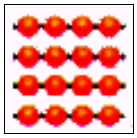
Universidade Federal do Ceará

Mestrado em Ciência da Computação



Dissertação de Mestrado

WVM: Uma Ferramenta para Processamento Distribuído de Alto Desempenho na Web

Paulo Benício Melo de Sousa

Orientador: **Prof. Dr. Mário Fiallos Aguilar**

Fortaleza

Outubro, 2001

Informações Gerais

Título: WVM: Uma Ferramenta para Processamento Distribuído de Alto Desempenho na Web

Autor: Paulo Benício Melo de Sousa

Orientador: Mário Fiallos Aguilar

Departamento: Ciência da Computação

Grau: Mestrado

Local e Data: Fortaleza, Ceará – Outubro de 2001

Agradecimentos

Primeiramente, agradeço ao Deus Eucarístico, verdadeira fonte de vida e sabedoria, pelo maior dom que recebi: a vida e por ter me concedido a graça de ter chegado até aqui. Não posso começar sem reconhecer a meu orientador, o prof. Mário Fiallos, pelo seu empenho e paciência: apesar de suas ocupações e os enormes desafios de sua lenta recuperação, sempre estava pronto a atender meus questionamentos, sugerir propostas e avaliar o andamento de meu trabalho.

Devo ainda agradecer a todo o corpo técnico do Centro Nacional de Processamento de Alto Desempenho no Nordeste (CENAPAD-NE), em especial à Sra. Silvia Lustosa e ao meu amigo Wagner Melcíades, que me estimularam nos momentos em que precisava dispor de mais tempo para a redação desta dissertação. Um carinho especial às duas mulheres da minha vida: Regina e Lília, que souberam entender minhas ausências, aceitando as dificuldades surgidas e me apoiando nos problemas do percurso. Não posso esquecer ainda o Orley da Secretaria do Mestrado que, por várias vezes, soube me ajudar com os vários problemas enfrentados perante a Pró-Reitoria de Pós-Graduação.

Agradeço, enfim, a todos os que, direta ou indiretamente, me auxiliaram nos trabalhos.

AD ASTRA PER ASPERA
(Uma estrada tortuosa leva às estrelas)

APRESENTAÇÃO

Resumo

Este trabalho apresenta uma alternativa para programação de aplicações paralelas-distribuídas que utilizam `PVM`, `MPI` ou `Java`. A ferramenta, que se utiliza de conceitos extremamente simples, é chamada WVM (*Web Virtual Machine*), numa alusão à possibilidade de se utilizar a World-Wide-Web como alternativa para configurar e utilizar aglomerados (*clusters*) remotos para processamento.

WVM dá suporte às várias fases de desenvolvimento de aplicações, tais como conexão remota, edição de códigos, compilação, execução e monitoramento básico de programas. Além disso, WVM permite que usuários ligados à WWW através de *browser*, acessem os aplicativos (bibliotecas) de um sistema de PAD (Processamento de Alto Desempenho) específico.

O objetivo final é apresentar um modelo simples e expansível e fornecer uma ferramenta que possa auxiliar no desenvolvimento de programas distribuídos em diferentes plataformas, tais como *clusters* de PCs, estações de trabalho e computadores de grande porte, como o existente no campo piloto do protótipo, em execução no Centro Nacional de Processamento de Alto Desempenho (CENAPAD-NE).

Palavras-Chaves

Processamento paralelo e distribuído, `PVM`, `MPI`, `Java`, WWW.

Abstract

This work presents an alternative for programming of parallel and distributed applications that use PVM, MPI or Java. The tool is called WVM (Web Virtual Machine), alluding to the possibility for using the World-Wide-Web as an alternative to configure and to utilize specific remote clusters for processing.

WVM supports the different phases of application development: remote connection, code editing, compiling, execution and programming monitoring. WVM also allows that users interconnected to the WWW access applications (libraries) from a High Performance Computing Center.

The tool, which is experimentally running at the National High Performance Computing Center in the Northeast of Brazil (CENAPAD-NE) has been aiding in the development of programs distributed in different platforms, such as clusters of PCs, of work stations and computers of great load.

Keywords

Distributed and parallel processing, PVM, MPI, Java, WWW.

Sumário

Índice das Figuras

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

CAPÍTULO 5

CAPÍTULO 6

CAPÍTULO 7

APÊNDICES

Índice das Tabelas

Introdução

i. Considerações Iniciais

A busca por maior velocidade no processamento das informações foi sempre um desafio para a computação e, ao mesmo tempo, um dos maiores responsáveis pelo seu desenvolvimento. Muito embora a revolução tecnológica tenha trazido avanços até recentemente inimagináveis, percebemos que ela é incapaz de acompanhar as exigências cada vez maiores de recursos computacionais. Isto se resume na clássica premissa de que as necessidades dos usuários tendem sempre a levar à exaustão dos recursos existentes, por mais poderosos que eles sejam.

Não são poucas as situações em que o uso do chamado Processamento de Alto Desempenho (PAD) se torna necessário: problemas complexos como modelagem numérica para previsão de tempo, gerência de grandes bancos de dados, tomadas de decisão em estratégias de defesa, projeto computacional de circuitos, dentre outros. De fato, estas situações são exemplos práticos de problemas para os quais é muito difícil (ou caro) se obter resultados em tempo hábil. Mesmo problemas aparentemente simples como resolução de sistemas de equações, fatoração numérica ou cálculo de caminhos ótimos, podem exigir um grande esforço computacional, muito além das possibilidades efetivas de processamento das várias plataformas disponíveis, devido à complexidade algorítmica ou ao grande volume de dados envolvidos.

As primeiras alternativas propostas para o uso de PAD surgiram com o desenvolvimento de *hardware* extremamente sofisticado, que iam de *mainframes* a computadores vetoriais e paralelos. No entanto, tais benefícios eram acessíveis apenas a um número bastante reduzido de pessoas, em geral, pesquisadores de grandes centros acadêmicos ou militares, devido aos seu altíssimo custo de instalação e manutenção. O desenvolvimento de *software* e de bibliotecas de programação, bem como a utilização de computadores em redes, mais especificamente em *clusters* ou aglomerações de estações de trabalho, tornaram possível disponibilizar as vantagens do PAD para uma massa de pesquisadores ainda maior. Neste contexto, encontramos, por exemplo, as bibliotecas de passagem de mensagens (as chamadas "*Message Passing Libraries*" ou MPLs), que permitem utilizar um ambiente distribuído como um computador paralelo virtual.

Recentemente, a explosão da Internet e da World-Wide-Web (WWW) permitiram interligar mundialmente as redes existentes, facilitando ainda mais o acesso aos recursos distribuídos. Finalmente, o aparecimento da linguagem Java [Gosling 1996], enquanto projeto da Sun Microsystems, veio a preencher uma importante lacuna: a possibilidade da programação independente da plataforma, susceptível à execução via Internet.

O panorama atual que aqui apresentamos é, portanto, marcado por dois universos de programação: de um lado, temos a Internet no ambiente WWW, que, com o uso da linguagem Java, permite a execução anônima de aplicações. De outro lado, temos a chamada programação distribuída de alto desempenho, em que os usuários utilizam recursos computacionais de forma explícita em máquinas que pertencem a um mesmo domínio.

ii. Objetivo da Dissertação

Em função das necessidades impostas pelos paradigmas anteriores, este trabalho propõe uma alternativa para conciliar aspectos da programação Internet com aspectos da programação de PAD distribuída. Mais especificamente, é apresentada uma integração das etapas envolvidas na programação paralela-distribuída, tais como edição, compilação, execução, conexão (Telnet) e transferência de arquivos (FTP), num único ambiente de programação, baseada nas tecnologias Internet e WWW (protocolo HTTP).

Nosso estudo envolverá os seguintes aspectos:

- Breve introdução aos conceitos relacionados com a computação paralela-distribuída de alto desempenho;
- Justificação da ferramenta proposta;
- Análise de requisitos da ferramenta e descrição das características das aplicações e linguagens utilizadas;
- Definição da organização e funcionamento do *software*;
- Apresentação do protótipo de processamento e avaliação dos resultados.

iii. Notação Utilizada

O texto se utilizará de uma notação padronizada, para facilitar a leitura, obedecendo às seguintes regras:

- Termos em *itálico* indicarão palavras ou expressões em língua estrangeira (normalmente inglesa). As traduções podem ser feitas, desde que venham a facilitar a legibilidade do texto e obedeçam à utilização comum;

- Termos em **negrito** serão usados para apresentar alguma definição ou idéia principal. Isto será feito, normalmente, quando se julgar conveniente destacar um conceito específico;
- Textos em fonte `tipográfica` (`Courier New`) indicarão nomes de tecnologias, como linguagens de programação, bibliotecas, padrões industriais ou produtos de *software*, bem como porções de código em uma determinada linguagem;
- O uso de texto em fonte caligráfrica (Umti10) está reservado para a representação de citações de outros autores ou expressões idiomáticas;
- As referências bibliográficas no texto são ilustradas com o uso de uma fonte estilizada (Arial Narrow) de fácil destaque. Referências a tabelas, seções e figuras também utilizam o mesmo estilo, evidenciando ligações a outros contextos dentro ou fora do texto da dissertação;
- Nomes de empresas, organizações, entidades civis e projetos estão grafados com uma fonte diferenciada (Verdana) para ilustrar agentes corporativos mencionados no texto.
- O uso de uma fonte gótica será restrita à apresentação da ferramenta, suas funcionalidades, elementos e operações;
- Finalmente, a utilização de notas de rodapé será feita sempre que aspectos particulares ou comentários sejam necessários. Normalmente, serão tratadas considerações práticas de arquitetura ou implementação, para facilitar o entendimento do texto e posicioná-lo com o estado da arte na época da elaboração do texto (2001).

iv. Organização do Trabalho

Este trabalho está organizado em três partes:

PARTE I: Aqui, apresentamos uma introdução ao Processamento Paralelo-Distribuído, com ênfase na evolução e terminologia dos sistemas, suas classificações e as alternativas para suas implementações. Esta parte compõe-se dos seguinte capítulos:

Capítulo 1 - Conceitos Básicos: Nesta capítulo introdutório são apresentados os conceitos fundamentais relacionados com sistemas de *hardware* e *software*, juntamente com uma breve apresentação das terminologias e um resumo histórico da evolução da área nos últimos anos;

Capítulo 2 – Processamento de Alto Desempenho e Programação Paralela: No segundo capítulo são analisados os tópicos relacionados com aplicações paralelas e as

características de uma das soluções mais comuns para a implementação de ambientes de processamento paralelo: as bibliotecas de passagem de mensagem (com ênfase ao PVM e MPI);

Capítulo 3 – Processamento na Internet e na Web: Aspectos relevantes para o processamento distribuído feito via *Web* são abordados neste capítulo, apresentando ainda o conceito de metacomputação e as alternativas utilizadas para sua implementação.

PARTE II: Nesta segunda parte, a partir dos conceitos apresentados, são descritas a organização, arquitetura e funcionamento da ferramenta WVM.

Capítulo 4 – Apresentação de WVM: Neste capítulo são ilustrados os principais aspectos da ferramenta que propomos para conciliar a programação na Internet com o PAD, o que inclui a descrição de suas características, objetivos e limitações.

Capítulo 5 – Arquitetura: A arquitetura da ferramenta, seus componentes e os aspectos funcionais de WVM são apresentados neste capítulo.

PARTE III: Na última parte, de cunho mais prático, descrevemos a implementação da ferramenta WVM, incluindo aspectos de desenvolvimento do protótipo e estudo de caso, com análise dos resultados obtidos.

Capítulo 6 – Desenvolvimento do Protótipo: Este capítulo trata das alternativas utilizadas para o desenvolvimento da ferramenta e as características do protótipo e do ambiente de execução para a ferramenta;

Capítulo 7 – Conclusão: Uma análise dos resultados experimentais obtidos para a validação da ferramenta, em um estudo de caso, seguindo-se alguns comentários finais relativos às dificuldades encontradas, trabalhos futuros e objetivos alcançados são descritos neste capítulo.

PARTE I

PROCESSAMENTO PARALELO-DISTRIBUÍDO

Capítulo 1

Conceitos Básicos

Introdução

Neste primeiro capítulo, damos uma introdução geral aos conceitos envolvidos no processamento distribuído, envolvendo aspectos históricos, que vão desde noções básicas de sistemas computacionais (hardware e de software) até tópicos relacionados com aplicações distribuídas. Os conceitos aqui apresentados serão importantes para a compreensão dos capítulos seguintes.

1.1. Considerações de *Hardware*: Sistemas Computacionais

1.1.1 – Evolução e Desafios

♦ A. Breve Histórico

A revolução tecnológica que assolou o mundo no século XX possibilitou o advento da chamada Era da Informação. Isso alterou sensivelmente a sociedade, criando novas estruturas, formas de relacionamento entre os indivíduos e culminando com o surgimento de uma sociedade baseada na informação. A velocidade de obtenção de dados significativos para tomadas de decisão ou análises científicas se tornou crucial: os sistemas computacionais deviam ser cada vez mais velozes e eficientes para prover tais resultados mais rapidamente.

A contínua evolução das arquiteturas de computadores, expandindo os horizontes da limitação de armazenamento de dados e velocidade, são comprovações inequívocas disso. A tabela 1.1 (adaptada de [Tanenbaum, 1999]) ilustra sucintamente um quadro comparativo entre as gerações de computadores e seus crescentes desempenhos[1].

[1] A classificação histórica das gerações de computadores não é consenso entre os principais autores e sempre devem ser observadas algumas restrições. [Hwang, 1998], por exemplo, subdivide a 4ª. geração em 2 outras: a geração dos **VLSIs** (Very Large Scale Integrated) – de 1978 a 1989 - onde figuram computadores como o VAX 9000, o Cray X/MP e, mais amplamente, o IBM PC; e a geração dos **ULSIs** (Ultra Large Scale Integrated) –

Geração	Modelo	Autor / Empresa	Ano	Características	Memória	Veloc. (ciclo)
Zero (-1945)	Mark I	Howard Aiken	1944	Computadores Mecânicos / relés	72 palavras × 23 dígitos	6 s
1º (1945-1955)	IAS	John v. Neuman	1950	Válvulas - 5 partes: Memória, ULA, UC, Entrada e Saída	4K palavras de 40bits	Ordem de ms.
2º (1955-1965)	7094	IBM	1965	Transistores – Uso de barramento	32K palavras de 36 bits	2 μs
3º (1965-1980)	System 360	IBM	1975	CIs – Multiprogramação – emulação	512K palavras de 36 bits	250 ηs
4º (1980-)	IBM/SP	IBM	1995	VLSI – Múltiplas categorias (PCs, minis e supercomputadores)	Gigabytes	4 ηs / 1Gflops

Tabela 1.1: Comparação entre modelos das gerações de computadores

Uma observação importante é a de que a organização atual das arquiteturas, por suas várias categorias (microcomputadores, estações de trabalho, mainframes e supercomputadores) segue uma estrutura piramidal [Fox, 1997b] [Hwang, 1998] de acordo com duas variáveis: custo/desempenho e volume de vendas, conforme ilustrado na figura 1.1 (adaptada de [Hwang, *ibid.*]). O que se pode observar é que a quantidade de computadores topo de linha é relativamente pequena, devido basicamente aos seus elevados custos.

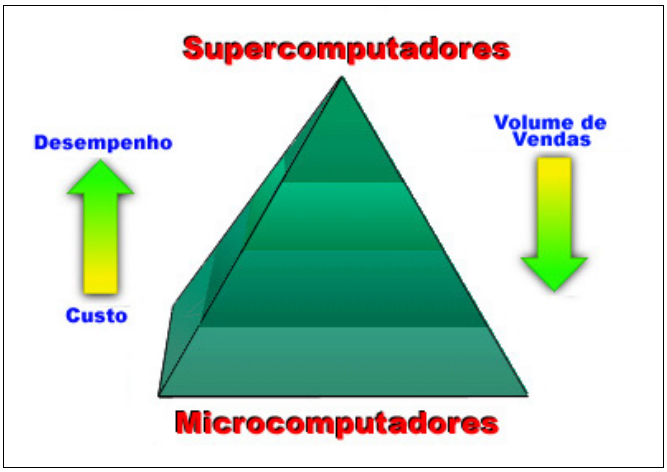


Figura 1.1: Distribuição das categorias de computadores

Pela figura, percebe-se que a medida em que se desce na estrutura piramidal, a partir do topo (onde ficam os chamados "supercomputadores"), encontramos bases cada vez maiores de *mainframes*, seguidos de estações servidoras e, finalmente, milhões de microcomputadores. Uma observação importante é a de que, embora se percebam sensíveis diferenças nos custos, a tecnologia atual pôde prover microcomputadores com desempenhos cada vez melhores. Isso é uma das principais motivações para a utilização das redes de computadores, onde existe uma relação custo/desempenho extremamente favorável.

A existência desta espécie de estratificação da base computacional atual nos leva a um questionamento relativo ao papel que cada um destas categorias possui. Para entendermos melhor a situação, apresentamos um quadro geral, baseado em [Long, 1994]. Uma tabela

---

que se estende de 1990 aos nossos dias - englobando desde o IBM SP e SP/2, às redes virtuais de processamento via WWW. Referências extras podem ser vistas em [Laudon, 1999].

comparativa entre os diferentes modelos ilustra a variação entre preço e desempenho no final da década de 90:

Categoria	Características	Fabricantes /Exemplos	Preços (faixas)	Desempenho (faixas)
Microcomputadores	Finalidade geral – uso doméstico / profissional – produtividade individual – baixo custo – Categorias: *workstations*, *desktops*, servidores, *laptops* e *notebooks*.	IBM/Aptiva, Apple/iMac, Compaq, Toshiba.	US$ 500,00 – US$ 3.000,00	Até alguns MFlops.
Computadores de médio porte	Mais rápidos e com maior capacidade de armazenamento – uso em organizações de médio porte, fábricas e sistemas de controle de tempo real – Quantidade de terminais de 20 a 100.	DEC (VAX), IBM(AS400), HP (HP9000)	US$ 8.000 – US$ 700.000 [(2)]	Algumas centenas de MFlops
Mainframes	Uso em grandes corporações com grandes volumes de dados ou governamental – múltiplos dispositivos para conexão – operação remota – Centenas ou milhares de terminais.	IBM S/390	US$ 50.000 a 3 milhões [(3)].	Até alguns GFlops
Supercomputadores	Computadores de alto poder de processamento – Aplicações: Decisões estratégicas, previsão atmosférica, cálculos massivos.	IBM (SP/2), Cray (XMP)	US$ 3 a 20 milhões [(34)].	Até dezenas de GFlops

Tabela 1.2: Comparação entre as categorias de computadores atuais (1999)

Vale destacar que, conforme podemos constatar pela evolução tecnológica ocorrida nos últimos anos, as distinções entre o que se pode classificar como "microcomputador" ou "estação de trabalho" estão se tornando cada vez mais sutis, ao mesmo tempo em que se percebe uma contínua diminuição no preço dos equipamentos e programas [Laudon, 1999]. No entanto, além do fator custo, duas observações tornam-se importantes para relativizarmos as conquistas até o momento: as necessidades dos usuários e a dicotomia entre *hardware* e *software*.

- (i). Necessidades dos usuários

Primeiramente, podemos facilmente perceber que, apesar dos avanços, a tecnologia é incapaz de satisfazer completamente às necessidades dos usuários, isto é, de acompanhar as exigências cada vez maiores de recursos computacionais mais rápidos, potentes e baratos, uma vez que os problemas a serem resolvidos tendem sempre a levar à exaustão os recursos existentes (por mais poderosos que sejam). Como exemplos de problemas deste gênero, podemos citar otimizações em teoria de grafos, tratamento de imagens, previsão meteorológica, tomadas de decisão estratégicas para defesa, gerência de grandes bancos de dados, simulações do mundo real e análise mercadológica [Foster, 1995].

---

[2] De acordo com preços divulgados na Internet por Martin J. Garvey em Information Week (http://www.iweek.com) – Abril, 1999.

[3] De acordo com dados obtidos na Internet de IDC – International Data Group (http://www.idc.com) – Outubro, 1999.

[4] De acordo com preços obtidos na Internet por CENAPAD-NE pelos fabricantes IBM (http://www.ibm.com), Cray (http://www.cray.com) e Sun (http://www.sun.com) – Novembro, 1999.

No entanto, mesmo em nosso dia-a-dia, nos deparamos constantemente com alguns problemas que, embora de simplicidade aparente, constituem verdadeiros desafios computacionais. Este é o caso de sistemas de equações, fatoração numérica, cálculo de caminhos ótimos ou mesmo teoria dos jogos. Tudo isso se deve, seja devido à complexidade algorítmica ou ao grande volume de dados, levando-nos comumente a soluções proibitivas com explosões combinatórias. Um exemplo claro pode ser encontrado, dentre as inúmeras referências na literatura, no clássico problema do caixeiro viajante (cf., por exemplo, [Vianna, 1998]). A conclusão óbvia é a de que, independentemente do avanço tecnológico e do surgimento de máquinas cada vez mais velozes, a natureza de determinados problemas, determina fortemente as possibilidades de processamento em tempo hábil (confira, por exemplo, os problemas de otimização combinatória).

- (ii). *Hardware versus Software*

O segundo aspecto que não podemos ignorar é o de que o ritmo de evolução do *hardware* não é adequadamente acompanhado pela velocidade com que o *software* se desenvolve [Hwang, 1998]. Ainda que se desconsiderem questões relativas aos custos de implantação de sistemas computacionais de grande porte, deparamo-nos com a dificuldade inevitável de se escolher ferramentas, técnicas e sistemas adequados ou mais eficientes para o desenvolvimento das aplicações que tentarão solucionar os problemas anteriormente propostos. Em outras palavras, resta responder à questão de qual *software* utilizar, a fim de que se possam se aproveitar ao máximo os benefícios do *hardware* existente. A resposta satisfatória a essa pergunta, além de difícil, implica, quase sempre, em altos investimentos.

♦ B. O Problema do Processamento

Considerando os elementos relacionados no item anterior, percebe-se que a evolução tecnológica elevou o poder computacional a patamares antes não imaginados, conforme mostrado na figura 1.2 (adaptada de [Foster, 1995]). No entanto, algumas lacunas podem mostrar que o problema relativo à busca de processamento cada vez mais eficiente ainda representa um desafio:

- A necessidade de ampliar ou aproveitar o parque tecnológico instalado, em especial, a grande quantidade de EPs (microcomputadores, em geral) subtilizados na rede;
- A exigência de *software* adequado para se atingir o nível de processamento desejado, a partir das arquiteturas existentes;
- A carência de técnicas que possibilitem o uso inteligente dos recursos, a fim de se resolver o problema desejado em tempo hábil.

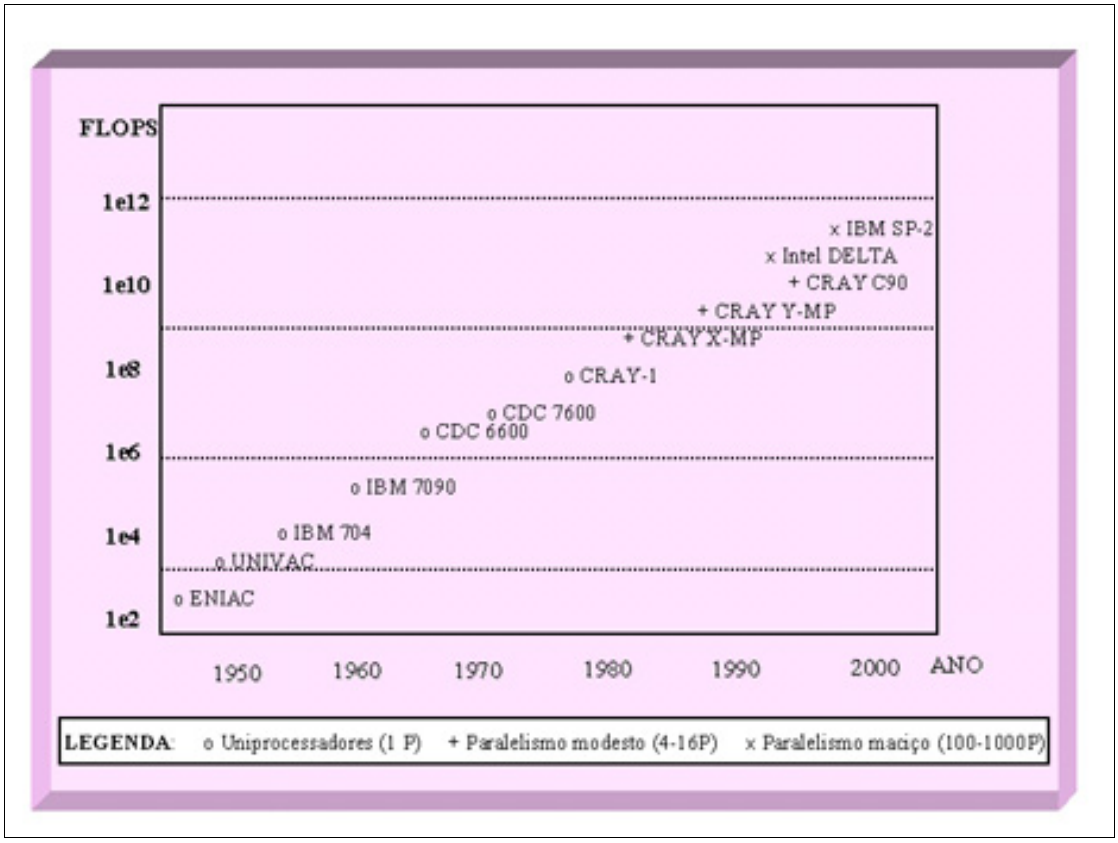


Figura 1.2: Evolução das arquiteturas em termos de FLOPs

A pergunta aqui, portanto, reside em **como** resolver cada um destes pontos, em vista da otimização do uso dos recursos, obtendo um processamento mais rápido. Em outras palavras, buscamos conciliar o desenvolvimento do *hardware*, com o surgimento de novos sistemas, linguagens e técnicas de programação. Uma observação importante é que esse esforço não envolve pura e simplesmente uma agregação de máquinas de elevado poder computacional: vai muito além disso, englobando alternativas de *software* e novos paradigmas para resolução algorítmica dos problemas. Este será o contexto do chamado **Processamento de Alto Desempenho** (PAD), melhor detalhado no próximo capítulo[5].

Durante muitos anos, no entanto, a busca pelo ganho na velocidade de processamento efetuou-se em *hardware* especializado, como as arquiteturas paralelas ou vetoriais. Apesar do bom desempenho, seu alto custo de instalação, manutenção e treinamento de pessoal dificultou em muito a sua utilização generalizada. Isso fez com que tais benefícios fossem acessíveis apenas a um número bastante reduzido de usuários (em geral, restritos a centros acadêmicos ou militares), notadamente bancados por instituições governamentais de fomento à pesquisa ou indústrias.

[5] Aproveitando-se da definição dada pela SOCINFO (Sociedade Brasileira da Informação), o conceito de **Processamento de Alto Desempenho** (PAD) é usado como sinônimo para o termo inglês *High Performance Computing* (HPC).

Recentemente, a proposta de utilização de sistemas multicomputadores (em ***clusters*** ou **aglomerações**), surgiu como alternativa para a obtenção de PAD, a baixo custo, dando origem ao conceito de HPCC (*High Performance Cluster Computing*). A figura 1.3, que se tornou clássica, ilustra de uma forma bastante pictórica, a funcionalidade pretendida com a utilização de redes ou *clusters* de estações de trabalho[6].

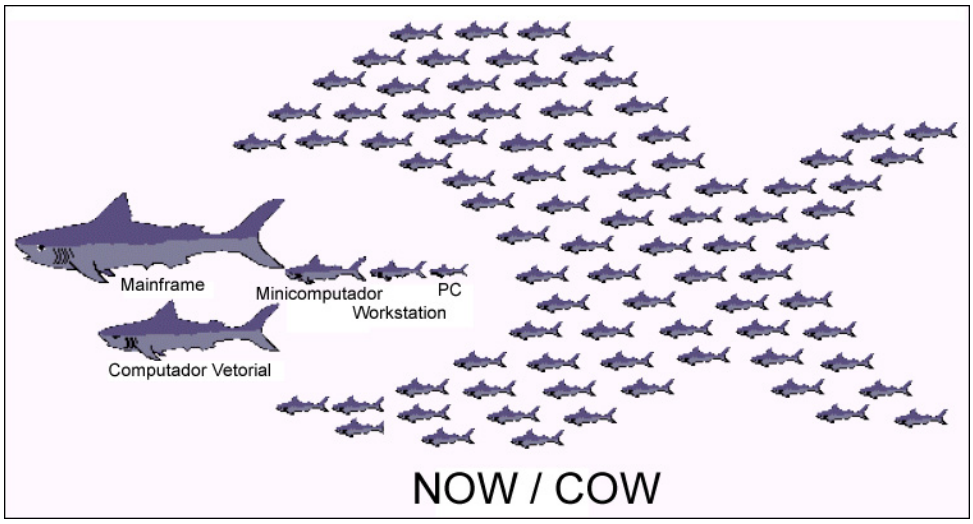


Figura 1.3: Representação das NOWs / COWs

As principais características das NOWs, como boa relação de custo *versus* desempenho, rapidez de processamento, modularidade, confiabilidade e concorrência, serviram para colocá-los como uma das propostas mais atraentes e viáveis nos mais diversos ambientes. Ao mesmo tempo, o desenvolvimento de *software* e bibliotecas de programação, como as bibliotecas de passagem de mensagens (chamadas de *Message Passing Libraries* ou MPLs), tornaram mais fácil a configuração e utilização de ambientes distribuídos como computadores virtuais paralelos. Essas bibliotecas paralelas – mencionadas nos próximos capítulos – popularizaram o uso do PAD para um número ainda maior de pessoas.

Finalmente, vale mencionar a grande quantidade de estudos teóricos e pesquisas na área de computação paralela-distribuída desenvolvida nos últimos anos que permitiram ampliar o leque de técnicas de programação e ferramentas de linguagem capazes de auxiliar no desenvolvimento de programas, sob este novo paradigma.

♦ C. Desafios para o Processamento

No decorrer dessa dissertação, mencionaremos alguns conceitos relevantes aos desafios para a implementação de PAD. Eles derivam tanto de práticos quanto teóricos e representaram, muitas vezes, os grandes limitadores para a adoção dos paradigmas de

[6] Apesar das controvérsias – e da grande complexidade da idéia (que envolve considerações de *hardware*, *software* e aplicações) – a ilustração quer sugerir que redes ou aglomerados computacionais, (do inglês, *Network Of Workstations* – NOWs – ou *Cluster Of Workstations* – COWs), quando adequadamente combinados, podem obter um poder computacional eqüivalente ao de grandes sistemas de processamento.

processamento paralelo-distribuído. Um exemplo clássico pode ser citado: a Lei de Amdahl [1967], que estipulou limites teóricos para o ganho de desempenho em máquinas paralelas, de acordo com uma variável básica: o percentual de código paralelizável. Contudo, para uma boa parte de problemas, tais ganhos no sistema podem realmente agilizar a obtenção de soluções em tempo hábil, compensando, portanto, os investimentos feitos.

Um outro aspecto de relevância com relação aos desafios inerentes ao PAD decorre da diversidade de tecnologias, seus custos de implantação e os tipos de problemas que podem ser tratados por tais alternativas. Para podermos entender isso melhor, partiremos para uma descrição breve da evolução e classificação destes sistemas.

1.1.2 – Classificações das Arquiteturas

♦ A. Considerações Históricas

Um exemplo clássico do modelo computacional pode ser descrito pela máquina **RAM – Random Access Machine** ([Aho, 1974]), baseada na idéia de um sistema uniprocessador seqüencial. Vale a pena notar que o tratamento formal já é bastante antigo, como pode-se perceber dos trabalhos de Alan Turing (1937), na clássica sistematização das atividades desempenhada pelos algoritmos. Note que o modelo também é bastante semelhante à arquitetura real proposta por John von Neumann algumas décadas depois (1963).

A partir da arquitetura RAM básica, podemos compreender melhor a sucessão de modelos que surgiram como propostas para o desenvolvimento de sistemas paralelos-distribuídos. Alguns dos primeiros estudos teóricos desenvolvidos em termos de arquitetura para a obtenção de PAD, podem ser encontrados ainda no início da década de 70 [Flynn, 1966][Aho, 1974]. Eles tornaram possível antever classificações e, mais do que isso, limitações teóricas para o desenvolvimento de sistemas computacionais paralelos.

Em toda a literatura, a classificação proposta por Michael Flynn [1966] para arquitetura de computadores é muito utilizada e a principal razão decorre de sua simplicidade, já que apenas são levadas em conta duas variáveis: o fluxo de instruções e o fluxo de dados (vide figura 1.4), sejam simples ou múltiplos, donde os termos: SISD, SIMD, MISD e MIMD.

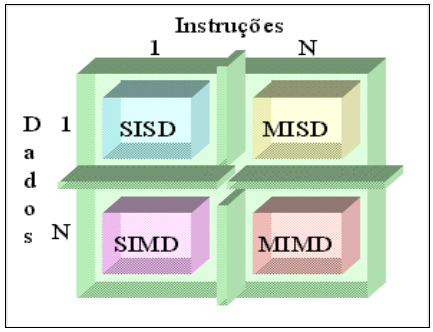


Figura 1.4: Classificação de Flynn

Note que a classificação inicialmente proposta não se preocupava com aspectos tecnológicos de implementação, apenas com as implicações funcionais do modelo. A tabela 1.3 apresenta, sucintamente, as principais características dos modelos apresentados por Flynn.

Categoria	**Características**

SISD *(Single Instruction / Single Data)*	• Arquitetura mais elementar: possui um único processador; • Uma instrução é executada de cada vez, manipulando um único fluxo de dado; • Caracteriza o modelo seqüencial (Arquitetura de von Neumann), cujo princípio básico de funcionamento identifica a grande maioria dos computadores.
SIMD *(Single Instruction / Multiple Data)*	• Controle único de instruções; • Cada instrução pode manipular múltiplos dados simultaneamente; • Opera no modo síncrono: cada instrução é executada simultaneamente em cada processador.
MISD *(Multiple Instruction / Single Data)*	• Possui um mesmo fluxo de dados que flui através de um conjunto de processadores, executando diferentes instruções; • Apesar da complexidade, alguns autores categorizam os computadores sistólicos nesta categoria [Murdocca, 2000].
MIMD *(Multiple Instruction / Multiple Data)*	• Essencialmente formada por grupos de processadores interligados e independentes; • Cada processador ou computador executa suas instruções independentemente dos outros processadores.

Tabela 1.3: Características dos modelos de Flynn

Vale observar que muitos autores, como [Hwang 1993][Tanembaum, 1998], estendem a hierarquia de modelos SIMD e MIMD, propondo uma classificação mais abrangente para a comparação das várias arquiteturas (cf., por exemplo, a figura 1.5).

Devemos salientar ainda que o desenvolvimento de novas tecnologias proporcionaram consideráveis melhorias ao modelo SISD, com a utilização de técnicas como *caching* (introdução de pequenas memórias extremamente rápidas), *pipelining* (utilização de múltiplas unidades funcionais) e *interleaving* (segmentação da memória em regiões). Desta forma, várias limitações originalmente existentes na arquitetura clássica de von Neuman foram corrigidas e melhoradas, conferindo um poder computacional maior às máquinas SISD[7].

No tocante às máquinas SIMD, encontramos paralelismo em nível de processador, havendo uma subclassificação usualmente aceita [Tanembaum, 1998] que consiste na identificação de dois grandes conjuntos: os processadores matriciais e os computadores vetoriais[8]. No primeiro caso, encontramos um grande número de processadores idênticos, normalmente dispostos matricialmente (donde o nome), coordenados pela UC (unidade de

[7] Alguns autores classificam máquinas monoprocessadoras com capacidade de *pipeline* como pertencente a outra categoria de Flynn (MISD) [Tanenbaum, *Ibid.*] [Murdocca, 2000].

[8] [Patterson, 1997] distingue os processadores vetoriais dos matriciais por uma sutil diferença: enquanto os primeiros operam com unidades funcionais projetadas para operar com poucos elementos por ciclo de *clock*, no caso dos computadores SIMD "puros" (matriciais) – tal como proposto por Flynn – as operações atuam simultaneamente sobre todos os elementos no mesmo ciclo. [Hwang, 1995], no entanto, classifica as categorias PVP (*Parallel Vector Processor*) e SMP (*Simmetric MultiProcessor*) como pertencente ao modelo MIMD.

controle)[9]. O exemplo clássico da literatura (`ILLIAC IV`), colocado em funcionamento na década de 70 para a NASA apresentava uma grade de 8×8 processadores, cada um com sua memória própria e visava um poder computacional equivalente ao de todos os demais computadores do mundo (1 GFLOP na época), apesar de, efetivamente, alcançar apenas o desempenho de 15 MFLOPS após vários anos de atraso ([Tanenbaum *apud* Falk, 1976]).

Já os processadores vetoriais funcionam de uma forma um pouco diferente: todas as operações aritméticas são feitas por uma única ULA (unidade lógica e aritmética) que pode operar em *pipeline* sobre os registradores vetoriais da máquina. Apesar de muito semelhantes do ponto de vista do programador, no caso dos computadores vetoriais, têm-se uma considerável simplificação do projeto de hardware ao mesmo tempo em que se obtém um desempenho extremamente eficiente para determinadas aplicações graças à capacidade das instruções em acessarem blocos de memória (diminuindo a latência) e operarem sobre massas de dados (permitindo maiores profundidades nos estágios de *pipeline*) [cf. Patterson, 1997].

Existem ainda muitos outros modelos abstratos de classificação, que vão desde o clássico PRAM [Fortune, 1974] ao CGM [Dehne *et al*, 1994], passando pelo BSP [Valiant, 1990] e pela LogP Machine [Culler *et all,* 1993][10]. Cada um destes modelos visa fornecer parâmetros mais realistas para uma real avaliação do desempenho das máquinas, apresentando características próprias de sincronização (controle), *overhead* (custo para execução de certas operações) e granulosidade (a ser visto adiante). No entanto, a utilização da classificação de Flynn nos permite analisar de uma forma simples a organização dos sistemas.

Finalmente, resta-nos mencionar aspectos relacionados a uma última técnica – a do processamento distribuído em rede (***networking***). Como são muito amplos, eles serão analisados na próxima seção. Vale notar que, neste caso, estaremos tratando não mais de apenas um computador isolado – que de agora em diante, denotaremos apenas por EP (elemento processador)[11] – mas da conjunção de várias máquinas SISD.

---

[9] Do ponto de vista da organização física, os processadores podem se valer de diferentes topologias, como totalmente conectada (cada processador intercomunica-se com todos os demais), grade (matriz simples), *double torus* (grade com bordas conectadas) ou *hiper-cúbica* (onde a distância máxima entre dois processadores é de valor logaritmico). Maiores informações, em [Foster, 1995] e [Hwang, 1998].

[10] Enquanto o modelo RAM (*Random Access Machine*) descreve o funcionamento básico da arquitetura clássica de von Neumann, sua extensão, chamada PRAM (*Parallel RAM*), propõe um modelo síncrono baseado na classe SIMD, formado por processadores idênticos com memória compartilhada. Suas operações teóricas de leitura e escrita, valem-se de dois tipos: concorrentes e exclusivas, gerando 4 combinações, com ordens de complexidade crescente: EREW, CREW, CROW e CRCW, havendo algoritmos especialmente projetados para o primeiro e último tipo (cf., por exemplo, [Cormen, 1996], [Hwang, 1998] e [Atallah, 1999]). Vale ainda ressaltar que os modelos BSP e CGM são considerados como modelos mais realísticos, por apresentarem resultados bem mais compatíveis com os parâmetros verificados nos experimentos [Cáceres, 2001].

[11] EP designará no texto um equipamento de hardware que apresenta uma unidade central de processamento responsável pela execução de instruções de um programa. Neste sentido, possui um caminho (*bus*) de dados –

♦ B. Sistemas MIMD

Deste ponto em diante, nos ocuparemos com a análise dos modelos pertencentes à classe MIMD de Flynn, já que a grande maioria dos sistemas de PAD enquadra-se desta forma. Por este motivo, daqui em diante mencionaremos alguns detalhes sobre este modelo, a começar pelas possíveis formas de classificá-lo.

➢ (i). Quanto ao Acoplamento

A arquitetura MIMD de Flynn pode ser subclassificada em 2 diferentes grupos, de acordo com o seu grau de acoplamento [Tanembaum, 1997]: sistemas com memória compartilhada (ou **fortemente acoplados**) e sistemas de memória distribuída (**fracamente acoplados**). A figura 1.5 (adaptada de [Tanembaum, *ibid.*]) mostra a taxionomia utilizada neste texto[12].

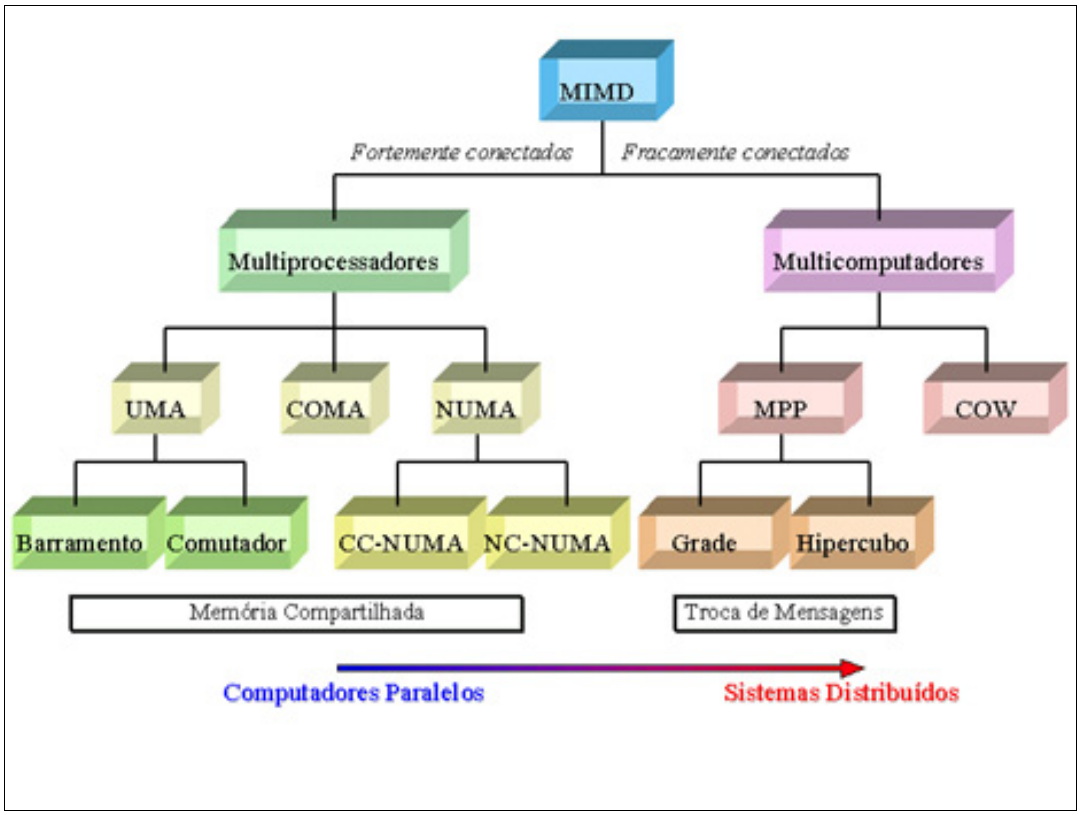


Figura 1.5: Classificação da arquitetura MIMD quanto ao acoplamento

- **Multiprocessadores (Memória compartilhada):** O conceito de **sistema fortemente acoplado** (que dá origem aos chamados **multiprocessadores**) [Enslow 1974], implica em EPs compartilhando recursos e com função

para interligar a entrada e a saída com a memória e uma unidade de controle (cf. definição apresentada em [Patterson, 1997]).

[12] [Hwang, *ibid.*], conforme anteriormente salientado, apresenta uma taxonomia um pouco diferente, embora os mesmos conceitos se encontrem definidos naquele texto. Nesta seção, tecemos considerações relativas apenas à classificação geral entre multiprocessador e multicomputador. O apêndice C.1 apresenta maiores detalhes sobre as arquiteturas incluídas nesta e naquela classe.

cooperativa. Isto, por sua vez, exige o uso de um *hardware* com **memória compartilhada** e de um sistema operacional específico, dependente do *hardware*. Sua grande vantagem reside na rapidez de obtenção dos dados entre os processadores. No entanto, além do custo, outras dificuldades deste modelo devem ser observadas: a limitação física para a expansão do número de processadores e a necessidade de técnicas de baixo nível para sincronização de leitura e gravação. A segurança do sistema é feita por mecanismos de controle, de forma que somente um processador pode acessar um endereço específico de memória por vez.

- **Multicomputadores (Memória distribuída):** Os **sistemas fracamente acoplados** (também chamado de **multicomputadores**) [Eckhouse, 1978] são caracterizados pela utilização de EPs independentes e interligados por uma rede, o que permite uma alta **escalabilidade**, ou seja, capacidade de crescimento e atualização do sistema. No entanto, existem também algumas desvantagens: o *overhead* de comunicação, já que os pacotes obrigatoriamente atravessam a rede e, principalmente, o fato de que o usuário é responsável pelo sincronismo e pelo recebimento dos dados (enquanto nos sistemas de memória compartilhada, o controle do usuário pode ser facilitado com o uso de monitores).

Além desta classificação (que leva em conta o grau de acoplamento), outros critérios podem ser considerados para organizar as arquiteturas existentes [Kirner, 1989], como mencionado a seguir[13].

➢ (ii). Quanto à Dependência dos Processadores e Granulosidade das Tarefas

Define-se **granulosidade** como sendo a computação executada pelos EPs que participam do processamento distribuído entre duas operações de interação quaisquer [Hwang 1998]. Neste sentido, percebemos que o tamanho do **grão** (ou seja, o intervalo de tempo em que um determinado código é processado isoladamente num EP) a ser utilizado é fortemente determinado pela organização física dos EPs e pelos seus mecanismos de comunicação. Desta forma, podem-se caracterizar dois grupos (semelhantes às definições dos sistemas fraca e fortemente acoplados, respectivamente):

[13] Uma classe adicional na classificação das máquinas MIMD quanto ao acoplamento envolve a idéia de sistemas DSM (*Distributed Shared Memory*), isto é, máquinas de memória distribuída que compartilham suas memórias locais. Este esquema é bem mais amplo que os modelos NC-NUMA e são implementados em um ambiente COW, em nível de *software*, sendo por isso chamados de SC-NUMA (*Software-Coherence NUMA*). Além disso, vale mencionar a existência de técnicas como a utilização de DSP's (*Digital Signal Processor's*), que atuam como coprocessadores de um processador central [Hwang, 1998]. Arquiteturas DSP e *transputers* não são analisados neste texto, pois o enfoque principal se destina a arquiteturas baseadas em multicomputadores.

- **Redes de Computadores e Sistemas Distribuídos[14]:** No qual os EPs são completamente independentes e podem se encontrar geograficamente espalhados. Desta forma, as tarefas (*jobs*) devem possuir alta granulosidade, isto é, pouca comunicação entre processadores e longos períodos de processamento. Isto se explica porque a existência de uma comunicação intensiva tenderia a comprometer fortemente o desempenho final.
- **Máquinas Paralelas[15]:** Neste caso, os EPs se encontram dentro de ambientes compactos (gabinetes), que são interligados através de um *hardware* específico, o que resulta numa velocidade consideravelmente maior de comunicação. Isto permite que o processamento faça possível uso de baixa granulosidade para as tarefas, sem excluir a possibilidade de utilização de grãos com tamanhos maiores. Máquinas como o `Cosmic Cube`, `nCube-2`, `iPSC/80`, `Symmetry`, `IBM SP`, `KSR-1`, `FX-8` e `CM-5` são enquadradas nesta categoria [Vianna 1998:176].

