. Também inicia os atributos dos
     * processos de acordo com os valores das colunas dos comandos executados
     *
     * @return Returns Lista de nodos com seus processos
     */
    public synchronized ListaNodo getDadosProcessoExterno() {

```

```

if (COMANDOS.size() > 0) {
    ListaNode resp = new ListaNode();
    Propriedades props = LeitorPropriedades.propriedades;
    BUFFER_LISTA_PROCESSOS = new Vector();
    for (int j = 0; j < COMANDOS.size(); j++) {
        try {
            Process process = Runtime.getRuntime().exec(
                (String) COMANDOS.get(j));
            BufferedReaderMonitor buffer = new BufferedReaderMonitor(
                new InputStreamReader(process.getInputStream()),
                props.getTamanhoBuffer());
            String nomeCluster = buffer.readLine();
            if (nomeCluster != null) {
                nomeCluster = nomeCluster.replaceAll("\\\\*", "");
                BUFFER_LISTA_PROCESSOS.addElement("***" + nomeCluster
                    + "***");
                // A segunda lista o primeiro nodo
                String aux = buffer.readLine();
                if (aux != null) {
                    Node nodoAtual = getNode(aux);
                    BUFFER_LISTA_PROCESSOS.addElement(aux);
                    /* linha com os nomes das colunas */
                    String linhaAtual = buffer.readLine();
                    BUFFER_LISTA_PROCESSOS.addElement(linhaAtual);
                    ArrayList nomeColunas = getColunas(linhaAtual);
                    resp.add(nodoAtual);
                    while (linhaAtual != null) {
                        linhaAtual = buffer.readLine();
                        if (linhaAtual != null) {
                            BUFFER_LISTA_PROCESSOS
                                .addElement(linhaAtual);
                            ArrayList atributosProcesso =
                                getColunas(linhaAtual);

                            Processo processo = new Processo();
                            for (int i = 0; i < atributosProcesso
                                .size(); i++) {
                                processo.setAtributo(
                                    (String) nomeColunas.get(i),
                                    (String) atributosProcesso
                                        .get(i));
                            }
                            nodoAtual.getProcessos().add(processo);
                            processo.setNodoPai(nodoAtual);
                        }
                    }
                    buffer.close();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return resp;
}
return null;
}

```

```

do
/**
 * Cria um objeto nodo dada a linha do comando de monitoração com o nome
 * nodo
 */
private Nodo getNode(String linha) {
    String nomeNodo = linha.replaceAll("-", "");
    nomeNodo = nomeNodo.trim();
    return new Nodo(nomeNodo);
}

/**
 * Quebra a linha em colunas. Cada coluna deve ser separada por um ou mais
 * espaços em branco
 */
private ArrayList getColunas(String linhaAtual) {
    ArrayList resp = new ArrayList();
    linhaAtual = linhaAtual + " ";
    while (linhaAtual.length() > 1) {
        int ind = linhaAtual.indexOf(" ");
        String coluna = linhaAtual.substring(0, ind);
        linhaAtual = linhaAtual.substring(ind + 1).trim() + " ";
        resp.add(coluna.trim());
    }
    return resp;
}

public Vector getBufferProcessosCluster() {
    return BUFFER_LISTA_PROCESSOS;
}

/**
 * Altera o comando de monitoração de atualiza os parâmetros de
 * monitoração
 * caso o comando seja diferente do comando usado na última monitoração
 *
 * @param novoComando
 *         novo comando de monitoração
 */
public void altereComandos(Vector novoComando) {
    alterou = false;
    if (novoComando != null && novoComando.size() > 0) {
        if (novoComando.size() != COMANDOS.size()) {
            COMANDOS = novoComando;
            alterou = true;
        } else {
            for (int i = 0; i < novoComando.size(); i++) {
                if (!novoComando.get(i).equals(COMANDOS.get(i))) {
                    COMANDOS.setElementAt(novoComando.get(i), i);
                    alterou = true;
                }
            }
        }
    }
    if (alterou) {
        // Atualizar lista
        getDadosProcessoExterno();
    }
}

```



```

    }

}

public Vector getProcessosSolicitadosUsuario() {
    return processosSolicitadosUsuario;
}

public void setProcessosSolicitadosUsuario(
    Vector processosSolicitadosUsuario) {
    this.processosSolicitadosUsuario = processosSolicitadosUsuario;
}

public boolean isAlterou() {
    return alterou;
}

public void setNodosSolicitadosUsuario(ListaNodo nodos) {
    this.nodosSolicitadosUsuario = nodos;
}

public ListaNodo getNodosSolicitadosUsuario() {
    return nodosSolicitadosUsuario;
}
}

```

LeitorPropriedades.java