➢ (iii). Quanto à Sincronização

Outra característica importante dos sistemas paralelos-distribuídos é determinada pela existência ou não de mecanismos para sincronização e controle do processamento. Neste caso, apresentamos a seguinte classificação ([Kirner, 1989]):

- **Sistemas Síncronos:** O processamento ocorre de forma sincronizada no tempo. O modelo teórico PRAM, as arquiteturas *pipeline* e as máquinas SIMD com processadores matriciais enquadram-se nesta categoria.
- **Sistemas Assíncronos:** Cada EP trabalha de forma independente, necessitando, portanto, de algum mecanismo de sincronização entre eles. Constituem a grande maioria dos sistemas, tais como as máquinas MIMD.

Na seção que se segue, trataremos melhor dos conceitos relacionados com redes de computadores, que serão, em última instância, o ambiente em cima do qual teremos os *clusters* de processamento para a ferramenta WVM.

♦ C. Sistemas Distribuídos e Redes de Computadores

O conceito de **sistema distribuído** é definido como sendo "um conjunto de computadores independentes que, do ponto de vista dos usuários do sistema, funcionam como

[14] Conforme pode ser visto no apêndice C.1, as redes e sistemas distribuídos correspondem ao modelo COW dos sistemas multicomputadores.

[15] As máquinas paralelas enquadram-se no modelo MPP mencionado no apêndice C.1.

um único computador" [Tanembaum 1997:2][16]. Neste sentido, surgem dois aspectos distintos da arquitetura: uma independência em nível de *hardware* e uma integração em nível de *software*, o que lhe confere uma funcionalidade transparente. Resumidamente, suas vantagens residem nos seguintes pontos (cf. [Soares 1995:7-8], [Tanenbaum 1997:3-5]):

- **Economia:** Os investimentos necessários e os resultados obtidos têm demonstrado que os sistemas distribuídos apresentam uma das melhores curvas de custo / desempenho dentre as arquiteturas existentes;
- **Velocidade (Responsividade):** *Clusters* de processamento distribuído podem apresentar desempenho comparável a alguns dos maiores (e mais caros) sistemas existentes.
- **Modularidade (Escalabilidade):** O sistema como um todo pode ser facilmente expandido, permitindo aumentar ainda mais o poder computacional.
- **Confiabilidade:** A falha de um EP não destrói necessariamente o processamento como um todo, de forma que o sistema pode continuar processando;
- **Transparência:** Os recursos podem ser acessados pelos usuários, independentemente de onde estejam localizados.

Uma prova da emergência das redes no contexto do processamento (já que, sob outros aspectos, as redes – em especial a Internet - já revolucionaram definitivamente a civilização moderna) é o surgimento de estudos da chamada "*grid computing*", ou computação em rede, tida como uma das "infra-estruturas emergentes que mudarão fundamentalmente o modo como a computação é pensada e usada" [Foster *et al*, 1999]. Na realidade, as **malhas computacionais** equivalem transcendem o conceito das NOWs, permitindo que vários aglomerados (*clusters*) possam participar de um macrosistema de processamento.

Por outro lado, não se pode deixar de mencionar alguns pontos fracos existentes, tais como as dificuldades de gerência dos recursos, a complexidade dos *softwares* e programas utilizados e as baixas velocidades de comunicação em redes heterogêneas. [Tanenbaum, 1995] menciona ainda outros problemas como saturação da rede e brechas de segurança. Fatores como estes não podem ser ignorados, ainda que cada um deles esteja sendo paulatinamente corrigido. Vale observar, contudo, que as vantagens apresentadas continuam sendo extremamente atrativas para a adoção das redes como um todo. [Eckhouse, 1978] define diferentemente um sistema distribuído como "um conjunto de módulos processadores (MPs) conectados por uma rede de comunicação". Este conceito irá se aproximar da definição de

[16] Os sistemas distrubuídos irão definir os ambientes DSM (*Distributed Shared Memory*) anteriormente mencionados – também classificados como SC-NUMA.

redes de computadores dada anteriormente, enquanto conjunto de MPs com capacidades de comunicação e compartilhamento de recursos [Soares 1995:10]. Os elementos mais importantes deste sistema de comunicação são os meios de transmissão - enlaces físicos - e os protocolos.

Nos dedicaremos basicamente ao estudo da utilização de sistemas distribuídos em redes, envolvendo tanto aglomerados (*clusters*), quanto arquiteturas paralelas, para a obtenção de processamento de alto desempenho. Antes, porém, verificaremos melhor um outro ponto de extrema relevância ao assunto: o *software*, isto é, as características das aplicações e programas utilizados nestes ambientes.

1.2. Considerações de *Software*: Sistemas e Aplicações Distribuídas

Na seção anterior, tecemos considerações relacionadas com o *hardware*, incluindo aspectos de sua arquitetura e classificação. Agora, deslocaremos nossa atenção para o *software* dos sistemas computacionais. Isso será importante porque, como veremos, poderemos construir ambientes de PAD, através de *clusters* de computadores interligados em rede tendo em comum programas ou bibliotecas de comunicação para apoio ao processamento.

1.2.1 – Breve Histórico

Ao mesmo passo em que o *hardware* evoluía, podemos observar um desenvolvimento cada vez mais aprimorado do *software*. Na tabela 1.4, apresentamos uma síntese da evolução do *software* ocorrida nas últimas décadas (adaptada de [Tanenbaum, 1995, 1999]).

Geração	Descrição
1ª (1945-1955)	Programação em código absoluto (em painéis de *hardware*) – Inexistência de SOs e LPs – Maior evolução: Introdução de cartões perfurados.
2ª (1955-1965)	Surgimento dos sistemas em lote (*batch*) – Surgimento dos primeiros SOs (`FMS` e `IBSYS`) – Utilização das primeiras linguagens (`Fortran`)
3ª (1965-1980)	Surgimento de compatibilidade entre os sistemas (`IBM` System/360) – Multiprogramação - Aparecimento da técnica de SPOOL – Surgimento dos primeiros sistemas de tempo compartilhado (*time sharing*) – Desenvolvimento do `MULTICS`, predecessor do `Unix` – Padronização (ANSI/ISO) das LPs.
4ª (1980-???)	Surgimento de *softwares* amigáveis (*user friendly*) – Desenvolvimento de interpretadores de comandos (*shells*): `DOS`, `Unix` - Explosão dos ambientes GUI (Mac, `Windows`) – Surgimento de ambientes gráficos para desenvolvimento (IDEs) e linguagens visuais.

Tabela 1.4: Comparação entre modelos das gerações de computadores

O surgimento da engenharia de software, com o desenvolvimento de uma metodologia mais formal para a construção de sistemas e aplicações, contribuiu imensamente para atenuar os elevados custos de criação e manutenção de programas. Esse enfoque, sentido

tanto em termos de *software* básico (sistemas operacionais, *drivers* de dispositivos, etc.) quanto aplicativo (editores, planilhas, sistemas de controle, etc.) possibilitou o surgimento de uma visão mais madura na lógica dos sistemas computacionais e na forma de como melhor aproveitar o potencial do *hardware* sem, necessariamente, prender-se a ele no nível de programação (como ocorria nas primeiras gerações).

Um exemplo da evolução ocorrida pode ser verificado com relação às linguagens de programação (LPs), que, se deslocando do nível de dependência da máquina (típico das linguagens de montagem), passou a ter enfoque nos nível mais próximos das necessidades do usuário e da melhor representação de sua realidade. Diversas visões de como melhor adaptar as características das LPs surgiram e deram origem aos vários paradigmas de programação hoje existentes, apresentados esquematicamente na tabela 1.5 (adaptada de [Brookshear 1998] e [Sebesta 2000]).

Paradigma	Características	Exemplos
Imperativo	Procedimental. Baseia-se no ciclo tradicional de execução de instruções: busca – decodificação – execução. Implica obrigatoriamente numa análise do problema.	`Fortran` – `Cobol` – `Algol` – `Basic` – `APL` – `C` – `Ada` – `Pascal`
Declarativo	Fundamenta-se na natureza do problema, buscando-se um algoritmo geral para solucioná-lo. Utiliza a lógica formal para expressar o problema.	`GPSS` – `Prolog`
Funcional	Os programas são construídos em blocos funcionais, a partir de primitivas que são conjugadas para desenvolver os programas, através de uma visão *top-down*.	`LISP` – `ML` – `Scheme`
Orientação a Objeto	Dados vistos como objetos com métodos próprios, apresentando vinculação dinâmica (herança múltipla e polimorfismo) e encapsulamento.	`Simula` – `Smaltalk` – `C++` - `Ada95` – `Java`

Tabela 1.5: Paradigmas Tradicionais de Programação

Merece destaque o crescimento do nível de abstração – especialmente em se considerando o paradigma OO (Orientação a Objetos) – que tornou possível uma melhor modelagem dos sistemas, com uma sensível economia nos custos de desenvolvimento e, em especial, manutenção de códigos. Além disso, o surgimento de linguagens voltadas para ambientes distribuídos, portadoras de bibliotecas e facilidades para suporte à rede, abriram caminho para que o processamento de alto desempenho pudesse se tornar realidade a investimentos cada vez menores e de uma maneira cada vez mais próxima aos usuários.

1.2.2 – Classificação dos Sistemas

Dentre as muitas formas de estudo para analisar a relação entre *hardware* e *software*, pode-se observar o papel dos sistemas operacionais (SOs) e classificá-los em três categorias, com suas características próprias, conforme ilustrado na figura 1.6 [Tanenbaum 1997:16-22][17]:

[17] O fato de levarmos em conta aspectos relativos ao SO e, paulatinamente nos deslocarmos para o contexto das aplicações distribuídas, deriva basicamente da forma estruturada dos vários níveis de máquina e sua ordem

Figura 1.6: Classificação dos Sistemas Operacionais

- **Sistemas Multiprocessados:** Correspondem ao maior grau de dependência lógica entre os processadores, refletindo a arquitetura física dos sistemas multiprocessadores onde operam. Nesta situação, existe apenas uma única cópia do mesmo sistema operacional executando nos múltiplos processadores. A comunicação é feita, portanto, através de ambientes fortemente acoplados. São sistemas proprietários, específicos para cada máquina e extremamente caros.
- **Sistemas Distribuídos:** Neste nível intermediário, temos a cópia de um mesmo sistema operacional sendo executada em cada processador. Embora não exista mais dependência em nível de *hardware*, os usuários ainda têm a impressão de que operam em cima de um único computador. A comunicação entre os processadores é feita através de mensagens internas do sistema.
- **Sistemas em Rede:** Finalmente, a configuração mais utilizada, existente em sistemas fracamente acoplados (comum em redes locais - LANs) é a de sistemas operacionais em rede. Aqui, cada módulo processador possui seu próprio sistema que pode, inclusive, ser distinto dos demais. A comunicação se torna possível através de protocolos comuns, em nível de camada de rede e transporte (como o `TCP/IP`), os quais permitam reconhecer recursos compartilhados ou outros computadores conectados. Desta vez, no entanto, o nível de transparência é bem menor, de forma que os usuários muitas vezes devem conhecer **onde** se encontram os recursos remotos. Embora isto possa ser um problema, a utilização desta alternativa permite utilizar o parque de máquinas já existente, conferindo um baixíssimo custo ao sistema como um todo.

Considerando esta classificação, nosso estudo aqui se centralizará nas aplicações distribuídas processadas em sistemas em rede. Tanto as bibliotecas de paralelização quanto os problemas que enfocaremos aqui utilizam EPs independentes, conectados entre si, seja em redes locais, *clusters* de processamento ou comutadores (switches) de alta velocidade. Desta forma, nos distanciamos ainda mais dos níveis mais baixos das camadas de rede (física, enlace, rede e transporte), passando a estudar o comportamento do sistema em nível mais

---

crescente de complexidade: lógica digital, microarquitetura, conjunto de instruções, sistema operacional, linguagem de montagem e linguagem de alto nível (maiores informações em [Tanenbaum, 1999]).

elevado (camadas de sessão, apresentação e, principalmente, aplicação). Será por isso que, a seguir, passaremos ao estudo das aplicações distribuídas.

1.2.3 – Modelo Funcional C/S

O modelo **cliente-servidor** (C/S) é, sem dúvida, o mais conhecido paradigma para provimento de serviços em rede, amplamente usado pela sua simplicidade e eficiência. Sua idéia reside em grupos de processos cooperativos que se comunicam, suprindo serviços. Conforme fica explícito, os **clientes** são os processos que requisitam determinada aplicação ou recurso, enquanto o **servidor**, por sua vez, é responsável por atender aos clientes, provendo o serviço solicitado[18] (figura 1.7).

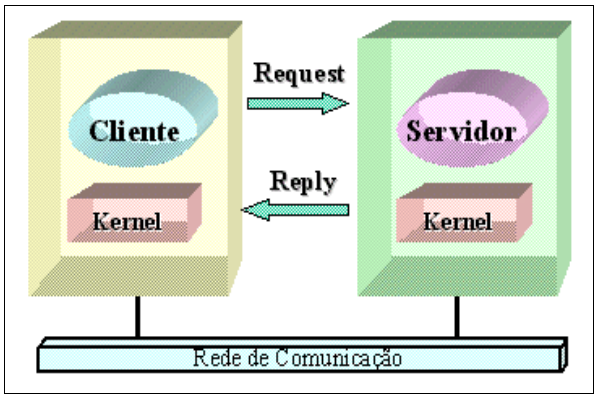


Figura 1.7: Modelo Cliente-Servidor

Um elemento importante aqui é que são os **processos** (aplicações) que definem quem requisita e quem provê determinados serviços, de forma que em sistemas em rede, máquinas podem ser (e normalmente o são) clientes de alguns serviços e servidoras de outros serviços para outros *hosts*. Note que, devido à sua simplicidade, o modelo permite a implementação de qualquer protocolo do gênero "requisição/resposta" (do inglês, ***request/reply***) orientado ou não à conexão. Uma outra facilidade é a abertura para a invocação remota de procedimentos, conforme será visto mais adiante. Este será o modelo a ser utilizado nas aplicações distribuídas, conforme veremos a seguir[19].

1.2.4 – Aplicações Distribuídas

- A. Definição

---

[18] Várias modificações podem ser feitas sobre o modelo C/S clássico, incluíndo um ou mais níveis intermediários (chamados de *middleware*) entre o cliente e o servidor. O objetivo é o de especializar funções nas camadas intermediárias, ao mesmo tempo em que alivia a sobrecarga de métodos no servidor. Um exemplo clássico são os ORBs (*Object Request Brokers*) do CORBA [Orfali, 1998].

[19] Maiores detalhes sobre o paradigma C/S, podem ser encontrados em [Tanenbaum, 1995], o qual fornece considerações sobre as várias classes de primitivas de comunicação (blocantes, "*bufferizadas*" e confiáveis), fornecendo ainda primitivas para uma implementação básica do modelo.

Denominamos **aplicações distribuídas** a "aplicações cujo processamento é feito através de múltiplos computadores conectados em rede, estando ou não geograficamente distribuídos" [Farley, 1998]. Elas são capazes de servir concorrentemente a vários usuários e, dependendo do projeto, podem otimizar a utilização dos recursos. Aplicações distribuídas são usadas constantemente pelos usuários: elas vão desde a simples navegação na *Web* (onde a comunicação cliente-servidor é efetuada via HTTP) e o uso do correio eletrônico, a sistemas multimídia de conversação, teleconferência ou trabalhos cooperativos, como os chamados "*whiteboards*" (telas gráficas compartilhadas entre dois ou mais usuários)[20]. Em linhas gerais, o particionamento de atividades (que será visto mais adiante, na seção relacionada com paralelismo), a necessidade de redundância em sistemas que necessitam de tolerância a falhas e a consulta a servidores remotos são boas motivações para a implementação de aplicações distribuídas.

Sistemas como estes têm sido aplicados a vários setores do mercado, seja da indústria ou de serviços, e em várias instituições de pesquisa. A união de tais aplicações com tecnologias como *data warehouse* (bancos de dados), sistemas especialistas (inteligência artificial) e redes de alta velocidade (telecomunicações) foram responsáveis pelo surgimento de serviços que vão desde o *home-banking* à visualização científica cooperativa. Mas apesar dos avanços, muitos acreditam que o que vemos atualmente, é, no entanto, apenas uma leve sombra do que se nos espera já nos próximos anos [Foster, 1995]. Vale, observar contudo, que, no caso das aplicações distribuídas, a falha de EPs remotos (muitas vezes, não conhecidos pelo usuário), pode resultar no término anormal de execução[21].

♦ B. Desafios

O problema de se implementar uma solução cooperativa (aqui no sentido de distribuída) para o processamento, portanto, não pode ser visto como algo trivial. Muitos fatores de considerável complexidade influem na política de controle dos processos, bem como na otimização de seu funcionamento. Isto porque, para além da complexidade teórica, uma gama de elementos das mais diversas naturezas dificultam, na prática, a implementação e o desempenho de um sistema distribuído como um todo:

- **Em nível de rede:** heterogeneidade das arquiteturas, topologia, conectividade e outras características físicas;

[20] Os "*whiteboards*" destacaram-se dentre as primeiras aplicações cooperativas, tanto pela sua potencial aplicação – já que os usuários poderiam construir objetos complexos (e não apenas operar sobre textos, como no caso dos IRC's) – quanto pela sua simplicidade – graças à existência de um servidor multitarefa e de clientes capacitados para a execução de chamadas remotas (normalmente feitas utilizando RMI). Maiores informações podem ser encontradas em [Farley, *ibid.*].

[21] Obviamente, tais características dependem da natureza e estrutura da aplicação distribuída, bem como do suporte fornecido pelo sistema e pela rede, conforme mencionado.

- **Em nível de elementos processadores:** limitações de *hardware* (memória, discos, etc.), arquitetura interna, acessibilidade aos recursos;
- **Em nível de software:** restrições do sistema operacional, protocolo de rede, serviços e programas disponíveis (paralelização automática ou bibliotecas de comunicação), etc.;
- **Em nível do problema:** dimensionamento, complexidade, granulosidade e tipos de dependências existentes (conceitos que serão vistos em seguida).

Dois são os fatores determinantes para um bom desempenho de um sistema de processamento distribuído: a **computação** e a **comunicação** envolvidas [Fox, 1997]. Vale observar que, enquanto o primeiro atua positivamente, o outro ajuda a denegrir o desempenho. Com os avanços da tecnologia, tanto uma quanto outra têm sofrido consideráveis melhorias. No entanto, muitos desafios permanecem e ainda estamos longe de uma solução definitiva para o problema da computação distribuída, conforme veremos a seguir.

♦ C. Componentes de Aplicações Distribuídas

As aplicações distribuídas podem ser formadas por diferentes elementos que podem coexistir dentro de uma mesma aplicação [Farley, 1998]:

- **Processos:** Classicamente, designam programas em execução, com capacidades de acesso a recursos da máquina (memória, disco, etc.). Ainda em sistemas com apenas um processador, técnicas como *interleaving* (acesso independente da memória), *pipelining* (execução independente de ciclos de processamento) e *caching* (acesso mais rápido de dados e instruções) permitem a execução mais rápida de processos no sistema, desde que suportado pelo SO.
- ***Threads:*** São fluxos de execução dos processos que operam independentemente entre si. Cada programa possui pelo menos uma *thread*. No entanto, sistemas operacionais multitarefas podem permitir a execução de várias *threads* simultâneas dentro de um mesmo processo. As *threads* podem ser controladas por mecanismos de sincronização que tornam possível manter a coerência e acessar de forma adequada as regiões críticas do sistema.
- **Objetos:** São os elementos fundamentais das linguagens OO (como `C++` e `Java`), identificando dados e métodos que modelam instâncias com atributos e propriedades característicos. Um processo pode ser caracterizado por vários objetos.
- **Agentes:** São elementos funcionais com certas finalidades dentro de uma aplicação distribuída. Alguns autores adicionam características como **inteligência** e **autonomia** [Bigus, 1998]. No entanto, podemos simplesmente considerá-los como

objetos com funções particulares dentro do sistema, capazes de serem executados em *threads* específicas. Uma aplicação distribuída pode, em última instância, ser considerada como uma coleção de agentes trabalhando para alcançar um determinado resultado.

- ***Hosts:*** Correspondem às máquinas sobre as quais processos e agentes executam. São elementos especialmente importantes quando **programação explícita** é considerada. No decorrer de nosso trabalho, irão constituir o chamado **Domínio de Processamento**.

A soma do conjunto de elementos acima resulta, portanto, em uma **aplicação**, que corresponde aos "códigos a serem executados pelos agentes no sistema remoto" [Farley, *ibid.*]. Note que as aplicações constituem, de fato, no elemento mais importante e devem ser acessados a partir de um local (***site***[22]) previamente definido pelo usuário.

♦ D. Requisitos para Aplicações Distribuídas

Nem todo problema pode ser resolvido através de uma solução distribuída. Existem critérios para que isto se torne em primeiro lugar, **possível** (de acordo com fatores como a dependência de dados) e, em segundo lugar, **viável** (dependendo de elementos como a comunicação). Alguns pontos devem ser, pois salientados:

- **Particionamento:** O critério mais importante é que um determinado problema possa ser dividido em diferentes partes capazes de serem executadas concorrentemente. Esta atividade, que, auxiliada com a atividade de escalonamento de tarefas, serve de base para o desenvolvimento de aplicações distribuídas, será chamada, de agora em diante de "**paralelização**". Note que ela se constitui no primeiro passo a ser dado, sendo necessário que sejam consideradas todas as variáveis do problema: natureza dos dados, tipos de procedimentos e dependências. O particionamento pode ser feito tanto em nível de objetos, *threads*, procedimentos ou dados, dependendo tanto da natureza da linguagem de programação usada, quando das características do sistema operacional. Este assunto será melhor estudado no próximo capítulo, que trata de processamento de alto desempenho;
- **Comunicação:** A principal variável que determinará a viabilidade de um problema particionado é o nível de comunicação (intenso, periódico ou elementar) entre os EPs. Isso se torna problemático quando consideramos que estamos trabalhando sobre sistemas em rede. A comunicação depende tanto da natureza do

[22] O termo inglês *site* pode ser naturalmente substituido pelo equivalente em português "sítio", no entanto, foi mantida a notação mais usual (hoje comum), embora em lingua estrangeira.

particionamento, quanto dos protocolos de nível intermediário (rede e transporte) existentes.

- **Multitarefas:** Outra variável importante é a natureza das tarefas utilizadas para solucionar o problema: o uso de múltiplas *threads* (desde que permitidas pelo sistema), podem aumentar a eficiência da aplicação, otimizando o uso de CPU, alocação de recursos e largura de banda utilizada. Em alguns casos, as *threads* não são apenas *convenientes*, mas *necessárias*, principalmente quando são considerados agentes que operam assincronamente, em ambientes heterogêneos. Um particionamento errado pode fazer com que estas tarefas distribuídas, inadequadamente dimensionadas, criem verdadeiros gargalos no sistema, diminuindo a eficiência pretendida.
- **Segurança:** O aspecto mais delicado é o que diz respeito à segurança, tanto pelas transações de informações na rede, quanto pela necessidade de privacidade e autenticação dos usuários. Situações em que dados são compartilhados (em que o uso de áreas de sincronização são necessárias) merecem especial atenção. Uma segurança mínima para aplicações distribuídas deve permitir que agentes possam ser autenticados, definir os recursos acessíveis a estes agentes e permitir uma comunicação confiável. O uso de técnicas como encriptação de dados, pode ser tanto recomendável quanto necessária, dependendo da natureza do problema. Além deste critério, questões relativas à confiabilidade da própria aplicação, devem ser cuidadosamente analisadas, a fim de garantir sua adequada execução.

♦ E. Alternativas de Implementação

Desde o seu surgimento, várias alternativas têm sido propostas para a implementação de aplicações distribuídas. Elas podem ir desde níveis mais baixo de comunicação (RPCs e DCE) a paradigmas de orientação a objetos (como CORBA, RMI ou OpenDoc). Eis as características de algumas destas propostas:

- ***RPCs:*** As *Remote Procedure Calls* (Chamadas de Procedimento Remoto) foram o mecanismo idealizado por Birrell e Nelson (1984) e patenteado pela Sun Microsystems (1988) para facilitar o desenvolvimento de aplicações distribuídas em arquitetura Cliente-Servidor (cf. figura 1.8).

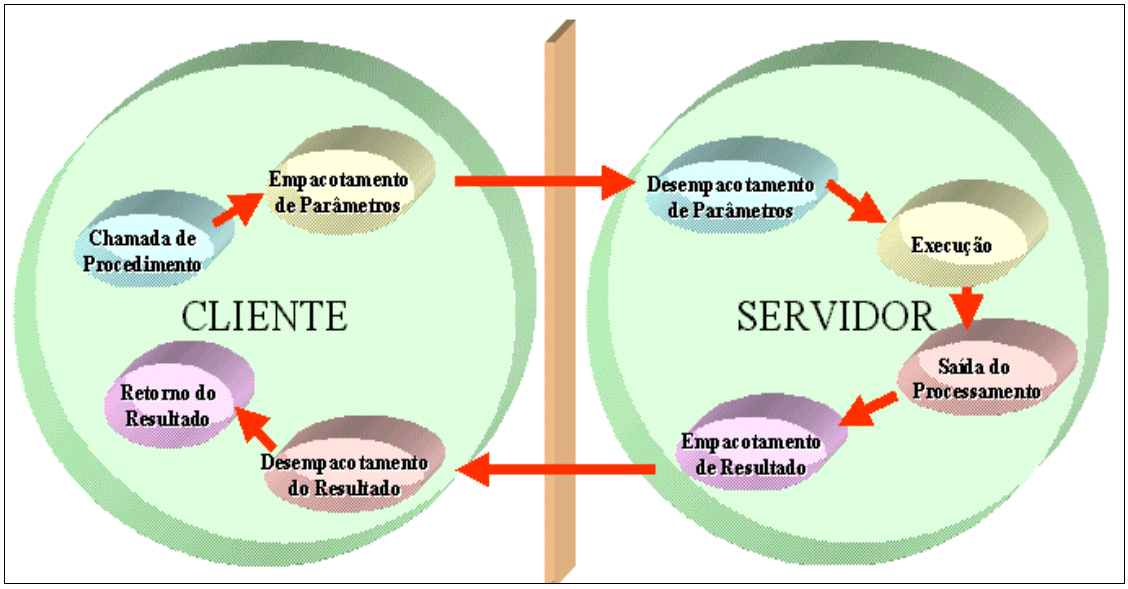


Figura 1.8: Modelo de Funcionamento das RPCs

A idéia básica das RPCs é a de que um processo de uma máquina (cliente) pode invocar **remotamente** a execução de um procedimento existente em outro EP (o servidor), de forma que o programador não necessita manipular diretamente as mensagens, bastando definir quais os procedimentos compartilhados (isto é, registrados), a fim de poderem ser executados por outras máquinas. Note que a comunicação é possível graças à formatação dos parâmetros e dados repassados de um EP a outro.

- ***DCE*** *(Distributed Computing Environment):* Diferentemente de um sistema operacional distribuído, o DCE é, na realidade, um ambiente de computação que executa sobre um SO já existente. Propriedade da OSF (Open Software Foundation), este ambiente visa basicamente prover uniformidade a sistemas distintos, executando no espaço do usuário e fornecendo um grande número de ferramentas para auxiliar o processamento distribuído entre diferentes máquinas (*hosts*).

Figura 1.9: Camadas do DCE

Sua organização modular (figura 1.9) permite que os múltiplos serviços existentes possam ser facilmente configurados e administrados. O DCE tornou possível algo até então impraticável: uma uniformidade na configuração de ambiente para os usuários, independente de máquina e de arquitetura. No entanto, o sistema necessita ser suportado adequadamente pelo sistema operacional, para que todas as suas funcionalidades sejam possíveis.

- ***RMI** (Remote Method Invocation):* Análogo às funcionalidades dos RPCs, o RMI é outra interface proposta pela Sun Microsystems para a implementação de chamadas remotas, desta vez em Java (figura 1.10).

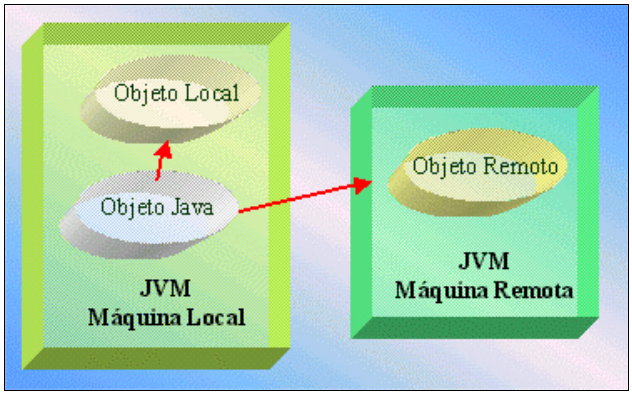


Figura 1.10: Modelo de Funcionamento do RMI

Além da independência de plataforma da linguagem, outra vantagem da RMI é a utilização do paradigma de orientação a objetos (OO). Uma observação importante é que, apesar da facilidade de implementação, o uso de RMI é restrito a ambientes de programação que utilizem Java, o que reduz consideravelmente a sua utilização na prática.

- ***CORBA:*** O *"Common Object Request Broker Architecture"* (ou "arquitetura para corretagem de requisições de objetos comuns") representa um esforço unificado de desenvolvedores em padronizar o modelo de objetos distribuídos. Resulta dos esforços de um consórcio - o Object Management Group (OMG) - formado por "pesos pesados" da indústria de *software* (incluindo a Sun Microsystems como um dos fundadores). O CORBA não é um produto, mas uma especificação aberta, independente de plataforma e linguagem. Sua independência de implementação é a grande vantagem em relação ao RMI. É formado basicamente por ORBs (Object Request Brokers) – canais que permitem a comunicação entre objetos; interfaces

para especificação de objetos distribuídos (como IDLs[23]) e um protocolo de comunicação entre ORBs (IIOP – Internet Inter-ORB Protocol). Tais elementos são mostrados esquematicamente na figura 1.11.

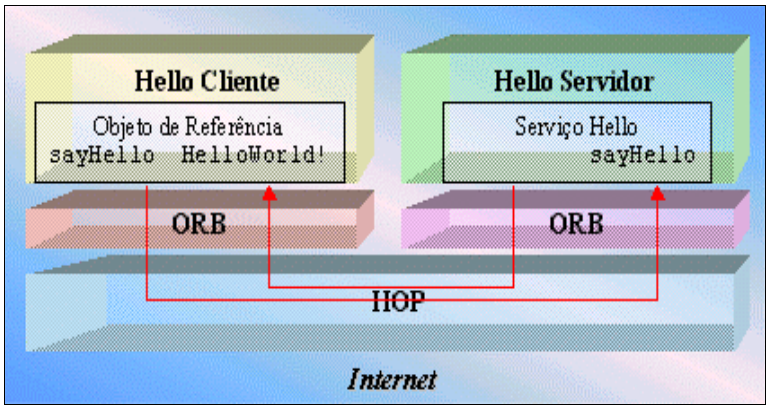


Figura 1.11: Elementos da arquitetura CORBA

- ***DCOM*** *(Distributed Component Object Model):* A proposta do *Distributed Component Object Model* (DCOM) da Microsoft é análoga ao CORBA, mas conta com o benefício da imensa base de sistemas Windows hoje existente[24]. Sucintamente, o modelo fornece uma visão modular, separando a interface do objeto de sua implementação, trabalhando com negociação para a definição de acesso aos objetos. Maiores informações, podem ser obtidas em [Orfali, 1998]. Note que, no caso das alternativas proprietárias, encontramos peculiaridades e restrições, normalmente presas às suas características do fornecedor, de forma que não existe ainda um concenso definitivo para a questão relativa ao processamento distribuído (figura 1.12).

[23] A IDL ou Interface Definition Language se constitui, como o próprio nome diz, numa linguagem para definição de interfaces. Sua grande vantagem está no fato de ser independente de implementação, sendo suportada por boa parte das atuais linguagens de programação com suporte a OO, como C++ e Java.

[24] Obviamente, existem diferenças patentes. Por exemplo, o modelo DCOM se apóia na IDL do *Distributed Computing Environment* (DCE), não sendo compatível com o CORBA e nem suportando herança múltipla.

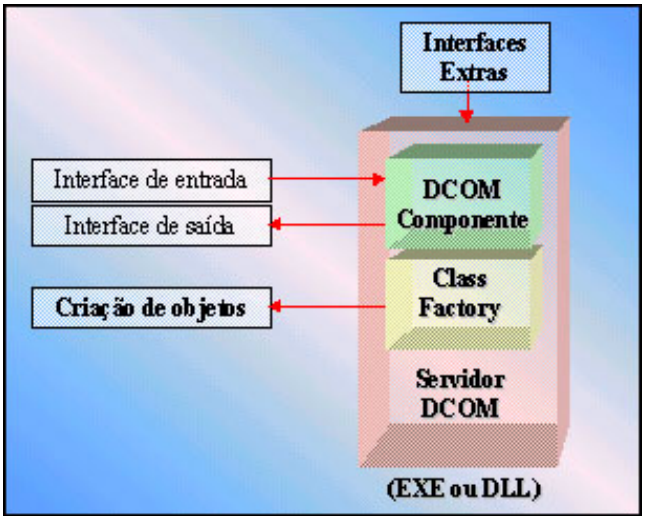


Figura 1.12: Objeto DCOM (estrutura do servidor)

Vale observar ainda que certos modelos apresentados até o momento, podem ser, de alguma forma, incorporados a outro. Por exemplo, o uso de `Java IDL` permitiria a utilização de `CORBA` na JVM; já no caso da biblioteca de mensagens `PVM`, utilizamos um modelo interno de `RPC`s para a execução de procedimentos remotos. Desta forma, a cobinação de alternativas surge muitas vezes como a solução para o desenvolvimento das aplicações distribuídas.

O próximo capítulo irá mencionar detalhadamente duas alternativas de baixo custo a serem utilizadas no escopo desta dissertação: as bibliotecas paralelas e a computação distribuída com `Java`.

Resumo

Neste capítulo, apresentamos alguns conceitos básicos e uma breve descrição das arquiteturas, em termos de hardware e software, alguns dos quais necessários ao longo da dissertação. Desde o início do capítulo, vimos como a tecnologia tem apresentando alternativas para a solução do problema de velocidade do processamento. No próximo, consideraremos as questões relativas à junção de alternativas para a obtenção de processamento de alto desempenho.

Capítulo 2

Programação Paralela - Distribuída

Introdução

Neste capítulo, teceremos detalhes sobre uma das duas alternativas mais utilizadas para PAD e que será bastante utilizada na ferramenta apresentada nesta dissertação: o processamento paralelo baseado no uso de bibliotecas de passagem de mensagem. Nosso objetivo aqui consiste em levar em conta os requisitos mínimos de hardware e software (alguns dos quais mencionados no capítulo anterior) e que serão necessários para possibilitar a utilização de uma rede local ou cluster, como um computador virtual. Para tanto, daremos um embasamento nas principais questões relacionadas com este assunto, indo desde o conceito propriamente dito de processamento de alto desempenho, à programação paralela e, por fim, tecendo detalhes sobre as bibliotecas de passagens de mensagens (vide modelo na figura 2.1)[25].

Figura 2.1: Enfoque do capítulo 2.

[25] O diagrama mostrado considera apenas uma divisão didática da ordem a ser tratada neste capítulo: em nenhum momento, no entanto, restringe-se a programação paralela como a única alternativa ao PAD, ou considera-se a solução de *message-passing* como sinônimo para a programação paralela (que poderia envolver, por exemplo, ferramentas automáticas de paralelização de códigos). Observe que a utilização de computadores vetoriais (mencionado no Capítulo 1) e de programação distribuída via Internet (cf. Capítulo 3), são propostas para obtenção de PAD que não são consideradas neste capítulo.

2.1. Processamento de Alto Desempenho

2.1.1 – Definição

Até recentemente, o termo "**supercomputação**" era utilizado para designar sistemas de elevado poder computacional. Em geral, coincidiam com máquinas de grande porte, extremamente caras, com elevado custo de manutenção e com SO e linguagens proprietários. No entanto, esta definição está irremediavelmente sujeita ao estado tecnológico da época: o que era um "super" ontem, não o é hoje e, muito menos, o poderá ser amanhã. Um exemplo disso são os nossos PCs de hoje em dia: sua velocidade e capacidades de armazenamento são equivalentes aos poderosos computadores de algumas décadas atrás. Não é de se surpreender, portanto que, paulatinamente, o deslocamento das atenções foi migrando de um sistema centralizado onde imperava o *mainframe* (vide tabela 1.2) para sistemas distribuídos ou em rede (como ocorre nas NOWs), formados por estações de trabalho interconectadas, com desempenho comparável aos grandes computadores.

O termo **Processamento de Alto Desempenho** (PAD) – do inglês, *High Performance Computing* (HPC) – apresentado sucintamente na seção 1.1.1, define esta nova visão em que *hardware* e *software* se conjugam com as novas tecnologias (como modelos de programação, redes de alta velocidade e técnicas de otimização), em busca de um único objetivo: rapidez no processamento[26]. Porém, como já mencionado, isto vai muito além da utilização isolada de equipamentos de última geração: envolve o uso de novas tecnologias para comunicação e processamento como um todo.

Deve-se salientar que o alto desempenho (ou otimização) provém, quer da velocidade de processamento, quer da relação maximizada entre velocidade e custo (cf. [Brasil, 1999a]). Aplicações típicas de PAD são as de engenharia complexa, como por exemplo, projeto estrutural de veículos e plantas industriais, prospecção de petróleo, previsão de tempo e clima, pesquisa da estrutura da matéria, geração de imagens para desenho animado e efeitos especiais em cinema, etc. (cf. [Brasil, *ibid.*])[27].

2.1.2 – Alternativas Atuais

---

[26] De acordo com o chamado "Livro Verde" da Sociedade Brasileira da Informação (SOCINFO), *High Performance Computing* (HPC) corresponde 'à tecnologia de computação, abrangendo equipamentos e programas de computação, visando a otimização dos serviços computacionais para determinadas aplicações ou tipos de aplicações".

[27] A abrangência do tema tem saído das esferas puramente técnicas e alcançado os planejamentos governamentais, sendo tratado como verdadeira política de desenvolvimento. Para se obter melhor compreensão sobre a atualidade do tema, vale consultar algumas referências adicionais relativas ao modelo americano (baseado no chamado "High-Performance Computing Act", de 1991) e ao programa brasileiro (com base no Projeto da Sociedade Brasileira de Informação – SOCINFO).

Apesar de estar se concretizando como uma tendência mundial, a utilização de sistemas de PAD ainda hoje constitui um desafio. Por um lado, apesar dos Centros de Processamento existentes[28], seu uso atual – essencialmente acadêmico – pode representar uma dificuldade e um desafio a mais para a sua absorção pelo mercado. Uma outra alternativa, consiste em montar ambientes de alto desempenho exclusivamente corporativos, o que, contudo, pode apresentar riscos e necessidade de investimentos ainda maiores. Assim, nos deslocamos daquela que seria a pergunta central da questão ("Como obter um processamento eficiente?") para considerarmos as limitações impostas pela relação custo/desempenho de um dado ambiente. Ou seja, recaímos em uma questão prática: "Como aproveitar ao máximo os recursos computacionais existentes?"

Por trás dessa interrogação, deve-se definir quais os recursos disponíveis, bem como quais as características das aplicações a serem executadas, suas restrições e as ferramentas possíveis para isto. Além disso, todas as restrições anteriormente mencionadas para as aplicações distribuídas (como particionamento, comunicação e segurança), devem ser cautelosamente analisadas. Várias podem ser as soluções possíveis para resolver o desafio representado pelo PAD. Cada uma delas é influenciada pela relação custo × benefício advinda de sua implantação dos sistemas e do desenvolvimento das aplicações, que é particular para cada caso. Alguns exemplos são mencionados a seguir:

- A. Em nível de *hardware*

Várias alternativas de *hardware* podem ser empregadas na busca de PAD. O Capítulo 1 ilustrou algumas das alternativas para isso. Normalmente, hoje encontramos duas possibilidades (não mutuamente exclusivas):

- **Aquisição de computadores vetoriais ou paralelos:** Alternativa de altíssimo custo, que pode se tornar necessária para organizações que necessitam de elevado poder de processamento. A utilização desta tecnologia pode se justificar em situações onde a massa de dados a ser computada pode ser vetorizada ou em situações nas quais o tempo de obtenção das respostas é crítico[29];
- **Criação de *clusters* de alta velocidade:** Onde servidores multiprocessados e máquinas de ultima geração interligadas por meio de uma rede de alta velocidade, podem criar um ambiente propício para computação distribuída.

[28] Como ocorre nos CENAPADs, a serem mencionados no último capítulo.

[29] Um outro aspecto no qual arquiteturas proprietárias normalmente apresentam considerável melhoria em relação aos *clusters*, reside na segurança e privacidade dos dados, tanto pelo fato de que tais máquinas possuem acesso mais restrito, quanto pelo fato de que em ambientes multicomputadores, normalmente algumas brechas podem ser mais facilmente encontradas.

Note que, conforme mencionado, a utilização de COWs (NOWs) está crescendo consideravelmente. Boa parte dos grandes projetos da atualidade residem em ambientes de *clusters* (conforme ilustrado por [Patterson, 1997]).

♦ B. Em nível de *software*

Em termos de *software*, normalmente nos deparamos com duas alternativas distintas:

- **Softwares de paralelização automática**: São ferramentas para geração automática de códigos, de acordo com especificações bem delimitadas do *hardware* (sendo, portanto, proprietárias e de altíssimo custo). Desta forma, tais programas somente funcionam em arquiteturas homogêneas, trazendo como benefício principal, a transparência para os desenvolvedores[30];
- **Bibliotecas de comunicação:** Por outro lado, englobam *softwares* diversos, podendo operar em arquiteturas heterogêneas, com implementação de baixíssimo custo e a vantagem da escalabilidade e a possibilidade de interoperabilidade. As bibliotecas de passagem de mensagem (*message-passing libraries*) encontram-se neste grupo e, dentre estas, destacaremos o PVM e o MPI, a serem melhor analisados adiante.

Observe que, dependendo da situação, determinadas arquiteturas de *hardware* exigem a contrapartida de *software* correspondente. Por exemplo, em computadores com memória compartilhada, a necessidade de um SO multiprocessado é determinante, em especial quando se considera a complexidade e o custo da arquitetura[31]. Na seção que se segue, daremos lugar a um estudo um pouco mais detalhado da alternativa do paralelismo, com ênfase na utilização de *clusters* e de bibliotecas de comunicação.

2.2. Processamento Paralelo

2.2.1 – Definição

Dentre as técnicas de programação utilizadas em vista de um PAD, o **processamento paralelo** é uma das alternativas que tem obtido êxito na resolução de grandes desafios computacionais [Zomaya, 1996], graças à demanda crescente por alto desempenho, baixos

---

[30] Uma técnica conhecida como *Data Parallelism* (paralelismo de dados) vale-se da uniformidade das arquiteturas para que compiladores possam executar a mesma operação em múltiplos processadores, correspondendo ao modelo SPMD – a ser visto logo adiante (cf. [Foster, 1995]).

[31] Sistemas de memória compartilhada são indicados por [Foster, *ibid.*] como uma das alternativas para PAD (ao lado das bibliotecas MP e do paralelismo de dado implícito).

custos e produtividade[32]. Definimos **paralelismo** como o "processamento simultâneo de programas ou partes de um mesmo programa em múltiplos processadores" [Brookshear, 1999]. O objetivo básico é eliminar o chamado "gargalo de von Neumann", isto é, a seqüencialização na requisições de instruções através de um barramento da memória para a CPU [Tanenbaum, 1999] [Pacheco, 1997].

2.2.2 – Tipos de Paralelismo

A bibliografia enumera vários critérios para classificarmos os tipos de paralelismo. Aqui, nos utilizaremos de algumas taxionomias mais utilizadas:

♦ A. Quanto à Programação

Este critério está relacionado com a forma com que o código paralelo foi gerado [Hwang, 1993, 1998]:

- **Paralelismo Implícito:** Visa automatizar o trabalho do programador na detecção de dependências. Fazem uma análise e transformação de dados de forma autônoma. Contudo, devido à falta de padronização e às peculiaridades dos programas a serem paralelizados, as técnicas utilizadas não se adequam a todos os casos. Exemplos de ferramentas: `High Performance Fortran` (Visual Numerics) e `ForgeExplorer` (IBM);
- **Paralelismo Explícito:** Nesta situação, a participação direta do programador é imprescindível. É o caso das *bibliotecas de passagem de mensagem*. Aqui, os programas são gerados da forma tradicional (por exemplo, em linguagem `C` ou `Fortran`), com a inclusão de chamadas específicas para a comunicação entre processos concorrentes. Exemplos: `PVM` (Mississipi State University) e `MPI` (Oak Labs).

Embora as ferramentas automáticas tragam enormes facilidades, alguns problemas podem ser fatores decisivos para a sua não adoção: primeiramente, a possibilidade de não resolução das dependências de código (o que implica na impossibilidade de sua paralelização). E, em segundo lugar, percebe-se que elas possuem um desempenho restrito, já que em muitos casos, o programador deve conhecer as características do seu programa. Em outras palavras, apesar das vantagens da paralelização implícita, a eficiência desejada está condicionada aos padrões de código sob as quais elas atuam [Hwang, 1998]. Note que, no caso das bibliotecas explícitas, sua principal deficiência reside na necessidade de intervenção direta do programador.

---

[32] Conforme mencionado, outras técnicas como a computação vetorial e a programação distribuída em agentes podem levar, cada uma a seu modo, à obtenção de PAD.

♦ B. Quanto ao Particionamento

Por este critério, nos preocupamos mais diretamente com a natureza do problema a ser paralelizado (particionamento de dados ou de funções) [Fox, 1996]:

- **Paralelismo de Dados (Decomposição de Domínio):** Nesta técnica, os *dados* são particionados entre os vários EPs do sistema. Cada processador executa as mesmas instruções sobre dados diferentes. É aplicado, por exemplo, em programas que utilizam matrizes (resolução de sistemas de equações) e para cálculos de elementos finitos[33].

Os chamados *data parallel programs* (como `DataParallel C`) se enquadram nesta categoria. A figura 2.2 ilustra um modelo esquemático para este tipo de paralelismo.

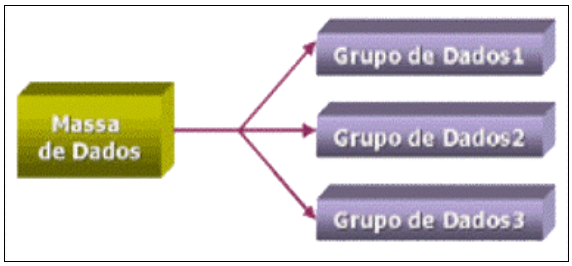


Figura 2.2: Diagrama de um Paralelismo de Dados

- **Paralelismo Funcional (Decomposição Funcional):** Também chamado de **paralelismo de controle**, nesta situação, cada processador executa instruções diferentes que operam sobre os dados (que podem ou não ser distintos).

Aplicações como programas do tipo produtor-consumidor, simulações e tratamento de dados (imagens) se enquadram neste modelo. Vide modelo esquemático na figura 2.3.

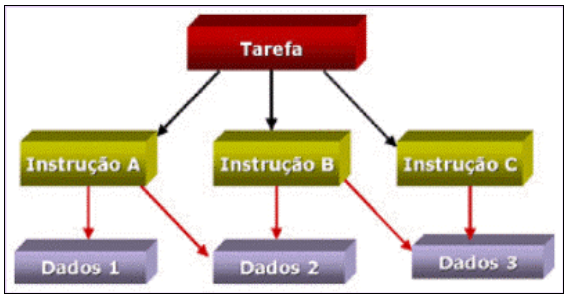


Figura 2.3: Diagrama de um Paralelismo Funcional

O paralelismo funcional é normalmente aplicado em programas dinâmicos e modulares onde cada tarefa será um programa diferente. Em termos práticos, a programação *multi-thread* opera hoje de acordo com este modelo de programação.

Vale a pena mencionar que [Fox, *ibid.*] amplia as classes, especificando detalhes sobre a natureza do que está sendo paralelizado: em nível de objetos ou de metaproblemas (cf. apêndice C.2). Para todos os efeitos, simplificamos o modelo neste texto.

---

[33] Em algumas situações, a quantidade de comunicação pode ser tão intensa que a realização de certos cálculos se tornaria proibitiva, analisando-se somente os custos das comunicações entre processadores. No entanto, em arquiteturas fechadas (como no caso de computadores vetoriais ou sistemas MPP), a troca intensa de informação por um canal de comutação extremamente rápido pode viabilizar tais programas.

♦ C. Quanto ao Programa (homogeneidade dos processos)

As arquiteturas MIMD permitem implementar duas formas mais comuns de programação[34]:

- ***SPMD (Single-Program Multiple-Data):*** No qual todos os processadores executam o mesmo programa, diferenciando-se apenas pelos dados que são tratados em cada máquina;
- ***MPMD (Multiple-Program Multiple-Data)***, pelo qual cada processador pode executar um programa distinto e operar sobre um conjunto diversificado de dados. Este modelo, no entanto, requer uma atenção especial para a organização dos processos e a sincronização dos mesmos.

2.2.3 – Paradigmas de Programação

A programação paralela pode ser implementada de diversas maneiras[35]. Consideraremos uma classificação bastante simples de acordo com as características dos processos em execução (figura 2.4):

- **Modelo Clássico SPMD:** Neste caso, todos os processos executam um mesmo programa, normalmente operando sobre dados distintos (paralelismo de dados)[36]. Esta definição prévia do código de cada tarefa permite que todas possam ser simultaneamente lançadas pelo sistema no ambiente de processamento, tal como é feito no paralelismo de dados ou em ferramentas de bibliotecas.
- **Modelo Mestre-Escravo:** Aqui, um único processo é inicialmente criado (o **mestre**) e este é responsável pela criação de todos os demais processos (**escravos**), bem como inicialização, coleta e impressão dos resultados. No entanto, são os programas escravos que realizam os cálculos propriamente ditos. O modelo mestre-escravo é convencionalmente utilizado no PVM.
- **Modelo Divisão e Conquista:** Neste caso, cada processo (com exceção dos últimos) é responsável pela criação de pelo menos um outro processo até que se forme a estrutura de processamento necessária. Este modelo pode criar uma

---

[34] Note que existe uma sutil diferença entre os modelos SIMD e SPMD (que pertence à classe MIMD de Flynn): no primeiro caso, nos deparamos com paralelismo em **nível de instrução**, enquanto a segunda classe trata de paralelismo em **nível de programa**, de forma que, nesta última situação, cada processador pode executar instruções independentes dos demais, ainda que pertinentes a um mesmo e único programa (cf. [Hwang, 1998], [Foster, 1995]).

[35] Maiores informações (inclusive com aplicações de outros modelos), podem ser conferidas em [Tanenbaum, 1999 e [Hwang, *ibid.*].

[36] Nada impede, contudo que, do ponto de vista da implementação do código, um programa apresente condições de controle sobre a identificação de cada processo, determinando, de uma forma distinta, o que deve ser feito (note que, neste caso, estamos nos deslocando para o modelo do paralelismo funcional).

generalização da topologia em árvore, não havendo uma coleta centralizada de resultados.

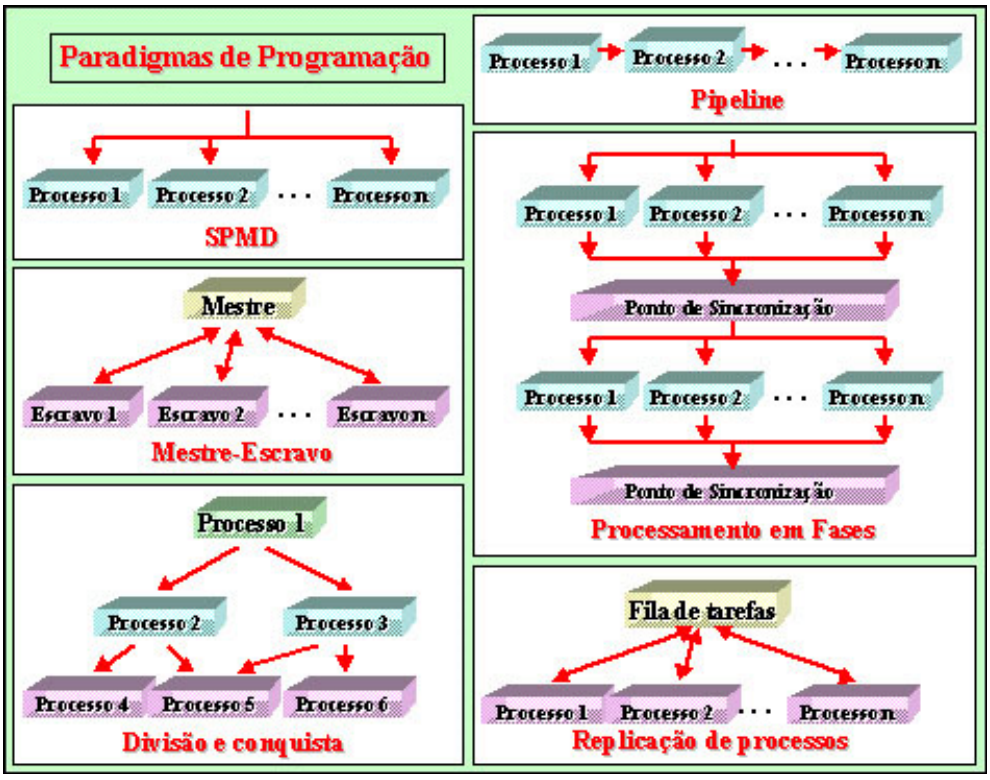


Figura 2.4: Paradigmas de Programação Paralela

- **Pipeline:** Nesta situação, existe um fluxo contínuo de dados, que são tratados diferentemente pelos processadores. Se os fluxos de dados for extenso o suficiente, todos os processos têm chance de estar ocupados durante todo o programa.
- **Paralelismo em Fases:** Consiste num modelo de paralelismo extremamente comum, sendo bem mais realista do que as propostas anteriores (cf. [Cáceres, 2001]). Aqui, a computação é dividida em duas fases: a computação (C), que são paralelamente processadas e a sincronização (S), no qual as atividades são sincronizadas entre si para partir para uma próxima fase de processamento.
- **Replicação de Processos:** Também chamado de "*work pool*", este modelo se vale de uma fila centralizada para a realização das atividades, que são consumidas (e produzidas) pelos processos, sob a gerência de um coordenador específico. É o modelo utilizado para variáveis compartilhadas: quando um processo está pronto para execução, obtém um pedaço da tarefa no reservatório e pode gerar novos objetos ou consumir os que já existem.

2.2.4 – Aspectos Funcionais

♦ A. Fatores Determinantes para o Desempenho

Alguns fatores são fundamentais para a determinação do desempenho em processamento paralelo: a **granulosidade**, o **balanceamento de carga** e a **escalabilidade**.

- (i). Granulosidade

É uma medida usada para indicar a relação entre o tempo de execução de cada tarefa e o tempo total de execução do programa, ou seja, é a razão entre computação e comunicação, medida pelo tempo de processamento ou o número de instruções em ponto flutuante (PF) entre duas fases de sincronização [Hwang, 1998].

Um "**grão**" é, portanto, uma medida da duração de cada tarefa. Essa definição permite classificar os processos em dois tipos de granulosidade (figura 2.5):

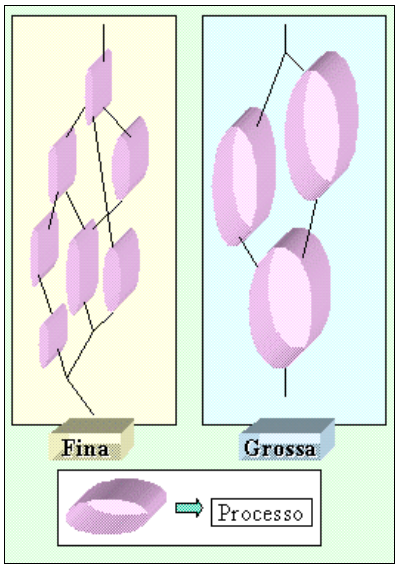


Figura 2.5: Tipos de Granulosidade

- **Granulosidade Fina:** Quando a diferença de tempo entre dois momentos de sincronização (recepção de dados, barreira, etc.) é relativamente pequeno. Em outras palavras, existe uma considerável freqüência de comunicações, o que tende a diminuir o desempenho do programa. Embora não haja uma classificação exata, valores abaixo de 200 operações em PF podem ser enquadrados nesta classe [Hwang, 1998];
- **Granulosidade Grossa:** Quando, por outro lado, existe um considerável período sem comunicação ou sincronização, permitindo a livre execução do processamento. Isso permite um ganho de velocidade (já que existe um espaço menor para a comunicação), mas pode dificultar o balanceamento de carga entre processos, no caso de haver uma variação muito grande entre o tempo de duração dos programas. Valores acima de alguns milhares de operações em PF caracterizam a granulosidade grossa.

- (ii). Balanceamento de Carga / Sincronização

O **balanceamento de carga** é o mecanismo responsável pela distribuição **otimizada** dos processos entre os diferentes EPs do sistema, fazendo com que o tempo de execução dos processos em paralelo seja mais eficiente ou ao menos satisfatória [Brookshear, 1999]. Embora existam ferramentas automáticas para tratar o balanceamento (como o `LoadLeveler` da `IBM` e o *freeware* `DQS`), não existe nenhuma solução

definitiva, sendo que, na maioria das vezes, apenas estimativas e testes empíricos podem ajudar na elaboração de uma política para a correta fragmentação do problema[37].

A **sincronização** irá se preocupar em como coordenar adequadamente os múltiplos grãos a fim de poder obter um melhor valor de *speed-up*. De fato, uma sincronização mal feita pode provocar além de queda de desempenho, erros de lógica no valor das variáveis de um programa. Um exemplo disso é mostrado na figura 2.6: Suponhamos que um processo (A) deve receber de outro (B) uma variável ("a") que serve ao cálculo de um novo valor ("b") a ser retransmitido, isto é, enviado desta vez de A para B, onde finalmente será utilizado para um cálculo. Observe que isto implica numa determinada ordem para a comunicação e o processamento. Tal seqüência é ilustrada na parte superior da ilustração.

No entanto, suponhamos que os programas foram inadequadamente estruturados, de forma a se alterarem os momentos de comunicação entre os processos (no caso, o processo A retransmite o valor de b antes que ele tenha sido atualizado corretamente). Este simples fato altera o fluxo de execução dos comandos, fazendo com que os resultados possam ser completamente diferentes dos valores esperados, conforme ilustrado na parte inferior da ilustração. Observe ainda que outros erros mais graves, como a simples permutação entre linhas de comandos *send*/*receive* ou a ausência de uma transmissão no dado, pode inclusive levar o processo a um travamento ou *dead-lock*.

[37] Vale a pena observar que uma metodologia proposta por [Foster, 1995] para o desenvolvimento de programas paralelos reforça bem a importância da granulosidade e do balanceamento de carga: o paradigma PCAM compõe-se das fases de particionamento **"P"**, (decomposição do problema), comunicação "**C**" (coordenação das tarefas), aglomeração "**A**" (recombinadas das tarefas para reduzir a comunicação) e mapeamento "**M**" (**a**ssociação das fases aos processadores). Normalmente, a metodologia deve ser continuamente reavaliada e, se possível, analisada em tempo de execução, a fim de ser otimizada.

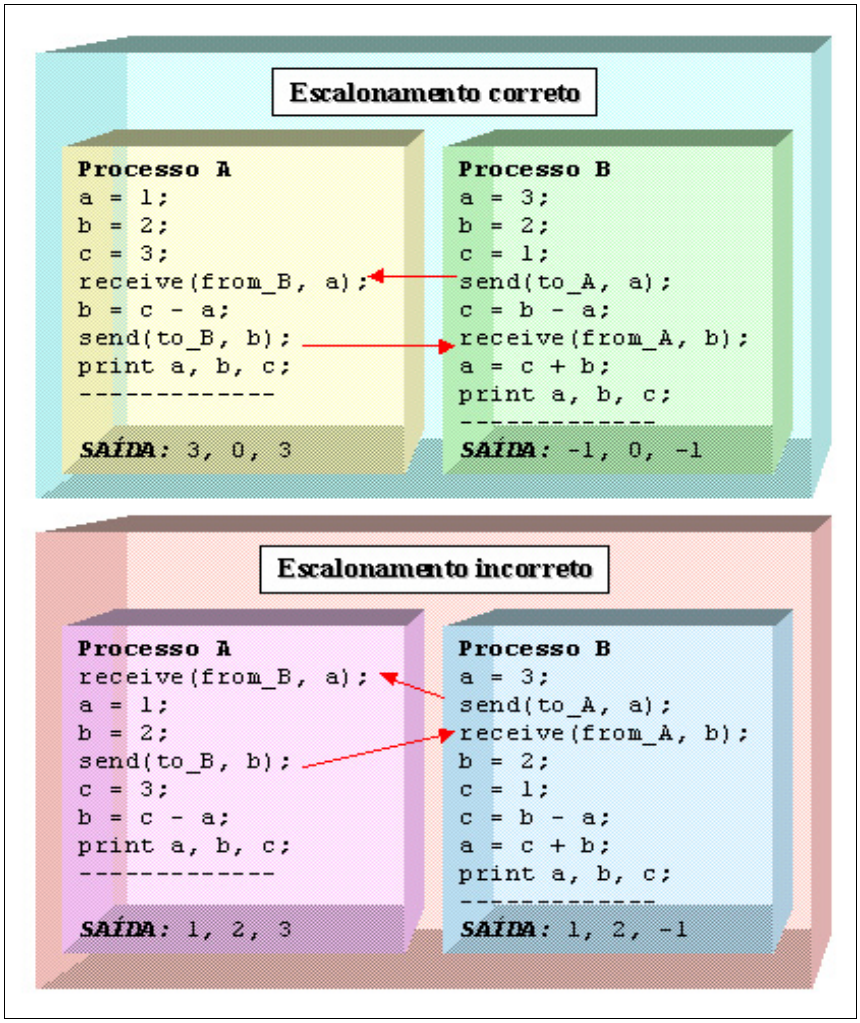


Figura 2.6: Exemplo de erros de escalonamento

- (iii). Escalabilidade

O conceito de **escalabilidade** inclui tanto elementos de *hardware* quanto *software* e refere-se à possibilidade de se poder ampliar um determinado sistema (isto é, aumentar os seus recursos) a fim de obter melhor desempenho e funcionalidade ou à necessidade de se reduzi-lo, a fim de se diminuir os custos a ele associados [Hwang, 1998].

Diretamente relacionado com o problema do particionamento, a questão relativa à escalabilidade trata da capacidade de se melhorar a relação custo/desempenho do sistema. Uma observação importante é que o puro e simples aumento do número de processadores, em si mesmo, não representa uma melhoria para o sistema. O problema reside em que, apesar do crescimento potencial, o fator **comunicação** pode começar a se tornar crítico, de forma que a curva de *speed-up* (aceleração) chegue a um limite a partir de certo ponto.

♦ B. Dificuldades no Paralelismo

A adoção do processamento paralelo para resolver um problema se depara com algumas dificuldades. Alguns destes problemas como a sincronização entre os processos, dependem diretamente dos fatores acima mencionados (granulosidade e balanceamento). Outros, porém, são intrínsecos à própria natureza do paralelismo e do problema a ser tratado, tais como:

- **"Nem tudo é paralelizável":** A necessidade de particionamento e as restrições relativas à independência dos códigos são fatores que limitam fortemente quais as aplicações que podem ser adequadamente convertidas em código paralelo;
- **Ferramentas:** O fato de que existe uma quantidade limitada de compiladores e ferramentas para paralelização automática, bem como as dificuldades da programação explícita e da própria visão do paradigma do paralelismo, também são dificuldades que não podem ser ignoradas;
- **Custos:** Há um considerável tempo gasto pelo programador em analisar um código fonte seqüencial, para paralelizá-lo e recodificá-lo de forma paralela. A própria mudança de paradigma, já que se torna necessário rever o programa original, avaliar seus pontos de sincronização e as formas de comunicação a serem empregadas, representam um esforço adicional considerável;
- **Dificuldades de depuração:** Em se tratando de programação explícita, percebe-se que as limitações das ferramentas para rastreamento e depuração dos códigos existentes são desafios adicionais enfrentados pelos programadores. Na maioria dos casos, a alternativa empregada é a utilização de arquivos para rastreamento de execução, chamados *tracefiles*, que contêm um conjunto de dados gerados durante a execução, para que sejam posteriormente analisados pelo usuário (quando da conclusão do programa). Exemplos: `Xmpi`, `Paragraph`, `Xpvm`, `Upshot` e `PE` (cf. [Zomaya, 1996]).
- **Outras limitações:** Finalmente, o maior desafio consiste em **saber investir** corretamente nesta nova tecnologia: não são poucos os casos em que a relação custo × benefício é inferior ao planejado e os limites teóricos impostos devem ser criteriosamente analisados.

♦ C. Justificando o Paralelismo

Algumas orientações podem ajudar a definir **onde** utilizar adequadamente a programação paralela para obter um ganho de desempenho no processamento:

- Para a resolução de grandes desafios computacionais (problemas de considerável complexidade), nos quais o tempo de processamento seqüencial é insatisfatório;

- Em problemas que possam ser facilmente particionados, originando subproblemas com granulosidade grossa adequadamente balanceados;
- Em sistemas multicomputadores onde a velocidade de transmissão de dados entre os EPs não seja crítica;
- Finalmente, em situações onde existe intenso processamento e pouca necessidade de comunicação entre os processos.

Admitindo que haja razões para a utilização do paralelismo na resolução de nosso problema, podemos agora conhecer uma das soluções mais baratas e largamente empregadas ainda hoje: a utilização de bibliotecas de passagem de mensagem.

2.3. Passagem de Mensagem

2.3.1 – Definição e Características

A utilização de comunicação por **passagem de mensagens** (***message-passing*** - MP) é um dos principais mecanismos empregados para permitir processamento distribuído em multicomputadores conectados via rede.

O modelo funcional é o de um conjunto de EPs que têm somente memória local, mas cujos processos são capazes de se comunicar através do envio e recepção de mensagens [Gropp, 1994], conforme ilustrado na figura 2.7.

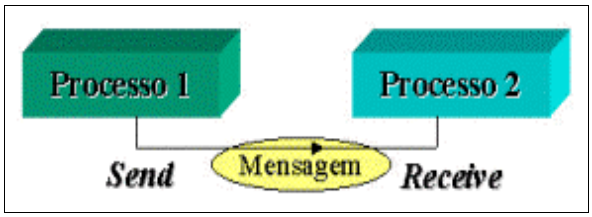


Figura 2.7: Comunicação via Message-Passing

De agora em diante, ao falarmos em programação paralela, estaremos implicitamente dizendo que a mesma é feita via passagem de mensagem. Estritamente falando, o paradigma MP define apenas um **padrão de comunicação**, não incluindo sintaxes de linguagem, nem bibliotecas e é completamente independente do *hardware*.

♦ A. Vantagens

A comunicação via passagem de mensagem apresenta-se apenas como uma das alternativas mais viáveis, devido a muitas vantagens, tais como [Gropp, 1994]:

- **Generalidade (Universalidade):** Pode-se construir um mecanismo de passagem de mensagens para qualquer linguagem, como ferramentas de extensão das mesmas (bibliotecas) e sob qualquer tipo de rede (lenta ou rápida), para os mais diferentes tipos de máquinas, que vão desde os supercomputadores aos PCs mais simples.

- **Aplicabilidade:** A adequação a ambientes distribuídos faz deste modelo, uma alternativa perfeitamente viável para uma grande gama de aplicações e ambientes.
- **Expressividade:** O modelo MP tem sido utilizado como um modelo completo para o desenvolvimento dos mais diversos algoritmos para processamento paralelo;
- **Possibilidade de depuração:** Considerando-se as características de programação, é perfeitamente possível realizar a depuração no modelo MP, já que ela evita uma das causas mais comuns de erros: a corrida crítica e escrita indevida da memória, devido ao controle explícito do modelo;
- **Baixo custo:** As mais conhecidas bibliotecas de MP (`PVM` e `MPI`) são programas do gênero *freeware*, ou seja, de domínio público. Sua instalação requer apenas conhecimentos básicos e, portanto, permitem formar ambientes de processamento paralelo de baixíssimo custo;
- **Desempenho:** Finalmente, a mais importante razão permanece sendo a possibilidade de se obter um ganho de desempenho com a utilização de *clusters* de processamento intercomunicados via MP.

♦ B. Limitações

Porém, vale a pena ressaltar que este modelo apresenta algumas restrições:

- **Necessidade de programação explícita:** Primeiro de tudo, o programador é diretamente responsável pela paralelização. Isto requer que haja uma mudança na forma algorítmica para desenvolvimento dos programas: de um modelo seqüencial, para uma visão cooperativa (distribuída) do processamento.
- **Problemas de intercomunicação:** Como seria de se esperar, os custos de comunicação podem tornar extremamente proibitiva a transmissão de mensagens em certos ambiente. Assim, os fatores mencionados na seção 2.2.4 (granulosidade, balanceamento de carga e escalabilidade) devem ser cuidadosamente analisados antes de se empregar paralelismo via MP.

2.3.2 – Tipos de Rotinas

A implementação de comunicação via passagem de mensagem é feita normalmente com o uso de **bibliotecas** que são incorporadas às linguagens utilizadas pelos programadores (comumente, `C` ou `Fortran`), de forma que sua utilização se torna consideravelmente mais simples e prática.

Via de regra, as bibliotecas de MP possuem rotinas com finalidades bem específicas, que podem ser identificadas em vários grupos:

- **Rotinas de gerência de processos:** São responsáveis pelas atividades básicas de inicialização e finalização dos processos. Além disso, permitem determinar o número de processos em execução e identificá-los.
- **Rotinas de Comunicação Ponto-a-Ponto:** Constituem o modelo mais elementar de comunicação, permitindo que dois processos distintos possam enviar e receber mensagens.
- **Rotinas de Comunicação em Grupo:** Permitem que vários processos possam transmitir mensagens, realizar *broadcast* (difusão) ou executar operações específicas como redução, coleta ou espalhamento de dados e ainda criar barreiras de sincronização.
- **Rotinas Auxiliares:** São rotinas que fornecem ferramentas para verificação de mensagens, estado da rede, análise de erros e testes no sistema.

2.3.3 – Tipos de Comunicação

Uma comunicação através de MP pode ser classificada de acordo com diferentes parâmetros [Geist *et al*, 1993] [Gropp, 1994] [CENAPAD-NE, 1998]:

♦ A. Quanto ao Bloqueio

Existem dois tipos básicos (ilustrados na figura 2.8):

- **Comunicação "blocante" (*blocking communication*):** Quando a finalização da execução da rotina é dependente de determinados eventos, ou seja, existe uma espera por determinada ação antes de se permitir a continuação do processamento. Dois casos típicos são a espera pelo esvaziamento de um buffer antes de sua reutilização e a existência de barreiras de sincronização;
- **Comunicação "não-blocante" (*non-blocking*):** Neste caso, a finalização da chamada não espera qualquer evento que indique o fim ou sucesso da rotina.

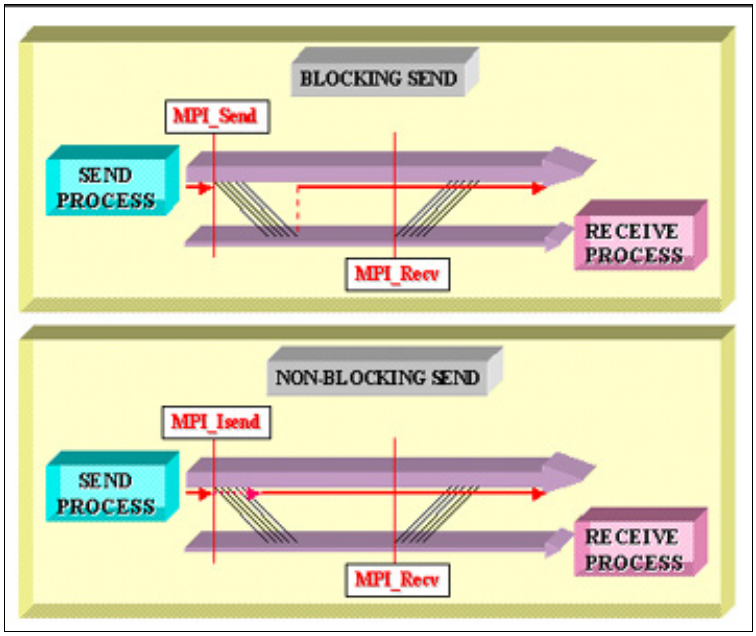


Figura 2.8: Modelos de comunicação quanto ao bloqueio.

♦ B. Quanto à Sincronização

Da mesma forma, temos duas classes de comunicação (vide figura 2.9):

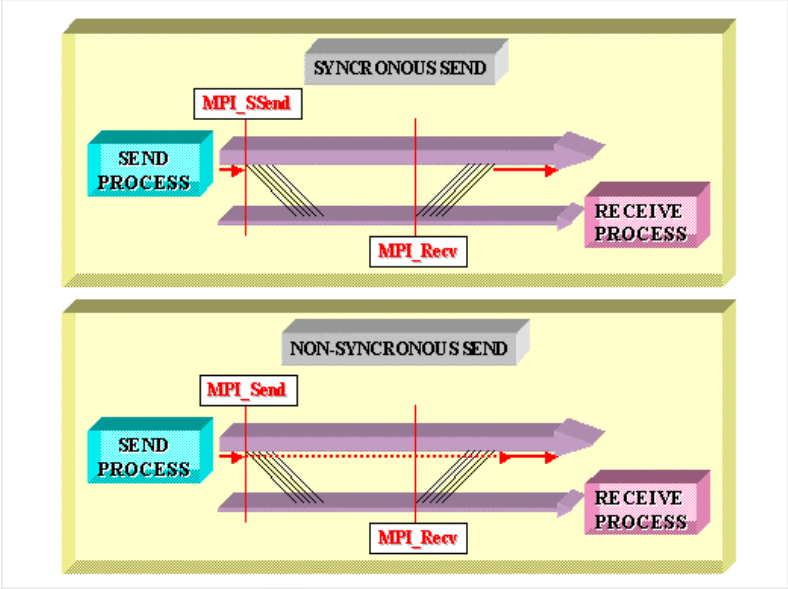


Figura 2.9: Modelos de comunicação quanto à sincronização.

- **Comunicação síncrona:** É a comunicação na qual o processo que envia a mensagem, não retorna à execução normal enquanto não houver um sinal do recebimento da mensagem pelo destinatário, isto é, uma confirmação de êxito da rotina. Corresponde ao modelo mais seguro de transmissão de dados.
- **Comunicação assíncrona:** Quando não existe nenhuma restrição para a confirmação da rotina, o que torna a sua execução consideravelmente mais rápida.

♦ C. Quanto ao Armazenamento da Mensagem

Muito embora qualquer biblioteca de passagem de mensagens utilize internamente áreas temporárias para envio e recepção de dados (*send buffer* e *receive buffer*), aqui levamos em conta a criação **explícita** de áreas para armazenamento temporário definidas pelo usuário, ao qual denominamos ***application buffer***. O modelo esquemático é mostrado na figura 2.10:

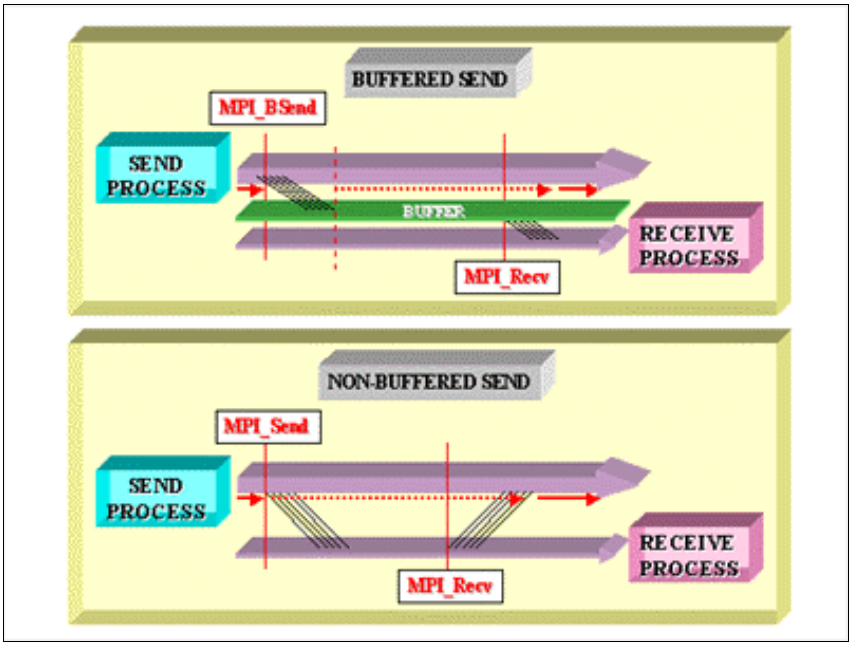


Figura 2.10: Modelos de comunicação quanto à "bufferização"

- **Comunicação protegida ("bufferizada"):** Caracteriza-se pela existência de endereços de memória (*buffer*) para armazenamento temporário de dados a serem enviados durante um processo de comunicação, o que é feito explicitamente pelo programador (vide figura 2.11).

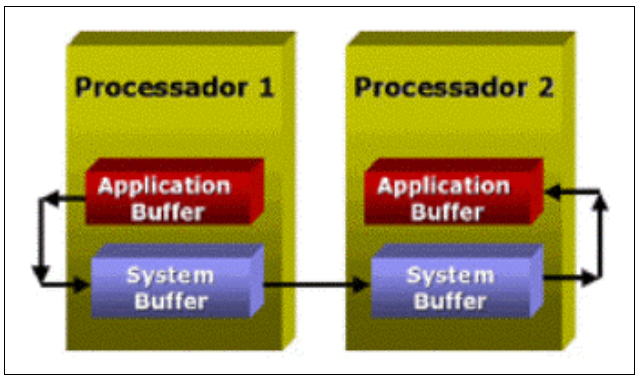


Figura 2.11: Comunicação indireta via Buffer

- **Comunicação direta ("não-bufferizada"):** Quando é o próprio sistema da biblioteca de MP que se encarrega de manipular o dado diretamente de sua localização original na memória, para os *buffers* de envio ou recepção.

2.3.4 – Principais Bibliotecas de Passagem de Mensagem

Problemas que envolvem processamento paralelo podem ser resolvidos com a utilização das bibliotecas de passagem de mensagem (*Message Passing Libraries* – MPLs), que permitem utilizar uma rede de estações de trabalho (*Network of Workstations* - NOWs) ou de computadores pessoais (*Network of Personal Computers* - NOPCs).

Existem várias implementações de message-passing disponíveis. No entanto, aqui destacaremos as duas principais: o **PVM** [Sunderman, 1990] e o **MPI** [MPI Forum, 1993], por motivos bastante simples: são bibliotecas de *software* gratuitas, facilmente obtidas na rede e que possuem um grande número de usuários.

Vale ressaltar que, embora distintas, essas duas MPLs são perfeitamente análogas entre si e a avaliação comparativa de seus desempenhos tem mostrado uma similaridade considerável [Geist 1996]. De maneira semelhante, ambas pretendem, idealmente, reduzir o processamento em até N vezes, onde N corresponde à quantidade de processadores, aumentando a eficiência do sistema como um todo.

♦ A. Características Comuns

Tanto PVM quanto MPI apresentam as seguintes características similares:

- **Rápida curva de aprendizagem:** Uma vez que são bibliotecas para linguagens já tradicionalmente utilizadas, cabe aos programadores apenas conhecer a sintaxe das funções nos pontos de comunicação e transmissão de dados;
- **Economia:** Permitem a utilização do parque tecnológico já existente, bastando que os EPs se comuniquem através de uma rede TCP/IP. Isto implica que boa parte dos códigos já existentes podem ser adequadamente convertidos para programação paralela;
- **Interoperabilidade:** Possibilitam a comunicação de máquinas de diferentes arquiteturas e sistemas. A única exigência é que tenham o mesmo protocolo (hoje, universalmente, TCP/IP);
- **Portabilidade:** Atualmente, existem versões destas ferramentas para os mais diversos ambientes, que vão do Windows aos sistemas Unix.

Alguns pontos negativos também são também comuns às duas bibliotecas, tais como:

- Necessidade de programação explícita (como seria de se esperar, pela própria definição da comunicação via MP);

- Pouca "amigabilidade" com o usuário, pela inexistência de interfaces gráficas padronizadas[38];
- Observação da seqüência de comunicação: Isto significa que as mensagens a serem recebidas devem obedecer exatamente à seqüência de envio.

Nos próximos itens, detalharemos separadamente as duas principais MPLs de nosso estudo: num primeiro momento, o PVM e, em seguida, o MPI.

♦ B. PVM (Parallel Virtual Machine)

➢ (i). Introdução

Iniciado em 1989 pelo Oak Ridge National Laboratory (ORNL) e hoje contando com o apoio de universidades americanas, o Projeto da **Parallel Virtual Machine** (PVM) desenvolveu a MPL mais conhecida de domínio público, hoje portável para praticamente todos os ambientes, incluindo a vasta família do Unix [Geist 1993] e algumas versões do Windows com interface para programação de aplicações em 32 bits (API32) [Fischer 1994] [Hwang & Xu, 1998].

O PVM (atualmente na versão 3.4) baseia-se na decomposição de uma dada tarefa em processos que serão simultaneamente executados por processadores conectados em uma rede IP e que constituem, virtualmente, uma única máquina. O modelo de programação utilizado na maioria dos casos baseia-se no paradigma Mestre-Escravo (vide seção 2.2.3, item A).

Alguns autores salientam que, o PVM é **autocontido**, enquanto o MPI (em algumas versões), não. Em outras palavras, enquanto esta última MPL necessita interagir com o subsistema de arquivos do SO, a primeira inclui todas as funções necessárias, internamente [Hwang & Xu, 1998].

➢ (ii). Funcionamento

O sistema de funcionamento do PVM compõe-se basicamente de dois elementos (figura 2.12): um ***daemon*** (pvmd) residente em cada máquina participante do processamento, responsável pela comunicação e sincronização entre os processos e uma **biblioteca** (libpvm3.a) acessível às máquinas que executam os códigos com as chamadas de passagem de mensagem. Finalmente, a existência de uma ***console*** (pvm) auxilia o usuário a configurar adequadamente a sua máquina virtual, acrescendo determinadas máquinas da rede para o processamento.

---

[38] Na realidade, a existência de interfaces como o XPVM e o LAM MPI, tornam certas distribuições de MPLs mais robustas. Seu funcionamento é feito via *tracefiles*. No entanto, quando os programas necessitam de interação com o usuário, tais ferramentas são de pouca valia.

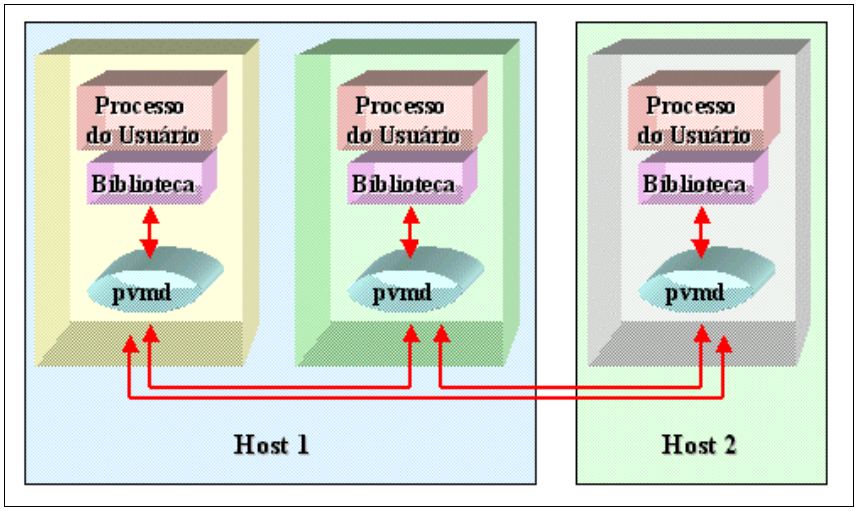


Figura 2.12: Funcionamento do PVM

Um exemplo de código em PVM é mostrado na figura 2.13, no conhecido padrão "HelloWorld", apenas como ilustração. Aqui, um processo mestre (identificado por `PvmNoParent`) é responsável por iniciar um programa escravo (a saber, o próprio programa, tal como ocorria num `fork()` do `Unix`), enquanto aguarda por suas informações. No escravo (reconhecido por possuir um processo PVM pai, dado por `ptid`), é gerada uma mensagem de "Hello", com a identificação da máquina (o `hostname` da máquina). Este texto é enviado ao processo pai (via `pvm_send()`) que, ao recebê-lo (via `pvm_recv()`), finalmente o exibe na tela.

Neste exemplo, alguns elementos importantes podem ser notados:

- Apenas um processo é executado inicialmente (o "pai" ou "mestre"), sendo responsável por inicializar todos os demais (via `pvm_spawn()`);
- A identificação dos processos obedece a um valor próprio do PVM (chamado de *task identifier* ou **`TID`** e obtido por `pvm_mytid()`), com valores de 32 bits. Uma observação importante é que os `TIDs` não são nem seqüenciais - indo de `0` a `N` - e também não correspondem aos **`PIDs`** (*process identification*) dos processos no sistema operacional;
- O mecanismo de MP no PVM trabalha com a formatação de mensagens (via sua inicialização no *send buffer* com `pvm_initsend`), a fim de driblar os problemas referentes às notações binárias de diferentes arquiteturas. O padrão usual (visto acima como `PvmDataDefault`) é o formato chamado **XDR** (*eXternal Data Representation*), independente de plataforma;
- Uma transmissão de dados pode conter vários componentes, como números inteiros, reais, cadeias de caractere, etc. Para isto, os dados tem de ser previamente **empacotados** no *send buffer*, de forma que, quando da execução de um `pvm_send()`, todos os elementos sejam transmitidos em forma de

rajada. Note que a seqüência de **desempacotamento** deve obedecer criteriosamente à seqüência de envio feita pelo outro processo;

- Processos filhos não têm normalmente, permissão para impressão na tela (console), que é propriedade do processo pai (a menos que uma função como `pvm_catchout` seja utilizada). Portanto, seus dados devem ser transmitidos ao mestre, para que este possa adequadamente mostrá-los na saída.

Programa PVM

```
#include "pvm3.h"
main()
 {
  int cc, tid, ptid;
  char buf[100];
  printf("i'm %x\n", pvm_mytid());
  ptid = pvm_parent();
  if (ptid == PvmNoParent)
   {
    cc = pvm_spawn("pgma.exe","host",&tid);
    if (cc == 1)
     {
      cc = pvm_recv(tid,1);
      pvm_upkstr(buf);
      printf("from t%x:%s\n", tid, buf);
      }
    else
      printf("can't start slave\n");
   }
  else
   {
    strcpy(buf,"hello world from");
    gethostname(buf+strlen(buf),64);
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, 1);
   }
  pvm_exit();
 }
```

Figura 2.13: Exemplo de programa em PVM

➢ (iii). Principais Rotinas

Não cabe aqui citar todas as funções da biblioteca PVM, mas apenas salientar as principais rotinas utilizadas e seu significado, conforme mostrado na tabela 2.1.

Rotina	Sintaxe	Descrição
Identificação	`int pvm_mytid (void)`	Registra o processo pvm gerando um número de 32 bits (retorno). Normalmente é a primeira rotina pvm a ser chamada no programa.
Inicialização do escravo	`int pvm_spawn (char *task, char **argv, int flag, char *where, int ntask, int *tids)`	Cria uma ou mais tarefas (`ntask`) na máquina virtual, pela execução de um programa (`argv`). Pode-se especificar ou não o *host* (`where`) onde estes processos serão criados.
Finalização de processo	`int pvm_exit (void)`	Avisa ao pvmd local que o processo está abandonando a máquina virtual. É a última rotina PVM a ser executada.
Identificação do pai	`int pvm_parent (void)`	Retorna o número de identificação do processo pai de outro processo.
Inicialização do envio	`int pvm_initsend (int encoding)`	Limpa uma área específica de memória (*send buffer*) e a prepara para empacotamento e codificação de mensagens, segundo um formado (**encoding**, normalmente `PvmDataDefault` = XDR).
Empacota-mento (ex.: inteiro)	`int pvm_pkint (int *ip, int nitem, int stride)`	Prepara para transmissão (empacota) um conjunto (`nitem`) de dados (`ip`) de um determinado tipo (inteiro, por exemplo), especificando a contigüidade (`stride`, cujo padrão é seqüencial = 1, i.e., contíguo);
Desempaco-tamento. (ex.: inteiro)	`int pvm_upkint (int *ip, int nitem, int stride)`	Desempacota um conjunto (`nitem`) de dados (`ip`) de um determinado tipo (p.ex., inteiro), após sua recepção, de acordo com uma seqüência dada (`stride`).
Envio	`int pvm_send (int tid, int msgtag)`	Envia o conteúdo do buffer ativo, seu tamanho e tipo de dado, para um processo PVM (`tid`), com uma dada identificação (`msgtag`).
Recepção	`int pvm_recv (int tid, int msgtag)`	Bloqueia um processo até que ele receba uma mensagem com um rótulo (`msgtag`) específico, chegue de um processo (`tid`, cujo padrão é 0 = `PvmAnyHost`). A mensagem recebida que é colocada no buffer de recepção (*receive buffer*).
Finalização forçada	`int pvm_kill (int tid)`	Finaliza um processo específico (`tid`).
Direciona-mento de saída (filhos)	`int pvm_catchout (FILE *out)`	Possibilita ao processo pai obter informações ou dados da saída dos processos filhos (padrão: `FILE` = `stdout`);
Multicast	`int pvm_mcast (int *tids, int ntask, int msgtag).`	Distribui dados para vários (`ntask`) processos (`tids`) em uma única chamada, com um rótulo (`msgtag`).

Tabela 2.1: Principais rotinas do PVM

Além destas, cabe ressaltar a existência de rotinas de grupos dinâmicos, como `pvm_joingroup()` e `pvm_reduce()`; comunicação "não blocante", como `pvm_psend()` e `pvm_precv()` e rotinas de sincronização, como por exemplo, `pvm_barrier()`, dentre outras.

No próximo item, tecermos considerações análogas sobre a biblioteca `MPI`, tida, por alguns, como sendo a sucessora do `PVM`.

♦ C. MPI (Message Passing Interface)

- (i). Introdução

Surgido mais recentemente (1994), o *Message Passing Interface* na verdade se constitui numa poderosa tentativa de padronização para comunicação via MP entre multicomputadores. Seus proponentes, que constituem o chamado "**MPI Fórum**", incluem grandes empresas como IBM, Cray, Intel, NEC, nCUBE e outros grupos na área de processamento paralelo, como os grupos P4, Zipcode, Chamaleon, PARMACS e o próprio PVM.

Por ser um padrão, o Fórum em si mesmo não faz nenhuma implementação: apenas deixa as especificações para que possam ser independentemente implementadas pelos desenvolvedores. Versões públicas são disponíveis, destacando-se o MPICH, do Argonne National Laboratory e da Mississipi State University. Outras de caráter proprietário como o IBM MPI (para *clusters* System Parallel - SP), LAM MPI (Ohio Supercomputer Center) e PMPIO (NASA) podem ser citados.

Vale a pena mencionar que exatamente por surgir de um planejamento aberto, o MPI está se tornando um padrão de fato para o processamento via *message-passing*. Os problemas relacionados com as primeiras versões foram sendo paulatinamente corrigidos e hoje, já existem distribuições estáveis para a grande maioria das arquiteturas e sistemas.

- (ii). Funcionamento

Em termos de funcionalidade, o MPI apresenta um conjunto de instruções bem maior (mais de 125) do que as existentes em PVM, ainda que, apenas algumas poucas funções elementares sejam necessárias para uma programação básica (formando o que se chama "MPI Básico", conforme se pode ver na tabela 2.2). A tendência é que, paulatinamente, o padrão MPI possa ir sendo absorvido pela indústria e genericamente utilizado em processamento paralelo. As características do MPI são definidas detalhadamente em [Gropp 1994] e [Snir 1996].

Uma observação importante é de que, dependendo da distribuição (pois existem inúmeras), o MPI pode funcionar de forma completamente distinta ao PVM. Um exemplo é o MPICH que se utiliza de *scripts* para a compilação (como o mpicc e o mpif77) e execução (via mpirun) de programas que utilizem essa biblioteca. Note que nenhum *daemon* é ativado neste caso[39].

[39] No caso do LAM MPI, percebe-se a existência de *daemons* e consoles semelhantes aos encontrados no PVM.

Programa MPI

```
#include "mpi.h"
main(int argc, char *argv[])
 {
  char message[20];
  int i, rank, size, type=99;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if(rank == 0)
   {
    strcpy(message, "Hello, world");
    for (i=1; i<size; i++)
      MPI_Send(message, 13, MPI_CHAR, i, type, MPI_COMM_WORLD);
   }
  else
    MPI_Recv(message, 20, MPI_CHAR, 0, type, MPI_COMM_WORLD, &status);
  printf( "Message from node =%d : %.13s\n", rank,message);
  MPI_Finalize();
 }
```

Figura 2.14: Exemplo de programa em MPI

Na figura 2.14 traduzimos um código em `C` para o `MPI`, análogo ao mencionado para o `PVM`. Vale a pena notar as diferenças entre este código e o anterior. Novamente, podemos pegar este exemplo didático e salientar algumas características próprias do `MPI`[40]:

- Todos os processos da máquina virtual (num total de `N`) começam a executar o mesmo programa, cada um recebendo uma identificação própria que vai de `0` a `N-1`. Note que qualquer um dos processos pode ser escolhido arbitrariamente para sincronizar os demais, recebendo os dados e finalizando o programa: ou seja, para exercer o papel de "mestre";
- A primeira função, `MPI_Init()`, responsável pela inicialização do processo, fornece uma identificação para o mesmo (*rank*), dentro do conjunto de processos (*size*). Tais valores são capturados, respectivamente, pelas funções `MPI_Comm_rank()` e `MPI_Comm_size()`, a fim de permitir sua adequada identificação dentro do conjunto;
- Toda a comunicação se faz através de um domínio ou contexto, identificado por um ***communicator***. O principal é genericamente chamado de `MPI_COMM_WORLD`;

[40] Em se tratando das características do `MPICH`.

- O programa define que um processo (com rank zero) enviará uma mensagem para os demais. O envio de dados é feito por uma função `MPI_Send()`, onde se descreve a informação a ser enviada (`message`, no caso), seu tamanho (`13`), tipo (`MPI_CHAR`), identificação (dado pela variável `i`) e classe (`MPI_COMM_WORLD`). Uma observação importante é a de que neste exemplo, não houve um **empacotamento** do dado, mas, simplesmente, o seu envio; em caso de necessidade de transmissão de diferentes tipos de dados, uma função do tipo *packing* deve ser **explicitamente** invocada (`MPI_Pack()`);
- Os demais processos (com *rank* não nulo) se encarregam de receber os dados enviados, de acordo com seus parâmetros de identificação, atualizando a informação de estado da recepção (`status`);
- Finalmente, a função `MPI_Finalize()` é responsável por concluir as chamadas de rotinas da biblioteca `MPI`.

➢ (iii). Principais Rotinas

Da mesma forma que feito para o `PVM`, ilustramos na tabela 2.2 algumas das principais funções utilizadas pelo `MPI`. Apesar de muito mais numerosas (chegando até 125), destacamos aqui apenas algumas das rotinas mais comuns.

Rotina	Sintaxe	Descrição
Inicialização	`int MPI_Init (int *argc, char * argv[])`	Inicializa um processo MPI, devendo ser a primeira rotina a ser invocada.
Identificação do processo	`int MPI_Comm_rank (MPIComm comm, int *rank)`	Retorna a identificação de um processo dentro de um grupo, entre um valor seqüencial de 0 a N-1.
Enumeração de grupo	`int MPI_Comm_size (MPIComm comm, int *size)`	Retorna o número de processos sob um determinado *communicator* (**comm**).
Envio	`int MPI_Send (void *buffer, int count, MPIDatatype type, int dest, int tag, MPIComm comm)`	Envia de forma blocante, o conteúdo de **buffer**, de tamanho **count**, com dados do tipo **type** para um processo destino (**dest**), com identificação (**tag**), sob o *communicator* **comm**.
Recepção	`int MPI_Recv (void *buffer, int count, MPIDatatype type, int source, int tag, MPIComm comm, MPI_Status status)`	Bloqueia um processo até que ele receba uma mensagem com um rótulo (**tag**) específico, chegue de um processo (**source**). A mensagem recebida é colocada em **buffer**, de tipo **type** e tamanho **count**.
Finalização	`int MPI_Finalize (void)`	Finaliza a invocação de rotinas MPI.