```

package br.ufsc.ine5328.monitor;

import java.beans.XMLDecoder;
import java.beans.XMLEncoder;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

/**
 * Essa classe é responsável por ler as configurações do monitor.
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class LeitorPropriedades {
    public static final String fileName = "./monitor.properties.xml";

    /** Contém as propriedades de configuração do sistema */
    public static Propriedades propriedades;

    public static void main(String[] args) {
        try {
            LeitorPropriedades.leiaPropriedadesMonitor();

            System.out.println(LeitorPropriedades.propriedades.isOscarSystem());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}

/**
 * Lê as propriedades de configuração do monitor inicializando o objeto
 * propriedades
 */
public static void leiaPropriedadesMonitor() throws FileNotFoundException
{
    XMLDecoder d = new XMLDecoder(new BufferedInputStream(
        new FileInputStream(fileName)));
    propriedades = (Propriedades) d.readObject();
    d.close();
}

/**
 * Salva os valores dos atributos do objeto propriedades no XML de
 * configuração do sistema
 */
public static void salvePropriedadesMonitor() throws FileNotFoundException
{
    XMLEncoder e = new XMLEncoder(new BufferedOutputStream(
        new FileOutputStream(fileName)));
    e.writeObject(propriedades);
    e.flush();
    e.close();
}
}

```

ListaNodo.java

```

package br.ufsc.ine5328.monitor;

import java.util.ArrayList;
import java.util.Vector;

/**
 * Representa uma coleção de nodos. Possui métodos para facilitar a
 * manipulação
 * de dados de uma coleção de nodos.
 */
public class ListaNodo extends ArrayList {
    private static final long serialVersionUID = 3068370102096481014L;

    public ListaNodo() {
        super();
    }

    public ListaNodo(Vector nodos) {
        super();
        for (int i = 0; i < nodos.size(); i++) {
            this.add(new Nodo((String) nodos.get(i)));
        }
    }

    public Nodo getNode(int posicao) {

```

```

        return (Nodo) this.get(posicao);
    }
}

```

ListaProcesso.java

```

package br.ufsc.ine5328.monitor;

import java.util.ArrayList;

/**
 * Possui métodos para facilitar a manipulação de dados de uma coleção de
 * processos
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class ListaProcesso extends ArrayList {

    private static final long serialVersionUID = -5562959410723233084L;

    public Processo getProcesso(int posicao) {
        return (Processo) this.get(posicao);
    }

    /**
     * Cria um novo objeto ListaProcesso dado uma lista com os nomes dos
     * processos
     */
    public static ListaProcesso criaListaPorNome(ArrayList nomeProcessos) {
        ListaProcesso resp = new ListaProcesso();
        for (int i = 0; i < nomeProcessos.size(); i++) {
            resp.add(nomeProcessos.get(i));
        }
        return resp;
    }
}

```

Mensagem.java

```

package br.ufsc.ine5328.monitor;

import java.util.Vector;

/**
 * Objeto que representa a mensagem enviada ao sistema cliente.
 */
public class Mensagem {
    /** Código identificador da mensagem */
    private int codigo;

    /**
     * Conteúdo da mensagem, cada índice do vetor será separado pela String
     * especial FIM_CONTEUDO.
     */
    private Vector conteudo;
}

```

```

/**
 * Número de caracteres do código da mensagem
 */
public static final int TAMANHO_COD_HEADER = 3;

/**
 * Número de caracteres do código da mensagem indicam o tamanho do
conteúdo
 * da mensagem.
 */
public static final int TAMANHO_ARRAY_HEADER = 10;

/**
 * String de separação entre conteúdos diferentes de uma mesma mensagem.
 * Deve sempre aparecer no fim de cada conteúdo
 */
public static final String FIM_CONTEUDO = "ÿ";

/**
 * Código da mensagem que solicita ao servidor informações sobre possíveis
 * erros com processos monitorados
 */
public static final int CODIGO_SOLICITACAO_ERROS = 1;

/** Código da mensagem de resposta da solicitação de erros */
public static final int CODIGO_RESPOSTA_ERROS = 2;

/** Código da mensagem de solicitação da lista de processos */
public static final int CODIGO_SOLICITACAO_LISTA_PROCESSOS = 3;