Tabela 2.2: Principais rotinas do MPI

Outras rotinas, além das básicas, merecem destaques, tais como: os vários tipos de *Send* "blocantes" e "não blocantes": síncronos, imediatos ou "bufferizados" (como `MPI_Ssend()`, `MPI_Rsend()` e `MPI_Bsend()`); rotinas para manipulação de *buffers* (`MPI_ Buffer_ attach()`, `MPI_ Buffer_ detach()`); mecanismos de sincronização (`MPI_Wait()`, `MPI_Barrier()`) e funções de comunicação coletiva (`MPI_Bcast()`, `MPI_Scatter()`, `MPI_Gather()`, `MPI_Reduce()`).

Antes de concluir o capítulo, vale sintetizar alguns critérios para estudo da programação PAD, que serão importantes para estudo posterior.

2.3.5 – Características da Programação Distribuída de Alto Desempenho

Algumas das características da programação relacionada com PAD já foram identificadas anteriormente. No entanto, vamos salientar aqui os pontos mais relevantes:

- **Tipo de concorrência:** Tanto o paralelismo de dados quanto o funcional podem ser empregados. Compete ao programador estudar e avaliar, de acordo com a metodologia empregada (como o paradigma PCAM anteriormente mencionado), estudar qual o nível de paralelismo a ser empregado;
- **Nível de complexidade:** Os problemas devem ter um nível razoável de complexidade, de forma que o investimento feito na conversão seqüencial – paralela possa ser adequadamente compensado com ganho de desempenho;
- **Utilização dos recursos:** Os recursos existentes podem ser exaustivamente acessados pelos usuários (desde que assim configurado pela administração local), a fim de acelerar o processamento. Apenas limites para a garantia do funcionamento do sistema devem ser observados;
- **Possibilidade de comunicação:** Em decorrência da natureza do problema tratado, pode haver a necessidade de que o particionamento gere alguma granulosidade e que os processos necessitem transmitir dados ou entrar em sincronização. Isto implica que a rede deve ser consideravelmente rápida;
- **Localidade:** Os programas e as máquinas pertencem geralmente a um mesmo domínio Internet, de forma que o processamento é feito em *clusters* bem definidos;
- **Segurança e identificação:** O acesso ao sistema é, via de regra, controlado por senhas e os usuários têm de ser identificados para acessar os recursos;
- **Continuidade:** Uma vez iniciado o processamento, ele continuará a execução até que seja encerrado ou interrompido pelo usuário. Em um sistema confiável e bem configurado, dificilmente os fatores externos (como problemas na rede), interferirão no processamento.

Resumo

O uso do paralelismo conseguiu trazer para um grande número de usuários, a possibilidade de se alcançar um PAD de forma simples e econômica, utilizando um ambiente computacional já existente. Com o uso de MPLs, pôde-se facilitar o trabalho do programador e tornar a programação paralela, uma alternativa viável.

No próximo capítulo, veremos um outro paradigma de processamento distribuído, centrado na utilização da Internet. que tem conseguido grande número de adeptos. A partir dos conceitos apresentados, poderemos então entender a proposta da ferramenta que é objetivo desta dissertação.

Capítulo 3

Processamento na Internet

Introdução

A explosão da Internet e seus serviços é reflexo da globalização que tem caracterizado este final de milênio. No entanto, por trás disso, o que surgiu foi uma alternativa barata e de enorme potencial para o processamento distribuído que, pelas suas características é aqui chamado de ***computação global*** *[Christiansen 1997:1].*

As possibilidades de utilização da Internet para processamento distribuído ainda são objetos de estudos e muitas são as referências de consulta disponíveis, tanto da parte de pesquisadores e estudiosos, quanto de empresas e laboratórios interessados neste paradigma. Neste capítulo, mencionaremos algumas das abordagens utilizadas, as tecnologias propostas e os novos conceitos envolvidos, que serão importantes para a compreensão da segunda parte desta dissertação.

3.1. Internet e Computação Distribuída

3.1.1 – Histórico

Até recentemente, a aplicação distribuída mais universalmente utilizada em rede era o serviço de correio eletrônico (`email`). Além de eficiente, o sistema é extremamente simples, de forma que outros mecanismos de comunicação de dados e informações teriam de ser desenvolvidos.

Em 1989, o CERN, laboratório europeu para estudo de partículas, localizado em Genebra na Suíça, desenvolveu um serviço distribuído de informações que pretendia servir de meio de comunicação entre pesquisadores para veicular dados de diferentes tipos, como texto, imagens e arquivos num mesmo documento (daí o conceito de **hiperdocumento**). Nascia a *World-Wide-Web* (WWW ou *Web*), que viria a ser, a partir de 1993, o maior responsável pela

"explosão" da Internet em todo o mundo, graças às suas facilidades, que combinam simplicidade com versatilidade.

Funcionalmente, a *Web* baseia-se em três padrões [Harold, 1997] [Farley, 1998] [Jaworski, 1999]:

- ***HTTP** (HyperText Transfer Protocol):* Um protocolo TCP (orientado à conexão) *stateless* para a transferência de hiperdocumentos, baseado na arquitetura cliente-servidor.
- ***HTML** (HyperText Markup Language):* Uma linguagem para a codificação destes documentos, compatível com o padrão SGML – *Standard General Markup Language* – e baseado no uso de declarações (*tags*) para identificar diferentes tipos de objetos e mídias (textos, imagens, sons e vídeos).
- ***URL** (Uniform Resource Locator):* Um padrão para a localização de recursos existentes na rede e referenciados pelas páginas na *Web*. As URLs constituem um tipo de identificador (URI ou Uniform Resource Identifier), juntamente com as URNs (Uniform Resource Names). A diferença básica é que essas últimas apresentam apenas ponteiros para recursos, sem explicitar uma localização particular [Harold, 1997].

Basicamente, o protocolo HTTP se utiliza de uma conexão baseada nos seguintes passos [Tannembaum, 1997b]:

1. Um usuário se utiliza de um programa cliente (um programa navegador ou *browser*) e indica uma URL desejada;
2. O *browser* pergunta ao sistema de resolução de nomes (DNS) o endereço IP da URL indicada;
3. O DNS responde com o endereço IP e o navegador estabelece uma conexão TCP com a porta de serviço HTTP daquele servidor (*default*: porta 80);
4. O *browser* então envia comandos para a obtenção dos conteúdos desejados[41];
5. O servidor envia os arquivos necessários ao cliente;
6. A conexão TCP é liberada e o *browser* exibe os conteúdos requisitados pelo usuário.

A utilização de extensões ao padrão aberto MIME - *Multipurpose Internet Mail Extensions* (RFCs 822, 1341 e 152) – uma codificação de diferentes tipos de dados

[41] As requisições C/S em nível de aplicação encontrados no TCP/IP normalmente utilizam comandos como "GET", como ocorre em boa parte dos protocolos, incluindo FTP, SMTP e HTTP.

reconhecidas pelos protocolos de correio eletrônico e HTTP - tornou a *Web* essencialmente um catalisador de serviços. O surgimento de serviços como CGI (*Common Gateway Interface*) implementado em linguagens como C, Perl ou AppleScript, trouxe certo **dinamismo** às páginas *Web*, o que lhe permitiu ser utilizado, por exemplo, como ferramenta para o ensino de computação e engenharia [cf. Marchioro, 1997]. No entanto, isso ainda não implicava na existência de processamento distribuído, uma vez que apenas operações no servidor eram executadas.

Em 1994, um grupo de pesquisadores da Sun Microsystems, interessados em resolver problemas relacionados com o controle de dispositivos e a transmissão de vídeo através da rede, desenvolveu o protótipo de uma linguagem (originalmente chamada Oak) que seria suficientemente leve, portável e distribuída para suas finalidades. O Java Language Project evoluiu deste projeto e deu origem a uma linguagem que se adaptou perfeitamente à *Web*, não só criando uma dinâmica nova com conteúdo executável [Srinivas, 1997], mas, principalmente, permitindo que o processamento distribuído na Internet se tornasse realidade.

3.1.2 – Java

♦ A. Considerações Iniciais

Definida como "uma linguagem simples, orientada a objetos, distribuída, interpretada, robusta, segura, independente de arquitetura, portável, de alto desempenho, multitarefa e dinâmica" [Gosling 1996a], o Java, por definição, possui características que lhes permitiram tornar ideal para o processamento distribuído na Internet: suporte à programação em rede, código leve, capacidade de manipulação de *threads* (múltiplos canais de execução) e confiabilidade.

Por trás de tudo isso, reside uma importante característica: por ser uma linguagem de programação (**LP**) sofisticada, e fortemente tipada, as máquinas virtuais Java (JVMs[42]) responsáveis pela interpretação dos códigos - ***bytecodes*** - podem ser implementadas nos mais variados ambientes. Isso permitiu que fabricantes de *browsers* (os clientes HTTP responsáveis pela interpretação de códigos HTML e URLs), como a Netscape e a Microsoft, inserissem JVMs em seus produtos, permitindo que código Java pudesse ser carregado, via rede e

[42] Uma JVM constitui uma máquina abstrata (ou virtual) implementada em nível de software especialmente projetada para o processamento de código Java – *bytecodes* – sobre uma arquitetura qualquer. Seu funcionamento é possível graças ao padrão definido na *Java Language Specification* [Gosling, 1996b]. Atualmente, são encontradas JVMs nos principais sistemas operacionais e *browsers* do mercado.

executado **localmente** nas máquinas[43]. O caminho para o processamento distribuído via *Web* estava aberto.

No entanto, vale destacar que, apesar das vantagens, a linguagem apresenta um sério desafio: por ser interpretada e executar sobre uma **máquina virtual** que está acima do nível de sistema operacional, forçosamente a execução dos *bytecodes* se torna lenta. Técnicas como **JIT** - *just-in-time compilation* – desenvolvida pela Sun Microsystems (cf. [Yellin, 1996]) e, em alguns casos, a invocação de métodos nativos[44] foram algumas das alternativas propostas para resolver este problema.

A utilização de produtos comerciais que incorporam a tecnologia **NET** (*Native Executable Translation*) podem ainda auxiliar neste processo [Hsieh, 1997]. Um estudo mais aprimorado sobre o desempenho de Java pode ser encontrado na literatura, comprovando uma considerável melhoria neste aspecto da linguagem [Mangione, 1998].

♦ B. Computação Distribuída com Java

As propriedades da linguagem tornaram Java, sem dúvida, uma opção bastante atraente para o desenvolvimento de aplicações distribuídas, fornecendo uma infra-estrutura básica para a utilização da Internet [Pramanick, 1998]. Vários pontos ilustram este fato:

- O fato de ser uma linguagem essencialmente orientada a objetos[45], permite que sejam criados agentes de razoável nível de complexidade, necessários ao processamento distribuído. Em pouco tempo, alguns produtos comerciais começaram a aparecer exatamente com estas características [Srnivas, 1997].
- As interfaces abstratas permitem implementar agentes no sistema para se comunicarem com interfaces específicas, sem que se saiba, previamente, como elas são realmente implementadas na classe. Este dinamismo confere à linguagem capacidade para lidar com elementos críticos durante a execução (como ocorre com o uso de RMI – *Remote Method Invocation*);
- As técnicas de manipulação de erros conferem aos programas escritos em Java robustez e condições para a criação de aplicações tolerantes a falhas (desde que previamente definidas pelo programador) que são essenciais em ambientes

[43] Alguns críticos consideram as JVMs inseridas nos *browsers* comerciais como versões **proprietárias** do interpretador de *bytecodes* Java da Sun, devido às extensões incorporadas nestes programas.

[44] Os métodos nativos – implementados pelo uso de uma interface denominada JNI (*Java Native Interface*) - comprometem fortemente a **portabilidade** do código Java.

[45] Algumas linguagens estendem padrões de programação para incluir suporte ao conceito de classes, adaptando-as ao paradigma OO (como ocorre no C++ em relação ao C). Em Java, no entanto, a presença de classes é um fator imperativo em qualquer contexto de programação.

distribuídos, permitindo ainda, a elaboração de condições de erro manipuláveis, de acordo com cada situação.

- A interface de programação de aplicações (API) confere um excelente suporte à programação em rede, provendo facilidades (como *sockets*) para a implementação de rotinas de comunicação;
- A existência de procedimentos para verificação interna do código durante sua carga (`Class Loader`), permitem assegurar a validade do mesmo. Além disso, serviços de criptografia e de assinatura são providos, o que reforça ainda mais sua segurança, permitindo a elaboração de transações remotas confiáveis;
- O suporte a *threads*, incluindo capacidades de criação, interrupção e finalização de processos, bem como os mecanismos de sincronização permitem gerenciar melhor o funcionamento de aplicações multitarefas;
- Finalmente, o fato dos *bytecodes* gerados poderem ser executados por qualquer JVM existente, lhe confere uma total independência de plataforma (*hardware* e SO) e deixam a linguagem adequada ao processamento global distribuído.

Estudos têm comprovado a eficiência da utilização de `Java` para a computação distribuída e vários autores tem explicitamente defendido que esta LP está em vias de se tornar "a maior linguagem dentro da ciência da computação" [Fox, 1997b:1] [Philippsen, 1997:7].

♦ C. Usando Java para Comunicar Processos

Pelas características apresentadas, pode-se perceber que a linguagem permite facilmente a implantação de serviços distribuídos através da rede. A figura 3.1 ilustra o funcionamento de uma aplicação na arquitetura C/S implementada em `Java`.

Do ponto de vista funcional, um dado cliente faz um *download* a partir de um determinado servidor, de uma aplicação escrita em `Java`. Isto pode ser feito, por exemplo, através de um *browser* habilitado com uma JVM (que cria um ambiente para a execução dos *bytecodes*). Executando sobre o sistema operacional local, a máquina virtual abre conexão com o servidor original, onde existem classes locais e bibliotecas que permitem a execução de outras aplicações e interagem com ela. É importante notar que tanto o cliente quanto o servidor podem executar aplicações e transferir informações ou resultados. Outros clientes podem ainda, interagir independentemente com o servidor (desde que este tenha suporte a tratamento *multi-thread*), porém, cada conexão cria um espaço próprio (***User Space***) garantido pelos níveis de segurança impostos pela linguagem. Cabe ao servidor gerenciar adequadamente os múltiplos canais de execução ativos.

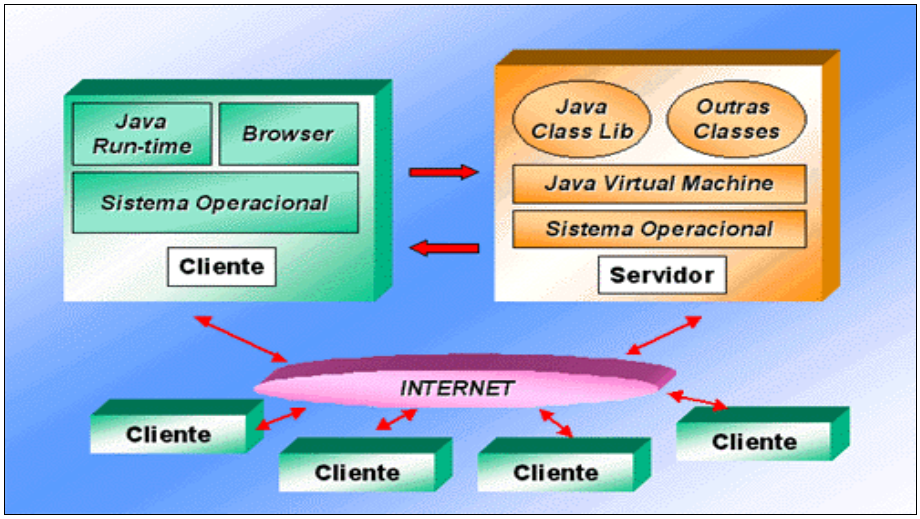


Figura 3.1: Arquitetura de uma Aplicação Java na Internet

Do ponto de vista da implementação, o mecanismo de comunicação é facilmente construído com extensões das classes de rede (pacote `java.net`) e a confiabilidade é assegurada de acordo com o nível de segurança adotado (configurado pelo usuário). Em outras palavras, "a vantagem do Java é que ele não é apenas uma outra linguagem, mas todo um ambiente de programação" [Stankovie, 1998:2].

♦ D. Tendências Futuras

Após 5 anos de existência, costuma-se perguntar se Java continua realmente tendo todo aquele apelo ou, ao menos, o fôlego necessário para continuar sendo alvo de tantos projetos. A resposta pode ser encontrada quando se percebe que Java foi escolhida como a linguagem do ano por inúmeros usuários[46], superando expectativas de novas tendências como XML[47] e PHP[48]. Segundo o IDC (International Data Group), uma importante instituição de pesquisas e estudos estatísticos americana, existiam ao final de 1999, quase 2 milhões de programadores Java no mundo, dos quais apenas 8000 pessoas certificadas no Brasil. Estima-se, no entanto, que cerca de 60% das empresas em todo o mundo, venham a aderir à linguagem até 2004 (cf. [EXAME, 2000]).

♦ E. Outras Tecnologias

---

[46] Vide reportagem "Eles arrasaram em 2000", na seção "*Linguagem do Ano*" de Informática Exame, edição de Dezembro de 2000.

[47] O eXtended Markup Language (XML) é um padrão proposto pelo W3C (World-Wide-Web Consortium) para formato universal de documentos estruturados e dados. O XML está se tornando uma das grandes tendências na programação via Internet, pelo dinamismo que confere às páginas na Internet, identificado tipos de informações usadas nos documentos, tal que este possa ser reformatado para o processamento de informações. Maiores detalhes em: http://www.w3.org/XML/.

[48] O Hypertext Preprocessor (PHP) é uma ferramenta que permite a criação de páginas dinâmicas, funcionando como uma linguagem de *script* interna, independente de plataforma usada para substituir componentes CGI (Common Gateway Interface). Maiores informações em: http://www.php.net/.

➢ (i). Aplicações proprietárias

Além de Java, existem outras alternativas que permitem a utilização da Internet para o processamento de informações. O modelo básico baseia-se sempre na arquitetura **cliente-servidor** (que, diga-se de passagem, pode ser implementado em qualquer linguagem), de forma que não existem, virtualmente, restrições para a abrangência da rede envolvida.

Um exemplo recente disto (iniciado em maio de 1999), está sendo o esforço conjunto que milhares de voluntários estão fazendo no sentido de participar do projeto de computação distribuída desenvolvido pelos pesquisadores do **SETI** (Search for Extra-Terrestrial Intelligence Institute) e da Universidade da Califórnia. Pode-se conferir maiores informações em: http: // setiathome. ssl. berkeley.edu.

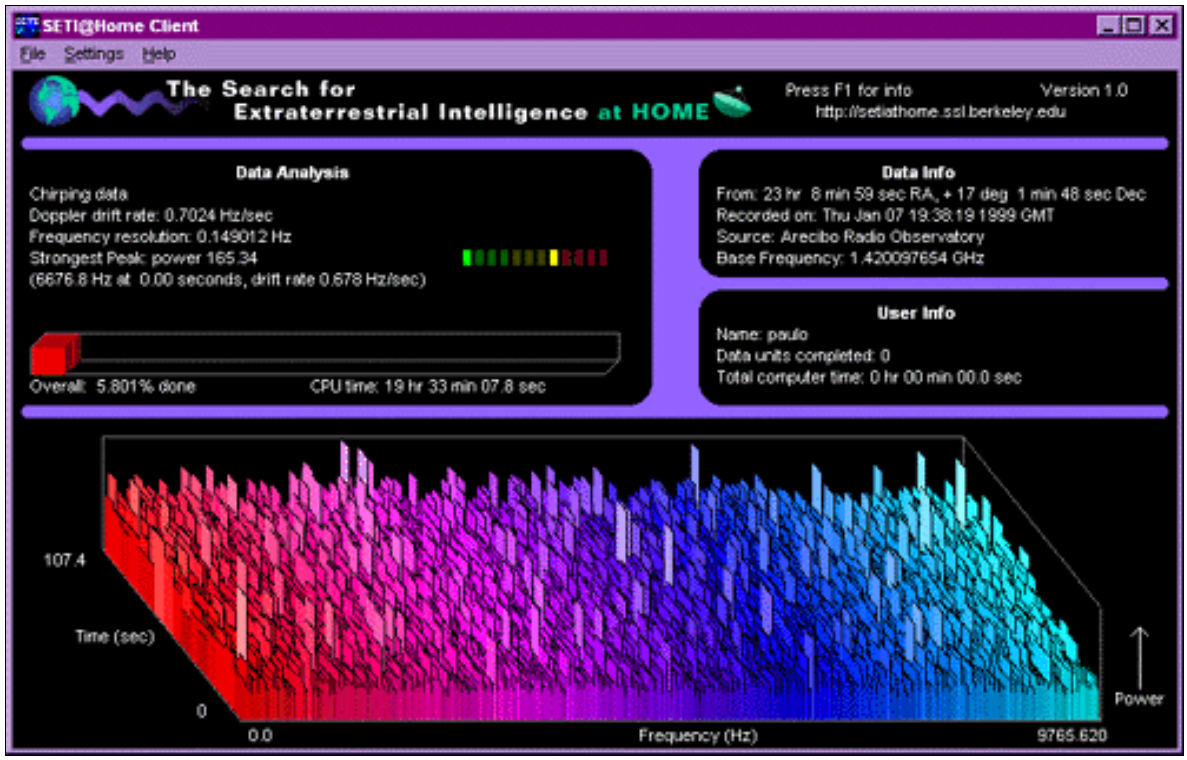


Figura 3.2: Tela do *SETI@home*

Após a instalação do *software* (sugestivamente chamado SETI@home), os clientes fazem um *download* de um pequeno conjunto de dados que será exaustivamente processado. Os resultados, são então, retornados ao servidor e analisados. Deve-se notar que a comunicação entre cliente e servidor ocorre apenas em dois momentos: no início (recebimento dos dados) e após o processamento (envio dos resultados), através de *sockets* TCP/IP convencionais.

Uma novidade por trás deste exemplo é que o programa cliente (figura 3.2) pode ser executado em *background*, (como proteção de tela, por exemplo). Vale destacar que, apenas durante a primeira semana, 700.000 unidades de processamento foram feitas - o equivalente a

350 anos de computação contínua em um microcomputador padrão - e a previsão é de que o projeto envolva milhares (ou até mesmo milhões) de usuários.

- (ii). Novas linguagens de programação

Apesar de ter sido alvo de várias críticas desde o seu lançamento, a linguagem Java continua sendo a principal ferramenta para desenvolvimento de soluções para a Internet. No entanto, propostas similares têm começado a surgir, graças ao promissor mercado de serviços que gigantes do software, como a HP, a Oracle e a Microsoft, além da própria Sun, vislumbram na rede mundial. Uma destas propostas é a linguagem C# (leia-se "C sharp"), lançada no pacote do Visual Studio. A idéia principal é a de geração código baseado em uma linguagem praticamente universal (como é o caso do C), com ligeiras adaptações, permitindo a construção de componentes acessíveis via WWW, chamados de programas ".NET", suportados pela Microsoft.[49] No entanto, o futuro da proposta C# continua permanecendo uma icógnita e somente o tempo determinará o seu sucesso.

3.1.3 – Características da Programação Internet

A seguir, destacamos os aspectos mais relevantes da programação distribuída via *Web*. Um ponto importante é notarmos as diferenças entre as características da programação Internet e as apresentadas anteriormente, em relação à programação paralela.

- **Tipo de concorrência:** As características do Java (enquanto linguagem essencialmente OO), permitem que os mais diferentes tipos de paralelização sejam utilizados, já que, ao lado da paralelização de dados e funcional – possível em linguagens imperativas – permite ainda o uso de paralelização de objetos. Além disso, o modelo de **metaproblemas** é especialmente caracterizado como aplicado à *Web* [Fox, 1996];
- **Nível de complexidade:** Os problemas tratados neste caso não podem apresentar a complexidade dos desafios propostos ao PAD (vide capítulo anterior). Vários fatores, como as restrições impostas para a utilização dos recursos e a instabilidade da rede, contribuem para que se deva optar por um processamento modesto nos clientes;
- **Utilização dos recursos:** Existem limites impostos pela natureza da linguagem para assegurar que nenhum processo remoto possa ultrapassar certos limites de utilização de CPU e memória. Além disso, *applets* comuns (não assinadas) não possuem

[49] Apesar de trabalhar com o conceito de classes, a linguagem C# é consideravelmente mais simples e restrita que o C++, sendo fortemente tipada e não apresentando macros, nem herança múltipla. Note que em vários aspectos, C# se assemelha à Java, de forma que muitos consideram esta nova linguagem como apenas uma tentativa de conquista do mercado crescente de programação Internet pela Microsoft, que é a única a dar suporte à plataforma .NET.

permissão para escrever no disco ou acessar determinados recursos da máquina remota;

- **Nível de comunicação:** O particionamento do problema em tarefas deve ser tal que a necessidade de comunicação seja mínima. Vale a pena salientar que uma das restrições de segurança do `Java` determina que uma *applet* apenas pode se comunicar com o servidor do qual ela proveio. Desta forma, não é possível haver comunicação de dois clientes entre si (a menos que condições especiais, como a assinatura da *applet* seja feita);
- **Distributividade**: Onde reside uma das grandes vantagens do Java: não existem quaisquer restrições sobre **onde** os canais de execução (*threads*) estão sendo executados, desde que se observem as questões inerentes à segurança do sistema;
- **Anonimato**: Um cliente, ao carregar uma *applet* não precisa ser identificado (a menos que se exija). Isto ocorre porque para a grande maioria das aplicações distribuídas, não há relevância de quem (ou onde) está sendo executado aquele processamento;
- **Instabilidade**: O grande desafio da programação Internet é superar os problemas de comunicação e garantir um bom termo ao processamento distribuído. A instabilidade das redes e a presença do **fator humano** (por exemplo, o simples fato de uma página com *applet* Java ser substituída por outra pelo usuário), podem comprometer seriamente a continuidade do processamento.

Ao analisarmos estas características, podemos ter uma noção do que pode efetivamente ser feito na computação distribuída via rede. Contudo, um conceito recente e de cada vez maior importância, deve ser mencionado quando tratamos deste assunto: o de metacomputação, a ser estudado na próxima seção.

3.2. Metacomputação

Embora toda esta evolução seja muito recente, e inúmeras pesquisas estejam em andamento, ao mesmo tempo em que novas tecnologias vão surgindo, alguns conceitos relacionados com este novo paradigma de programação distribuída, feita via Internet, precisam ser definidos.

3.2.1 – Definições e Características

♦ A. O que significa Metacomputação

A origem do termo **metacomputação** (creditada a Larry Smarr, diretor do Projeto CASA, em 1989) deriva de "μετα", sufixo grego que indica uma "ação em comum", de forma que podemos interpretar o termo no sentido de vários computadores compartilhando recursos e agindo juntos para resolver algum problema em comum [Baker, 1999].

Nesta dissertação, chamamos de **metacomputação** o processamento distribuído feito em redes, sejam elas locais, *intranets* ou na Internet [Gray, 1997] e que permite criar um "**supercomputador virtual**" [Foster, 1997] [Vanhelsuwe, 1997], possibilitando "combinar os recursos de milhões de computadores conectados à Internet" [Christiansen, 1997].

De fato, uma das pioneiras visões de metacomputação era a do "uso de poderosos recursos computacionais transparentemente disponíveis para serem usados por meio de um ambiente de rede" [Cattlett, 1992], o que reforça a idéia de um **computador virtual**. Isso é possível tanto pelo aumento do poder computacional das máquinas (considere-se, por exemplo, as características dos nossos *desktops* de hoje) quanto pelo desenvolvimento de novas tecnologias para o aumento da velocidade de transferência de dados em rede, com garantia de qualidade de serviço (como ocorre com a utilização de ATM ou Gigabit Ethernet).

O objetivo principal deste novo paradigma é o de aproveitar o poder de processamento ocioso de centenas de milhares de computadores conectados à Internet. Este potencial de processamento pode ser facilmente observado quando se compara numericamente a quantidade de máquinas das mais diversas plataformas, conectadas à rede, com o número de computadores de grande porte, disponíveis para PAD (vide figura 1.1). Em outras palavras, busca-se colocar a rede mundial a serviço do processamento. Esta não é uma tarefa trivial até porque muitos são os desafios a serem superados, como diversidade de plataformas, velocidade de comunicação, controle dos processos e viabilidade no particionamento das tarefas.

O ideal futuro, no entanto, é de que se possa utilizar a metacomputação de forma análoga à utilização de energia gerada pelas redes elétricas atuais: que, tal qual um usuário simplesmente liga um dispositivo elétrico, mesmo sem saber de onde provém a energia, ele possa se utilizar transparentemente dos recursos de rede na resolução de um problema. Este exemplo ilustra a definição para o conceito das **malhas computacionais** mencionadas anteriormente na seção 1.1.2.C. Neste sentido, as malhas constituem a infra-estrutura tecnológica para a provisão de PAD independente da distribuição geográfica dos recursos e dos usuários [Foster, 1998].

♦ B. Características

Embora relacionado com o processamento feito via *Web*, a metacomputação não se prende a uma plataforma ou tecnologia. Deve-se notar que, por trás deste conceito, persistem

as características descritas na seção anterior sobre a programação Internet. Contudo, três passos definem basicamente esse paradigma [Baker *et al*, 1999]:

1. A integração dos recursos individuais de software e hardware;
2. A implementação de uma interface (*middleware*[50]) para prover uma visão transparente dos recursos anteriores;
3. O desenvolvimento e a otimização de aplicações distribuídas.

Uma observação importante é que, estritamente falando, os componentes (ou recursos) individuais, não têm importância. Ao contrário, o que realmente importa é a forma como tais componentes trabalham juntos e possibilitam uma visão uniforme do conjunto. Alguns princípios gerais identificam exatamente as principais funcionalidades da metacomputação:

- Não interferência nos níveis de administração de um sistema;
- Não substituição de sistemas, protocolos ou serviços existentes;
- Permissão para que domínios remotos possam participar do ambiente distribuído virtual;
- Provisão de suporte a recursos heterogêneos.

Além destas metas, alguns pontos de considerável complexidade são considerados dentro dos princípios ideais para a construção de ambientes virtuais para processamento:

- Provisão de infra-estruturas para recuperação de erros e falhas;
- Não determinismo das tecnologias a serem utilizadas, de forma que os usuários não necessitem se prender a paradigmas, linguagens ou ferramentas específicas para a utilização do sistema[51] [Baker *et al*, 1999];
- Elaboração de políticas de sincronização adequadas.

♦ C. Componentes

A formação de um computador virtual apresenta uma organização que reflete os passos anteriormente definidos [Baker, *ibid.*]:

---

[50] O conceito de *middleware* é, conforme o nome indica, um elemento intermediário entre dois níveis de máquinas abstratas, funcionando como interface entre um e outro. Algumas fontes de referência se baseiam no uso desta camada intermediária para implementação de componentes de *software* (vide, p.ex., [Shoffner, 1998]).

[51] Note que isso somente se torna possível através de uma padronização tecnológica, tal como ocorreu com o `TCP/IP`, em termos de protocolos de rede. Isto demanda tempo e prende-se a um fator tecnológico determinado. No tocante à metacomputação, essa tendência pode ser percebida, por exemplo, com a inclusão de `JRE's` nos mais diversos sistemas.

- **Recursos físicos (*hardware*):** Formado basicamente por processadores e memória, representam o *hardware* que efetivamente prover o poder computacional do sistema;
- **Recursos de *software*:** Incluem os programas (envolvendo sistemas e linguagens) e as bibliotecas que permitem a interoperação dos vários recursos disponíveis na rede, provendo, portanto, um nível de comunicação entre os EPs;
- **Ambiente virtual:** Compreende os módulos responsáveis pela configuração, gerência e manutenção do ambiente de metacomputação. Ele envolve basicamente duas entidades: os sistemas administrativos (nível de gerência) e operativos (em nível de usuário);
- **Acesso remoto e recuperação de dados:** Engloba rotinas específicas para tratamento dos dados que atuam no sistema, com o objetivo de obter melhor desempenho com segurança e tratamento para eventuais falhas no processamento.

3.2.2 – Modelos de Metacomputação

Várias alternativas têm sido propostas para a utilização da *Web* em processamento distribuído. Aqui, vamos enquadrá-las em dois grandes grupos: as que envolvem o desenvolvimento de **classes de comunicação** e as que se relacionam com a criação de ferramentas que propõem não apenas bibliotecas, mas **arquiteturas** para este tipo de processamento.

♦ A. Classes de Comunicação e Interfaces

A abordagem aqui adotada é a de que se pode efetuar o processamento através do uso de uma dada linguagem (como `Java`), com a inclusão de classes que permitam a comunicação entre diferentes EPs que participam da computação distribuída. A comunicação entre dois pontos utiliza, basicamente, *sockets*, introduzindo a idéia de **DAMPPs** - *Distributed Applet-based massively Parallel Processing* [Vanhelsuwe, 1997] como alternativa para o processamento distribuído. A vantagem deste modelo é sua praticidade e facilidade de implementação, já que é análogo ao modelo das bibliotecas de passagem de mensagens na programação de PAD. A literatura nos apresenta alguns exemplos:

- **MOM** [Shoffner, 1998]: O ***Message-Oriented Middleware*** permite modelar mensagens como eventos a serem tratados diretamente pela API do programa, provendo um conjunto de serviços disponíveis para os clientes. Este modelo apresenta a vantagem de prover passagem de mensagens diretas, sem custos adicionais (como a conversão para código nativo) e a implementação em `Java` provê a portabilidade necessária para a independência de plataforma. No entanto, vários pontos críticos podem ser observados: a necessidade de familiarização com o

mecanismo de mensagens (que não é padronizado), a inexistência de mensagens blocantes e a extrema simplicidade, que compromete a generalização do modelo [ib, p.4]. A figura 3.3 ilustra o funcionamento esquemático do MOM, destacando a ação do *broker*, empregado em situações onde o uso de `CORBA/RMI` é demasiadamente complicado e, por outro lado, nas quais o uso de RPCs é insuficiente.

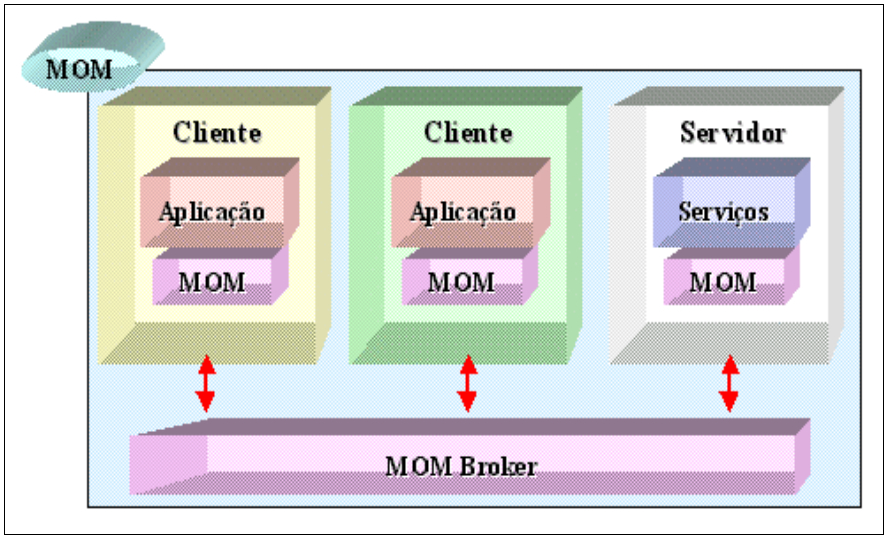


Figura 3.3: Modelo de funcionamento do MOM

- O **MPI Java Wrapper** [Chang, 1996] consiste de um pequeno conjunto de classes que provêem métodos públicos e **nativos** análogos às rotinas `MPI` e que são traduzidos internamente para as chamadas implementadas naquela própria MPL (figura 3.4). Este modelo fornece algumas vantagens como familiaridade e padronização, devido à compatibilidade com o padrão `MPI`; bom desempenho, versatilidade e flexibilidade de implementação. No entanto, a necessidade de métodos nativos traz como principal desvantagem, a perda de portabilidade, já que o sistema necessita da existência daquela MPL para sua operação. Outros problemas como a incompatibilidade de implementação de certas funções (como `MPI_Request` e `MPI_Op`), além da constatação de problemas durante a implementação de processos concorrentes, foram observadas [ib., p.8].

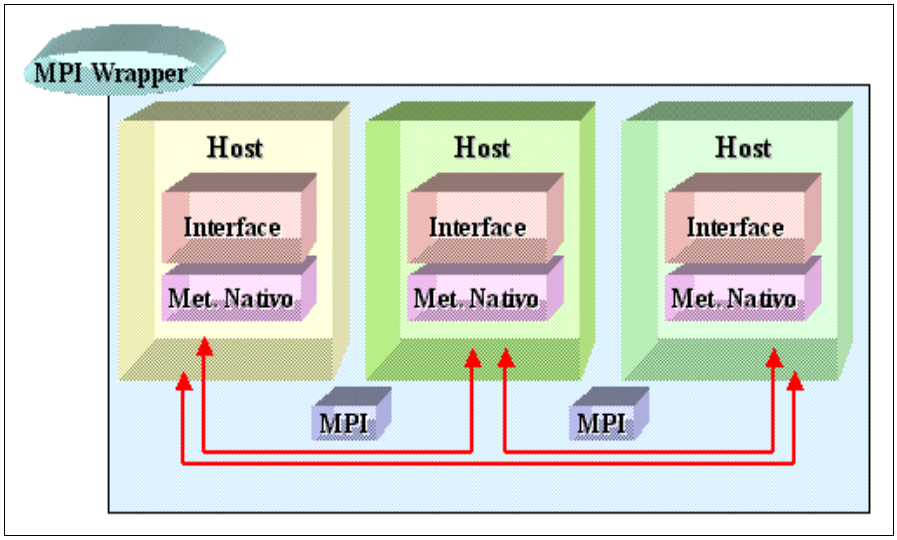


Figura 3.4: Modelo de funcionamento do MPI Wrapper

- O **Visper** [Stankovie, 1998] é uma ferramenta que simplifica o desenvolvimento e testes de programas paralelos através de uma abordagem modular de objetos (figura 3.5). O ambiente se constitui de uma console com interface gráfica e de um *cluster* para processamento formado por estações que executam um *daemon* como aplicação Java. No nível de aplicação, as primitivas de MP provêem de uma classe (`Vcomms`), responsável pela implementação de mensagens que seguem o padrão `MPI`, incluindo suporte a funções "blocantes", "não-blocantes" e coletivas. Note que, de forma semelhante ao que acontece com o `PVM`, o Visper apresenta *daemons* responsáveis pela comunicação entre os clientes, além de uma console para os comandos.

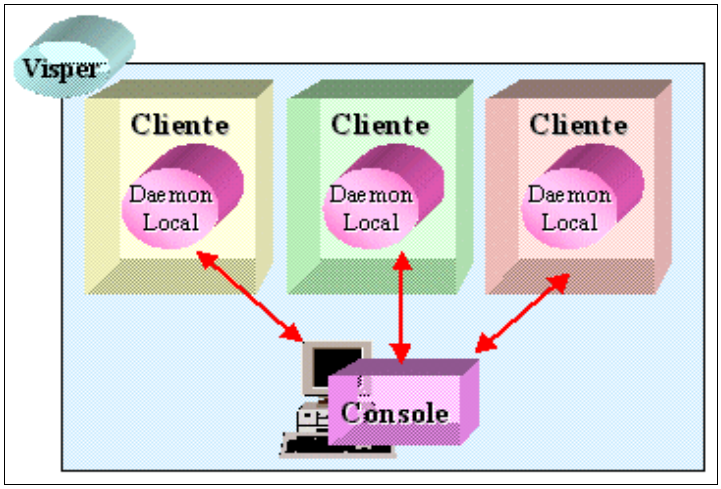


Figura 3.5: Arquitetura Visper

Outros exemplos da utilização de classes de comunicação podem ser encontrados em inúmeras outras referências, mas apresentam funcionamentos análogos aos anteriores. Merece destaque, por exemplo, o **`IceT`** [Gray, 1997], que apresenta características como divisão dinâmica das máquinas virtuais e ambiente cooperativo para programação em múltiplas

máquinas virtuais, utilizando uma sintaxe análoga ao PVM (assim como o MPI Java Wrapper assemelha-se ao MPI).

- B. Ferramentas e Arquiteturas baseada em Agentes

Neste segundo caso, não existe apenas uma utilização de classes para comunicação (como no caso anterior): há toda uma proposta de arquitetura e utilização de ferramentas, com o objetivo de tornar o processamento distribuído mais eficaz, o que torna os modelos aqui apresentados consideravelmente mais complexos e eficientes. Alguns exemplos irão ilustrar melhor este modelo:

> **Javelin** [Christiansen, 1997]: A proposta desta ferramenta é criar uma infra-estrutura para a execução de aplicações paralelas numéricas de granulosidade grossa. Embora apenas uma *applet* seja visível para os usuários, a arquitetura usada envolve um conjunto de agentes (*brokers*) que são os responsáveis por coordenar e suprir a demanda pelos recursos computacionais distribuídos. O esquema da figura 3.6 ilustra o funcionamento do Javelin, com os seus cinco componentes (tarefas, *hosts*, clientes, *brokers* e servidor): quando um cliente requer uma tarefa, o *broker* é responsável por escalonar uma determinada tarefa e enviá-la ao cliente requisitante. Os usuários automaticamente tornam suas máquinas disponíveis para *hosts* que participam do processamento distribuído, através de um *download* da *applet* que dispara uma pequena *thread*, a qual fica esperando pelas chamadas. A implementação utiliza RMI e *servlets*[52] (programas que estendem a funcionalidade do servidor), e que permitem uma melhor integração, provendo serviços como *upload* e gerenciamento de processos.

[52] Os *servlets* – que possuem uma terminologia análoga aos *applets* dos clientes – são classes que ficam no servidor e possuem as funcionalidades dos conhecidos programas CGI – Common Gateway Interface – permitindo uma adequada comunicação com os clientes, processando suas requisições e retornando suas respostas. Este padrão proposto pela Sun Microsystems pode ser encontrado em: http://java.sun.com/products/servlet/index.html.

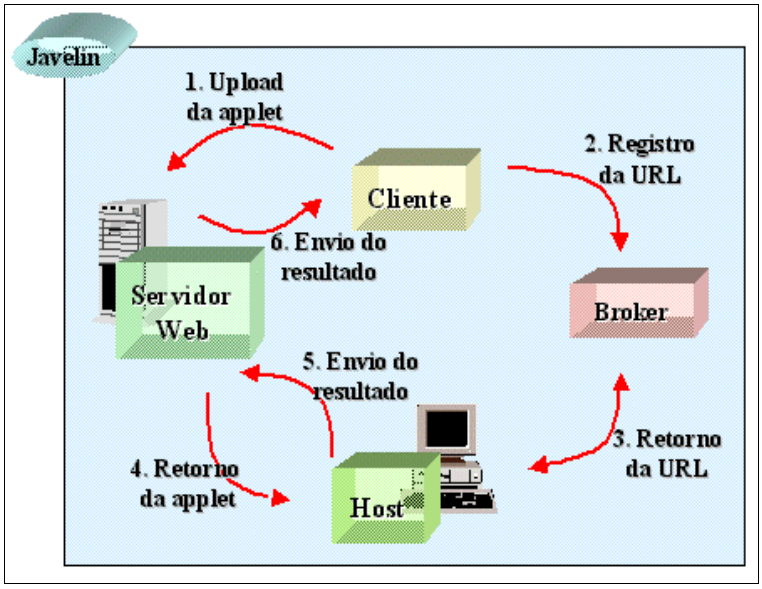


Figura 3.6: Modelo de funcionamento do Javelin

- **TANGOsim** [Beca, 1997]: Esta ferramenta combina um ambiente colaborativo de trabalho com filtros para passagem de mensagens. Seu esquema de funcionamento é mostrado na figura 3.7. O sistema provê uma sessão de gerência, verificação de *logs* e eventos e sincronização, além de mecanismos de comunicação. A arquitetura baseia-se em dois elementos principais: um *daemon* local (implementado como *plug-in* `LiveConnect`) que permite a comunicação do cliente e um servidor central, responsável pelo controle de todos os clientes. A interação com BDs (bancos de dados) torna possível ainda a sua utilização com aplicações como projetos de aeronaves, simulações de clima e *whiteboards* compartilhados. Outros trabalhos, como a construção de **`GIS`** (*Geographical Information Systems*) tridimensionais em `VRML`, foram propostos para desenvolvimento.

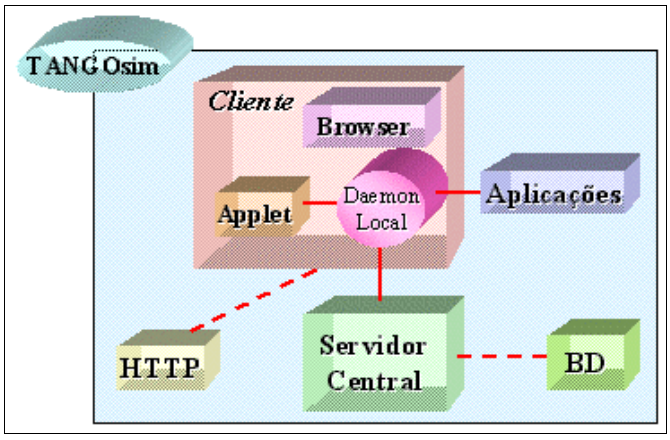


Figura 3.7: Modelo de funcionamento do TANGOsim

- **JAVADC** [Chen, 1997a e 1997b]: Denominado de "*Java for Distributed Computing*", o `JAVADC` é um modelo de computação distribuída baseado na *Web*, que permite a execução de programas SPMD que utilizam `PVM`, `pPVM` (uma extensão da MPL

anterior, que permite o uso de comunicação paralela) e MPI. Em resumo, ele permite que um usuário na Internet possa configurar um ambiente PAD, executar a aplicação e monitorá-la. Sua implementação utiliza o pacote proprietário IFC (*Internet Foundation Classes*) da Netscape, além de uma interface cliente, um servidor HTTP, um *cluster* para processamento e um *site* de aplicações (vide figura 3.8).

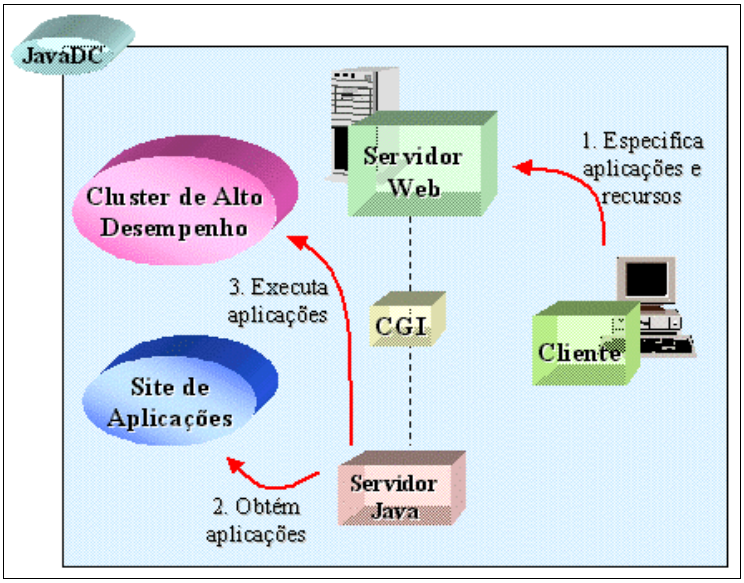


Figura 3.8: Arquitetura do JAVADC

A extensão do modelo originou outros ambientes como o **Arcade** e o **CDCE** (*Collaborative Distributed Computing Environment*) [Chen, 1997b], que são ambientes integrados para projeto colaborativo. Uma observação importante é que a arquitetura usada no JAVADC servirá de base para a implementação dada em nossa dissertação, com algumas importantes diferenças (cf. seção 4.1.3).