/** Código da mensagem de resposta da solicitação da lista de processos */
public static final int CODIGO_RESPOSTA_LISTA_PROCESSOS = 4;

/** Código da mensagem de alteração do comando de monitoração */
public static final int CODIGO_ALTERA_COMANDO = 5;

*/
/** Código da mensagem e resposta de alteração do comando de monitoração
public static final int CODIGO_RESPOSTA_ALTERA_COMANDO = 6;

public int getCodigo() {
    return codigo;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public Vector getConteudo() {
    return conteudo;
}

public void setConteudo(Vector conteudo) {
    this.conteudo = conteudo;
}

```

```

protected Mensagem() {
}

/**
 * Cria um objeto Mensagem dada uma String com a mensagem recebida pelo
 * servidor
 *
 * @param mensagem
 *         mensagem recebida
 *
 * @return Returns Novo objeto Mensagem
 */
public static Mensagem criaMensagem(String mensagem) {
    System.out.println("Servidor: Mensagem recebida -" + mensagem);
    if (mensagem != null && mensagem.length() > 0) {
        Mensagem msg = new Mensagem();
        msg.setCodigo(getCodigo(mensagem));
        msg.setConteudo(initConteudo(mensagem.substring(TAMANHO_COD_HEADER
            + TAMANHO_ARRAY_HEADER)));
        return msg;
    }
    return null;
}

/**
 * Cria um objeto Mensagem inicializando seu conteudo.
 *
 * @param codigo
 *         Código da mensagem
 * @param conteudoMsg
 *         Conteúdo da mensagem
 *
 * @return Returns Novo objeto Mensagem com conteúdo e código inicializados
 */
public static Mensagem criaMensagem(int codigo, Vector conteudoMsg) {
    Mensagem msg = new Mensagem();
    msg.setCodigo(codigo);
    msg.setConteudo(conteudoMsg);
    return msg;
}

private static int getCodigo(String msg) {
    return Integer.parseInt(msg.substring(0, TAMANHO_COD_HEADER));
}

/**
 * Formata o conteúdo em uma String para ser transmitida pela rede
 *
 * @return Returns Conteúdo do objeto Mensagem em formato de transmissão
 */
public String formateConteudo() {
    String conteudoMsg = "";
    if (this.conteudo == null)
        return "";
    for (int i = 0; i < this.conteudo.size(); i++) {
        conteudoMsg = conteudoMsg + conteudo.get(i) + FIM_CONTEUDO;
    }
}

```

```

    }
    return conteudoMsg;
}

/**
 * Formata o cabeçalho da mensagem em uma String para ser transmitida pela
 * rede
 *
 * @return Returns Cabeçalho do objeto Mensagem em formato de transmissão
 */
public String formateCabecalho() {
    String resp = String.valueOf(this.codigo);
    int diferenca = TAMANHO_COD_HEADER - resp.length();
    for (int i = 0; i < diferenca; i++) {
        resp = "0" + resp;
    }
    String aux = String.valueOf(this.calculaTamanhoMsg());
    diferenca = TAMANHO_ARRAY_HEADER - aux.length();
    for (int i = 0; i < diferenca; i++) {
        aux = "0" + aux;
    }
    return resp + aux;
}

/**
 * Calcula o tamanho do conteúdo da mensagem
 *
 * @return Returns tamanho do conteúdo da mensagem em bytes
 */
private int calculaTamanhoMsg() {
    if (conteudo == null)
        return 0;
    int total = 0;
    for (int i = 0; i < conteudo.size(); i++) {
        total += ((String) conteudo.get(i)).length()
            + Mensagem.FIM_CONTEUDO.length();
    }
    return total;
}

/**
 * Cria um vetor com os conteúdos da mensagem.
 *
 * @param msgConteudo
 *         Conteúdo da mensagem a ser lida
 *
 * @return Returns Vetor com os conteúdos da mensagem
 */
private static Vector initConteudo(String msgConteudo) {
    Vector resp = new Vector();
    while (msgConteudo.length() > 0) {
        int indice = msgConteudo.indexOf(FIM_CONTEUDO);
        resp.addElement(msgConteudo.substring(0, indice));
        msgConteudo = msgConteudo.substring(indice + 1);
    }
    return resp;
}

```

```

/**
 * Formata a mensagem em uma String para ser transmitida pela rede
 *
 * @return Returns mensagem em formato de transmissão
 */
public String formateMensagem() {
    return this.formateCabecalho() + this.formateConteudo();
}
}

```

Monitor.java

```

package br.ufsc.ine5328.monitor;

import java.io.FileNotFoundException;

/**
 * Classe inicial do sistema. Inicializa o gerenciador de mensagens e a
 * monitoração do cluster
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class Monitor {

    private GerenciadorMensagem gerenteMensagem;

    protected Monitor() throws FileNotFoundException {
        // Lê propriedades
        LeitorPropriedades.leiaPropriedadesMonitor();
        // Inicializa Gerenciador de mensagens
        gerenteMensagem = new GerenciadorMensagem();
        gerenteMensagem.start();
    }

    public static void main(String[] args) {
        try {
            Monitor monitor = new Monitor();
            monitor.execute();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Looping de monitoração
     */
    private void execute() {
        try {
            LeitorProcessos leitor = LeitorProcessos.getInstance();
            while (true) {
                ListaNodo nodos =
leitor.getDadosProcessoExterno();
                verifiqueErrosEstadoProcessos(nodos);
                verifiqueErrosEstadoNodos(nodos);

```

```

        Thread.sleep(LeitorPropriedades.propriedades
                    .getIntervaloMonitoracoes() *
1000);
        /* No arquivo de propriedades o tempo é
descrito em segundos */
    }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private void verifiqueErrosEstadoNodos(ListaNodo nodosLidos) {
    VisitanteVerificadorErro visitante = VisitanteVerificadorErro
        .getInstance();

    visitante.confiraNodosSolicitados(LeitorProcessos.getInstance().getNod
osSolicitadosUsuario(), nodosLidos);
    System.out.println("Servidor Erros:" + visitante.getErros());
}

private void verifiqueErrosEstadoProcessos(ListaNodo nodos) {
    if (nodos != null) {
        System.out.print("Verificando Erros");
        LeitorProcessos leitor = LeitorProcessos.getInstance();
        VisitanteVerificadorErro visitante =
VisitanteVerificadorErro
            .getInstance();
        visitante.clearErros();
        for (int i = 0; i < nodos.size(); i++) {
            nodos.getNodo(i).recebavisitante(visitorante);
        }
        visitante.confiraProcessosSolicitados(leitor
            .getProcessosSolicitadosUsuario());
    } else {
        System.out.println("Nenhum processo esta sendo
monitorado");
    }
}
}
}

```

Nodo.java

```

package br.ufsc.ine5328.monitor;

/**
 * Representa um nodo do cluster
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class Nodo {
    private String nome;

    private ListaProcesso processos;

    public Nodo(String nomeNodo) {

```



```

        this.nome = nomeNodo;
        processos = new ListaProcesso();
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public ListaProcesso getProcessos() {
        return processos;
    }

    public void recebevisitante(Visitante vis) {
        int cont = 0;
        while (vis.visitarProximo() && cont < processos.size()) {
            vis.visite(processos.get(cont));
            cont++;
        }
    }
}

```

Processo.java

```

package br.ufsc.ine5328.monitor;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;

/**
 * Contém atributos referentes aos processos monitorados
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class Processo {

    public static final String ATRIBUTO_ESTADO = "STAT";

    public static final String ATRIBUTO_COMANDO = "COMMAND";

    public static final String ATRIBUTO_PID = "PID";

    private LinkedHashMap propriedadesProceso = new LinkedHashMap();

    private Nodo nodoPai;

    public String getAtributo(String nomeAtributo) {
        return (String) this.propriedadesProceso.get(nomeAtributo);
    }

    public void setAtributo(String nomeAtributo, String valorAtributo) {
        this.propriedadesProceso.put(nomeAtributo, valorAtributo);
    }
}

```

```

    }

    public List getPropriedadesEmOrdem() {
        return new ArrayList(this.propriedadesProceso.keySet());
    }

    public List getValorPropriedadesEmOrdem() {
        return new ArrayList(this.propriedadesProceso.values());
    }

    public Nodo getNodoPai() {
        return nodoPai;
    }

    public void setNodoPai(Nodo nodoPai) {
        this.nodoPai = nodoPai;
    }
}

```

Propriedades.java

```

package br.ufsc.ine5328.monitor;

/**
 * Contém as propriedades do arquivo de configuração
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class Propriedades {
    private String nomeCluster;

    private int intervaloMonitoracoes; // em segundos

    private int porta;

    private int tamanhoBuffer;

    private boolean oscarSystem = false;

    public Propriedades() {
    }

    public void setNomeCluster(String nome) {
        this.nomeCluster = nome;
    }

    /**
     * @return Returns the nomeCluster.
     */
    public String getNomeCluster() {
        return nomeCluster;
    }