Para concluirmos, vale mencionar três das mais recentes propostas para o modelo de metacomputação hoje [Baker, 1999]:

- **GMT – Globus Metacomputing Toolkit** [Foster *et al*, 1998]: Provendo uma infra-estrutura para tratamento de ambientes distribuídos como máquinas virtuais, o projeto visa fornecer uma ferramenta para a construção de "malhas computacionais". Isto é possível se estabelecendo um conjunto básico de serviços e requisições disponíveis no sistema, através de uma arquitetura em camadas construídas por bibliotecas bem definidas (especificações que podem ser encontradas em http://www.globus.org). Basicamente, o GMT apresenta uma arquitetura complexa e se utiliza de uma combinação de recursos: serviços básicos de comunicação em Nexus, serviços de segurança e autenticação em **GSI** (*Generic Security Service API*), interfaces remotas para arquivos via **GASS** (*Global Access to Secondary Storage*), um serviço de informações sobre diretórios (**MDS** –

Metacomputing Directory Services) e um administrador para alocação de recursos **GRAM** (*Globus Resource Allocation Manager*). O sistema está disponível em Unix (algumas versões) e sendo portado para NT.

- **Legion** [Grimshaw *et al*, 1997]: Nessa abordagem, têm-se um modelo unificado de objetos que serão usados para construir os sistemas virtuais. Basicamente, tudo é encarado como um objeto (sejam componentes de *hardware* ou *software*), administrados por classes definidas pelos próprios usuários, com capacidades de persistência, utilização de cache, mapeamento lógico para endereços físicos e utilização de grafos dirigidos (DAGs) para organização da informação. As interfaces do sistema são descritas através de IDL (*Interface Definition Language*) e a implementação usa comunicação via TCP/IP, com resolução DNS, além de um núcleo de administração feito em Java, com validação de segurança no modelo de chaves públicas e privadas. Sumariamente, Legion utiliza uma base de informação sobre os recursos (Collection), um módulo para seu mapeamento (Schedule) e, finalmente, um agente responsável pela sua implementação (Enactor), sendo que a programação suporta tanto C++ (na versão MPL – *Mentat Programming Language*) e Fortran (BFS – *Basic Fortran Support*). Legion já foi portado para sistemas como Linux, Solaris, AIX, IRIX, DEC e Cray.
- **WebFlow** [Haupt *et al*, 1999]: Descreve um outro modelo de metacomputação via *Web*, tal que usuários possam compor aplicações distribuídas, usando módulos em uma arquitetura de três camadas, composto por *applets*, *servlets* (*middleware*) e códigos para bases de dados e simulações. Uma grande vantagem é a utilização de componentes visuais, formando grafos computacionais distribuídos, fornecendo assim, um alto nível de comodidade para os usuários. Na realidade, o WebFlow é implementado se utilizando do GMT (incluindo serviços como MDS, GRAM e GASS). A comunicação entre os componentes se utiliza de IIOP e o sistema suporta o emprego de outros protocolos como o HTTP e o FTP, através de objetos em Java que permitem encapsular módulos em Fortran, C, Pascal e Java, seja em sistema Solaris, IRIX ou NT.

No último item, destacaremos algumas aplicações práticas em metacomputação que são um destaque a parte: ao invés de apenas proporem modelos, elas buscam resolver concretamente problemas de forma econômica e eficiente.

♦ C. Metacomputação Aplicada

Talvez o maior impacto da utilização deste novo paradigma possa ser sentida nas aplicações de metacomputação. A seguir, mencionamos dois projetos inicialmente concebidos dentro deste contexto (citados em [Baker, 1999]).

- **FAFNER**: Descrito em http://www.npac.syr.edu/factoring.html como um sistema cooperativo para a fatoração de grandes números, o projeto FAFNER (1995) busca utilizar o ambiente distribuído da *Web* para quebrar chaves de criptografia – como as utilizadas pelo algoritmo RSA (vide http://www.rsa.com). Se utilizando basicamente do protocolo HTTP, o FAFNER realiza uma elegante combinação de técnicas (como o NFS – *Number Field Sieve* ou "peneiração de campos numéricos") e tecnologias que tornaram a abordagem eficiente, tais como:
  - Algoritmos simples capazes de executar tanto em máquinas de baixo custo (com 4MB de memória), quanto em computadores superescalares;
  - Registro anônimo, o que possibilita um nível elementar de segurança para os usuários;
  - A utilização de scripts e a organização de uma rede hierárquica que necessita de uma mínima intervenção humana.
- **I-WAY**: Buscando fornecer recursos para visualização científica de dados, o projeto experimental I-WAY (*Information Wide Area Year*, de 1995), envolvendo 17 diferentes domínios interligados por diferentes redes e protocolos. A utilização de servidores Unix para prover autenticação uniforme, criação de processos e funções de comunicação entre os clientes, com níveis de segurança e confiabilidade para aplicações de realidade virtual.

Outras propostas para a utilização de metacomputação aplicada podem ser encontradas em programas como o **WebWindows** [Fox e Furmanski, 1997], **Net Pellpack** [Markus, 1997], **Netsolve** [Casanova, 1995] e **Aglets** [Harrison, 1995]. Aplicações como a interface de recursos para biologia do NCSA, a simulação de tempo distribuída da NASA, a interface do simulador NIST para ambiente paralelo e o desenvolvimento de **VPLs** (*Virtual Programming Laboratories*) são citados como exemplos de peso da utilização de metacomputação [Fox e Furmanski, 1997a e 1997b].

Resumo

Partindo dos conceitos discutidos neste capítulo (relacionado com processamento na Internet) e as noções da importância do PAD para a computação (capítulo anterior), estamos, agora, em condições de apresentar a ferramenta que é objeto de nossa dissertação de mestrado e que pretende se

utilizar da Internet, para prover facilidades que auxiliarão o desenvolvimento de programas paralelos.

PARTE II

DESCRIÇÃO DA FERRAMENTA

Capítulo 4: Apresentação de WVM
Capítulo 5: Arquitetura

Capítulo 4

Apresentação de WVM

Introdução

Neste capítulo, apresentamos a ferramenta objetivo desta dissertação e que será descrita na seção que se segue. Para tanto, nos valeremos dos conceitos dados no primeiro capítulo e de uma breve análise dos paradigmas de programação apresentados nos últimos capítulos.

A seção 4.1 se ocupará da comparação entre os modelos de programação na Internet e em PAD paralelo. Já a seção 4.2 sintetiza o significado e objetivos da alternativa que apresentaremos neste trabalho.

4.1. Conciliando PAD com a Programação Internet

4.1.1 – Objetivo

O deslocamento contínuo da atenção de projetistas de *hardware* rumo a ambientes de processamento distribuído e às alternativas de baixo custo para processamento de alto desempenho (seção 1.1) tem acompanhado a evolução dos sistemas de *software* (seção 1.2) a níveis cada vez mais elevados de programação, envolvendo paradigmas distribuídos e tecnologias até recentemente não imaginados.

Contudo, a gama de arquiteturas, sistemas computacionais e programas hoje existentes representam, por si mesmo, um desafio. A opção pela melhor alternativa, considerando-se basicamente o fator custo × benefício, pode não ser uma decisão fácil. Sob este aspecto, vamos nos concentrar basicamente em dois fatores (mencionados nos capítulos 2 e 3): a programação paralela-distribuída em *clusters* e a programação distribuída na Internet.

Nosso objetivo, aqui, consiste em analisar um e outro, a fim de buscarmos uma resposta à questão central de qual arquitetura, sistema e recursos utilizar para resolver

problema computacional, satisfazendo requisitos como custos de implantação, desenvolvimento e manutenção e buscando-se, acima de tudo, um desempenho satisfatório em termos de rapidez de processamento.

4.1.2 – Dificuldades

Se observarmos as características da programação paralela-distribuída (item 2.3.5), e as compararmos com os pontos mais relevantes relacionados à programação Internet (item 3.1.3), podemos elaborar um quadro comparativo entre uma e outra proposta, que está resumidamente apresentado na tabela 4.1:

Característica	Programação Paralela / PAD	Programação Internet
Paralelismo[53]	Dados / funcional.	Objeto / Metaproblemas.
Complexidade	Elevada (própria para processamento em *clusters* e computadores paralelos).	Pequena (adequada para um processamento global).
Recursos	Podem ser exaustivamente utilizados.	Apresentam sérias restrições para seu uso remoto.
Independência entre processos	Desejável. Permite-se a comunicação entre processos concorrentes (cliente-cliente).	Essencial. Na maioria dos casos, apenas se permite a comunicação do cliente com o servidor.
Comunicação	Pode ser considerável, devendo ser sempre muito rápida.	Extremamente lenta e deve ser fortemente evitada.
Ambiente	Local	Distribuído globalmente.
Segurança	Forte controle de acesso.	Ausência de identificação (anonimato) ou por certificações.
Granulosidade	Fina ou Grossa.	Grossa.
Continuidade	Garantida.	Não assegurada, devido à enorme instabilidade da rede.
Dependência de Arquitetura	Absoluta. O processamento é feito sob uma arquitetura bem específica.	Bastante flexível: dependendo da tecnologia empregada, pode ser ignorada.
Custos	Variáveis: altos, médios ou baixos, de acordo com as opções definidas. Contudo, em comparação com arquiteturas vetoriais existe uma considerável economia.	Baixos, já que permite o processamento em redes remotas.
Curva de Aprendizagem	Relativamente rápida, caso apenas chamadas de MPLs sejam necessárias em cima do código de uma linguagem já existente.	Considerável, especialmente pelo fato de ser uma tecnologia nova ainda em desenvolvimento.

Tabela 4.1: Comparação entre a Programação Paralela e Internet

Apesar de várias semelhanças, o que se destacam são os pontos divergentes entre as duas opções de programação. Assim, considerando-se tais características, a interrogação central é, sem dúvida, a de **como** conciliar essas duas alternativas. Para tanto, devemos analisar até que nível isto é possível e o que deve ou não ser priorizado. Serão essas as questões que nos ocuparão daqui em diante...

4.1.3 – Critérios

[53] Segundo a classificação proposta por Fox (1996).

Um ponto importante a ser destacado é o de que uma simples observação da tabela 4.1 já deixa claro que não se pode resolver o problema sem se definir algumas **prioridades** ou **critérios**, já que, sob vários aspectos, as alternativas anteriores excluem possibilidades comuns. A primeira coisa a ser feita é, portanto, uma análise mais criteriosa do problema a ser tratado, perante esses fatores.

Para isso, podemos dar algumas pistas (colocadas na forma de questionamentos), que nortearão o caminho a ser tomado. Vale a pena observar que devemos analisar a questão numa certa ordem, que irá progressivamente se aproximando da realidade física da implementação:

1. **Natureza do problema**: Qual o problema a ser tratado? Quais os pré-requisitos (entradas) e resultados (saídas) esperados? Qual a sua complexidade? Como pode ser fragmentado? Qual a sua granulosidade? Qual o nível de dependência entre os processos (e consequentemente, sua comunicação)?
2. **Recursos:** Quais os recursos a serem utilizados? Como podem ser acessados? Qual nível de segurança pode (ou deve) ser empregado? Existem alternativas? Qual a necessidade de continuidade no processamento? Como será a organização dos subprocessos? Qual o tamanho do *cluster* de processamento desejado?
3. **Outros fatores:** Por fim, pode-se partir para a análise de fatores secundários como a necessidade de uso de um ambiente não local, as necessidades de aprendizagem e a reusabilidade de código e questões referentes à portabilidade e interoperabilidade entre máquinas de arquiteturas diferentes ou sistemas distintos, com seus impactos para a sua adoção da parte dos usuários.

Algumas observações sobre as questões acima devem ser salientadas: primeiramente, nos preocupamos com aspectos relacionados com o **problema** em si e seu nível de complexidade (por exemplo, se existe ou não alguma explosão combinatória), bem como da possibilidade de se obter ganhos com sua paralelização, considerando-se todo o processo de comunicação e sincronização entre os processos. Sem uma análise criteriosa destas características teóricas, correríamos o risco de se criar altos custos para se obter soluções não satisfatórias, seja no tempo, seja na qualidade. Em seguida, partimos de aspectos práticos que derivam da **tecnologia** a ser empregada, às necessidades que definem o uso de uma alternativa tecnológica e as vantagens e desvantagens desta opção. Por fim, consideramos elementos de caráter mais **subjetivos** (embora ainda relacionados com os aspectos de programação), que vão desde a economia decorrente do modelo implementado – quando se consideram, por exemplo, as curvas de aprendizagem – à facilidade de utilização do mesmo.

As respostas a essas perguntas definirão, a alternativa a ser utilizada, embora, em alguns casos, se possa optar pela implementação de diferentes soluções. Por exemplo,

suponhamos que estejamos com um problema de granulosidade grossa e que o nível de comunicação seja elementar (apenas para obter os dados iniciais de processamento e enviar as respostas ao final dos cálculos). Neste caso, se a tecnologia o permitir, podemos optar por uma ou outra alternativa. Um exemplo: a possibilidade de se utilizar comunicação por mensagens, em si mesmo não implica que MPLs devam ser usadas: como vimos, a implementação de algumas interfaces e classes de comunicação, permitem que a programação na Internet – em especial, com o uso de classes Java específicas – possa ser adotada, desde que suas limitações sejam observadas.

Finalmente (e não menos importante), o critério que norteará a opção final deve basear-se numa variável fundamental: o fator **custo** × **benefício**. Os custos referem-se tanto aos gastos financeiros necessários para a implantação da solução desejada em todos os níveis (técnico, tecnológico ou pessoal), quanto à complexidade de implementação daquela alternativa. Já os benefícios estão fundamentalmente relacionados com o desempenho, isto é, o tempo que se dispõe para o processamento e a execução das tarefas. Caso este seja um fator crítico, não resta dúvidas de que o paralelismo é uma das melhores alternativas. Entretanto, isso não invalida que ferramentas da programação Internet possam ser utilizadas. Será exatamente em cima deste ponto que iremos apresentar a alternativa da ferramenta WVM.

4.1.4 – Propostas

Algumas das alternativas para conciliarmos o processamento distribuído em *clusters* com a Internet já foram apresentadas na seção de arquiteturas baseadas em agentes (seção 3.2.2.B): no protótipo `TANGOsim`, um *daemon* local permite que o programas possam ser executados nos *hosts* que participam do sistema. Já a ferramenta `JAVADC` permite que um determinado cliente execute programas em um ambiente de processamento completamente distinto. Ainda na seção 3.2.2.a (classes de comunicação), pudemos observar facilidades como a ferramentas `Visper`, que permite aos clientes iniciar processos a partir de uma *console*.

No entanto, algumas observações para esclarecer a diferença entre as alternativas anteriores e o modelo que apresentamos neste capítulo devem ser feitas:

- A maioria das ferramentas que utilizam `Java` (como `JAVADC`, `TANGOsim`, `IceT` e `Visper`) fazem uso das facilidades de classes especiais como as `IFC`s ou *Internet Foundation Classes* e o `LiveConnect` [cf. `http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/`], desenvolvidas por fabricantes como a `Netscape`. Assim, o código não pode ser considerado 100% aberto, estando sujeito a certas restrições, que aliás, são as responsáveis pelas facilidades implementadas;

- Outras ferramentas permitem apenas escalonar programas, já editados, para sua execução. Ambientes integrados ainda são restritos à programação Internet, não conciliando esta programação com o PAD;
- Finalmente, algumas arquiteturas propostas apresentam programas com sintaxe própria, não padronizados e com características particulares que dificultam a portabilidade dos códigos.

Em cima de tudo isto, podemos agora, apresentar a ferramenta WVM como um diferencial das alternativas existentes. Não é, como veremos, uma solução definitiva, mas uma ferramenta de apoio à programação e desenvolvimento em ambientes Internet para o processamento em *clusters*.

4.2. WVM - *Web Virtual Machine*

4.2.1 – Apresentação

A ferramenta aqui descrita e que, de agora em diante, passaremos a chamar de **WVM** (***Web Virtual Machine***) tem como objetivo conciliar vários aspectos da programação Internet com aspectos da programação distribuída de PAD.

Figura 4.1: Logotipo WVM

Em outras palavras, o que propomos é a integração das etapas envolvidas na programação paralela-distribuída, tais como edição, compilação, execução, conexão (`Telnet`) e transferência de arquivos (`FTP`), num único ambiente de programação, baseada nas tecnologias Internet e WWW.

♦ A. Considerações de Projeto

Todo o ambiente WVM foi desenvolvido em API Java padrão, sem a inclusão de classes proprietárias, isto é, de componentes de *software* (que em `Java` são chamados "*beans*"[54]) ou códigos de terceiros, sendo, pois, completamente compatível com os ambientes de desenvolvimento. Sobre este aspecto, existe uma dupla vantagem:

---

[54] Os *plugins* (do termo inglês "grãos") identificam componentes de software que podem ser acrescidos ao ambiente de desenvolvimento, permitindo um crescimento funcional da linguagem. Em termos de Java, vale a pena notar que o termo é bastante ilustrativo, levando-se em conta a identificação desta linguagem com a simbologia do café.

- Por um lado, isto permite a sua execução em qualquer computador com um suporte mínimo a uma máquina virtual Java (JVM), independente de aplicativos ou bibliotecas extras;
- Além disso, o fato de se evitar API proprietária visa permitir uma fácil manutenção de código, sem a necessidade de utilização de qualquer IDE específica[55].

♦ B. O que é WVM

Uma observação importante é de que não se pretende apresentar uma alternativa para a resolução de problemas por metacomputação propriamente dita, mas de permitir que o **PAD possa ser mais facilmente acessível a usuários de ambientes já existentes**, aproveitando-se das vantagens que a programação Internet apresenta. Isto significa que ainda focalizamos problemas de considerável complexidade, com granulosidade e comunicação variáveis, onde as exigências por recursos podem ser consideráveis e haja a possibilidade de reutilização de códigos. Porém, ao lado destas características (próprias da programação paralela), permitimos outras funcionalidades típicas da programação globalmente distribuída: acesso ao sistema de forma independente, a baixos custos, com facilidades de editoração, execução remota e monitoramento das aplicações.

Do ponto de vista do usuário, a interface cliente apresenta facilidades e funciona como um *middleware* (vide, por exemplo, [Shoffner, 1998]) entre este e o servidor (vide item 5.1.2.B). Internamente, WVM se vale de uma **metalinguagem** (ML)[56] (seção 5.3) para codificação dos comandos enviados ao servidor, com suas respectivas funções. Além disso, buscou-se elaborar um ambiente, ao mesmo tempo simples (de fácil compreensão) e dinâmico (configurável de acordo com o usuário), que será melhor explicado no próximo capítulo e cuja implementação será detalhada no Capítulo 6.

4.2.2 – Aspectos Práticos – Um Pouco de História

O que está por trás da ferramenta WVM são aspectos práticos que provém da realidade experimentada nos últimos anos no Centro Nacional de Processamento de Alto Desempenho no Nordeste – o CENAPAD-NE[57] – localizado em Fortaleza, um dos cinco grandes Centros de Processamento espalhados pelo país.

---

[55] A maioria dos ambientes integrados de desenvolvimento (IDE's) são proprietários e apresentam custos variados, de acordo com a sua finalidade e o público alvo, destacando-se basicamente duas versões: uma padrão (*standard*, também chamada de versão educacional ou individual) e outra corporativa (*professional* ou ainda *database* ou *enterprise*).

[56] O conceito de **metalinguagem** é muito comum em algumas áreas da computação e trata da utilização de uma gramática de ordem superior (com **sintaxe** – representação – e **semântica** – significado) para retratar alguma outra expressão lingüística. Ou seja, trata-se de uma linguagem sobre outra linguagem.

[57] Maiores informações sobre o CENAPAD-NE podem ser obtidas em http://www.cenapadne.br/.

Percebeu-se ali que a existência de recursos computacionais (como nós de processamento e servidores de disco) e sua disponibilidade, em si mesmo vantajosos, ainda apresentam alguns entraves que dificultam uma maior aceitação da alternativa de PAD perante a comunidade acadêmica e científica local. De fato, alguns pontos podem ser enumerados neste sentido:

- A necessidade de treinamento em processamento paralelo-distribuído, a fim de que esse novo paradigma de programação possa ser assimilado e adequadamente compreendido tanto pelos cientistas quanto programadores requer tempo e custos;
- A limitação nas **quantidade** de ferramentas existentes: em geral, utilizam-se MPLs sobre as linguagens convencionais – `C` e `Fortran` ou ambientes proprietários (como o `IBM PE`[58] para sistemas SP/2);
- A questão relativa à **qualidade** (ou natureza) dos programas disponíveis: a carência de ferramentas mais amigáveis e interativas, que permitam aos usuários sem muita experiência nos sistemas específicos de tais centros, utilizar o seu poder computacional;
- A existência de pré-requisitos consideráveis para a utilização das linguagens e bibliotecas disponíveis, como familiaridade com o `Unix` e bons conhecimentos de programação imperativa clássica (seja `C ANSI`, seja `Fortran 77`).

Em outras palavras, além de recursos de *hardware* e *software*, um terceiro fator – o de formação de recursos humanos – torna-se necessário a fim de prover o Centro de um corpo de usuários capacitados para a utlização adequada dos recursos ali existentes. A experiência mostra que, apesar do convencimento das vantagens ao se adotar o processamento em *clusters* ou computadores escalares, vários pesquisadores sentem-se inibidos em repassar por todo um processo de formação e treinamento nas novas tecnologias e não foram poucos os que apresentaram queixas acerca das dificuldades iniciais neste processo.

Percebe-se, ainda, que a mudança de uma abordagem tecnológica, bem como de um modelo tradicional de programação e processamento, não é uma atividade fácil, já que implica numa modificação do próprio paradigma de programação. Vários cursos ministrados no `CENAPAD-NE` nos últimos anos, relacionados com tecnologias de processamento paralelo demonstraram isso (cf. `anexo C.3`). A utilização de novas tecnologias e a proposta de ferramentas para auxiliar na formação de maior número de pessoas, promete ser um benefício

[58] O `Parallel Environment` corresponde a um ambiente integrado de facilidades, existentes na arquitetura SP/2 da `IBM`. Eles apresentam ferramentas como visualizador de *tracefiles* e uma biblioteca de *message-passing* análoga ao MPI.

considerável na continuidade de formação de pesquisadores e estudantes da área de PAD relacionado com o paralelismo em *clusters*, como é o caso do Centro.

4.2.3 – Objetivos de WVM

Deve-se salientar aqui uma observação importante: antes de ser propriamente uma ferramenta, WVM busca apresentar uma arquitetura (embora simples) baseada no modelo C/S, que possa responder satisfatoriamente a alguns pontos relacionados com a integração de tecnologias e a amigabilidade[59] em relação ao usuário final. Sucintamente, a proposta tem como objetivos principais os seguintes pontos:

- A. Integração

  Aproveitar a ampla divulgação e facilidades de utilização da Internet, do `Java` e da WWW para auxiliar na programação distribuída. Isto é enfocado sob dois aspectos:

  - Primeiramente, a capacidade de utilizar a rede para iniciar processos em *clusters* de PAD, de acordo com o comando dos usuários;
  - Em segundo lugar, a possibilidade de permitir que os clientes possam participar do processamento via *Web*. Isso pode ser conveniente, por exemplo, enquanto se espera os resultados dos programas em execução no ambiente paralelo.

- B. Facilidade

  Projetar e implementar um ambiente para a Internet e a WWW que permita a execução e o gerenciamento simples de aplicações distribuídas escritas em `PVM`, `MPI` ou `Java` executadas em *clusters* de computadores de tal forma que:

  - Usuários de diferentes domínios possam acessar computadores de um outro domínio;
  - Seja permitido aos usuários preparar o ambiente de execução e monitorar a execução das aplicações remotamente;
  - Sejam integradas, num mesmo ambiente de trabalho facilidades gráficas que dispensem a necessidade de comandos para conexão remota (`telnet`), transferência de arquivos (`FTP`), visualização básica, compilação e edição de arquivos, bem como a configuração das máquinas;

[59] O termo mais adequado para a tradução da expressão inglesa "*user friendly*".

- A ferramenta possa tornar-se independente do sistema local, onde é executada (portabilidade)[60].

Os resultados do projeto podem permitir, de um lado, facilitar as etapas de programação e, de outro, fornecer subsídios para avaliações comparativas entre as diferentes alternativas para o processamento, como o uso de bibliotecas de passagem de mensagens (`PVM` ou `MPI`) ou mesmo a utilização de `Java`.

4.2.4 – Considerações sobre o Desempenho

Uma questão importante que devemos comentar diz respeito às características funcionais da ferramenta. Dois pontos devem ser considerados:

- **WVM como ferramenta de ajuda à programação paralela-distribuída:** Nesta situação, a ferramenta fornece uma interface integrada e única, o que permite facilitar o processo de desenvolvimento de aplicações. Porém, enquanto interface, ela não interfere no processamento paralelo-distribuído propriamente dito, de forma que o desempenho final não é afetado, permitindo que o PAD prossiga sua execução, a menos que seja cancelado pelo usuário.
- **WVM como ferramenta para programação via Internet:** Por ser escrita em `Java`, WVM se beneficia do conceito de um ambiente uniforme para desenvolvimento, que pode ser carregado a partir de qualquer computador conectado à Internet; isto visa representar um ganho de produtividade e economia no desenvolvimento das aplicações para aglomerações de computadores, isto é, *clusters* (vide o conceito HPCC, mencionado ainda no item 1.1.1.B).

A arquitetura WVM deseja, portanto, beneficiar usuários e programadores, a fim de permitir um desenvolvimento mais rápido de programas. Portanto, uma primeira melhoria de desempenho ocorre em nível de engenharia de *software*. Um último fator a considerar, e que não pode passar desapercebido, é o de que a carga da aplicação ou *applet* pelas JVMs pode ser consideravelmente lenta, dependendo da velocidade de *download* e do ambiente de execução Java[61] onde os *bytecodes* serão interpretados.

Vale mencionar que, de acordo com as últimas tendências do mercado, WVM é um exemplo da chamada **tecnologia móvel**, baseada no conceito de **escritório *online***, ou seja, no fato de que a *Web* apresenta-se como um imenso repositório de aplicações (normalmente

---

[60] A portabilidade aqui mencionada trata-se da independência com relação ao sistema local (isto é, onde o cliente é executado). No entanto, a configuração do ambiente é dependente do sistema remoto (*cluster* de processamento).

[61] O ambiente para execução de programas em `Java` é mais comumente identificado pela sigla JRE (*Java Runtime Environment*), que se encontra documentado em `http://java.sun.com/products/jdk/1.1/jre/`, sendo responsável pela interação entre a JVM e o SO.

gratuitas), disponíveis para usuários como alternativas simples para uma grande variedade de programas[62].

4.2.5 – Limitações

Finalmente, não podemos deixar de mencionar as limitações que decorrem do modelo aqui apresentado. Em primeiro lugar, é importante deixar claro que a ferramenta WVM apresenta um escopo bem característico para sua utilização:

- **Tipo de Problema:** A utilização das ferramentas atuais se restringe a problemas em que a utilização de MPLs para PAD possa ser uma alternativa eficiente. A possibilidade de uso de metacomputação pode também vir a ser viável em processos apresentem pouca necessidade de comunicação e baixa complexidade.
- **Localidade:** A resolução de problemas em PAD restringe-se a certos ambientes bem definidos (chamados de domínios de processamento), onde ficam os *hosts* e *clusters* disponíveis para a execução remota dos clientes.
- **Registro dos usuários:** Por motivos de segurança no sistema, os usuários que pretendem utilizar a ferramenta devem estar previamente cadastrados, de forma que as permissões para acesso e conexão no servidor possam ser bem sucedidas.
- **Continuidade de Processamento:** Embora seja apenas uma interface, a ferramenta necessita estar em permanente conexão com o servidor, durante todo o processamento[63], a fim de que os resultados possam ser enviados adequadamente para os usuários.
- **Ambiente de Programação:** Embora WVM, em si mesma, seja independente de plataforma, a atividade de programação exige da parte do usuário, um prévio conhecimento do ambiente sobre o qual sua aplicação deve executar e as ferramentas de MPLs a serem utilizadas no processamento.
- **Restrições da *applet*:** Quando executada dentro de uma *applet*, WVM sujeita-se às restrições impostas pela JVM para a sua execução (vide seção 7.1.1.B). Assim, certas facilidades como salvamentos locais, somente serão possíveis caso procedimentos de assinatura sejam previamente feitos ou a ferramenta seja executada como aplicação.

[62] Maiores informações sobre esta tendência, pode ser encontrada na reportagem "Só é preciso um browser" da revista Info Exame, edição de dezembro de 2000.

[63] Nas etapas que precedem o processamento – como construção dos códigos – não existe a necessidade de interação contínua com o servidor, conforme será mencionado.

- **Limites para conexões simultâneas:** A quantidade de clientes simultâneos é restrita à limitação que o servidor possui de atender a todas as *threads* em um dado momento.
- **Impossibilidade de interação:** Por serem processos independentes, disparados no sistema de PAD, os programas a serem executados no *cluster* não devem requerer interação com o usuário, estando já configurados para execução direta e ininterrupta[64].
- **Simplicidade de monitoramento:** Os dados a serem monitorados pela execução dos processos devem apresentar saídas simples, a fim de permitir sua adequada exibição para o usuário. Isso, por um lado alivia o tráfego de mensagens enviadas do cliente para o servidor, mas, por outro, limita a visualização de maiores informações relacionadas com o processamento.
- **Não uniformidade da certificação:** Em alguns sistemas, a certificação dos usuários é feita de forma particular. O sistema mais utilizado (`NIS`[65]), normalmente é acrescido de melhorias (como o `NIS+` ou ainda o `SSH`[66]), a fim de fortalecer o nível de segurança. Em particular, no caso do ambiente SP/2 da `IBM`, existe um nível superior de proteção que se utiliza do sistema `Kerberos`[67]. Este nível crescente de complexidade inviabiliza a utilização de um sistema uniforme de certificação por parte da ferramenta. Em outras palavras, a autenticação da ferramenta se utiliza de critérios bem menos robustos para a validação dos usuários (vide item 5.2.1.A.i).

Resumo

Tendo contextualizado o problema, apresentamos neste capítulo, as características, limitações e objetivos da ferramenta WVM. No próximo capítulo, apresentaremos sua arquitetura que refletirá fortemente todos esses pontos. É

---

[64] A impossibilidade de iteração é comum a várias ferramentas para manuseio de *tracefiles*, como o `XPVM`. Além disso, as aplicações necessitam possuir granulosidade alta, a fim de que se possa ter uma visibilidade adequada da execução dos processos.

[65] A utilização do *Network Information Service* – `NIS` – é bastante generalizada, seja pela sua simplicidade, seja pela difusão de sistemas como o `Unix`. No entanto, as vulnerabilidades demonstradas – em especial no tocante à segurança – forçaram o aparecimento de novas complementações para este padrão.

[66] O *Security Shell* – `SSH` – é o interpretador de comandos normalmente utilizado em sistemas em rede, para garantir que toda a informação seja transmitida encriptada.

[67] O sistema `Kerberos`, desenvolvido no MIT, baseia-se no conceito de chaves privadas e públicas (periodicamente renovadas), a fim de possibilitar a execução de certos comandos (normalmente feitos pelo superusuário do sistema).

importante conhecê-la corretamente para que a utilização desta proposta possa ser uma alternativa prática e eficaz para a utilização de PAD.

Capítulo 5

Arquitetura de WVM

Introdução

Neste capítulo, vamos descrever a arquitetura e o funcionamento da Web Virtual Machine, identificando seus componentes e apresentando a estrutura funcional da ferramenta. Tudo isso leva em conta as características apresentadas no capítulo anterior e determinará as tecnologias utilizadas para o desenvolvimento da ferramenta (Capítulo 6).

5.1. Modelo Funcional

5.1.1 – Componentes da Arquitetura

Web Virtual Machine, assim como a grande maioria das alternativas propostas para o processamento feito via Internet, segue o paradigma do modelo cliente-servidor (seção 1.2.3). Isto implica que, de um lado, temos uma aplicação permanentemente ativa, atendendo requisições (o **servidor**) e, de outro, processos eventualmente concorrentes que, assincronamente, solicitam serviços (**clientes**). No entanto, além destes dois **componentes**, a arquitetura define um espaço - ou ambiente - para processamento, em cima do qual as aplicações definidas pelos clientes devem ser efetivamente processadas, sob o controle do servidor.

Assim, podemos definir os seguintes elementos que compõem a WVM:

- **Cliente (C):** Constitui uma *applet* ou aplicação *standalone* Java, carregada a partir do servidor pelos usuários através da *Web*. Cada cliente comunica-se com o servidor através de um canal de execução (*thread*) próprio.
- **Servidor (S):** Reside numa máquina com uma JVM ativa conectada à rede onde o PAD deve ser realizado. Ela deve possuir também um servidor *Web*, a fim de que as páginas HTML com código *applet* possam ser enviadas aos clientes. É responsável

pela captura das requisições dos clientes e pelo envio das saídas de erros (`stderr`) e de comandos (`stdout`) para cada um dos clientes ativos.

- ***Site* de Aplicações (SA):** Correspondem ao local onde residem as aplicações desenvolvidas pelos usuários, incluindo códigos fontes, arquivos de configuração de máquinas, arquivos textos, etc.
- **Domínios de Processamento (DP):** Onde reside o *cluster* de processamento para as aplicações desenvolvidas em MPLs. Será neste local que as aplicações serão efetivamente processadas.

Para efeito de simplificação, integramos o *site* de aplicações e o domínio de processamento numa mesma entidade na rede, isto é, sob um mesmo domínio, muito embora pudessem ser distintos. Neste caso, o servidor deve possuir comunicação direta com tais ambientes ou mesmo ser uma das máquinas envolvidas no *cluster*.

5.1.2 – Descrição Funcional

♦ A. Diagrama de Funcionamento

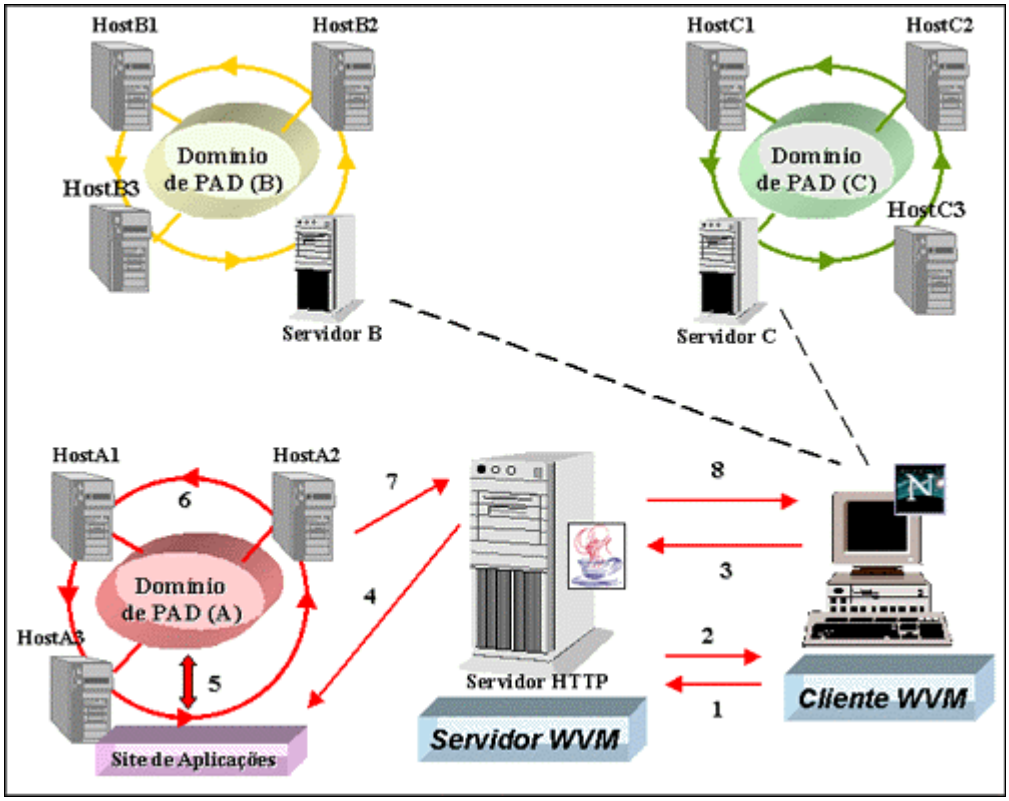


Figura 5.1: Arquitetura Funcional de WVM no PAD

A partir dos componentes anteriormente mencionados, podemos entender como se dá o funcionamento da ferramenta, através dos passos numerados no diagrama anterior. Isto envolve várias etapas - validação, preparação e processamento - cada uma com características

próprias, conforme explicado a seguir (os números no texto indicam as etapas ilustradas na figura 5.1).

- Etapa I: Validação / Configuração

O usuário se identifica perante o sistema e seu ambiente é configurado. O objetivo é fornecer tanto um nível elementar de segurança quanto uma adequação das opções disponíveis para o usuário. Neste sentido, podemos compreender os passos que descrevem como isto é feito:

- **(1).** Primeiramente, faz-se a conexão do usuário com o servidor HTTP pertencente a um determinado domínio de processamento. Isto pode ser feito através de um *browser* com capacidades para a execução de *applets* ou pela execução direta da aplicação *standalone* WVM.

Observação: No caso de execução do cliente em nível de aplicação Java, a utilização de um programa navegador seria substituída por um ambiente JRE adequadamente configurado, sem a necessidade de conexão via HTTP com o servidor *Web*. Neste caso, é feita uma conexão direta via *socket*, passando-se à etapa seguinte sem a necessidade de se fazer *download* de páginas HTML com *applets*.

- **(2).** O servidor HTTP responde às requisições dos *browsers*, enviando uma página com uma *applet* - o cliente WVM - com o qual o usuário interagirá. O primeiro procedimento é o de validação da conta do usuário (ou "*login*"). Isto serve por um lado, tanto para assegurar a privacidade dos clientes, quanto para lhes permitir uma correta configuração de seus ambientes (que serão carregados após a validação dos dados).

Observação: A validação dos usuários é feita com a comparação das senhas fornecidas durante o processo de *login* e a base de dados existente no servidor. O administrador do sistema deve, portanto, ter cadastrado previamente todos os usuários que farão conexão com o servidor[68].

- Etapa II: Preparação / Editoração

Boa parte do tempo, os usuários ficam envolvidos não com o processamento propriamente dito, mas com a **preparação** para o mesmo, o que inclui a definição dos

[68] O cadastramento dos usuários, como será visto, é uma das atividades necessárias ao componente servidor da ferramenta, conforme será visto na próxima seção.

ambientes, criação ou modificação de códigos fontes e organização de dados. Esta fase, portanto, envolve os seguintes passos:

- **(3).** A requisição de arquivos para leitura e alteração, que é feito pelo cliente e enviado ao servidor[69]. Por medida de segurança, o acesso aos arquivos é restrito apenas ao usuário que o possui.

  Observação: Essa restrição também inibe o acesso a outras pastas (diretórios) do sistema. A organização adotada é a de que os arquivo dos usuários devem estar localizados de acordo com o ambiente de processamento em uma organização bem definida de pastas, tais como Java, PVM ou MPI.

- **(4).** O servidor, então, se encarregará de buscar as aplicações no *site*, conforme a definição feita pelo usuário e repassá-las ao cliente. As etapas de retransmissão de dados aos clientes são ilustrados nos passos **(7)** e **(8)** e ilustram na figura, uma recepção de dados[70].

  Observação: O processo pode continuar indefinidamente, através de contínuas leituras e gravações de arquivos necessários ao processamento. Isto ocorre porque múltiplas requisições (para os vários arquivos envolvidos no processamento) são feitas, de forma que os passos 3 e 4 podem ser intercalados numerosas vezes.

➢ Etapa III: Processamento / Execução

Finalmente, considerando que todo o ambiente já está adequadamente configurado, o usuário pode passar à fase de processamento, a qual é feita nas seguintes etapas:

- **(5).** O servidor, tendo recebido do cliente os parâmetros de execução (vide passo 3), busca a aplicação e a executa no domínio de PAD, de acordo com as definições dadas pelo usuário (como variáveis de ambiente e parâmetros de execução);

- **(6).** O processamento é, então, iniciado e sua execução é feita, a partir daquele momento, **independente da ação do cliente**, definida dentro do ambiente previamente configurado pelo usuário (número de *hosts*, tipo de MPL, etc.), seja explícita, seja implicitamente.

---

[69] Vale observar que nada impede que a escrita dos códigos seja toda feita *in loco*, isto é, sem a leitura de arquivos previamente existentes no servidor. No entanto, nestes casos, a transmissão dos códigos locais para o ambiente de processamento deve ser feita a fim de possibilitar a execução dos arquivos criados localmente.

[70] A figura 5.1 foi propositadamente simplificada, a fim de evitar referências múltiplas de etapas. Um melhor esclarecimento sobre o funcionamento será mostrado na seção 5.3, quando será mostrada a metalinguagem.

- **(7).** Durante todo o processamento e, em especial, quando do seu término, os dados são repassados ao servidor, bem como eventuais mensagens do sistema[71].
- **(8).** Por fim, o cliente recebe as mensagens de sistema e saídas de dados em sua *console*.

Observação: A saída padrão no protótipo atual são dados **não formatados** em forma textual. No entanto, foi criada uma facilidade gráfica, a partir de um formato particular, para permitir a visualização de dados e sua "plotagem" no eixo cartesiano, muitas vezes úteis para certas aplicações (vide maiores informações no Capítulo 6).

♦ B. Considerações sobre o Modelo

Cabe agora salientar algumas considerações relevantes sobre o modelo, a fim de esclarecer pontos relativos ao seu funcionamento:

➢ (i). Domínios de Processamento (DP) × *Sites* de Aplicações (SA)

Apesar da existência de vários domínios de PAD possíveis (considere-se, por exemplo, todo a estrutura do sistema SINAPAD – vide Anexo C.3), capazes de serem configurados para execução de aplicações, um cliente WVM interage com apenas um servidor específico por vez. Para tanto, é necessário que o servidor WVM lá existente, esteja permanentemente ativo, pronto para a atender às requisições dos clientes.

Uma outra observação é a de que a simplificação do modelo, como visto, uniu as entidades SA e DP em um mesmo ambiente – um domínio de aplicações e processamento (DAP). No entanto, dependendo das características a serem adotadas, pode-se optar pela existência de dois servidores: um pertencente ao *cluster* de processamento e onde ficam as páginas HTML para *download* dos usuários; e outro encarregado de buscar aplicações e repassá-las ao primeiro servidor. Duas condições são necessárias para um funcionamento adequado: primeiramente, que haja comunicação eficiente entre os dois servidores, de forma a permitir que os arquivos possam ser enviados ao outro servidor. A segunda condição é a de que o servidor WVM tenha comunicação direta com o *cluster* ou ambiente de processamento, onde as aplicações serão executadas. Vale a pena notar que, em caso de múltiplos servidores, existe ainda um risco de degradação do desempenho, já que uma nova etapa de comunicação é necessária.

---

[71] Normalmente, as mensagens do sistema são classificadas de acordo com sua função básica: seja a de "saída padrão" (chamada `stdout`) ou "erro padrão" (chamada de `stderr`).

Observação: O protótipo utiliza o DAP, como uma abordagem simplificada de implementação do modelo. A separação dos servidores não foi implementada para evitar qualquer aumento de complexidade no funcionamento.

- (ii). *Applet* × Aplicação

As características da ferramenta permitem a utilização dual da ferramenta – tanto como *applet*, quanto como aplicação *standalone*[72] – no componente cliente. Assim, caso a utilização de um navegador habilitado para Java não seja possível, pode-se fazer o *download* dos códigos (*bytecodes*) e executá-la como aplicação através de algum JRE previamente existente (hoje já presentes na maioria dos sistemas operacionais comerciais)[73].

Observação: Algumas características apresentarão funcionalidades diferentes ou mesmo ficarão indisponíveis entre uma e outra forma de utilização. Um exemplo disso refere-se à opção de Ajuda (Help), tal como implementada no protótipo: visto que o modelo se utiliza das funcionalidades dos *browsers* para abrir páginas HTML relacionadas com um determinado contexto (PVM, MPI ou Java), o seu funcionamento numa aplicação *standalone* fica impraticável.

- (iii). Anonimato × Certificação

O objetivo dos procedimentos de verificação e validação do usuário é fornecer um nível mínimo de segurança aos usuários da ferramenta. Note que, pelas próprias características da programação Internet (item 3.1.3), o anonimato está presente. Assim, uma das alternativas para contornarmos este problema é a inclusão de rotinas que forcem aos usuários o fornecimento de uma senha (*password*) para conexão ao sistema. No entanto, como em qualquer sistema convencional, a vulnerabilidade desta alternativa é perceptível, já que qualquer outra pessoa, de posse dessas informações, poderia entrar no sistema e alterar as configurações de ambiente daquele usuário.

- (iv). WVM como *Middleware*

---

[72] Do ponto de vista da **implementação**, toda *applet* extende a classe `Applet` padrão, apresentando um método de inicialização (`init()`) e funcionando sempre sob um `Panel`; já as aplicações *standalone* apresentam um método estático de execução independente da classe principal (como a função `main()` derivada do `C`), permitindo, por exemplo, a carga de `Frames`. A conjunção destas duas alternativas se vale, dentre outros detalhes, da conciliação das duas alternativas: a inclusão do método inicializador da *applet* em `main()` e a invocação de uma classe derivada – responsável pela construção da aplicação propriamente dita – na classe principal.

[73] A utilização de JREs não plenamente compatíveis com a especificação da Sun (dada pela Java Language Specification - JLS), tem causado tormentos a vários desenvolvedores e sido a causa de algumas das mais recentes disputas judiciais entre a Sun Microsystems e grandes companhias de software (como a Microsoft).

Apesar de nos referirmos à arquitetura cliente-servidor como o modelo funcional da ferramenta, do ponto de vista do usuário, WVM opera como uma espécie de interface e *middleware*, já que o processamento final ocorre efetivamente num terceiro componente (uma máquina no *cluster*), não visível para a arquitetura C/S.

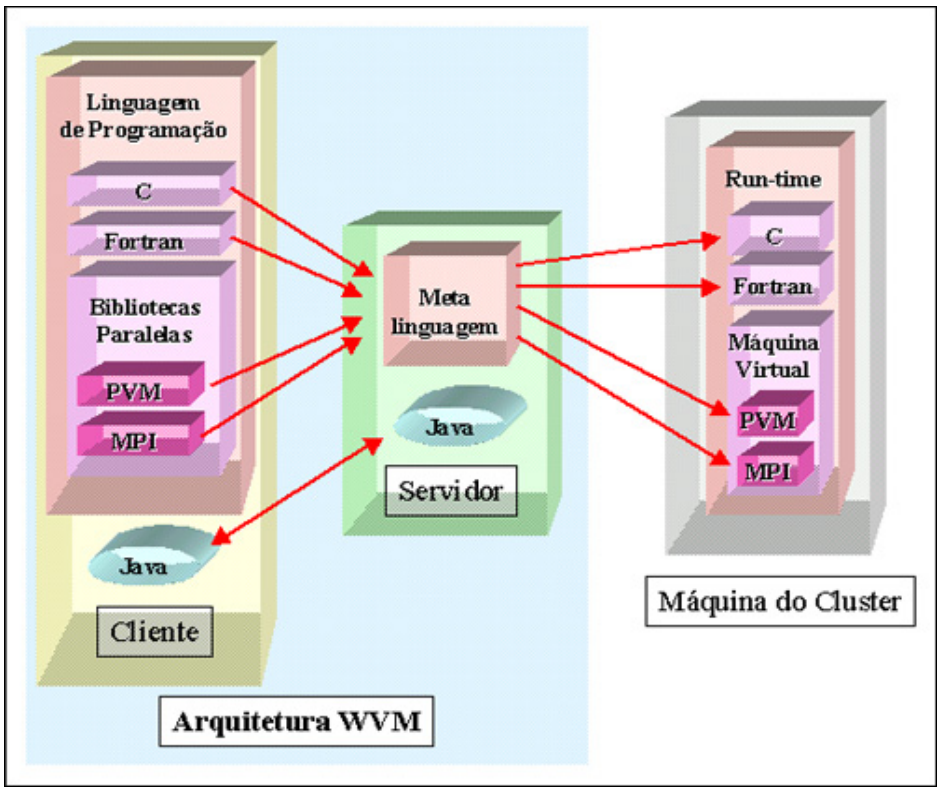


Figura 5.2: Funcionamento de WVM do ponto de vista do usuário.