    /**
     * @return Returns the intervaloMonitoracoes.

```

```

    */
    public int getIntervaloMonitoracoes() {
        return intervaloMonitoracoes;
    }

    /**
     * @param intervaloMonitoracoes
     *         The intervaloMonitoracoes to set.
     */
    public void setIntervaloMonitoracoes(int intervaloMonitoracoes) {
        this.intervaloMonitoracoes = intervaloMonitoracoes;
    }

    /**
     * @return Returns the porta.
     */
    public int getPorta() {
        return porta;
    }

    /**
     * @param porta
     *         The porta to set.
     */
    public void setPorta(int porta) {
        this.porta = porta;
    }

    public int getTamanhoBuffer() {
        return tamanhoBuffer;
    }

    public void setTamanhoBuffer(int tamanhoBuffer) {
        this.tamanhoBuffer = tamanhoBuffer;
    }

    public boolean isOscarSystem() {
        return oscarSystem;
    }

    public void setOscarSystem(boolean oscarSystem) {
        this.oscarSystem = oscarSystem;
    }
}

```

Visitante.java

```

package br.ufsc.ine5328.monitor;

/** Interface do padrão visitante */
public interface Visitante {
    public void visite(Object obj);

    public boolean visitarProximo();
}

```

VisitanteVerificadorErro.java

```

/**
 *
 */
package br.ufsc.ine5328.monitor;

import java.util.HashMap;
import java.util.Hashtable;
import java.util.List;
import java.util.Vector;

import br.ufsc.ine5328.monitor.il8n.Messages;

/**
 * Verifica erros nos processos e nodos
 *
 * @author Eduardo Bitencourt Milanese <milanese@inf.ufsc.br>
 */
public class VisitanteVerificadorErro implements Visitante {

    private Hashtable todosProcessos = new Hashtable();

    private Hashtable erros = new Hashtable();

    private static VisitanteVerificadorErro singleton;

    static {
        singleton = new VisitanteVerificadorErro();
    }

    public static VisitanteVerificadorErro getInstance() {
        return singleton;
    }

    private VisitanteVerificadorErro() {
    }

    /**
     * Verifica se algum dos processos monitorados está em estado de erro e
     * adiciona erros de acordo com o PID do processo Inicializa um Hash com
     * todos os processos do cluster
     */
    public void visite(Object obj) {
        Processo processo = (Processo) obj;
        String estado = processo.getAtributo(Processo.ATRIBUTO_ESTADO);
        if (estado != null) {
            if (estado.equals(Estado.DEFUNCT)) {
                String[] parametros = { processo.getNodoPai().getNome(),
                    processo.getAtributo(Processo.ATRIBUTO_COMANDO) };
                erros.put(processo.getAtributo(Processo.ATRIBUTO_PID),
                    Messages
                        .getString(
                            "visitanteVerificadorErro.erroProcessoZombie",
                            parametros));
            } else if (estado.equals(Estado.STOP)) {
                String[] parametros = { processo.getNodoPai().getNome(),

```



```
                .getString(
"visitanteVerificadorErro.erroNodeIndisponivel",
                params));
            }
        }
    }

    public Vector getErros() {
        return new Vector(this.erros.values());
    }

    public boolean visitarProximo() {
        return true;
    }

    public void clearErros() {
        this.erros.clear();
    }
}
```

ANEXO II – ARTIGO

Aplicativo para Monitoração de Processos em um Ambiente OSCAR

Mário A. R. Dantas

Departamento de Informática e Estatística (INE)
 Universidade Federal de Santa Catarina (UFSC)
 88040-900 – Florianópolis – Brazil
 Email: mario@inf.ufsc.br

Eduardo B. Milanese

Departamento de Informática e Estatística (INE)
 Universidade Federal de Santa Catarina (UFSC)
 88040-900 – Florianópolis – Brazil
 Email: milanese@inf.ufsc.br

Abstract

Cluster environments are usually complex in number of process and processors. Control systems, like process monitor, increase configuration trustworthiness and help to solve problems.

In this paper, we develop a cluster monitoring tool that warn the system administrator when occurs environments failures. The messages are sending to mobile devices by wireless network.

1. Introdução

A computação está cada dia mais presente em diversas áreas do conhecimento humano. A biologia, a estatística, a meteorologia, a física são apenas alguns exemplos. Junto com a expansão da computação e a complexidade dos sistemas computacionais, cresce a demanda por sistemas com alto desempenho. Em contrapartida, o ritmo de crescimento da capacidade de processamento dos computadores é cada vez menor, devido a proximidade da saturação da tecnologia atual para fabricação de microprocessadores.

Esse cenário ocasionou uma crescente popularização dos sistemas paralelos. Um sistema paralelo é formado através da união de sistemas menores, o que ocasiona uma multiplicação no desempenho. Entre os sistemas paralelos destacam-se os agregados computacionais ou clusters. Por serem formados por componentes de baixo custo e fácil aquisição no mercado, eles são uma solução eficiente e barata para sistemas que necessitam de alta performance.

Uma característica freqüentemente desejável em um ambiente paralelo é a alta disponibilidade. Sistemas de alta disponibilidade possuem mecanismos de recuperação e de tolerância a falhas, entre outros, visando manter-se disponíveis durante a maior fração de tempo possível.

Sistemas de agregado são mais complexos de instalar e manter. Tendo em vista as dificuldades, implementamos uma ferramenta que facilite a monitoração desse tipo de sistema, informando sobre as falhas ocorridas nos nodos e nas aplicações executadas no ambiente sem que o administrador precise realizar monitorações manuais do cluster a todo o momento. Foram desenvolvidos dois aplicativos: um a ser instalado no ambiente do agregado e o outro em um Palm. A comunicação entre os sistemas é feita através de uma rede local *wireless*.

2. Computação em Cluster

Na sua forma mais básica, um cluster é um sistema que compreende dois ou mais computadores ou sistemas (denominados nodos) na qual trabalham em conjunto para executar aplicações ou realizar outras tarefas, de tal forma que os usuários que os utilizam tenham a impressão que somente um único sistema responde para eles, criando assim uma ilusão de um recurso único (computador virtual) [9].

A idéia básica de um cluster, também denominado agregado computacional, é unir computadores para formar um sistema de alto desempenho, de alta disponibilidade no caso de aplicações críticas ou com as duas características. Cada computador do cluster é denominado de nó ou nodo. A “ilusão de um recurso único” trata-se da visualização do cluster por parte dos usuários. Para prover essa funcionalidade são necessários softwares de imagem única ou SSI (Single System Image em Inglês).

Segundo Dantas [4], existem cinco métricas principais para se classificar clusters:

- **Limite Geográfico:** Compreende o alcance geográfico do cluster, que pode compreender uma pequena sala ou até mesmo vários departamentos de uma grande organização.

- **Utilização de Nós:** Pode ser de dois tipos: utilização dedicada e não-dedicada. Na configuração dedicada os nodos são utilizados somente para executar as tarefas referentes ao cluster, enquanto que em configurações não-dedicadas os nodos são também utilizados para outros propósitos.