A figura 5.2 ilustra esquematicamente o funcionamento da arquitetura, através do mapeamento dos comandos entre os componentes, ou seja, como as diretivas feitas no componente cliente são interpretadas pelo servidor, a fim de serem processados no *cluster*. Cabe tecer algumas observações sobre o seu funcionamento:

1. Na programação para PAD, o usuário desenvolve sua aplicação em uma dada linguagem (`C` ou `Fortran`) e opta pela utilização de uma das MPLs disponíveis (`PVM` ou `MPI`). No entanto, o cliente opera apenas com as etapas que antecedem à execução dos programas: a preparação dos ambientes e digitação dos códigos. Note que a utilização de `Java` é possível, sendo que, neste caso, o modelo atual interagirá apenas com o servidor, já que as máquinas do *cluster* se destinam à execução de programas paralelos[74] (maiores informações sobre as futuras

[74] Existe na realidade uma forte razão pela qual `Java` não é suportada no *cluster*: o sistema operacional existente ou não possui suporte para JVMs nativas (como no caso do `AIX 3.2.5` existente no ambiente SP/2 do `CENAPAD-NE`), ou tal suporte ainda é experimental (como no caso dos *clusters* `Linux`, cuja JVM ainda provém de versões beta).

modificações serão esclarecidas na seção referente à **metacomputação** no sistema, no último capítulo).

2. Quaisquer comandos são enviados ao servidor. A existência de uma metalinguagem (mencioanda na seção 5.3.1) para a determinação das ações a serem executadas nas máquinas do *cluster* é algo completamente transparente para os usuários (ainda que necessário para o projeto do servidor);

3. Em cada máquina do *cluster* selecionada para o processamento (o que pode ser feito pelo usuário), são realizadas as execuções dos comandos transmitidos pelo servidor, como compilação, verificação de erros, e transferência de dados. Note que, somente neste momento, os códigos previamente digitados pelos clientes nas suas respectivas configurações (C/Fortran e PVM/MPI) são de fato executados. Vale tecer aqui algumas considerações sobre implementação (detalhadas no Capítulo 6): A programação em Java poupa o programador das tradicionais preocupações referentes à migração para os vários ambientes onde determinado sistema funcionará. No entanto, como aqui utilizamos uma metalinguagem de comandos, algumas parametrizações estarão fortemente sujeitas às características e restrições dos sistema remotos. Por exemplo, os comandos para a criação de um novo ambiente (digamos "mpi") ou para a remoção de um arquivo (por exemplo, "testempi.c"), serão distintos em ambientes Linux ou Windows. Desta forma, a utilização da ferramenta como *middleware* serve para a conversão adequada de comandos deste tipo, de acordo com o ambiente de execução[75].

No item 5.2.2 mencionaremos um outro aspecto que está relacionado basicamente com as funcionalidades do servidor WVM, especialmente no que diz respeito à gerência de *scripts* de teste e verificação no *cluster*, bem como à administração do ambiente.

- (v). Espaço para Metacomputação

Um último fator a ser considerado trata das questões referentes à utilização de metacomputação no modelo e será motivo de discussão para o capítulo final desta dissertação. É importante notar que somente a utilização de Java possibilitará o acréscimo desta funcionalidade. Note que isto explica uma diferença entre os aspectos de programação Java e C/Fortran (ilustrado na figura 5.2). Muito embora não seja objetivo inicial de WVM, sua organização abre espaço para esta forma de processamento distribuído. Disto derivam algumas modificações na arquitetura básica

[75] Para resolver como o sistema executará independentemente cada comando, um teste prévio de identificação é feito, passando-se, então, à utilização de *scripts* de *shell* específicos para a execução de cada tarefa (como a criação de diretórios e a remoção de arquivos).

proposta a seguir (seção 5.2), motivo pelo qual somente depois detalharemos estes aspectos.

5.2. Arquitetura dos Componentes

Para que se possa entender melhor o funcionamento de WVM, vamos isoladamente verificar os **componentes** cliente e servidor, analisando como estes se relacionam a partir do modelo funcional apresentado na seção anterior. Antes, porém, de detalharmos a arquitetura proposta, devemos mencionar alguns aspectos comuns, tanto ao cliente, quanto ao servidor. Em primeiro lugar, cada componente subdivide-se em ambientes e estes, por sua vez, em módulos.

- Os **ambientes** indicam partes independentes do programa (ainda que inter-relacionadas), com objetivos bem definidos dentro da arquitetura. Se considerarmos as questões de implementação, perceberemos que os ambientes corresponderão sempre a **`classes`** de objetos em Java.
- Já os **módulos**, irão representar atividades básicas que se relacionam entre si para a execução de uma dada operação num dado ambiente. Em termos da linguagem, são implementados sobre as formas de **`métodos`** de classes.

A figura 5.3 esquematiza os elementos desses diagramas: um dado ambiente é dividido em módulos e estes são apresentados em termos de funções, cujas atividades são sucintamente indicadas na parte inferior. Vale a pena notar que módulos e ambientes do cliente podem interagir – e normalmente o fazem – com módulos e ambientes do servidor e vice-versa. Desta forma, embora estudemos em separado cada um dos componentes (seções 5.2.1 e 5.2.2), percebe-se que existem tanto elementos comuns quanto complementares no funcionamento da arquitetura. Nas seções que se seguem, serão apresentados os diagramas funcionais que ilustrarão a arquitetura dos componentes.

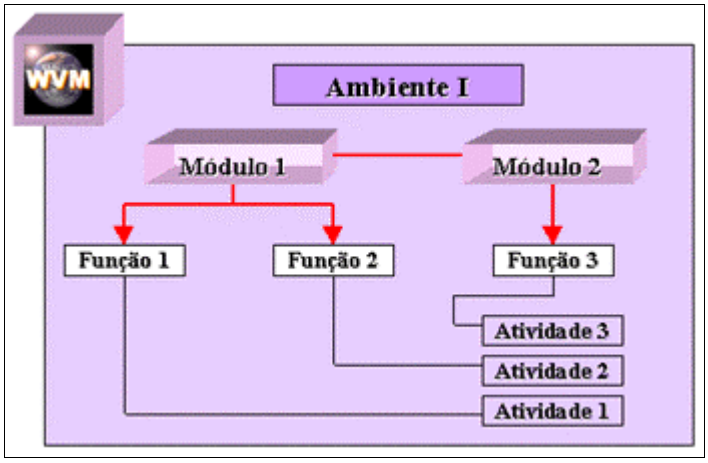


Figura 5.3: Diagrama modelo para apresentação dos componentes.

5.2.1 – O Cliente WVM

O cliente WVM é a *interface* responsável pela interação com o usuário, fornecendo-lhe uma aplicação GUI de fácil compreensão para seu trabalho. Além disso, é responsável por capturar os comandos do usuário e enviá-los para o servidor.

Funcionalmente, o cliente WVM apresenta ambientes que se relacionam e podem ser ativados a cada momento pelo usuário. A organização deste componente é mostrada na figura 5.4.

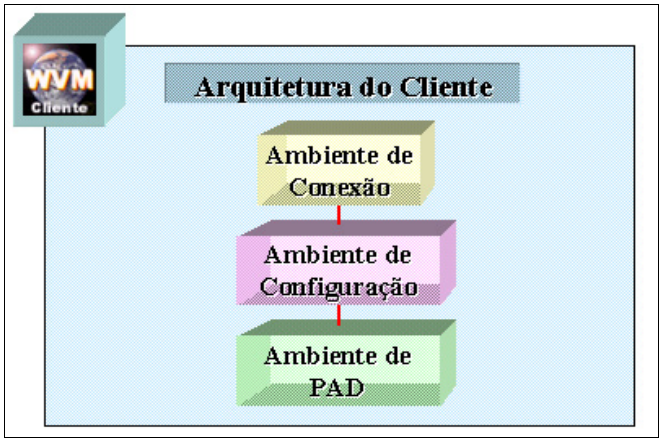


Figura 5.4: Componentes do Cliente WVM

A comunicação C/S é feita via *socket* Internet (já fornecido pela API Java padrão), o que traz algumas vantagens como facilidade e compatibilidade. Vale relembrar a capacidade de funcionamento dual do cliente, tanto como aplicação *standalone* sobre um JRE existente (normalmente já fornecido nos sistemas operacionais), quanto através de uma *applet*, em um *Web browser* habilitado com JVM interna, obedecendo-se às restrições de segurança impostas (vide seção 7.1.1.B). A seguir, identificaremos melhor cada um destes ambientes, suas funcionalidades e sua organização em termos de módulos. Alguns destes conceitos serão análogos ao componente servidor da arquitetura, conforme veremos adiante.

- A. Ambiente de Conexão

Como o próprio nome diz, é o ambiente responsável pela validação do usuário e configuração inicial da interface. Ele é comum aos dois componentes da arquitetura C/S e se subdivide em dois módulos, conforme ilustrado na figura 5.5:

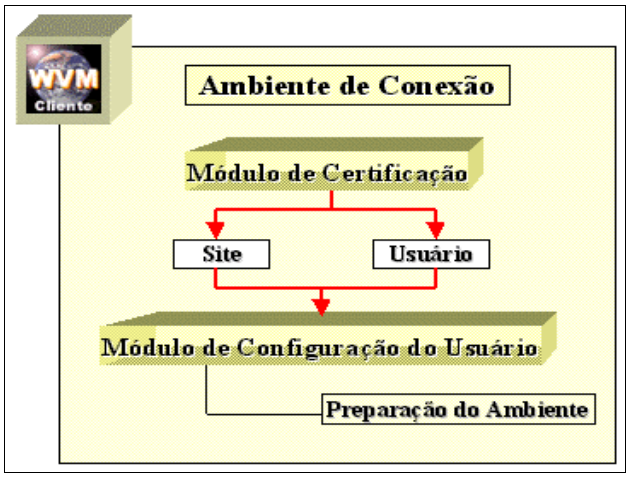


Figura 5.5: Organização do Ambiente de Conexão no Cliente.

- (i). Módulo de Certificação

Se constitui no primeiro módulo a ser executado, logo quando da chamada do cliente. Caso seja invocado sobre uma *applet*, é chamado imediatamente após o *download* das classes via rede[76]. Este módulo é ativado sempre que se pretende mudar a identificação do usuário, ou acessar um outro domínio de PAD.

Funcionalmente, o módulo de certificação envia ao servidor informações que correspondem ao ***login*** (nome da conta no sistema de *cluster*) e ao ***password*** (senha) do usuário, para sua validação. As identificações são necessárias e não podem ser burladas para a penetração no sistema, já que somente com a certificação correta (*login* e *password*) é que o sistema poderá ser utilizado pelo usuário, lhe permitindo acesso aos arquivos remotos. Uma observação importante é que cada *site* de processamento possui sua própria lista de usuários e de senhas, de forma que é necessário interagir com os administradores locais a fim de se criar uma adequada configuração do sistema.[77] Cabe lembrar aqui o que foi mencionado na seção 4.2.5: a autenticação é feita de forma diferenciada, em relação ao mecanismo de validação dos usuários no Unix, como aqueles baseados em sistemas como o NIS. A implementação

[76] Na realidade, existe toda uma hierarquia de passos para a execução de um programa Java, que vão desde a carga da JVM à instanciação da classe em si, passando pelas etapas de vinculação de dados e inicialização (especialmente quando se trata de *applets*). Na discussão aqui apresentada, nos abstemos destes detalhes sobre o JRE.

[77] Por não existirem restrições relativas ao cadastramento dos usuários e de suas senhas nos servidores WVM, é possível que um usuário possua diferentes *logins* ou *passwords* nos vários DPs onde ele esteja cadastrado. A correção deste problema implicaria na necessidade de uma centralização das bases de dados, o que se torna proibitivo, considerando-se as dimensões da rede.

se utiliza de um protocolo próprio (descrito na sua metalinguagem) e as senhas devem ter sido previamente cadastradas pelos administradores (conforme será visto no tocante ao componente servidor).

- (ii). Módulo de Configuração do Usuário

Nesta etapa, que ocorre após um *login* bem sucedido do usuário, as informações relativas ao seu ambiente, bem como arquivos e configurações da sua conta, são remotamente transferidas do servidor ao cliente. A partir deste momento, o usuário pode desenvolver seu trabalho, seja editando arquivos, executando aplicações ou alterando o próprio ambiente[78]. Para simplicidade do modelo, as configurações estão sujeitas a algumas restrições como:

- Possibilidade para criação apenas dos três ambientes de processamento: PVM, MPI e Java. Quaisquer arquivos devem estar localizados dentro destas pastas (ou diretórios);
- Os arquivos e pastas estão sujeitos às limitações do sistema: tanto os seus nomes quanto as suas formatações, dependerão do SO remoto, onde residem;
- Dualidades entre os sistemas locais e remotos podem ser conflitivas: devido à diversidade nas configurações dos SOs, arquivos podem ser tratados diferentemente pela configuração local e remota[79].

♦ B. Ambiente de Configuração da VM

Além de fornecer informações sobre as máquinas componentes do *cluster* e permitir a invocação de URLs para acesso às páginas de documentação, este ambiente é responsável por permitir configurar as máquinas que compõem a *virtual machine* (VM). Os módulos são mostrados na figura 5.6.

[78] No próximo capítulo será visto um artifício utilizado para economizar acessos remotos ao servidor, em busca de identificações dos ambientes e listagem dos arquivos.

[79] Um exemplo disso, são os arquivos texto: em sistemas baseados no `Windows`, a mudança de linha é feita por dois caracteres especiais: *line feed* (`LF`) e *carriege return* (`CR`); já no `UNIX`, apenas `CR` é utilizado. Desta forma, a criação de tais arquivos num sistema, influi na exibição deles em outro.

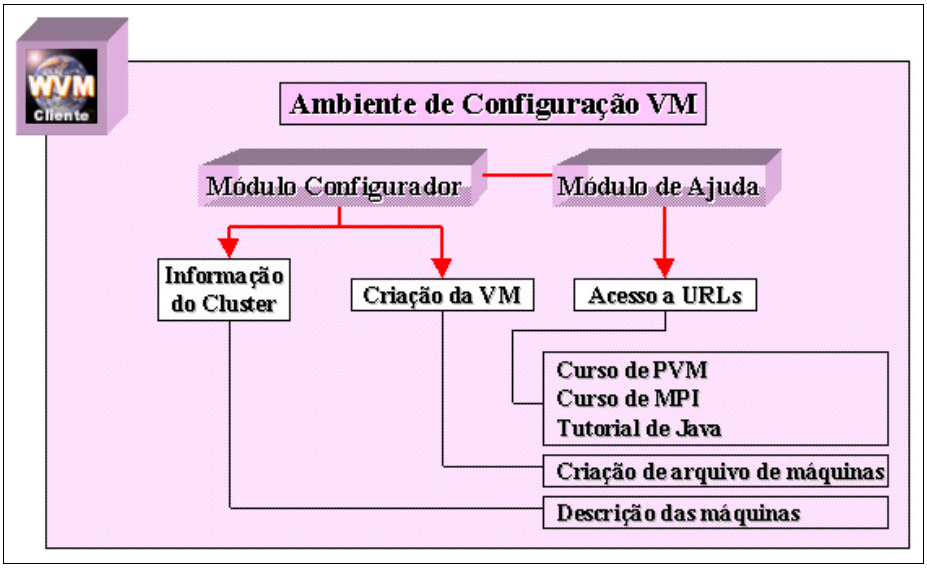


Figura 5.6: Organização do Ambiente de Configuração VM no Cliente.

- (i). Módulo Configurador

A utilização de um mecanismo dinâmico para detecção das máquinas previamente definidas para um dado *cluster* é a primeira tarefa executada pelo configurador. Neste módulo também é possível obter uma sucinta descrição de cada uma das máquinas disponíveis (previamente definidas pelo administrador), o que pode ser relevante para optar pela inserção de cada uma delas no processamento.

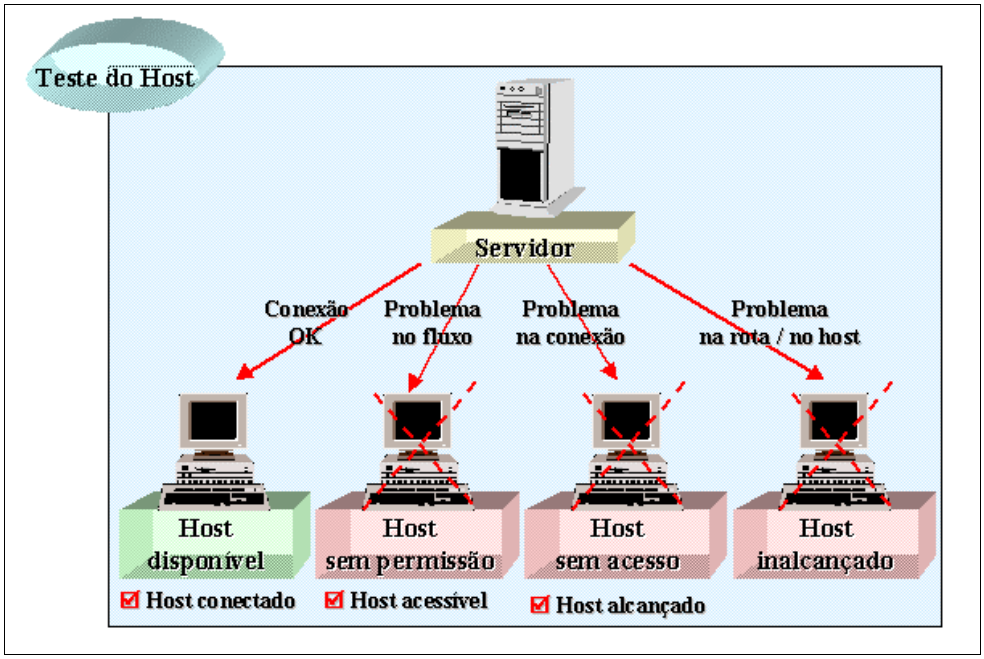


Figura 5.7: Possibilidades de Falhas na comunicação C/S.

O mecanismo adotado na checagem das máquinas consiste basicamente na avaliação da tentativa de abertura de conexão via rede: o sucesso determina a disponibilidade da mesma; sua falha, no entanto, pode indicar uma gama de possíveis

causas (no próximo capítulo, serão detalhados aspectos da implementação destes testes). O quadro da figura 5.7 indica algumas das possibilidades previstas para as falhas. Note que, uma vez testadas, tais máquinas ficam disponíveis para a seleção do usuário, a fim de criar a VM a partir das máquinas existentes no **Domínio de PAD**.

Vale notar que existem duas etapas para a formação da *virtual machine*: previamente, são detectados os EPs do *cluster* e então, cada um deles é incluído no chamado **arquivo de máquinas** (utilizado pelas bibliotecas MPL) para seu processamento. Note que um dado EP pode, contudo, estar presente no domínio e não participar, por alguma razão, da computação. Tais razões podem variar desde problemas de hardware (máquina danificada), a problemas na comunicação (demora na obtenção das respostas) ou mesmo pela simples escolha do usuário (que pode acabar por optar em não incluir uma determinada máquina que pode, por exemplo, estar sendo usada por vários outros usuários).

- (ii). Módulo de Ajuda

Para auxiliar a utilização tanto das MPLs quanto da linguagem Java, beneficiando-se da Internet, o modelo apresenta um módulo para propiciar uma ajuda aos usuários da ferramenta. Isso se beneficia de dois pontos:

- A estrutura das *applets* permite naturalmente a invocação de páginas HTML a partir do navegador;
- A existência de cursos e tutoriais *on-line* pela Internet (como os existentes no CENAPAD-NE). Desta forma, a partir de uma URL específica, informações sobre uma determinada biblioteca, como PVM, MPI ou Java podem ser obtidas, sem qualquer sobrecarga no desenvolvimento do protótipo. Isso torna a rede como um banco-de-dados em potencial a serviço dos clientes.

Desta forma, conciliando-se os pontos acima, pode-se permitir o acesso a material de estudo e treinamento para os usuários. Isso terá um reflexo na implementação: conforme mencionado, somente *applets* poderão executar adequadamente este módulo, devido às especificações da JVM[80].

♦ C. Ambiente de PAD

[80] Nada impede que o módulo de ajuda se utilize de outras alternativas, não baseadas em conteúdos HTML. Exemplos disso poderiam ser tanto a criação de objetos locais (como os *helpers* em formas de *menu* comumente encontrados nas implementações de LPs convencionais), quanto a invocação de comandos via *socket*, que permitam ao servidor fornecer os tópicos de ajuda requisitados pelos clientes.

Sendo o principal ambiente do componente cliente, é o responsável pela execução de tarefas relacionadas com o processamento paralelo executado no Domínio de PAD ao qual o cliente está conectado. Outras atividades preparatórias, como a elaboração dos códigos e os ajustes das opções de compilação também são tratados neste ambiente. Sua funcionalidade permite dividi-lo em três módulos, conforme mostrado na figura 5.8.

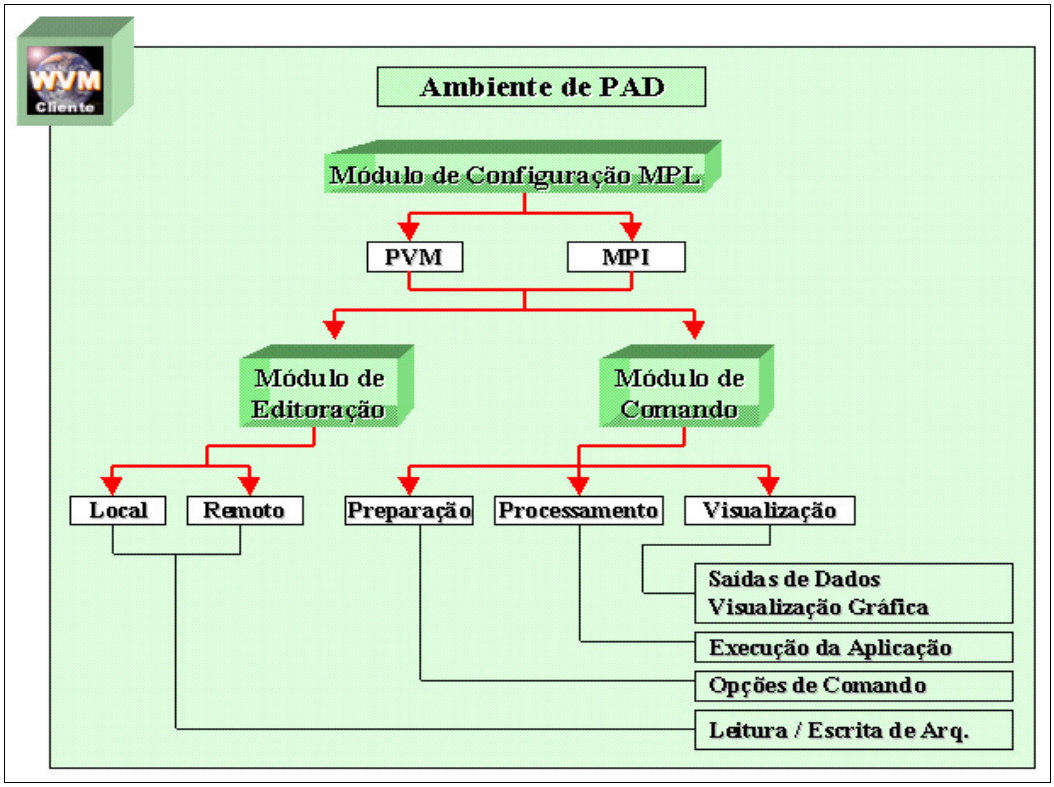


Figura 5.8: Organização do Ambiente de PAD no Cliente.

- (i). Módulo de Configuração MPL

A importância deste módulo deve-se ao fato de que os usuários definem, antes de mais nada, o tipo de biblioteca de passagem de mensagem com o qual deve operar: `PVM` ou `MPI`. Isto é importante porque os comandos de compilação, configuração e execução, são ajustados de acordo com a biblioteca utilizada. Os demais módulos dependem, pois, da definição feita neste fase. Alguns pontos importantes devem ser salientados:

- Dependendo da MPL selecionada, o modelo de programação (seja mestre-escravo, seja SPMD ou ainda árvore) deve ser criteriosamente escolhido, conforme mencionado na seção 2.2.3 e atendo-se às restrições das bibliotecas utilizadas[81];

[81] Note-se, por exemplo, que o `MPI` funciona com o modelo SPMD e qualquer outro tipo de organização deve ser simulado funcionalmente (isto é, tratado pela diferenciação dos processos); já no caso do `PVM`, o modelo real

- Os arquivos de máquinas devem ser previamente configurados para permitir o processamento no *cluster*. Quando a quantidade de processos é superior à quantidade de máquinas, a alocação das tarefas é, normalmente, feita em esquema rotativo (*round-robin*), indo do início ao fim do arquivo;
- Como visto, as sintaxes das bibliotecas, apesar de semelhantes, são distintas e suas funcionalidades nem sempre correspondentes. Isto pode ser visto, por exemplo, quando se analisa o modelo de criação de processos no PVM e no MPI. Cabe ao usuário definir adequadamente a MPL a ser utilizada (maiores informações, podem ser encontradas na seção 2.3.4).

➢ (ii). Módulo de Editoração

Este módulo, que sucede à execução do módulo anterior, é responsável pela editoração dos arquivos existentes no ***Site* de Aplicações**, ajustando-os adequadamente para futuro processamento. Vale a pena notar que boa parte do tempo é consumida neste processo. No caso de edição de arquivos fontes, por exemplo, o usuário define as chamadas da MPL utilizada, preparando os códigos para sua execução, seja em PVM, seja em MPI (vide seção 2.3.4.B e C). Algumas observações importantes relativas ao Módulo de Editoração podem ser feitas:

- Primeiramente, a editoração de arquivos remotos é possível desde que o sistema de segurança da *applet* esteja autenticado ou o cliente esteja executando como aplicação. A vantagem desta alternativa é que os arquivos podem ser salvos localmente, permitindo sua edição após a desconexão com a Internet;
- Em segundo lugar, o mecanismo de conexão foi implementado apenas nos momentos de transferência de dados, isto é, quando da recepção de dados (do servidor ao cliente) ou da transmissão (no sentido inverso, do cliente ao servidor). Durante a editoração, porém, não existe qualquer necessidade em se manter a conexão aberta, aliviando o tráfego na rede.

➢ (iii). Módulo de Comando

Aqui são realizados todos os comandos relativos à preparação do ambiente de processamento, sejam relacionados com a leitura ou escrita de arquivos, sejam invocações de procedimentos específicos para auxiliar na configuração da VM. Tais

---

de funcionamento baseia-se no paradigma Mestre-Escravo e os processos somente podem ser criados pelo programa mestre.

comandos são executados remotamente, a partir do cliente, no servidor, através da metalinguagem (mencionada na seção 5.3).

As atividades de processamento das aplicações, propriamente dito, são, portanto, efetuadas neste módulo, bem como as operações de compilação e configuração das máquinas virtuais. Durante as execuções dos comandos, os clientes ficam à espera dos resultados e mensagens recebidos pelo servidor. Dentre os principais comandos, destacam-se:

- Criação e remoção de ambientes (mencionados no Módulo de Configuração);
- Criação, salvamento, modificação e remoção de arquivos (vide Módulo de Editoração);
- Compilação e ajuste dos arquivos, seja feito diretamente, seja via arquivos *makefile*[82];
- Execução de arquivos binários e programas no *cluster*;
- Visualização das saídas.

No tocante à visualização dos resultados de processamento, algumas observações devem ser feitas: a saída padrão (`stdout`) e o redirecionamento para arquivos continuam sendo possíveis, já que, apesar da invocação remota, o processamento é feito a partir de um componente real do *cluster* (o servidor). No entanto, como muitas vezes a visualização gráfica é desejada ou mesmo necessária para a interpretação dos dados de saída, o Módulo de Comando apresenta uma **função de visualização** analisa os dados e se responsabiliza por fazer uma plotagem de pontos, de acordo com as macros definidas para a formatação da saída (feita pelos comandos da metalinguagem)[83].

Vale a pena lembrar que estes ambientes e módulos são intercomunicáveis e ajustáveis, de forma que um usuário pode a cada momento fazer uma invocação de operações de um determinado ambiente. Isto será mostrado mais detalhadamente na seção 5.2.3, quando se estudará a metalinguagem e no próximo capítulo, quando será descrita a implementação do protótipo.

5.2.2 – O servidor WVM

[82] A utilização do comando `make` nos sistemas `Unix` representou uma considerável economia, já que permitiu o processamento em lote de comandos de compilação (nos chamados arquivos `makefile`). Este padrão tornou-se comum ente os usuários, sejam do `C`, sejam do `Fortran`, e é ainda muito utilizado, especialmente em pacotes de software.

[83] No próximo capítulo, é mencionado como o trabalho de visualização dos dados é feito, bem como quais as aplicações e limitações desta forma alternativa.

O servidor WVM, complementa o cliente, sendo responsável pela transferência e conversão dos comandos de usuário (enviados pelo cliente via *socket*), em chamadas diretas ao sistema distribuído de PAD. Tais comandos incluem desde a simples transferência de arquivos, até a execução dos programas definidos pelos usuários nos *clusters* de processamento.

As atividades desenvolvidas pelo servidor possibilitaram organizar a sua arquitetura em dois ambientes distintos, como mostra a figura 5.9. Cada um desses componentes, suas funcionalidades e organização serão descritos a seguir, apresentando seus módulos (conforme feito com o cliente WVM).

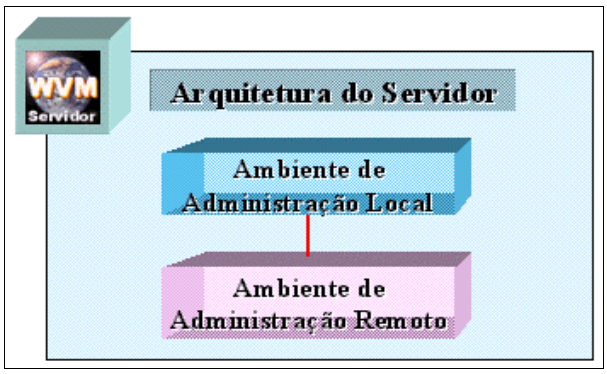


Figura 5.9: Componentes do Servidor WVM

Não participando da interface com o usuário, o servidor deve ser executado como **aplicação *standalone***, devido às restrições impostas pela JVM para as atividades de chamadas de funções remotas, controle e monitoramento de *threads* e aplicações. Caso o domínio de PAD e o *site* de aplicações sejam distintos (apresentando, portanto, servidores distintos), o servidor deverá prever a necessidade de fazer *download* dos programas requisitados pelo usuário naquele domínio, o que torna a implementação deste componente mais complexa.

♦ A. Ambiente de Administração Local

Este ambiente é responsável pelas atividades de **gerência** e **organização** do servidor WVM. Dentre suas funções, citam-se o controle das informações dos usuários e a configuração de seus ambientes (identificação do domínio de PAD). Esquematicamente, é composto dos três módulos ilustrados na figura 5.10.

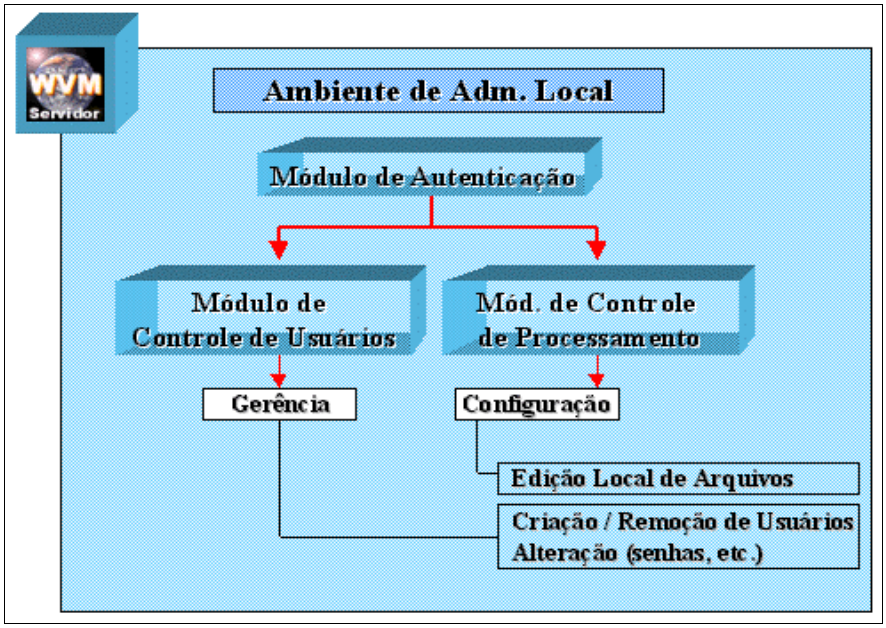


Figura 5.10: Organização do Ambiente de Adm. Local no Servidor

- (i). Módulo de Autenticação

Análogo ao existente no cliente, este módulo fornece um nível básico de segurança, permitindo que somente pessoas autorizadas (neste caso, administradores) possam ter acesso ao servidor. O módulo de autenticação cuida também da configuração e prepara para a utilização dos ambientes de processamento, permitindo também configurar as características das contas e a organização do sistema.

Vale ressaltar, porém que, diferentemente do que ocorria com o componente anterior, no servidor, a operação de *login* baseia-se apenas na conta local. Como nos sistemas `Unix`, a conta administrativa é comumente definida como **`root`**, mas nada impede de haver várias contas administrativas para o gerenciamento do ambiente.

- (ii). Módulo de Controle dos Usuários

Aqui, são feitas as operações de **gerência das contas**, incluindo criação, alteração e remoção das mesmas, bem como registros e modificações de senhas. Procedimentos auxiliares de teste – como por exemplo, os que auxiliam na identificação do sistema operacional remoto – são aqui invocados, a fim de poderem ser utilizados para executar tais tarefas.

Outros testes incluem a verificação de contas existentes antes da execução de determinados comandos (tais como os de alterações de senha) e a configuração do sistema (incluindo remoção de diretórios e arquivos quando da eliminação de contas)[84].

[84] Em sistemas com compartilhamento de recursos, a alteração de um ambiente em uso – como, por exemplo, a eliminação de um determinado diretório ou pasta – pode apresentar os seguintes efeitos colaterais: (i)

Vale notar, pelo que foi acima descrito, que o módulo de controle é invocado tanto explícita quanto implicitamente. No primeiro caso, para o cadastramento de informações próprias das contas de usuários. Já no segundo caso, para os procedimentos de identificação necessários às execuções de comandos do módulo de comando existente no cliente.

- (iii). Módulo de Controle de Processamento

Rotinas para a **configuração do sistema**, incluindo edição local de arquivos são feitas neste módulo. Por executar como aplicação *standalone*, o servidor possui liberdade para escrever no sistema e realizar alterações em arquivos do domínio de PAD[85]. Alguns arquivos, gerenciados pelo servidor e úteis nas atividades do sistema merecem destaque:

1. O arquivo de senhas (`passwd`): no qual são registrados os usuários do sistema;
2. O arquivo de definição das máquinas (`hostsdef`): onde se encontram as informações sobre os EPs disponíveis no domínio de processamento.
3. O arquivo de cargas (`workload`): em que ficam registrados os históricos das medidas de uso de CPU e memória do sistema, coletadas periodicamente.

Vale a pena notar que os dois primeiros arquivos são normalmente encontrados na maioria dos sistemas operacionais: nos ambientes `Unix`, por exemplo, localizam-se no diretório de configuração `/etc`. No entanto, os formatos adotados pela WVM se distinguem em termos de sua funcionalidade (conforme será explicado no Capítulo 6). Já no tocante ao último arquivo (`workload`), existem algumas observações a serem feitas:

- A edição deste arquivo se torna algo secundário, já que, como ele é gerado automatica e periocamente no sistema, a única preocupação do administrador deve ser a de lhe reduzir o tamanho, no caso de exceder a algum limite considerado suficiente[86];
- A medida de carga de processamento de um sistema (também chamado de *workload*) é o elemento fundamental para a elaboração de qualquer

encerramento bruto das operações; (ii) impossibilidade de execução do comando; (iii) execução da atividade de maior prioridade; ou (iv) ativação de um estado inconsistente no sistema. Em todo o caso, é necessário considerável cautela antes da execução de comandos por parte dos administradores.

[85] Na realidade, a execução do servidor como aplicação deve-se fundamentalmente à necessidade de controle de *threads* e conexões *sockets* feitas pelos clientes.

[86] Diferentemente do que ocorria com os outros arquivos de gerência, este se comporta como um registro de logs (o que é comum em ambientes `Unix` e, portanto, pode crescer indefinidamente.

política para o escalonamento de tarefas. Os dados sobre o estado atual e o histórico do sistema permitem escolher, dentre um conjunto – *pool* - de máquinas, aquela(s) mais adequada(s) à submissão de um determinado programa para processamento. Cabe notar aqui um dado importante: não basta simplesmente a obtenção de uma medida instantânea isolada, mas também, de uma média de valores, a fim de se permitir analisar adequadamente uma determinada máquina[87];

- Existem variáveis de grande importância que devem ser cuidadosamente analisadas para a tomada adequada de decisão: o *clock* da máquina, sua capacidade de memória e o tipo de sistema. Uma forma de equacionar tais valores, seria utilizar alguma forma de normalização, fornecendo critérios adequados para uma comparação entre os diferentes equipamentos[88]. No próximo capítulo, teceremos considerações sobre formas de se obter as medidas do sistema e critérios para análise destes valores.

♦ B. Ambiente de Administração Remota

As atividades mais importantes do servidor são realizadas neste ambiente, que se relaciona com os clientes, cuidando da administração dos processos. Seus dois módulos estão mostrados na figura 5.11.

➢ (i). Módulo de Controle dos Clientes

O atendimento às solicitações dos clientes constitui a tarefa primordial do servidor. As respostas às requisições envolvem tanto a transferência de dados quanto a execução remota de comandos. Conforme mostrado na próxima seção (5.3), a interação entre o cliente e o servidor ocorre por meio de uma metalinguagem de comandos, de forma bastante simples.

---

[87] O sistema de contabilização dos recursos no Unix, por exemplo, utiliza medidas tomadas através de medições horárias, diárias e mensais.

[88] A análise da equivalência entre máquinas (normalização) é utilizada, por exemplo, nos Centros Nacionais de Processamento (CENAPAD's) para calcular a chamada **unidade de serviço** (US), medida de desempenho – usado para avaliar o custo das aplicações – que equivale a uma hora de utilização de CPU em um nó do sistema IBM SP/2.

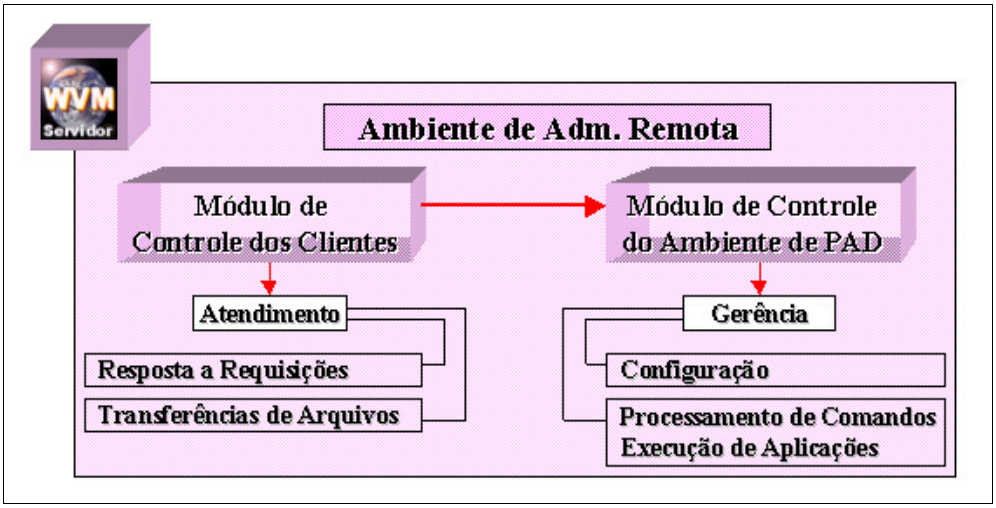


Figura 5.11: Organização do Ambiente de Adm. Remota no Servidor

Vale a pena notar que toda atividade requisitada pelos clientes, não diretamente relacionada com o processamento no *cluster*, é executada neste módulo. Exemplo disso são as operações de transmissão de dados, que independem do ambiente MPL. Dentre as preocupações inerentes a este módulo, se incluem:

1. Montagem de uma lista de clientes ativos com suas respectivas prioridades: Após cada validação na requisição de conexão, uma nova identificação é dada ao cliente até que este seja finalizado (ou não mais seja possível de comunicação). Cada *thread* segue o seu ciclo de vida em separado, podendo estar em estados específicos do processamento[89];

2. Controle da seqüência de conexões por clientes: Por razões de economia e simplicidade, a conexão de um cliente com um servidor não é contínua: apenas quando determinados comandos são requisitados, é que as conexões são estabelecidas. A gerência dos *sockets* TCP-IP fornece uma identificação seqüencial para as conexões, simplificando o tratamento e a organização destas comunicações.

Do ponto de vista da implementação, portanto, este módulo cuida da criação e gerenciamento de *threads*, se preocupando com a configuração mínima para cada cliente. Alguns comandos são implementados neste sentido, tais como a mudança de prioridade, a listagem das informações dos clientes e a finalização das *threads* (vide maiores informações no item 6.3.1.C). Com base nesta organização, pode-se criar um tratamento para falhas nas comunicações entre o servidor e os clientes, bem como abrir

[89] Os estados dos clientes são detalhadamente numerados na próxima seção, quando o diagrama da metalinguagem for descrito. Por hora, deve-se considerar apenas que tais mudanças de estado ocorrem quando das requisições dos clientes e do envio das respostas pelo servidor.

espaço para a implementação de uma comunicação fim-a-fim entre clientes, intermediada pelo servidor (necessária para a metacomputação).

- (ii). Módulo de Controle do Ambiente de PAD

Finalmente, neste último módulo, as atividades relacionadas com a execução das aplicações no domínio de PAD são executadas. Ao receber as informações dos clientes, o servidor envia ao sistema uma chamada para a execução dos programas em uma dada MPL, configurando o ambiente de execução, preparando a máquina virtual e, finalmente, deixando o sistema nativo executar a aplicação. As mensagens e saídas recebidas pelo sistema são repassadas, então, aos clientes.

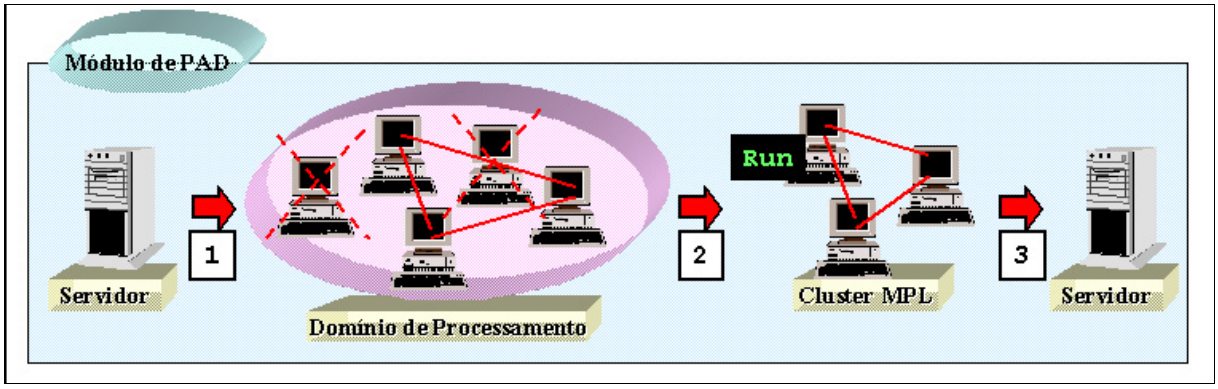


Figura 5.12: Passos do módulo de processamento

Além destes procedimentos, o controle dos ambientes de processamento (MPI e PVM) constitui uma atividade importante para o servidor. Neste caso, destacam-se tanto operações de verificação e modificação das variáveis de sistema, quanto a parametrização dos comandos de compilação e processamento das aplicações. Vale a pena observar que, do ponto de vista deste último módulo, o servidor WVM se constitui, funcionalmente, como um **cliente**, requisitando processamento no *cluster*. Isto reforça a necessidade de que este servidor tenha comunicação direta com ao menos uma das máquinas do ambiente de processamento. As funções deste módulo (ilustradas na figura 5.12) compreendem os seguintes passos:

1. Ativação da máquina virtual composta pelas máquinas do *cluster* previamente selecionadas para a execução da aplicação paralelo-distribuída;
2. Transmissão do comando de execução da aplicação na MPL determinada pelo usuário, com as parametrizações necessárias para o seu processamento direto[90];

[90] Conforme anteriormente mencionado, é necessário que o programa não possuia interatividade, já que ele é executado através do servidor nas máquinas do domínio de processamento.

3. Transmissão dos dados entre o *cluster* e o cliente, permitindo intermediar a sua exibição ou visualização.

Uma última observação, relativa ao servidor, será ainda vista quando tratarmos das modificações necessárias ao suporte de metacomputação, que será mostrado no último capítulo.

5.3. Comunicação em WVM

5.3.1 – Metalinguagem (ML)

Tendo verificado os modelos funcionais, tanto do cliente quanto do servidor no protótipo WVM, cabe finalmente, tecer detalhes sobre o relacionamento entre estes dois componentes. O que descreveremos aqui é um desdobramento do diagrama funcional (mostrado na seção 5.1.2), levando em conta os pormenores da comunicação. A metalinguagem – identificada, de agora em diante, por ML – apresenta alguns aspectos de implementação; contudo, o que nos interessa é apenas a descrição de como estes components se relacionam. Vale recordar que, como foi mencionado, a ML é mais facilmente percebida na visão de WVM como *middleware*.