- **Tipo de Topologia:** É o tipo de hardware empregado na confecção do cluster e na interconexão dos nodos. Os tipos de hardware são subdivididos em NOW (Network of Workstations ou Redes de estações de trabalho em Português), COW(Cluster of Workstations ou cluster de estações de trabalho em Português) e Clump (Cluster of SMPs).
- **Aplicações Alvo:** Consiste no tipo de aplicação para o qual o cluster foi projetado. Nessa classe existem duas métricas principais, o alto desempenho (HPC – High Performance Computing) e a alta disponibilidade (HA – High Availability).
- **Tipos de Nós:** São classificados com relação ao software e hardware. Os clusters de nós homogêneos possuem hardware e software semelhantes, já os sistemas heterogêneos são compostos por nodos com sistemas operacionais ou plataforma de hardware distintas.

Neste artigo utilizamos o pacote de software OSCAR (Open Source Application Resources) para a configuração do ambiente de cluster. O OSCAR possibilita a construção de um sistema de cluster de alto desempenho (HPC). Ele é um SSI (Sistema de Imagem Única), ou seja, o usuário do cluster visualiza um único sistema, apesar do cluster poder ser formado por inúmeros nodos.

3. Alta Disponibilidade

Alta Disponibilidade não é apenas um produto ou uma aplicação que se instale, e sim uma característica de um sistema computacional[3]. Um sistema *A* terá maior disponibilidade que um sistema *B* caso permaneça mais tempo em funcionamento. Geralmente a alta disponibilidade é provida através da redundância de componentes de hardware e software.

Os sistemas de alta disponibilidade são projetados para detectar, mascarar e recuperar-se de falhas. Mascarar uma falha significa impedir sua visualização por um observador externo [8].

O HA-OSCAR (High Availability Open Source Cluster Application Resources) também é um projeto

open source (código aberto), assim como o OSCAR, e implementa técnicas de alta disponibilidade aplicadas em conjunto com a ferramenta OSCAR.

Para prover alta disponibilidade [5] a redundância de componentes é adotada nos clusters HA-OSCAR, visando eliminar o ponto único de falha (nodo mestre). O HA-OSCAR possui mecanismos de recuperação de falhas, failover e failback automáticos.

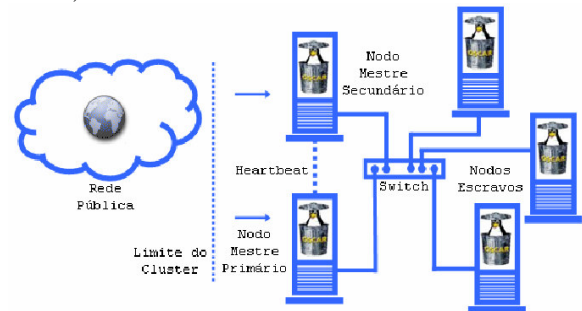


Fig. 1 – Esquema de um cluster HA-OSCAR [2].

Vemos no esquema acima a adição de um nodo mestre secundário. Como uma falha no nodo mestre compromete o funcionamento de todo o cluster, uma solução encontrada para prover alta disponibilidade é replicar esse nodo.

4. Sistema Desenvolvido

É muito comum que sistemas de agregados executem aplicações críticas que não podem ficar indisponíveis. Como todos sistemas computacionais são suscetíveis a eventuais ocorrências de falhas, surge a necessidade da intervenção humana na tarefa de monitoração do ambiente.

A proposta deste artigo é uma ferramenta que auxilie um administrador na tarefa de monitoração de um agregado, enviando mensagens de erro e informações de processos para um dispositivo móvel através de uma rede sem fio.

O desenvolvimento do projeto resultou em dois aplicativos distintos: um sistema servidor implementado em Java usando o pacote de desenvolvimento J2SDK 1.4.2, que deve ser instalado no cluster e é responsável por fazer a monitoração do mesmo. O outro aplicativo foi implementado utilizando a ferramenta de desenvolvimento SuperWaba 5.5 e é um aplicativo voltado a dispositivos móveis (sistema cliente), sendo no caso desse projeto, especificamente, um Palm Tungsten C com sistema operacional Palm OS 5.23.

A lógica do processo de monitoração foi implementada no sistema servidor, sendo que o sistema cliente é responsável por enviar ao sistema servidor os nomes dos processos e nodos do cluster que devem ser monitorados, além de receber eventuais mensagens de

erros. A comunicação entre os sistemas (cliente e servidor) é feita através de uma rede sem fio. O sistema cliente também pode solicitar uma lista com informações sobre os processos e nodos que estão sendo monitorados. Essa lista contém o nome do usuário que iniciou o processo, PID (Process ID ou identificador do processo), porcentagem de CPU ocupada, memória ocupada, nome do processo e estado do processo.



Fig. 2 – Aplicativo Cliente

5. Estudo de Caso

Os testes dos aplicativos foram feitos em apenas uma máquina, com sistema operacional Windows XP, de forma a simular o comportamento do ambiente de cluster. Para simular o funcionamento do palm foram utilizados o Palm OS Emulator v3.5 e o Palm OS Simulation 5.3. Primeiro, as interfaces de configuração do aplicativo cliente foram testada nos simuladores. Depois, foi criada uma configuração padrão para testar a ocorrência de falhas no agregado.

Para o primeiro estudo de caso foi atribuído ao estado dos processos java e mysqld (nodos 1 e 2 respectivamente) o valor “Z” (processo em estado *zombie*) e os aplicativos servidor e cliente foram iniciados. Após dez segundos as mensagens “Nodo oscarnde2 - O processo mysqld está em estado zombie” e “Nodo oscarnde1 - O processo java está em estado zombie” foram exibidas no visor do emulador conforme o esperado. Esperou-se mais trinta segundos para verificar se as mesmas mensagens de erro seriam enviadas novamente, o que não ocorreu conforme o esperado.



Fig. 3 – Mensagens de erro nos estados dos processos “java” e “mysqld”.

O segundo teste foi executado no simulador. As configurações padrão foram restabelecidas no sistema cliente e foi atribuído o valor “T” (processo parado) ao processo “mysqld” do nodo número dois. Foi adicionado no cliente o nodo de número 3 (inexistente) e a lista de configurações foi enviada para o sistema servidor (botão atualizar). A mensagem “O nodo 3 está indisponível” foi exibida no visor do simulador.

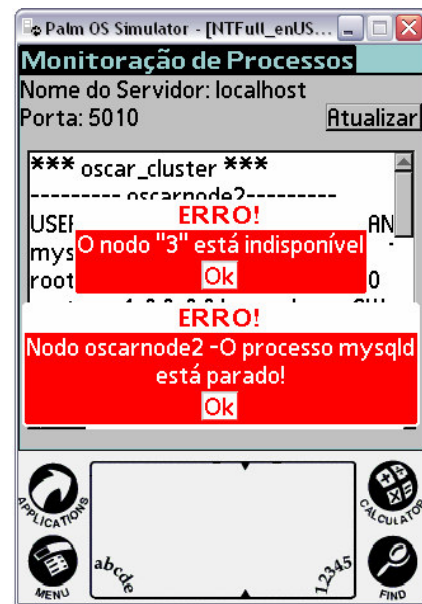


Fig. 4 – Mensagem de erro no nodo 3 e no processo “mysqld”.

No terceiro e último estudo de caso foram testados problemas de comunicação entre o sistema servidor e

cliente. Para realizar o teste o aplicativo servidor foi parado. Foi solicitada a lista de informação de processos do cluster pela interface do POSE. A mensagem “Erro ao tentar ler lista de processos do cluster” foi exibida e as informações anteriores da caixa de listagem foram excluídas. Logo após, o item “Iniciar Monitoração” da barra de menu foi selecionado. Segundos depois a mensagem “Erro ao ler mensagem do servidor” foi exibida. Esperou-se um minuto para verificar se a mensagem de erro seria exibida novamente, o que não ocorreu como era esperado.



Fig. 5 – Cluster indisponível, erro ao ler lista de processos.



Fig. 6.23 – Conexão com servidor restabelecida.

6. Conclusões e Trabalhos Futuros

As possíveis aplicações de sistemas de alto desempenho e disponibilidade são bastante amplas e ambientes de cluster estão se tornando cada vez mais populares. Os clusters possibilitam a criação de sistemas de alto desempenho de alta disponibilidade a custos relativamente baixos se comparados a soluções específicas.

No decorrer deste trabalho observamos a importância da monitoração não só no nível de hardware como também dos serviços disponibilizados pelo ambiente de alto desempenho; esta monitoração é viabilizada através da monitoração dos processos críticos executados no ambiente.

Referências

- [1] ANDERSEN, T.; LEE, P. A. *Fault tolerance principles and practices*. Prentice Hall, 1981.
- [2] BAGGIO, R. K.; DANTAS, Mário A. R. *Uma Ferramenta para Monitoração e alertas SMS em um Ambiente de Cluster de Alta Disponibilidade*. 2004
- [3] Conectiva. *Guia do Servidor Conectiva Linux*. Disponível em <http://www.conectiva.com/doc/livros/online/9.0/servidor/ha.html>. Acessado em Setembro 2005.
- [4] DANTAS, Mário A. Ribeiro. *Computação Distribuída e de Alto Desempenho*. Rio de Janeiro: Axcel Books, 2005. 278 p.
- [5] HA-OSCAR. *High Availability Open Source Cluster Application Resources*. Disponível em <http://xcr.cenit.latech.edu/ha-oscar/index.html>. Acessado em Outubro 2005.
- [6] OSCAR. *Open Source Cluster Application Resources*. Disponível em <http://oscar.openclustergroup.org/>. Acessado em Outubro 2005.
- [7] PalmSource. *Using Palm OS Emulator*. 2003. Disponível em <http://www.palmos.com/dev/support/docs/emulator.pdf>. Acessado em Setembro 2005.
- [8] PEREIRA, Nélio Alves Filho. *Serviços de Pertinência para Clusters de Alta Disponibilidade*. 2004. Disponível em <http://www.ime.usp.br/~nelio/mestrado/>.
- [9] PITANGA, Marcos. *Computação em Cluster*. 2003. Disponível em <http://www.clubedohardware.com.br/artigos/153/>. Acessado em Agosto 2005.
- [10] RISTA, C.; PINTO A. R.; DANTAS M. A. R. OSCAR: Um Gerenciador de Agregado para Ambiente Operacional Linux. 2004.