A figura 5.13 ilustra o o processo de comunicação entre o cliente e o servidor via comandos da ML. O diagrama está descrito sob a forma de autômato, dando ênfase aos procedimentos invocados e ao relacionamento entre eles. A seguir, serão mencionados alguns dos métodos existentes na implementação e que serão comentados para facilitar a compreensão, de acordo com as seguintes observações:

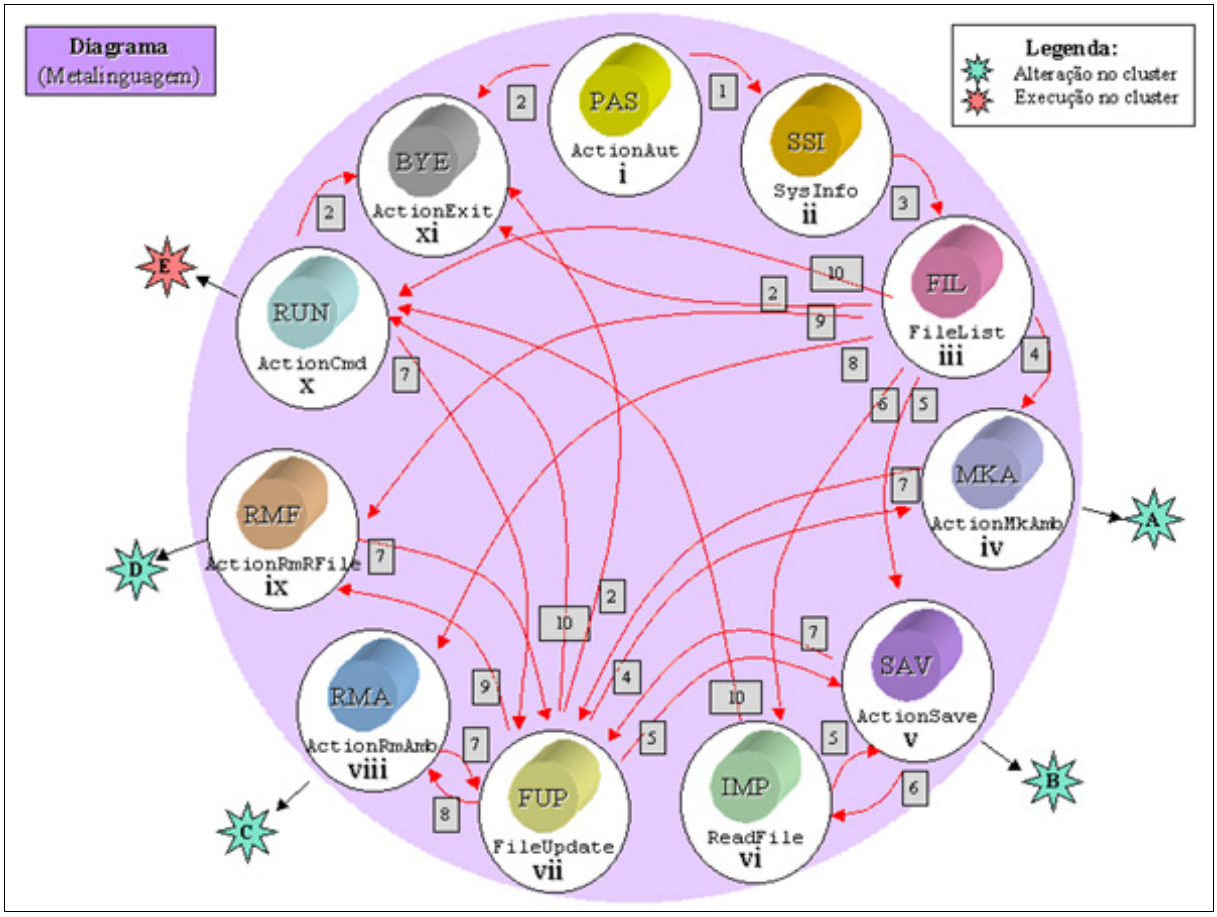


Figura 5.13: Diagrama funcional da ML

- Os mnemônicos (ou diretivas) em destaque indicam as macroinstruções[91] utilizadas no programa e que são enviadas do cliente ao servidor ou vice-versa. Os estados acionados por cada diretiva estão numerados de (**i**) a (**xi**).
- Um determinado estado pode ser alcançado de diferentes formas. Tais fluxos estão discriminados em números arábicos de [1] a [10].
- Os momentos em que são feitas mudanças no ambiente de processamento são ilustrados nos pontos A, B, C, D e E. A utilização de macros, embora possa não facilitar a legibilidade, favorece a simplicidade na explicação do modelo.
- Algumas diretivas são invocadas diretamente pelo usuário (como PAS, MKA e RUN), enquanto outras ocorrem implicitamente (como FIL e FUP), com o objetivo de ajustar o ambiente dos clientes.
- Além disso, devem ser observados algumas características dos autômatos: se a partir do estado inicial (**i**), é possível atingir o estado final (**xi**); a existência de

[91] Macroinstruções são comumente conhecidas simplesmente como **macros**, identificação essa que será usada daqui em diante.

apenas um único estado final e, finalmente, a alcançabilidade de todos os estados.

Para um melhor entendimento sobre o funcionamento da ML, serão descritos os passos mais comuns de execução, a partir da iteração inicial entre o cliente e o servidor. Vale a pena observar que, para a simplicidade da explicação, somente alguns fluxos do diagrama são considerados.

- Etapa (i):

  O primeiro passo consiste na autenticação do cliente (PAS) que, em caso de sucesso [1], levará à validação de conexão. Caso contrário [2], o procedimento de finalização é chamado e a aplicação encerrada[92]. Esta atividade corresponde ao módulo de certificação do cliente.

- Etapas (ii) e (iii):

  Supondo sucesso na autenticação, são obtidas informações sobre o sistema (SSI), que são repassadas ao servidor para registro. Tais dados fornecerão a base tanto para a seleção de *scripts* (como criação de ambientes, remoção de arquivos, etc.), quanto para a execução dos comandos informativos do servidor (vide seção anterior). A descrição dos ambientes e a listagem dos arquivos (FIL) conclui a preparação do ambiente do usuário. Note que os passos [2] e [3] ocorrem seqüencialmente, envolvendo tanto o módulo de configuração do usuário do cliente, quanto o módulo de controle dos usuários do servidor.

- Etapas (iv) e (v):

  A partir deste ponto, cabe ao usuário decidir o que fazer. Supondo a necessidade de criação de um ambiente [4], a pasta adequada - PVM, MPI ou Java - é criada no servidor (MKA). Um procedimento mais usual, no entanto, consiste na criação de arquivos no ambiente remoto; o salvamento de tais arquivos é feita por SAV [5]. Tanto em uma diretiva quanto outra, existe uma alteração a ser propagada no *cluster* remoto (A e B). Isto engloba as atividades dos módulos de configuração MPL e comando do cliente, bem como o módulo de controle do ambiente de PAD do servidor.

- Etapa (vi):

[92] O procedimento de finalização funciona diferentemente entre aplicações *standalone* e *applets*: enquanto na primeira, uma invocação do sistema operacional é possível (função `exit()`), na segunda, apenas se pode chamar o método `close()` para o término da *thread* que executa na *applet*.

A leitura dos arquivos remotos (para alguma eventual alteração) é a função mais comum [6] e pode ser feita pela diretiva IMP (impressão). Note que as modificações devem ser salvas a fim de que o trabalho não seja feito em vão [5]. O módulo relacionado com isto no cliente é o de editoração, que deve ser suportado pelo já mencionado módulo de controle do ambiente de PAD no servidor.

- Etapas (vii), (viii) e (ix):

  Outras funções de alteração (C e D), indicam respectivamente a remoção de um ambiente [8], pelo comando RMA e a exclusão de um arquivo [9], por RMF. Note que a primeira diretiva inclui obrigatoriamente invocações implícitas para a remoção de arquivos existentes na pasta. Vale mencionar que após qualquer alteração, um procedimento de atualização (FUP) deve ser chamado [7]. Ele é responsável por refletir no ambiente do usuário os comandos executados. Observe que essas atividades envolvem os mesmos módulos mencionados nas etapas (**iv**) e (**v**) anteriores.

- Etapas (x) e (xi):

  Finalmente, tendo sido feita toda a preparação necessária [10], a rotina de processamento no cluster (RUN) pode ser invocada. As execuções remotas podem ainda implicar em alterações no ambiente do *cluster* (E). Os resultados são, então, repassados aos clientes que podem, finalmente, concluir sua execução (BYE). O módulo de comando no cliente e o módulo de controle do ambiente de PAD no servidor são os responsáveis por essas atividades.

Observe que, embora tenhamos relatado o funcionamento, do ponto de vista do cliente, o componente servidor apresenta, igualmente, a mesma lista de macros, graças à sua função como *middleware*: é ele quem intercepta os comandos do cliente, feitos pelas diretivas, e os trata adequadamente, executando as funções requisitadas (vide seção 5.1.2.B).

Até o momento, temos nos detido nas informações relativas aos componentes cliente (C) e servidor (S), bem como no processo de comunicação entre eles, através de uma visão funcional. No próximo capítulo, nos deteremos nas questões relativas à implementação da ferramenta, mostrando como está operando o protótipo atual e qual o ambiente de processamento existente.

Conclusão

Neste capítulo, apresentamos a arquitetura e os aspectos funcionais da ferramenta WVM, que descrevemos ao longo de toda essa Segunda Parte.

No capítulo seguinte, descreveremos a implementação da ferramenta, tecendo considerações sobre a tecnologia empregada e as alternativas utilizadas no desenvolvimento do protótipo.

PARTE III

IMPLEMENTAÇÃO DA FERRAMENTA

Capítulo 6: Desenvolvimento do Protótipo
Capítulo 7: Conclusão

Capítulo 6

Desenvolvimento do Protótipo

Introdução

A arquitetura apresentada no capítulo anterior define as características da ferramenta. Para poder demonstrar sua funcionalidade, a implementação de WVM foi feita em um protótipo que aqui descreveremos. O objetivo é tecer considerações de cunho prático e levar em conta aspectos relativos à tecnologia adotada no desenvolvimento da ferramenta.

6.1. Aspectos Gerais

6.1.1 – Considerações sobre o *Software*

O desenvolvimento do protótipo buscou utilizar-se de tecnologia simples, mas eficiente o bastante para prover a implementação do modelo apresentado no capítulo anterior. O objetivo foi o de facilitar tanto sua programação quanto utilização, permitindo uma implementação rápida e de baixo custo dos componentes, com a funcionalidade pretendida.

- A. Considerações sobre a LP

Por ser uma linguagem de programação relativamente nova e ainda em amadurecimento, as implementações dos programas em Java vêm sendo modificados, à medida em que novas especificações foram sendo sugeridas (ou impostas)[93]. Um exemplo disso foi a mudança do tratamento de eventos (da versão 1.0 para a versão 1.1), o suporte a *threads* (da versão 1.1 para a versão 1.2) e a ampliação dos componentes gráficos, fornecendo

[93] Maiores infornações podem ser obtidas nas documentações da Java Language Specification (JLS) ou no *site* http://www.javasoft.com/.

maior versatilidade na construção das interfaces e melhor padronização para ambientes heterogêneos (iniciado na versão 1.2 e amadurecido na versão 1.3)[94].

Essas observações esclarecem os motivos pelos quais tanto a interface cliente quanto o servidor foram sendo modificados ao longo do desenvolvimento da dissertação. No estado atual, os protótipos seguem o padrão Java 2, obedecendo tanto à especificação 1.2 quanto à 1.3. A opção por estas versões visa, de um lado, a utilização de novos recursos implementados pelos modelos de eventos e as vantagens das classes dos pacotes `java.net` e `java.awt`, em relação à versão 1.0 e 1.1. Por outro lado, em se tratando das distribuições mais recentes, o objetivo maior é o de possibilitar a utilização de inovações gráficas, como as fornecidas pelo pacote `javax.swing`. Cabe salientar, contudo que, para algumas JVMs, certas especificações da versão 1.2 ainda não foram adequadamente implementadas nos *Web browsers* ou nas JREs dos sistemas operacionais[95].

Apesar de algumas vantagens (como a compatibilidade), a permanência nas especificações mais antigas da linguagem não representa, em si, uma alternativa vantajosa. De fato, à medida em que o tempo passa, as JVMs mais antigas vão ficando obsoletas e tendem a ser abandonadas, em prol de versões mais recentes (normalmente resultando em funções mais eficientes ou com maiores recursos). Um exemplo disso pode ser constatado quando encontramos métodos identificados como **depreciados** durante o processo de geração dos *bytecodes*[96]. Isso eqüivale a obedecer à regra de se buscar compatibilidade de tecnologia até certo limite.

Nas seções que se seguem, teceremos considerações sobre alguns pontos relevantes da implementação feita para os componentes cliente e servidor, no tocante tanto aos aspectos visuais, quanto à implementação dos módulos funcionais.

- (i). Com Relação aos Componentes Gráficos

---

[94] As descrições pormenorizadas das contínuas melhorias implementadas na linguagem podem ser encontradas na literatura. Para os nossos propósitos ressaltamos basicamente três aspectos que ilustram bem este quadro de evolução da linguagem: (1) a mudança do contexto de tratamento de eventos, que migraram de classes mais genéricas para elementos mais particulares e específicos, sendo, portanto, mais facilmente tratados; (2) a correção nos tratamentos de múltiplos canais de execução, fornecendo-lhes as políticas de segurança necessárias ao tratamento por parte da JVM e (3) o surgimento das interfaces e componentes independentes de arquiteturas, como é o caso da tecnologia `Swing`.

[95] Por exemplo, o `Internet Explorer 5` e o `Netscape Communicator 6` ainda não suportam adequadamente tecnologias como o `swing` e apresentam problemas no tratamento de múltiplas *threads*, chegando a travar os *browsers*. A alternativa normalmente utilizada por algumas distribuições dos navegadores para dar suporte às versões mais recentes das JVMs consiste basicamente na instalação de um *plug-in* (componente adicional) que instala o JRE da `Sun`.

[96] No pacote de desenvolvimento JDK padrão, os métodos detectados como **depreciados** (*deprecated*) durante a compilação de arquivos `.java`, geram mensagens indicativas das novas classes, métodos ou parâmetros a serem utilizados em substituição à chamada original, a fim de orientar os programadores.

Para melhorar o tempo de carga da *applet* via rede (o que ainda hoje representa um gargalo em Java), apenas controles mínimos foram inseridos nas interfaces, tanto cliente quanto servidora. Dependendo da versão escolhida, elas serão compatíveis com a distribuição JDK 1.1 ou 1.2. Porém, muito mais do que se buscar uma minimização no tamanho dos códigos (e, portanto, uma melhoria da velocidade na transferência dos *bytecodes* pela rede), o que se pretende é permitir o uso da ferramenta em JVMs simples, tais como aquelas disponíveis via *browsers*.

Sob este aspecto, vale a pena mencionar uma questão prática: a AWT padrão funciona como uma interface entre o programa Java e o sistema operacional da máquina. Assim, ao se especificar um componente gráfico (como um botão, por exemplo), a JVM repassa ao SO a invocação daquele componente, segundo seus parâmetros (como posição, dimensões, cores, etc). A grande vantagem desta abordagem é que isto alivia o trabalho da máquina virtual; no entanto, cria problemas como a dependência de *layout* e a necessidade de chamadas de sistema para a atualização da tela. Por outro lado, a idéia central de tecnologias como o Swing (versão 1.2), é a de permitir que a própria JVM gerencie os componentes gráficos. Uma outra facilidade adicionada consistem em que, desta vez, o usuário passa a ter uma maior versatilidade na construção da interface (por exemplo, a capacidade de escolha entre as possíveis formas de visualização de um componente com *look-and-feel*). No entanto, isso causa uma considerável sobrecarga na execução das aplicações, motivo pelo qual cabe ao usuário optar por utilizar uma ou outra alternativa (AWT ou Swing).

Um outro aspecto que ainda será detalhado é que tais componentes não fazem uso de nenhuma classe privada e fechada – isto é, particular a uma determinada ferramenta – o que poderia vincular o protótipo a um determinado ambiente de desenvolvimento[97]. Assim, apenas o pacote JDK da Sun foi utilizado.

- (ii). Com Relação à Comunicação C/S

A comunicação entre o cliente e o servidor se dá através de *sockets* TCP e, portanto, com controle na transmissão e recepção dos pacotes[98]. Para simplificar a transferência de dados, apenas um conjunto básico de comandos é necessário – vide seção 5.3.1. No entanto, a interação entre o ambiente de processamento e o cliente

[97] Apesar da grande quantidade de ferramentas atualmente disponíveis no mercado, como VisualCafé (Symantec), Visual J++ (Microsoft) e JBuilder (Borland / Inprise), para mencionar apenas o ambiente Windows, existe um ponto comum a todas: a incorporação de classes proprietárias, inseridas pelos desenvolvedores para auxiliar na composição visual das aplicações criadas em tais ferramentas. Apesar dos benefícios, isto acarretará dificuldades na portabilidade dos **códigos fontes**, que a algum custo, poderiam ser ajustados para outros ambientes e sistemas.

[98] A utilização de comunicação TCP é hoje genericamente utilizada, em detrimento de pacotes UDP, que apesar da simplicidade, não apresentam maiores atrativos, graças à necessidade de gerenciamento.

remoto, pode requerer tempo considerável, quando conteúdos de arquivos são transferidos ou massas de respostas são direcionadas para a saída do usuário.

Deve-se salientar que, estritamente falando, o servidor poderia ser implementado em qualquer outra linguagem, desde que a manipulação das *threads* clientes fosse possível[99] e a sintaxe da comunicação, adequadamente respeitada (de acordo com a ML descrita anteriormente). No entanto, a opção por utilizar `Java`, permite, dentre outras vantagens que:

- O código do servidor seja facilmente portável para qualquer outra arquitetura, o que facilita a inclusão de novos domínios de PAD, com os mais variados sistemas;
- O componente servidor seja construído de forma similar ao cliente, tanto em aspectos de interface, quanto em termos de certas funcionalidades existentes naquele componente.

♦ B. Ferramentas de PAD

Conforme visto no Capítulo 2, o ambiente de processamento paralelo pode ser implementado através de MPLs, com requisitos mínimos de *hardware* e *software* e a possibilidade da utilização de uma rede local ou *cluster* para processamento distribuído de baixo custo.

A implementação aqui utilizada permite o desenvolvimento de programas em linguagem `C` ou `Fortran`, com a utilização de bibliotecas `MPI` ou `PVM`. A escolha por tais linguagens e bibliotecas deve-se:

- À grande diversidade de programas existentes nestas LPs e à cultura de programação universalmente existente, baseada nestas duas linguagens científicas;
- À base já instalada de programas nas MPLs mencionadas, que, além de serem gratuitas, são compatíveis em nível de biblioteca com as LPs anteriores, sendo, pois, de fácil compreensão para aqueles programadores.

A única restrição que deve ser lembrada nos programas a serem tratados aqui é a de que os programas não devem possuir interatividade com o usuário, já que serão executados indiretamente, através de chamadas ao sistema feitas pelo servidor WVM. Isso, com certeza acarreta a necessidade de se buscar uma solução cuidadosamente pensada, onde todos os

[99] Obviamente, toda a preocupação com relação à implementação do tratamento dos clientes, bem como a metalinguagem de comandos, deve ser igualmente observada.

parâmetros necessários possam ser fornecidos ao programa, sem a necessidade de intervenção do usuário[100].

6.1.2 – Considerações sobre o *Hardware*

Tendo mencionado os aspectos de *software* acima e as características dos componentes da ferramenta, podemos tecer considerações sobre o *hardware* e o sistema em que o protótipo foi implementado:

- A. Clientes

Os clientes WVM requerem uma configuração mínima de *hardware* para a sua execução: um *Web browser* habilitado com JVM ou qualquer JRE disponível em nível de sistema (como as versões do interpretador `java` existentes em sistemas como o `Windows`, `Linux` ou adicionados ao sistema pela instalção do pacote `JDK`). O processo de carga dos *bytecodes* pode ser consideravelmente lento em máquinas com poucos recursos de memória ou em sistemas de rede com baixas taxas de transferência de dados (largura de banda)[101]. As configurações mínimas variam com a arquitetura e o SO de cada máquina, mas considera-se que, para um PC, valores como 32MB de RAM, CPU com pelo menos 200MHz de *clock* e uma taxa de transferência não inferior a 32Kbps são suficientes.

- B. Servidor

A máquina onde reside o componente servidor deve ter memória e recursos suficientes para dar resposta apropriada aos clientes. No caso do protótipo a ser apresentado, o servidor está em execução em uma máquina RISC IBM modelo 250, com sistema operacional `AIX versão 4.2`, processador de 300MHz e 256MB de memória RAM. Esta configuração é necessária por três motivos:

- A estação está diretamente conectada à rede de processamento, tornando possível, em nível de sistema, a invocação de chamadas responsáveis pela execução de processos diretamente no *cluster* de processamento;
- O S.O. possui uma versão IBM do `JDK` que segue as especificações da versão 1.1 da JLE elaborada pela Sun, tornando possível a execução do programa servidor como aplicação, da forma que a arquitetura o exige;

---

[100] As formas para se resolver o problema da interação – existente, aliás em várias outras aplicações – são basicamente duas: uma mais estática, incorporando ao código do programa valores pré-determinados para o seu funcionamento e outra, mais dinâmica (e usual), na qual o mesmo programa obtém os valores a partir da leitura de arquivos os quais podem ser novamente escritos, para fornecer alterações necessárias a um futuro processamento.

[101] Algumas configurações de máquina e versões de *browsers* podem apresentar um funcionamento bastante debilitado, ou mesmo encerrar prematuramente as aplicações, por conseqüência do *timeout*, isto é, da extrapolação do limite de espera máximo para a obtenção de dados ou mesmo para a carga da aplicação.

- Existe uma versão do servidor *Web* NCSA ativo, previamente compilado para a arquitetura. Conforme estabelece a arquitetura, o *daemon* (`httpd`) é necessário para que os usuários possam conectar-se à máquina e fazer o *download* de páginas HTML com a *applet* do programa cliente.

Finalmente, como mencionado no item anterior, o fato de o servidor ter sido construído em Java, abre possibilidades para que possa ser facilmente modificado e compilado para outros ambientes, se necessário, ajustando-o às características de um determinado domínio de processamento. Na seção que se segue, iremos detalhar a implementação atual do cliente e, em seguida, do componente servidor da arquitetura.

6.2. Implementação do Cliente WVM

6.2.1 – Características da Interface

♦ A. Considerações Gerais

O protótipo da implementação do cliente WVM, embora apresente uma interface simples, possui um conjunto de componentes gráficos, classificados de acordo com a sua funcionalidade. De acordo com a distribuição utilizada, pode ser acessado via `AWT` ou `Swing`. A figura 6.1 exibe o *layout* do cliente, com os respectivos painéis que o compõem, usando a interface a `Abstract Window Toolkit` padrão.

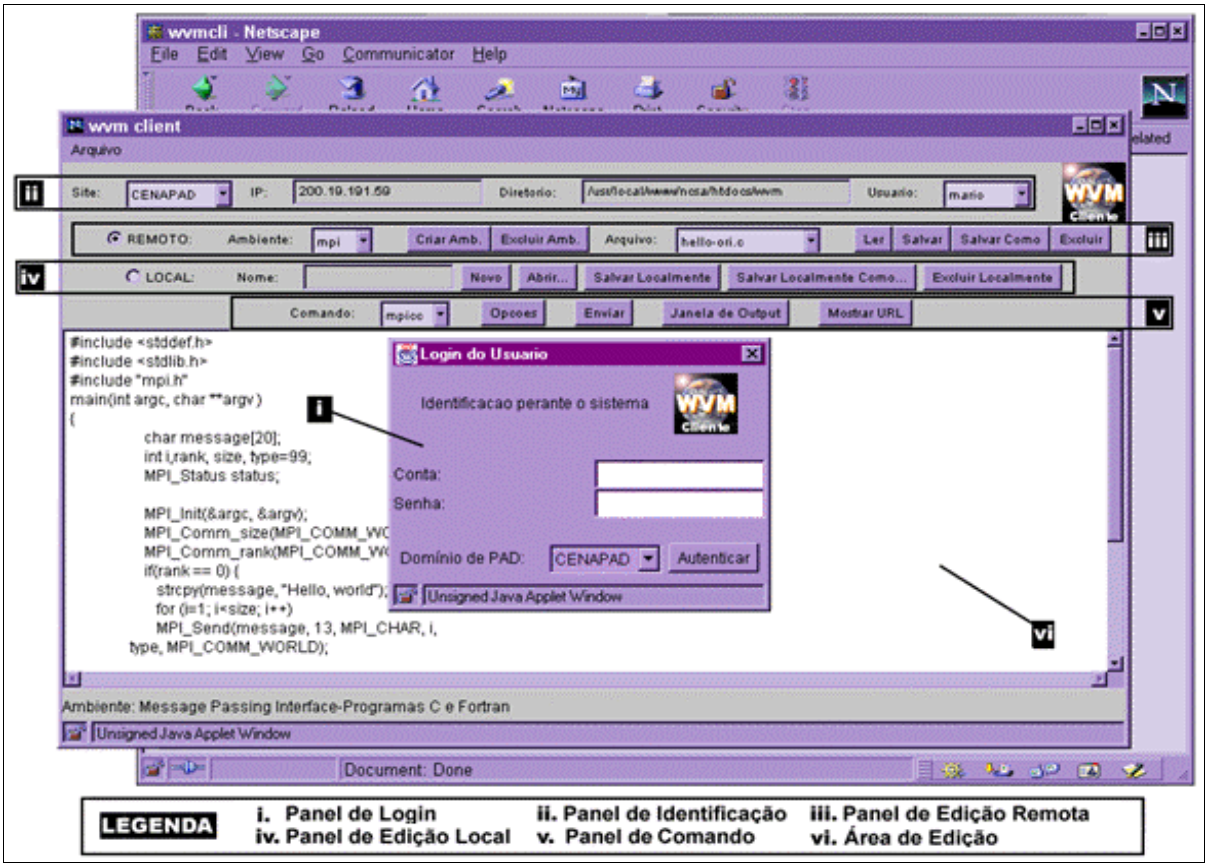


Figura 6.1: Interface AWT Padrão do Cliente WVM

Funcionalmente, o cliente WVM apresenta as opções necessárias para a gerência de comandos a serem executados: edição de arquivos (locais ou remotos), configuração do ambiente virtual (tipo de MPL, número de máquinas, etc.) e acompanhamento das execuções, o que permite subdividi-la em 7 partes, de acordo com suas funcionalidades (que serão detalhadas mais adiante):

1. Tela de autenticação (Panel de Login);
2. Área de sistema (Panel de Identificação);
3. Área de edição remota;
4. Área de edição local;
5. Área de execução de comando comando;
6. Área de Edição;
7. Telas de saída.

A figura 6.2 apresenta a mesma interface cliente, de acordo com o padrão `Swing` (especificação Java 2). Note que, apesar da organização ligeiramente diferenciada, os mesmos componentes podem ser identificados.

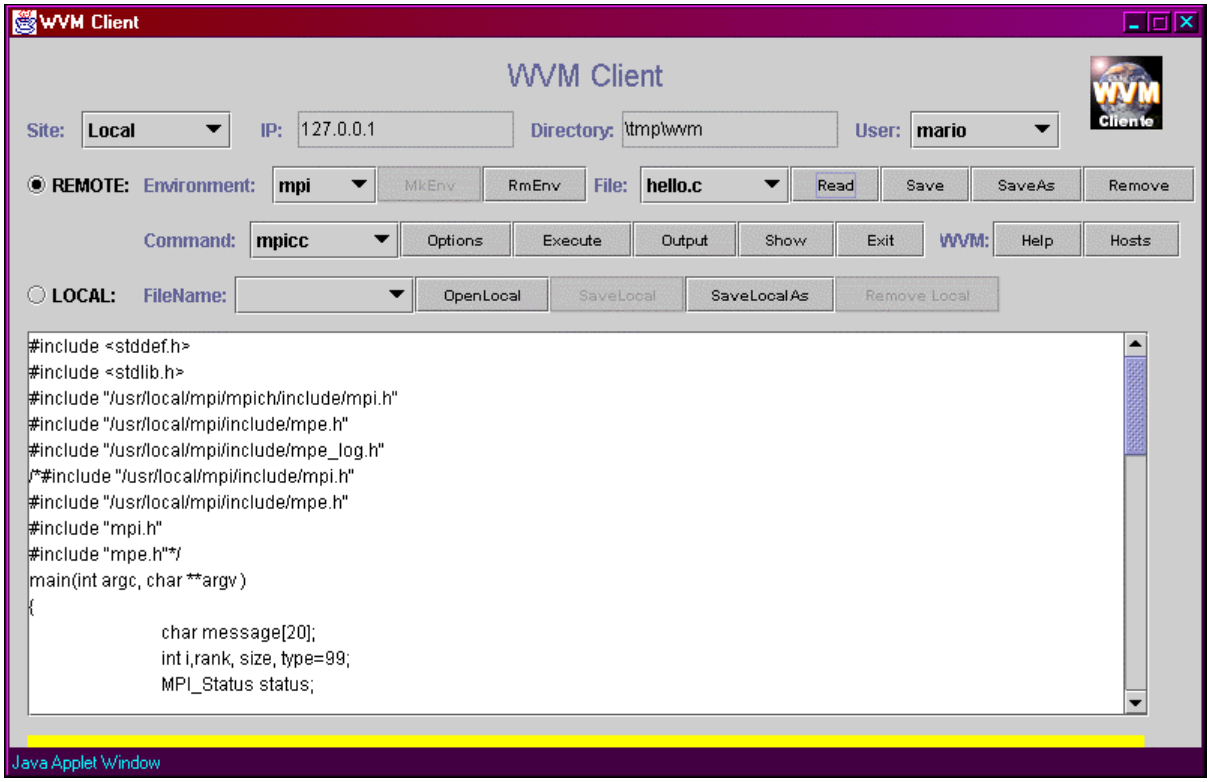


Figura 6.2: Interface Swing do Cliente WVM

♦ B. Componentes

➢ (i). Tela de Autenticação (Painel de *Login*)

A tela de autenticação (figura 6.3) é responsável pela implementação do Módulo de Certificação (Ambiente de Conexão do cliente).

Nele, o usuário fornece as informações necessárias para a conexão no ambiente de PAD, incluindo nome do usuário e senha, para um dado domínio de processamento.

Figura 6.3: Painel de *Login*

Conforme anteriormente dito, a tela de autenticação é sempre ativada quando um usuário requisitar mudança de *login*[102] ou de domínio de processamento. Note-se que, somente após a validação da identificação, é que o cliente pode interagir com o

[102] Deve-se perceber que denominamos **cliente** não ao usuário, mas ao programa. Assim, vários usuários podem, teoricamente, usar a mesma aplicação (ou *applet*). Note que, em todo caso, é necessário ao usuário possuir a senha da conta para a qual se pretende ingressar.

servidor. Para efeitos de implementação do protótipo, foram definidos 3 *sites* (domínios de processamento):

- CENAPAD: Centro de Processamento que conta com o *cluster* RISC SP/2 e que será descrito no último capítulo (seção 7.1.1);
- RMAV: Laboratório da Rede Metropolitana de Alta Velocidade, que conta com PCs interligados via rede ATM (maiores informações podem ser obtidas no item 7.1.1.A e no apêndice C.2);
- LOCAL: Conforme indicado, permite utilizar apenas a máquina local para o desenvolvimento dos trabalhos. Neste ambiente de teste extremamente simples, assume-se que os dois componentes encontram-se localizados na própria máquina.

Outros domínios de processamento podem ser posteriormente definidos, permitindo estender o uso da ferramenta (por exemplo, entre os vários Centros de Processamento mencionados sucintamente no apêndice C.2).

- (ii). Painel de Identificação

Aqui ficam as informações relativas ao domínio de PAD ao qual o cliente está conectado (figura 6.4) e a sua identificação (figura 6.5). Funcionalmente, este painel está ainda relacionado com o Ambiente de Configuração, provendo as funções do Módulo de Configuração do Usuário, conforme especificado no capítulo anterior (ver item 5.2.1.A.ii).

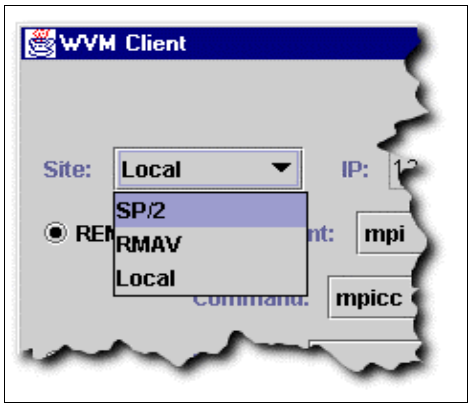


Figura 6.4: Painel de Identificação do *Site* de PAD

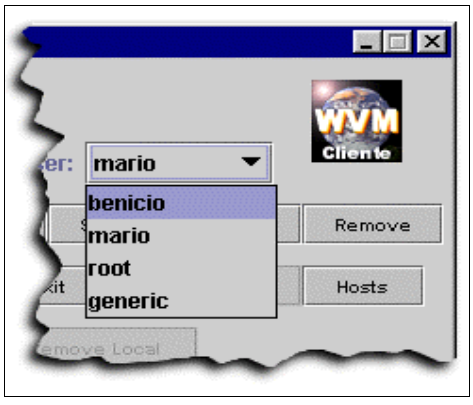


Figura 6.5: Painel de Identificação do Usuário

Sempre que houver uma requisição para mudança de *site* (domínio de PAD) ou de identificação, o módulo de certificação é invocado, a fim de validar a conexão ao novo sistema ou com esta nova identidade. Dentre as informações providas estão:

- Site: Descreve qual o Domínio de Processamento atual;
- IP: Identificação do *host* do servidor (com o qual o cliente se conecta);
- Directory: Especifica qual a pasta de trabalho atual;

- User: Identificação do *login* do usuário.

➢ (iii). Painel de Edição Remota

O painel de edição é responsável por abrir conexões com o servidor e buscar remotamente aplicações, arquivos ou informações, a fim de serem editadas localmente na *applet*. Dentro da arquitetura funcional, ele envolve o Módulo de Editoração para o Ambiente de PAD (item 5.2.1)[103].

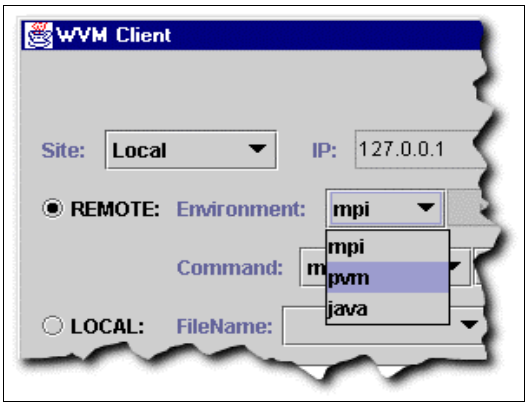


Figura 6.6: Caixa de Opções de Ambientes Remotos

A qualquer instante, o usuário pode definir o ambiente em que está operando (PVM, MPI ou Java), conforme anteriormente descrito nos Módulos de Configuração (5.2.1.B.i e 5.2.1.C) – figura 6.6. A cada modificação feita, a lista dos arquivos é ajustada para indicar corretamente aqueles contidos na pasta selecionada.

Dentro de uma determinada pasta, encontram-se os arquivos de trabalho dos usuários. Note que nessa seção apenas são tratados apenas objetos **remotos**; a possibilidade de utilização de arquivos locais será vista mais adiante.

Finalmente, o usuário pode especificar qual arquivo ele deseja editar (seja um código de programa, um arquivo de dados para processamento ou ainda informações), de acordo com as funções mencionadas na arquitetura funcional (itens 5.2.1.B.i e 5.2.1.C) – figura 6.7.

Figura 6.7: Caixa de opções de Arquivos Remotos

As definições de ambiente e de arquivos irão configurar as opções disponíveis no painel de comando, conforme adiante mostrado. As principais informações consistem portanto de:

[103] O mesmo se fazendo para o futuro ambiente de metacomputação (MC).

- Environment: Discrimina a MPL ou a linguagem Java;
- File: Identifica o atual arquivo de trabalho.
- Opções de comandos correlatos para manuseio das pastas ou arquivos: Mkenv (criação de ambiente), Rmenv (remoção de pasta), Load (carga de um arquivo remoto), Save e SaveAs (transmissão de dados para salvamento remoto), Remove (remoção de arquivos).

A execução dos comandos acima são ilustradas na ML da figura 5.13 e mencionados nas etapas de (iv) a (ix) mencionadas na seção 5.3.1. Vale, no entanto, esclarecer aspectos práticos da implementação: a execução propriamente dita do comando de edição remota é feita em várias etapas, conforme indicado na figura 6.8 a seguir:

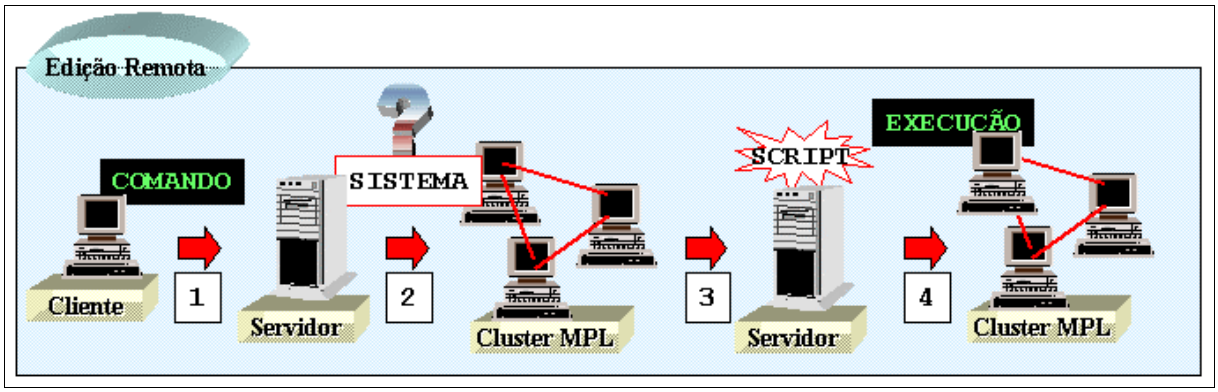


Figura 6.8: Passos para a execução de um comando de edição remota

1. Submissão do comando ao servidor;
2. Reconhecimento do sistema operacional do *cluster*;
3. Seleção do *script* do servidor responsável pela atividade;
4. Execução propriamente dita do comando originalmente discriminado.

Note que, devido à diversidade dos ambientes, a partir da requisição do cliente, é necessário um tratamento do comando até que este possa ser adequadamente traduzido para alguma linguagem do sistema remoto (o que é feito via interpretador de comandos do SO, ou seja, *scripts* de *shell*).

- (iv). Painel de Edição Local

Dependendo das permissões para a escrita de arquivos locais, o painel de edição local pode ser utilizado. Conforme anteriormente dito, a utilização de edição local permite uma desconexão com a rede, durante as etapas de construção dos programas, evitando o desperdício de comunicações. As funcionalidades são análogas às definidas no Módulo de Editoração para a edição remota (5.2.1.A.ii), incluíndo operações como: OpenLocal (carga), SaveLocal e SaveLocalAs (salvamento),

RemoveLocal (remoção de arquivo). A figura 6.9 ilustra esquematicamente o que ocorre durante uma edição local.

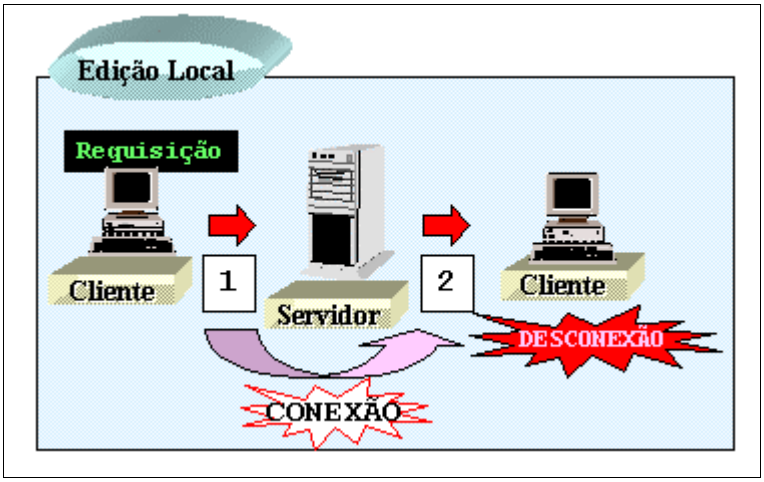


Figura 6.9: Esquematização das etapas para edição local

- (v). Painel de Comandos

Corresponde aos controles relacionados com a execução de comandos remotos, tais como compilação e execução de aplicações. Funcionalmente, implementa o Módulo de Comando (descrito na seção 5.2.1.A.iii) e é o mecanismo que permite a comunicação entre o servidor e o ambiente de PAD, através de chamadas ao sistema.

As opções do painel de comandos variam de acordo com a configuração do ambiente (PAD ou MC) e ainda de acordo com o arquivo atualmente selecionado. A figura 6.10 ilustra um exemplo da disposição de opções no painel de comando (no caso, arquivos em `C` no ambiente `MPI`).

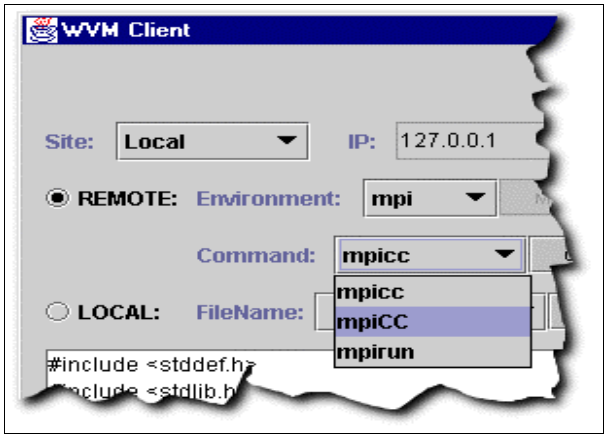


Figura 6.10: Painel de Comandos

Resumidamente, as facilidades implantadas no cliente WVM permitem que os usuários possam acessar mais comodamente certos recursos do ambiente de PAD, sem a necessidade de comandos explícitos para conexão remota (`telnet`), transferência de dados (`ftp`) e visualização de resultados.

A tabela 6.1 ilustra os comandos disponíveis de acordo com os arquivos selecionados. Note que alguns deles pertencem à programação `Java` e não propriamente às bibliotecas paralelas.

		Tipo de		

Processamento	Ambiente	Arquivo (extensão)	Edição	Comandos Relacionados
PAD	PVM	.c	Sim	Cc, gcc, xlc
		.f	Sim	F77, xlf
		.mak[104]	Sim	Make
		[Binários]	Não	Execução (RUN)
	MPI	.c	Sim	Mpicc, mpiCC[105]
		.f	Sim	Mpif77
		.mak	Sim	Make
		[Binários]	Não	Mpirun
MC[106]	JAVA	.java	Sim	Javac, javadoc
		.class	Não	Java, havah, javap
		.html	Sim	Appletviewer

Tabela 6.1: Relação entre Tipos de Arquivos e Comandos por Ambiente

Alguns formatos de arquivo são identificados como **especiais** e apresentam características próprias, algumas dependentes do ambiente, conforme mostrado na tabela 6.2.

Tipo de Arquivo (extensão)	Edição	Função
.sh	Sim	Arquivo em lote para execução direta – Ambiente Unix (RUN)
.bat	Sim	Arquivo *batch* para execução direta – Ambiente DOS (RUN)
.mac	Sim	Configuração da máquina virtual
.dat	Sim	Plotagem de gráfico (PLOT)

Tabela 6.2: Relação entre Tipos de Arquivos e Comandos Comuns

Observe que os primeiros tipos de arquivos (na verdade, *scripts* de *shell*), podem ser editados e executados, como normalmente o seriam em qualquer ambiente. Já os formatos que se seguem são responsáveis por duas operações internas às funcionalidades da ferramenta:

- Arquivo .mac: Configuração do **arquivo de máquinas** para a construção do ambiente virtual, necessário antes do início do processamento de PAD;
- Arquivo .dat: Configura os pontos para plotagem em um gráfico cartesiano (útil para a visualização dos dados)[107].

➢ (vi). Área de Edição

A área de edição é responsável por exibir o conteúdo dos arquivos (locais ou remotos), permitindo ao usuário alterá-los, salvando-os remotamente. Está relacionado

[104] O uso de arquivos do tipo **Makefile** (com extensão ".mak") no ambiente Unix é bastante comum, devido aos parâmetros de compilação, tanto para o chamado "programa mestre" quanto para o "programa escravo" (vide seção 2.2.3-A).

[105] Note que, como mencionado na seção 2.3.4 C.ii, a distribuição MPI utilizada (MPICH) apresenta os comandos mpicc, mpiCC, mpif77 e mpirun como *scripts* de compilação (e não arquivos binários).

[106] Embora o conceito de metacomputação (MC) apareça apenas no último capítulo, algumas facilidades já estão sendo implementadas no protótipo.

[107] A "plotagem" se faz mediante um formato específico nos dados, que deve ser obedecido, a fim de que parâmetros como os valores extremos e o tipo de curva, possam ser adequadamente configurados no gráfico. Maiores detalhes são mencionados a seguir.

com os painéis de edição local e remoto e opera de acordo com as permissões definidas no painel de comandos, constituindo com eles o Módulo de Edição.

- (vii). Telas de Saída

Ao lado destes painéis, existem janelas auxiliares como a tela de saída de comandos (que exibe para o usuário os resultados dos programas após sua execução) e as telas de mensagens do sistema. As saídas dos códigos podem ser vistas em telas como a mostrada na figura 6.11, obedecendo aos seguintes formatos:

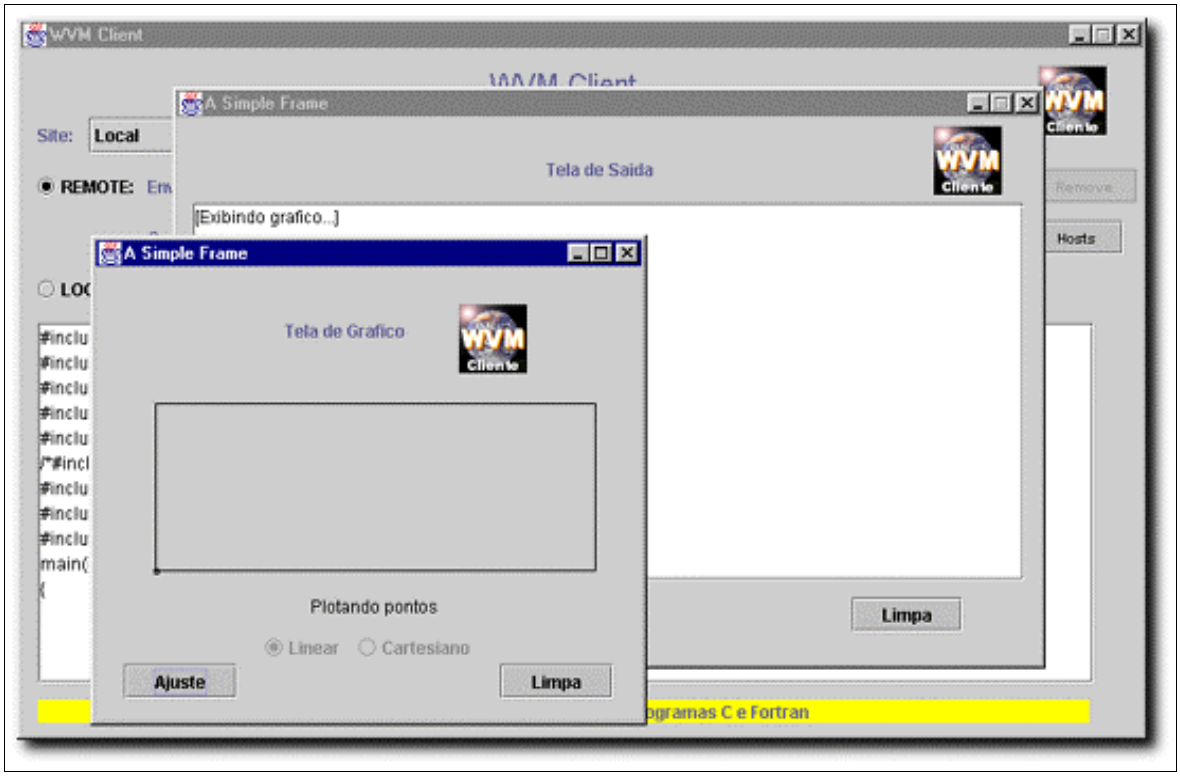


Figura 6.11. Telas de Saída de Comandos

- **Saída padrão:** Obtém os resultados via console, sendo pois, a alternativa mais usual e apresentando como vantagem a ausência de formatação e o uso generalizado de um fluxo de saída nas linguagens de programação (como o `stdout` do C);
- **Visualização gráfica:** Implementa a visualização de dados, através da formatação dos mesmos e impressão em uma interface para plotagem dos pontos. Isso permite interpretar as informações com mais facilidade do que apenas através da exibição dos dados[108].

[108] A formatação dos dados define como as informações obtidas devem ser exibidas na tela de saída, conferindo uma adequada interpretação gráfica. Para isso estão definidas diretivas de pré-processamento (como "GRAFDAT") e alguns comandos que ajustam o espaço de plotagem, tanto para exibição estática (na qual os pontos são plotados em uma única tela) ou dinâmica (que permite o deslocamento dos quadros, em intervalos de tempos, facilitando, por exemplo, a percepção de convergências dos dados no gráfico).

A tabela 6.3 faz um resumo das características da interface do cliente, relacionado-as com a arquitetura funcional descrita no capítulo anterior.

Ambiente	Estrutura Funcional	Componente(s) da Interface
Conexão	Módulo de Certificação	Painel de *Login*
	Módulo de Configuração do Usuário	Painel de Identificação
PAD	Módulo de Configuração MPL	Painel de Edição Remota
	Módulo de Editoração	Painel de Edição Remota, Painel de Edição Local, Área de Edição
	Módulo de Comando	Painel de Comando, Área de Saída

Tabela 6.3: Relação entre a Arquitetura Funcional e a Interface no Cliente WVM

6.3. Implementação do Servidor WVM

6.3.1 – Características da Interface

♦ A. Considerações Gerais

Apesar de mais complexo, servidor WVM apresenta uma interface mais simples que o cliente, já que não participa da interface com o usuário (figura 6.12). As funções do servidor, descritas na seção 5.2.2, envolvem basicamente a gerência das informações dos usuários e o controle dos processos concorrentes.

♦ B. Componentes

Da mesma forma que o cliente, o servidor WVM apresenta uma interface formadas por componentes que refletem os módulos funcionais anteriormente descritos. Eis os principais painéis da aplicação servidora:

Figura 6.12: Interface do Servidor WVM

- (i). Painel de *Login*

  Responsável pela validação do acesso ao ambiente, através da certificação do administrador. O funcionamento é análogo ao da interface cliente. Note que existe a necessidade um nível maior de segurança, já que no servidor se encontram funções como as de alteração de senhas e finalização de processos. Conforme já mencionado, ao contrário do que ocorria com os clientes, a operação de *login* baseia-se apenas na validação de uma conta local.

- (ii). Painel de Edição Local

  Onde arquivos locais ao sistema podem ser editados, incluindo as informações relativas aos usuários, tais como nome de contas e senhas. Diferentemente do que ocorre com o cliente, não existe aqui a necessidade de edição de arquivos remotos. Conforme mencionado no item 5.2.2.A, o administrador é responsável por arquivos relacionados com o(a):

  - Cadastramento dos usuários e de suas senhas (arquivo `passwd`);
  - Definição das informações das máquinas do *cluster* (arquivo `hostsdef`);
  - Controle do arquivo de registro das cargas do sistema (`workload`).

- (iii). Painel de Comandos:

  No qual o administrador pode executar comandos diretamente no sistema, como, por exemplo, em procedimentos de configuração de máquinas e alteração de ambientes de usuários (vide figura 6.13). A tabela 6.4 ilustra algumas funções existentes no servidor.

Comando	Significado
MKUSER	Criação de um usuário.
RMUSER	Remoção de um usuário
PASSWD	Modificação de senha de um usuário.
KILL	Finalização de um processo (cliente)[109].
PRIORITY	Mudança na prioridade de um cliente[110].
LSCLIENTS	Listagem dos clientes conectados.
VIEWCLIENT	Exibição das propriedades dos clientes.

Tabela 6.4: Rotinas do Módulo de Comando do Servidor WVM

[109] Sujeito às restrições do SO para gerenciamento de processos.

[110] A mudança de prioridade de uma *thread* (cliente) somente é possível se o sistema operacional suportar alterações de prioridade em tempo de execução (como ocorre no `UNIX`).

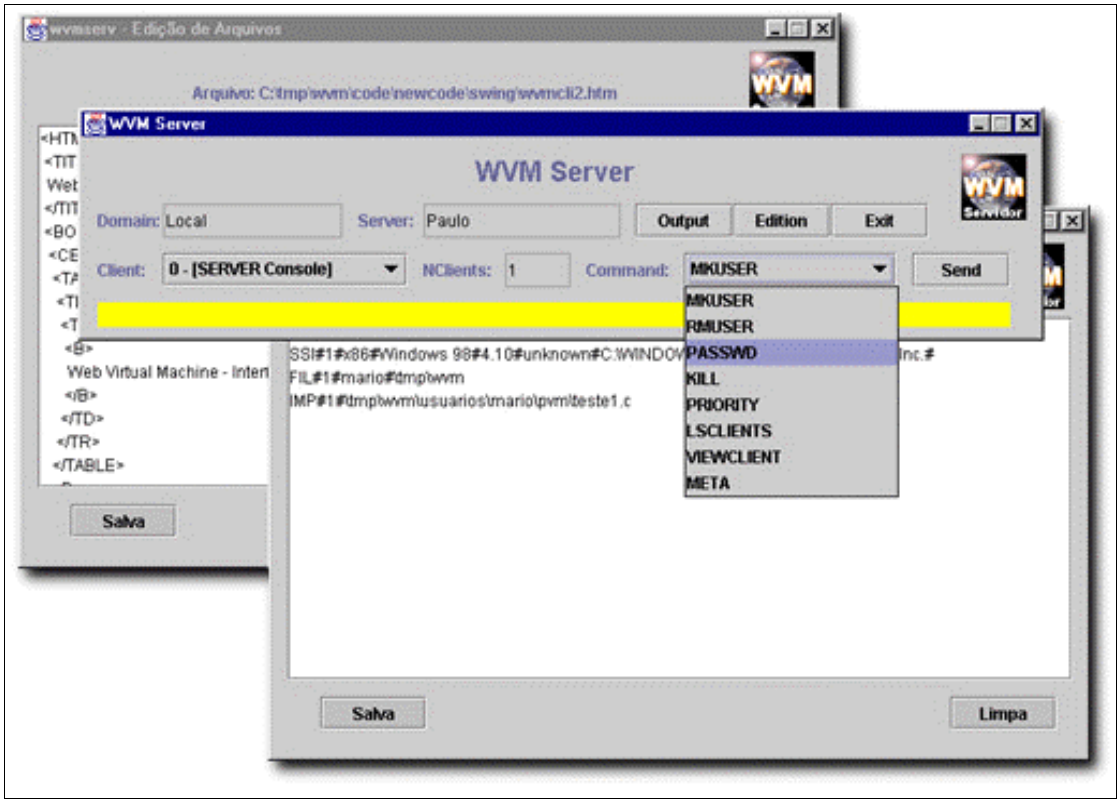


Figura 6.13: Telas do Servidor WVM

A seguir, são consideradas algumas observações referentes aos comandos do servidor:

- Os comandos relacionados com os usuários (MKUSER, RMUSER e PASSWD) alteram diretamente o arquivo de senhas do ambiente WVM, chamado de `passwd`, como no `Unix`. Para tanto, alguns testes são feitos internamente, como a verificação de contas preexistentes (o que validaria uma remoção ou alteração de senha, mas invalidaria a criação de uma conta);
- Os comandos relacionados com os processos ou *threads* (KILL e PRIORITY), dependem do suporte dado pelo sistema operacional nativo.
- Os comandos informativos (LSCLIENTS e VIEWCLIENT) fornecem informações recebidas quando da conexão inicial dos clientes, que indicam, por exemplo, qual a arquitetura e o sistema operacional existentes nas máquinas remotas[111].
- Comandos adicionais (como o META), estão sendo previstos para permitir a obtenção de informações sobre o ambiente MC.

A tabela 6.5 faz um resumo das características da interface do servidor, relacionado-as com a arquitetura funcional descrita no capítulo anterior.

[111] A obtenção de tais informações depende da obtenção de chamadas `System` feitas pela linguagem `Java`.

Ambiente	Estrutura Funcional	Componente(s) da Interface
Administração Local	Módulo de Autenticação	Painel de *Login*
	Módulo de Controle dos Usuários	Painel de Edição Local
	Módulo de Processamento	Painel de Edição Local, Painel de Comandos
Administração Remota	Módulo de Controle dos Clientes	Painel de Comandos
	Módulo de Controle de PAD	Painel de Comandos

Tabela 6.5: Relação entre a Arquitetura Funcional e a Interface no Servidor WVM

♦ C. Funcionalidades do servidor

Com relação ao servidor, cabe tecer algumas observações finais sobre dois aspectos distintos da implementação: o tratamento das *threads* (implementado no módulo de controle dos clientes) e as medidas de obtenção de cargas (presente no módulo de controle de processamento).

➢ (i). Tratamento de *threads* e processos

Um ponto importante relacionado com o servidor, diz respeito ao tratamento dos clientes pela aplicação. Antes de mais nada, vale salientar alguns pontos: como o servidor é *multi-thread*, o atendimento às múltiplas solicitações de conexão (feitas via *socket*) são respondidas através de uma classe extensão de `Thread`[112]. Isto significa que todos os clientes são tratados como **processos leves**. Observe que isto restringe fortemente o que um determinado usuário pode, de fato, fazer a partir de seu cliente.

Outro ponto que merece ser observado é que, a cada requisição satisfeita (isto é, após a validação do cliente), uma nova *thread* surge para ser tratada pelo servidor. A política de atendimento destes processos serve-se de um escalonamento estabelecido pela JVM. No entanto, uma política de prioridade das *threads*, quando suportada pelo SO nativo, pode ajustar os critérios de resposta dado pelo servidor[113].

Por outro lado, pode parecer interessante verificar como o processamento no *cluster* é feito internamente. Antes de mais nada, os clientes enviam apenas **comandos remotos**, com as devidas formatações, a fim de indicar o que deve ser processado e como deve ser executado (conforme sucintamente explicado na metalinguagem de comunicação usada no modelo C/S do protótipo). Cabe ao servidor obter tais informações e iniciar a execução de tais programas.

Deve-se notar que tais processamentos não podem se constituir em *threads*, pois indicam, de fato, processos "de peso" executados em `PVM` ou `MPI` no sistema. A

---

[112] Em `Java`, existem duas formas de se criar uma *thread*: uma é estender a classe `Thread` original (como é feito) e a outra é implementar a interface `Runnable`. As diferenças são sutis e dizem respeito aos métodos a serem herdados e à forma como a execução deve ser mantida (cf. SRIDHARAN, 1997**)**.

[113] As prioridades em Java podem variar de 1 (`MIN_PRIORITY`) a 10 (`MAX_PRIORITY`), sendo o valor normal igual a 5 (`NORM_PRIORITY`). Note que tais valores são consideravelmente limitados, quando comparamos com a grande faixa de prioridades de processos no `UNIX` (que vão de 0 até por volta de 65.000).

solução para isto, consiste na invocação direta da classe `Process`, interfaceada pela superclasse `System` da linguagem. Isto ocorre transparentemente para o cliente e o controle reside no servidor, de forma que somente este (sob a ordem daquele) pode executar alguma alteração no processamento. Vale lembrar que isto somente é possível devido à restrição anteriormente mencionada de que o servidor existe em uma máquina diretamente participante do *cluster*.

- (ii). Medidas de carga no sistema

O arquivo `workload` é responsável por registrar as medidas de carga no sistema. Tais dados são importantes porque favorecerão a escolha de um determinado *host* para compor uma máquina virtual para o processamento em uma determinada MPL.

Vale observar que a própria medida para obtenção do tempo é um elemento que deve ser cautelosamente analisado. Em sistemas `Unix`, por exemplo, o *shell* possui vários comandos simples para fornecer tais informações sobre o processamento, sejam através de uma medição instantânea, seja através de uma média, tal como ocorre nos sistemas nativos de contabilização de recursos, registrados no `/var/adm/acct`. Dentre os comandos para obtenção de um valor instantâneo da carga, podemos citar o de estatísticas do sistema (`vmstat`) e o verificador de processos (`which` ou `w`).

Em todo caso, na maioria dos sistemas, o tempo é avaliado através de três parâmetros distintos: o *user time* (**ut**), o *system time* (**st**) e o *elapsed* ou *iddle time* (**et**). O primeiro representa o tempo efetivo em que o processo ocupou a CPU; já o segundo fator, indica o tempo consumido pelo sistema operacional devido ao programa (com o uso de recursos como paginação, por exemplo) e, finalmente, o terceiro parâmetro computa todo o tempo, envolvendo os *delays* de entrada e saída ignorados nas outras duas medidas.

Vale mencionar, finalmente, que a utilização de um registro de *logs* para a gravação das medidas de carga permite elaborar uma análise mais realista para a política de processamento.

Resumo

Neste capítulo, apresentamos o protótipo da ferramenta WVM, incluindo as interfaces do cliente, do servidor e o ambiente de processamento utilizado.

No próximo capítulo, teceremos considerações finais sobre o modelo apresentado com relação aos resultados experimentais e trabalhos futuros relacionados com a ferramenta.

Capítulo 7

Conclusão

Introdução

Neste capítulo, apresentaremos alguns resultados sobre o funcionamento da ferramenta no ambiente de processamento do CENAPAD-NE. Faremos ainda observações sobre o estado atual e os trabalhos futuros referentes ao modelo apresentado ao longo da dissertação.

7.1. Resultados

7.1.1 – Ambiente Experimental Utilizado

Nos capítulos anteriores, mencionamos detalhes sobre o funcionamento da arquitetura, no que diz respeito aos componentes cliente e servidor. Para mencionarmos como o protótipo em execução se encontra, resta tecer considerações sobre o *site* de aplicações e o domínio de processamento que, conforme mencionado na seção 5.1.1, foram unificados no que chamamos de ambiente de processamento, e que será descrito a seguir.

- A. Ambiente de Processamento

O item mais importante, aqui mencionado, diz respeito ao ambiente de processamento e ao *site* de aplicações. Estes elementos externos foram unificados no ambiente do Centro Nacional de Processamento de Alto Desempenho no Nordeste (CENAPAD-NE), onde existem diferentes recursos computacionais disponíveis para processamento de alto desempenho. A organização atual do Centro é mostrada na figura 7.1.

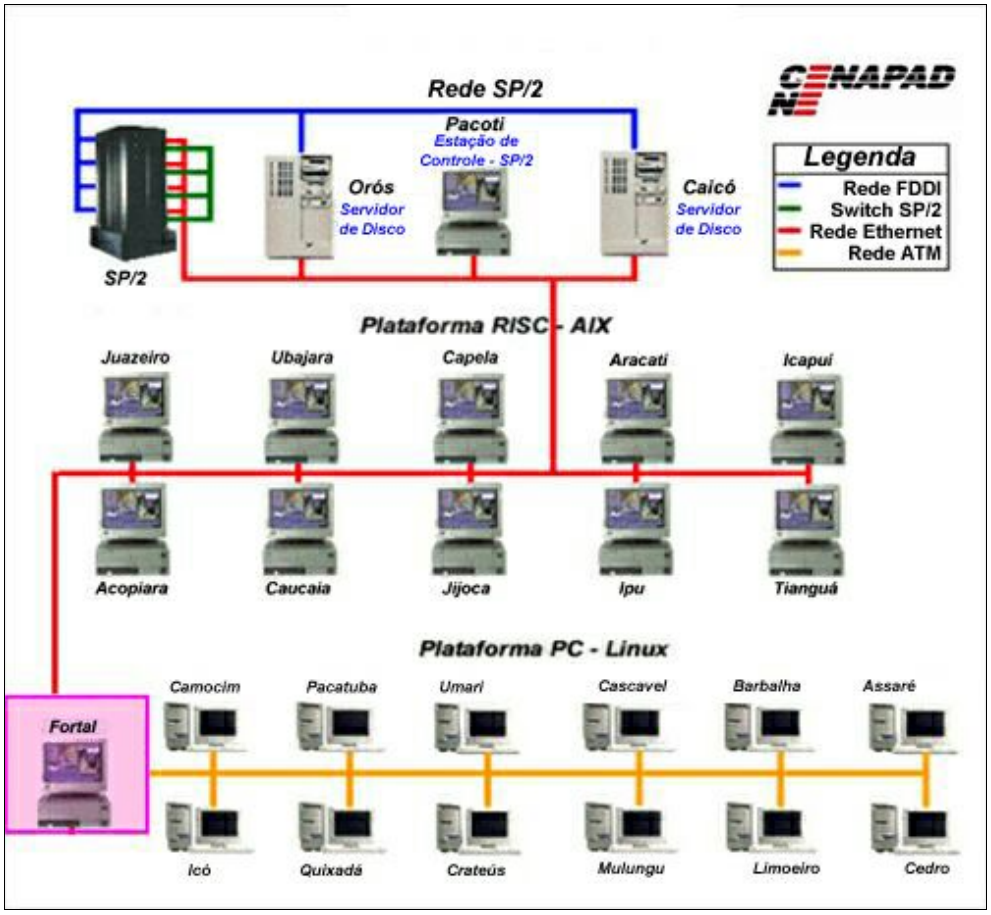


Figura 7.1: Ambiente de Processamento no CENAPAD-NE

- (i). Ambiente SP/2

Constituído por um computador IBM SP/2 (*Superscalable Power Parallel System*) modelo 9076 com 4 nós, cada um com 256MB de RAM, 1GB de espaço de paginação e capacidade de processamento total superior a 1Gflop. As várias conexões existentes no sistema apresentam taxas de transferência variadas:

- **Rede Switch:** Comutador de alta velocidade de 320 MB/s;
- **Rede FDDI:** Fibra ótica conectando o sistema SP/2 com os servidores de disco a 100MB/s;
- **Rede Ethernet:** Rede de difusão a 10MB/s.

Além disso, o ambiente SP/2 conta com 2 servidoras de arquivos modelo 590 com capacidade de 20GB de disco cada e 128MB de memória, além de uma estação de controle para o sistema. Todas as máquinas com sistema operacional `AIX` e conectados via *ethernet*.

- (ii). *Cluster* RISC

Formado por 10 estações de trabalho RISC IBM modelo 250, cada uma com 1GB de disco, sistema operacional `AIX` e conectadas em rede *ethernet*.

- (iii). *Cluster* ATM

  Constituído por 12 PCs rodando `Linux` conectados a uma máquina servidora F40 e um comutador (*switch*) ATM, também disponíveis para utilização. A conexão é feita através de 2 protocolos de emulação: LANE (*LAN Emulation*) e CIP (*Classic IP*). O primeiro, destinado a manter a compatibilidade com os demais membros da rede metropolitana e o segundo, a diminuir a pilha de protocolos, visando um desempenho melhor[114].

Conforme anteriormente dito, outros ambientes de processamento (além do `CENAPAD-NE`) podem ser anexados à configuração dos usuários, desde que estes possuam acesso à *Web* e suporte à JVM.

♦ B. Exemplos de Aplicações

Algumas aplicações foram testadas com a utilização da ferramenta. No entanto, vale destacar algumas observações, antes de relatar resultados:

- Conforme mencionado na 5.1.2.B, os programas devem se apoiar nas seguintes plataformas: linguagens `C` ou `Fortran` e bibliotecas MPLs como `PVM` ou `MPI`. A utilização de `Java` também é possível, atendo-se ao detalhe de que, neste caso, o modelo atual interagirá apenas com o servidor;
- A não interatividade com o usuário se torna necessária, por dois motivos: primeiramente, uma vez que os comandos são executados indiretamente, através de chamadas ao sistema feitas pelo servidor WVM, não é possível interagir diretamente com as máquinas do *cluster*. Em segundo lugar, a interface única apresentada apresenta condições para transmitir parâmetros (que devem ser passados cautelosamente ao programa) e obter resultados (inclusive com interpretação gráfica);
- A interface do programa no formato de *applet* exige, nos parâmetros atuais de configuração das JVMs instaladas nos *browsers*, que um procedimento de validação seja feito, a fim de permitir a submissão de tarefas remotas. Detalhes sobre os passos para a execução da assinatura, são descritos tanto

[114] A utilização de LANE utiliza um endereçamento particular para as máquinas, onde os 32 bits ATM são mapeados para 4 octetos da forma `100.X.X.X`. Já o CIP utiliza um endereçamento similar ao utilizado na rede convencional do sistema no `CENAPAD-NE` (domínio `200.19.191.Y`). Maiores informações sobre o funcionamento da RMAV pode ser encontrado em `http://www.cenapadne.br/rmav`.

na documentação JDK da Sun[115] e as informações constantes no apêndice C.5.

- Fora as considerações anteriores, não existem requisitos **intrínsecos** da parte da aplicação para que possa ser utilizado com a ferramenta WVM, isto é, não há dependências ou vinculações com outras eventuais bibliotecas do sistema, nem mesmo para adaptação de códigos fontes, já que apenas classes do próprio JDK foram utilizadas (cf. item 4.2.1.A). Deve-se salientar contudo que, apesar da não dependência relativa aos pacotes, certas funcionalidades do sistema necessitam da invocação explícita de diretivas (como "GRAFDAT" para ajustar a formatação dos dados na plotagem gráfica).

Considerando-se os pontos acima, percebe-se que não existem fortes restrições a fim de que o desenvolvimento de um dado programa possa ser feito se valendo da ferramenta proposta. Como exemplo, citamos a seguir, duas aplicações feitas por usuários do sistema no CENAPAD-NE.

➢ (i). APLICAÇÃO NATIVA: Combinatória utilizando *clique de grafo*

Problemas de explosão combinatória são bastante comuns na literatura e englobam uma enorme gama de aplicações (cf. [Garey, 1979], [Cormen, 1996] e [Atallah, 1999]). No nosso caso, enfocaremos aqui as questões relacionadas com um problema clássico em Teoria dos Grafos, conhecido como **clique máxima**. A curiosidade sobre esta primeira aplicação é que ela foi totalmente desenvolvida utilizando-se das ferramentas apresentadas. Vale a pena observar que, posteriormente, o programa foi utilizado **fora** do ambiente WVM sem quaisquer dificuldades.

Uma descrição rápida do problema consiste no seguinte: dado um grafo qualquer G=(V,E), pretende-se encontrar o maior subgrafo completo, isto é, o maior conjunto G'=(V',E'), com V' $\subset$ V e E' $\subset$ E, tal que para quaisquer vértices v1,v2 $\in$ V', existe uma aresta e $\in$ E'. O problema pertence à categoria NP de complexidade, tendo sido um dos primeiros problemas a serem reduzidos ao clássico SAT (cf. [Garey, *ibid.*]). O interesse por esta aplicação decorre de dois fatores:

- A alta demanda de processamento, resultante da explosão combinatória para os dados;
- A independência dos subconjuntos, que podem ser facilmente tratados em nível de decomposição de domínio (cf. item 2.2.2.B);

[115] Confira, por exemplo, http:// java.sun.com/ docs/ books/ tutorial/ applet/ practical/ security.html.

- Finalmente a possibilidade de plotagem estática do conjunto de dados (facilmente plotada na tela).

Uma observação importante é a de que nosso objetivo aqui não consiste em propor melhorias ou técnicas para a resolução da **clique máxima**, mas em aproveitar a natureza clássica do problema e utiliza-lo como experimento piloto. A seguir, ilustramos as principais etapas utilizadas no desenvolvimento e na execução da aplicação:

Etapa I: Construção do programa:

(1) Nesta etapa, foram feitas várias versões da aplicação, com a codificação dos dados em `C` e bibliotecas MPLs como `PVM` ou `MPI`.

(2) Criação da base de dados de teste: foram sucessivamente criados arquivos para a entrada de dados com tamanhos de 300, 1000, 3000 e 11000 vértices (sendo este último correspondente ao valor máximo para a carga no sistema SP-2).

Etapa II: Execução e resultados:

(3) A execução dos programas foi feita a nível comparativo com o programa seqüencial e paralelo para 2, 3 e 4 nós processadores (todos pertencentes ao SP-2).

(4) Os resultados comparativos, em termos de tempo de processamento (em segundos) são mostrados na tabela 7.1:

P \ |V|	100	300	1000	3000	10000
1	1.4	20	400	9100-	25340-
2	1.4	22	250	6003-	19300-
3	2	18	127	4403	12340-
4	2.4	14	87	2941	8230-

Tabela 7.1: Resultados da aplicação Clique

➢ (ii). APLICAÇÃO ADAPTADA: RNA utilizando *backpropagation*

Uma segunda aplicação testada, seguiu uma etapa inversa: desenvolvida por meio da programação direta (com múltiplas seções `telnet`, transferência de dados via `ftp` e visualização externa de resultados), passou-se em seguida à utilização da ferramenta para verificação da adequação da mesma.

A utilização de Redes Neurais Artificiais (RNA's) baseia-se na idéia de que muitos problemas computacionais se utilizam, de um modo geral, de algum tipo de solução algoritmica basicamente seqüencial, o que pode se tornar complexo ao se trabalhar com uma grande quantidade de dados em um contexto restrito. O estudo das

RNA's, por sua vez, tendo como modelo do cerebro humano, vem surgindo como alternativa para a solução dos problemas tratados de forma ineficiente pelo enfoque algoritmico, pois se baseiam em algumas caracteristicas comportamentais importantes da rede neural biologica, entre elas a capacidade de aprender e a generalizar a partir de estimulos a ela apresentados, diminuindo consideravelmente a quantidade de dados a ser analisada (referências completas podem ser encontradas em [Haykin, 1991]).

Dentre os vários algoritmos existentes na literatura, testou-se a utilização do *backpropagation*, que constitui num dos métodos mais utilizados para treinamento das RNAs. Como, em muitas situações, seu tempo de treinamento se torna inaceitável, mesmo com a utilização de PCs e estações de trabalho poderosos, o objetivo é contornar este problema com a utilização máquinas paralelas e/ou clusters de computadores para acelerar os cálculos envolvidos. Dos vários resultados apresentados por [Melcíades, 1999] resultam de uma análise dessas questões. Foram suas implementações nos modos *batch* (onde cada processador sempre utiliza os mesmos pesos para as chamadas "**sinapses**" da rede, havendo portanto, considerável comunicação) e *block* (no qual, por sua vez, somente após a execução de certos "blocos" de cálculo, é que dados são enviados ao mestre no paradigma mestre-escravo).

O interesse pelo estudo do caso estava basicamente sujeito à:

- Pré-existência da aplicação;
- Fácil ajuste para incorporação no sistema.

Disto, pode-se antecipar as etapas seguidas para os testes:

Etapa I: Construção do programa:

(1) A construção do programa (feito em C) e se utilizando de PVM foi feita sem o uso na ferramenta, de forma que essa parte foi ignorada.

(2) Apenas no tocante à execução não-interativa e à exibição dos dados, foram ajustadas linhas de código para acionamento das diretivas ("GRAFDAT"), a fim de serem visualizadas.

Etapa II: Execução e resultados:

(3) A execução dos programas, já realizada antes, foi feita a nível comparativo com as várias implementações do programa seqüencial e paralelo (relativos aos nós do SP-2).

(4) Os resultados comparativos são mostrados na tabela 7.2, ilustrando o tempo de processamento (em segundos), ao se utilizar uma quantidade variável de processadores:

P \ RNA	1	2	3	4

Batch	-	320	188	152
Block 1/2	-	385	220	215
Block 1/3	-	467	292	230
Block 1/4	-	564	345	278
OnLine	368	-	-	-
Batch seq.	331	-	-	-

Tabela 7.2: Resultados da RNA (cf. [Melcíades, 1999])

7.1.2 – Resultados Práticos

Finalmente, como resultado da utilização da ferramenta, podem ser citadas as seguintes considerações finais:

- A. Integração e Transparência

  A ferramenta facilita para o usuário a tarefa de desenvolvimento e execução das aplicações no *cluster*, dando maior transparência às atividades, aproveitando-se das facilidades de utilização remota da rede (através da integração de serviços para transferência de dados e execução de processos) para auxiliar na programação distribuída feita em *clusters*.

- B. Facilidade

  A utilização de aplicações pré-existentes ou feitas na ferramenta, tornou-se possível, com o adequado ajuste dos códigos. Porém, o fato de aplicações podem ser feitas internamente com a utilização da ferramenta e com maior flexibilidade, constitui uma facilidade adicional.

- C. Flexibilidade

  Ajustes no código podem ser feitas sem levar em conta questões de biblioteca particulares, permitindo um ajuste mais flexível da ferramenta ao ambiente de execução.

Resumidamente, os resultados do projeto podem permitir, de um lado, facilitar as etapas de programação e, de outro, fornecer subsídios para avaliações comparativas entre as diferentes alternativas para o processamento, como o uso de bibliotecas de passagem de mensagens (`PVM` ou `MPI`) ou mesmo a utilização de `Java`. Deve-se, contudo, constatar alguns problemas encontrados:

- D. Adaptação dos códigos e ambientes

  Conforme anteriormente mencionado, a necessidade de adaptar códigos para a execução de certas atividades representa uma necessidade. Embora isso não implique na utilização de novas bibliotecas, requer a interferência direta do usuário, o que não é feito, por exemplo, dentro do contexto da programação automática (cf. item 2.1.2.B) ou

no contexto da metacomputação (item 3.2.1.A). Além disso, para efeitos práticos, uma movimentação dos códigos deve ser feita para as localizações adequadas onde o servidor WVM pode visualizar os arquivos.

- E. Fluxos de dados

  Um problema mais grave foi percebido quando da utilização de intensa saídas textuais para visualização de dados. Isso porque boa parte do tráfego passava a ser percebida em dois caminhos: dos nós processadores ao servidor WVM e deste ao cliente, onde os dados eram efetivamente mostrados.

- F. Limitações na quantidade de clientes

  Nos testes realizados, o atendimento às *threads* dos clientes no servidor WVM, tornava-se bastante limitado pela própria natureza dos chamados "processos leves" (cf. item 6.3.1.C). Constatou-se que múltiplas requisições podiam causar problemas tanto na gerência dos processos quanto na resposta aos clientes.

- G. Eficiência

  Uma questão de particular interesse reside na influência de uma camada intermediária para interpretação dos comandos (o *middleware*, apresentado no item 5.2.1.B) impreterivelmente afetava surge como um elemento a mais entre as camadas de aplicação e transporte. No entanto, vale ressaltar que esta influência não atua no processamento direto no *cluster*, mas apenas na interface entre oos clientes e o servidor WVM e no processamento inicial encabeçado por este último.

7.2. Trabalhos Futuros

A utilização de WVM possui como finalidade básica prover recursos para viabilizar a computação paralela, utilizando bibliotecas de MPL em *cluster*. No entanto, uma outra possibilidade se abre às tendências do processamento distribuído: a utilização da metacomputação, melhor pormenorizado na seção 3.2. Enquadramos essa nova possibilidade nesta seção de trabalhos futuros, dando os detalhes de como isso pode ser feito.

7.2.1 – WVM como Ferramenta para Metacomputação

O modelo de metacomputação, tal como apresentado no item 3.2, se constitui numa tendência com grandes possibilidades de aplicação prática. Nesta seção, o que mencionamos são as possíveis implicações do uso deste paradigma, a partir da proposta atual.

- A. Descrição do Modelo

As fases de conexão e preparação se processam de forma análoga ao descrito no capítulo 5. No entanto, o processamento ocorre não mais no domínio de PAD, mas nos clientes habilitados e disponíveis para a metacomputação que recebem do servidor as informações sobre o código Java a ser executado (6) e para ele retornam os resultados (7). Somente então, é que este envia as informações para o cliente original que requisitou a tarefa (8).

A visão do projeto identifica duas condições para que um dado cliente WVM possa participar da metacomputação: a primeira é a de estar **habilitado**, isto é, disponível para este tipo de processamento (que é a configuração *default*). No entanto, o usuário pode desativar essa capacidade a qualquer momento, através dos comandos do ambiente MC. A segunda condição é a de que ele esteja disponível, em outras palavras, não já esteja participando de nenhuma outra metacomputação. Para evitar sobrecargas nas *applets*, um cliente somente possui uma *thread* (canal) para gerenciar esta atividade. Na figura 7.2, por exemplo, os clientes 2 e 3 estão passíveis de serem utilizados pelo cliente 1, enquanto o cliente 4 se encontra desabilitado.

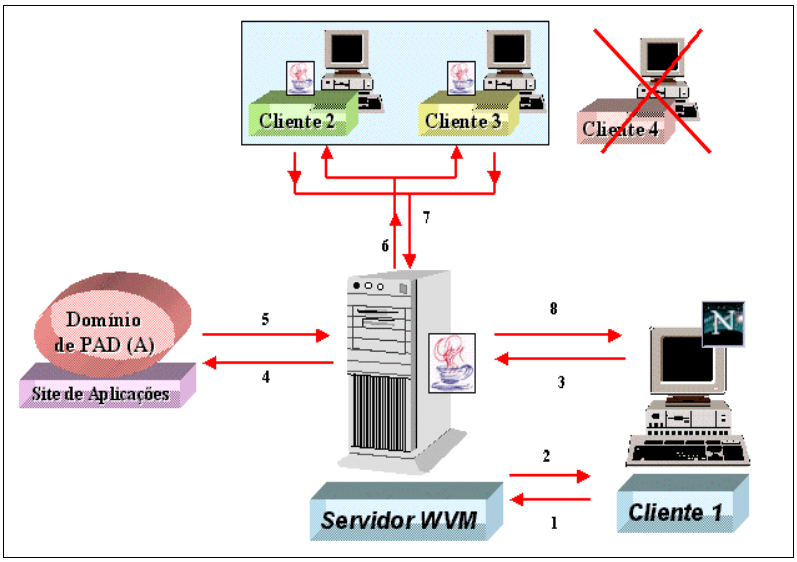


Figura 7.2: Metacomputação em WVM

Ainda que não haja possibilidade de conexão com os demais clientes, o usuário pode ainda fazer o *download* do código e processar a aplicação Java em sua própria máquina. Ainda que isto não represente um ganho de processamento, a capacidade de transferência das aplicações do servidor para qualquer cliente trazem como benefício uma independência quanto a localização do cliente.

Como resultado final desta organização de arquitetura, a cada momento, a rede pode possuir vários núcleos de processamento virtuais, além dos domínios de processamento ativos e disponíveis para PAD. A figura 7.3 mostra um modelo esquemático de como os clientes e *hosts* participantes de WVM podem se apresentar em um dado instante.

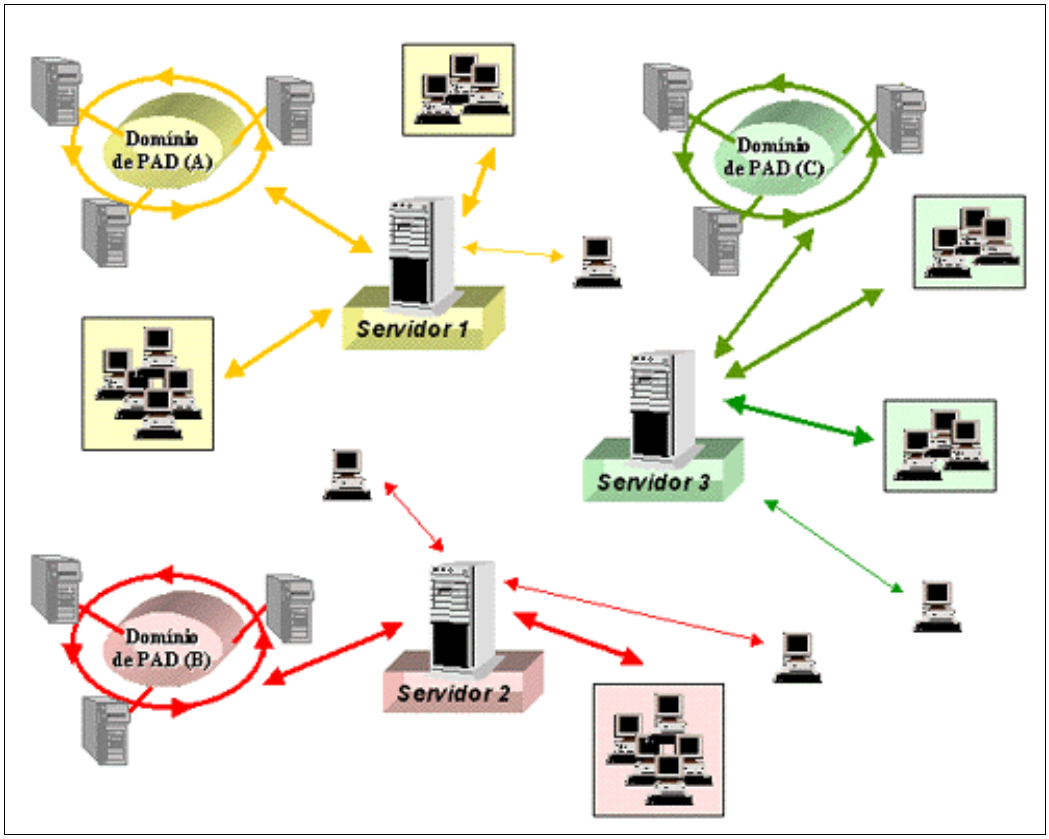


Figura 7.3: Diagrama Esquemático de uma Rede WVM

Nas próximas seções, vamos indicar as modificações de cada componente da arquitetura, que permitirão implementar o modelo funcional anteriormente descrito.

♦ B. Metacomputação no Cliente

O cliente WVM terá a necessidade de introduzir um novo componente, cuja estrutura é apresentada na figura 7.4:

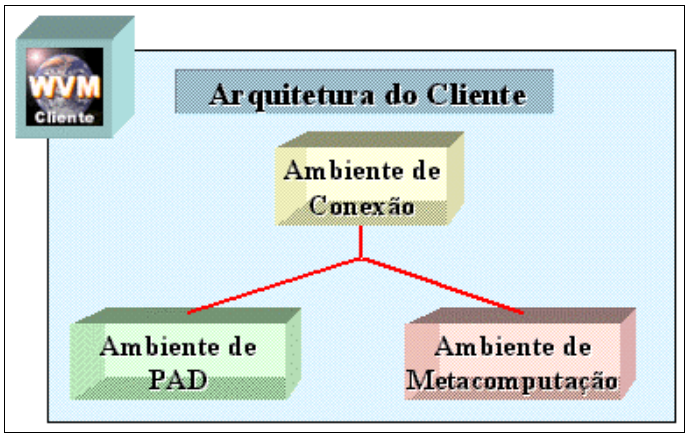


Figura 7.4: Componentes do Cliente WVM para suporte a MC

O ambiente de metacomputação é análogo ao sistema de PAD, pelas funcionalidades que apresenta, desta vez voltando-se para o processamento efetuado via

Internet, em outros clientes habilitados. Este ambiente apresenta três módulos: **configuração**, **editoração** e **comando** (figura 7.5). Somente neste último módulo pode-se fazer uso dos clientes para realizar o processamento.

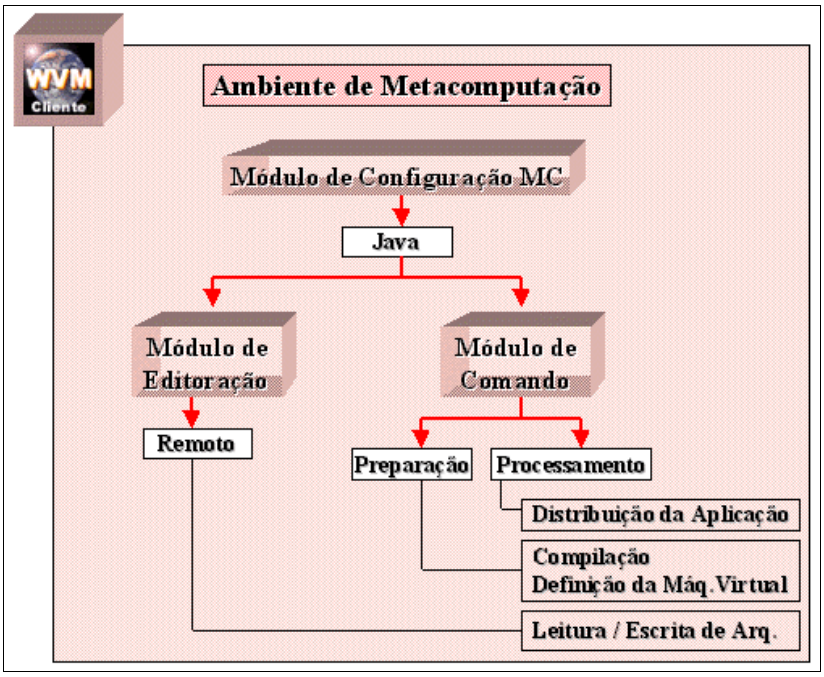


Figura 7.5: Organização do Ambiente de MC no Cliente

Note que a configuração não passa pela definição de uma biblioteca paralela (como `MPI` ou `PVM`), mas pela identificação da linguagem (`Java`), a partir do qual podem ser preparados arquivos ou executados comandos. Algumas diferenças devem ser novamente observadas: a menos que a *applet* seja assinada, o salvamento de arquivos é feito apenas remotamente (no servidor). Em segundo lugar, a execução da aplicação é melhor definida na atividade de *distribuição* do código entre os demais clientes participantes do processamento distribuído.

Algumas observações importantes devem ser salientadas:

- A edição dos arquivos não pode ser feita aleatoriamente em qualquer cliente. De fato, somente aquela aplicação requerente do processamento no ambiente MC, pode, de fato, ter acesso ao código fonte da aplicação Java. Os demais clientes, envolvidos no ambiente somente servem para prover recurso computacional para processamento distribuído.
- As aplicações (da mesma forma como ocorre com o `PVM` e o `MPI`) necessitam ser adequadamente planejadas e implementadas, a fim de se valerem das vantagens da computação distribuída. Note que problemas de comunicação são tratados de uma forma distinta, em relação ao modelo *message-passing*, já que soluções alternativas

como RMI é que são utilizadas para o envio e a formatação de parâmetros e métodos.

- Cabe ressaltar um problema relativo às restrições de segurança: como um cliente somente se comunica com o servidor, o processamento cliente-cliente (como o modelo MC dá a entender) deve se valer de fases distintas. Com essa alternativa, a presença do servidor serve tanto para cadastrar clientes disponíveis, quanto para enviar comandos de processamento e obter resposta.
- Além da preocupação sobre a natureza da aplicação e a tecnologia utilizadas, deve-se notar que outras questões, como aquelas relativas à granulosidade das tarefas, balanceamento de carga e escalabilidade (seção 2.2.4), devem ser igualmente analisadas.

Vale a pena observar que isso resulta em uma extensão da interface do cliente. A tabela 7.3 reúne características anteriormente mostradas (tabela 6.3) com novos elementos da arquitetura funcional voltada para o ambiente MC[116].

Ambiente	Estrutura Funcional	Componente(s) da Interface
Conexão	Módulo de Certificação	Painel de *Login*
	Módulo de Configuração do Usuário	Painel de Identificação
PAD	Módulo de Configuração MPL	Painel de Edição Remota
	Módulo de Editoração	Painel de Edição Remota, Painel de Edição Local, Área de Edição
	Módulo de Comando	Painel de Comando
MC	Módulo de Configuração MC	Painel de Edição Remota
	Módulo de Editoração	Painel de Edição Remota, Painel de Edição Local, Área de Edição
	Módulo de Comando	Painel de Comando

Tabela 7.3: Relação entre Arquitetura Funcional e Interface no Cliente para MC

♦ C. Metacomputação no Servidor

Vale lembrar que o servidor WVM é constituído por dois ambientes: o de Administração Local e o de Administração Remota. O ambiente de metacomputação irá introduzir mudança neste segundo ambiente, conforme identificado na figura 7.6.

No módulo de controle do ambiente MC, os clientes que se conectam ao sistema e que estão disponíveis para metacomputação, são monitorados. A gerência das aplicações disponíveis para execução remota nas *applets* é feita através de *threads* próprias que devem ser continuamente monitoradas, a fim de permitir o envio das informações aos usuários interessados.

[116] Os elementos gráficos se utilizam dos componentes anteriormente mostrados. O que muda, no entanto, é o objetivo ou a utilização destes mesmos elementos.

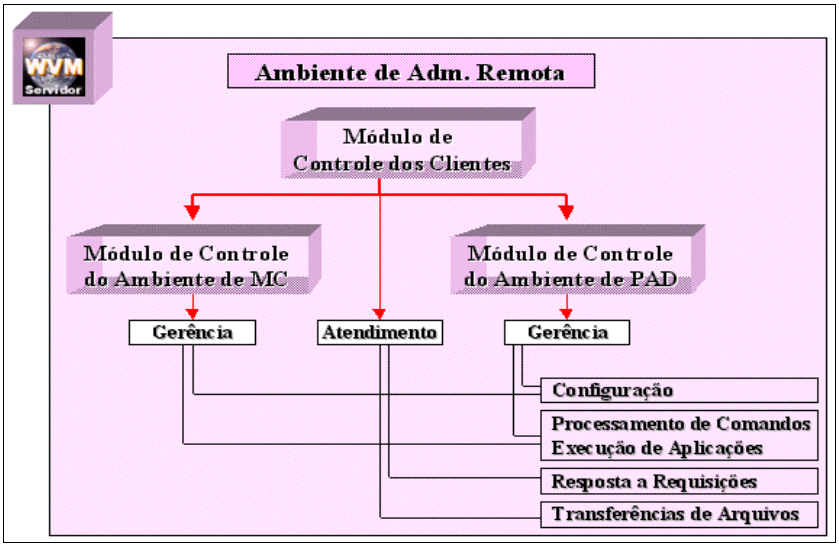


Figura 7.6: Organização do Ambiente de Adm. Local no Servidor

7.2.2 – WVM como ferramenta

O objetivo final da proposta WVM, conforme várias vezes mencionado, é fornecer uma alternativa para auxiliar no processamento distribuído em *clusters*, provendo certas facilidades. Uma das formas de se alcançar este objetivo seria, enfim, o de consolidar a ferramenta no atual ambiente constituído pelos Centros de Processamento (CENAPADs), mencionados na seção 2.1.2. Os resultados e testes obtidos permitiram corrigir e otimizar alguns recursos, que resultaram em melhorias e correções feitas constam no texto dessa dissertação[117]. Vale, contudo, mencionar que, para a utilização de cada ambiente de processamento, deve haver um prévio ajuste da ferramenta, a fim de que as entradas para a identificação dos novos *clusters* sejam fornecidas.

Espera-se, finalmente, que com a consolidação das futuras políticas e infra-estruturas para processamento (vide os comentários o modelo brasileiro, em especial a SOCINFO, no capítulo 2), seja possível prever novas alternativas, como a WVM, que venham a aproximar, ainda mais, o paradigma da computação paralela distribuída, a uma quantidade cada vez maior de usuários.

Resumo

---

[117] Visando isso, foram observadas algumas críticas e comentários resultantes da apresentação da ferramenta durante o 11º Simpósio Brasileiro de Arquitetura de Computadores e Computação de Alto Desempenho – SBAC-PAD'99 (SOUSA & FIALLOS, 1997).

Neste último capítulo, apresentamos algumas consideraç!oes sobre os resultados e fornecemos indicação das possíveis modificações que podem ser feitas na proposta atual, a fim de dar continuidade aos trabalhos iniciados.

Vale concluir afirmando que, num contexto tecnológico tão dinâmico quanto o atual, a apresentação de novas alternativas para o processamento paralelo-distribuído podem iluminar os caminhos que, num futuro bem próximo, serão responsáveis por alterações ainda mais profundas na sociedade moderna.

REFERÊNCIAS BIBLIOGRÁFICAS

I: Referências Principais
II: Fontes Auxiliares

I. Referências Principais

[AHO, 1974] - Aho, A.V., Hopcroft, J., Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Massachussets, 1974.

[AMDAHL, 1967] - Amdahl, G.M. "Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability". In: *Proceedings AFIPS Conf.* Pp.483-485. Reston, VA: 1967.

[ATALLAH, 1999] – Atallah, M. *et al*. "Algorithms and Theory of Computation Handbook". CRC Press, USA: 1999.

[BECA, 1997] - Beca, L., Cheng, G., Fox, G.C. *et al*. *Web Technologies for Collaborative Visualization and Simulation*. New York: Syracuse University, 1997.

[BAKER, 1998] - Baker, M. & Buyya, R. *Cluster Computing: The Commodity Supercomputing*. Queensland University of Technology, 1998.

[_____, 1999] – Baker, M. & Fox, G. *Metacomputing: Harnessing Informal Supercomputers*. Syracuse University, 1999.

[BECK, 1999] - Beck, M., Dongarra, J.J., Fagg, G.E. *et al*. *"HARNESS: a next generation distributed virtual machine."* In: FGCS – Future Generation Computer Systems. Vol. 15 – ContentsDirect – Elsevier Science B.V. - October, 1999.

[BARSEKAS, 1989] - Barsekas, D.P., Tsitsiklis, J.N. *Parallel and Distributed Computation – Numerical Methods*. Prentice-Hall International Editions. New Jersey: Syracuse University, 1989.

[BIGUS, 1998] - Bigus, J.P. & Bigus, J. *Constructing Intelligent Agents with Java*. John Wiley & Sons Inc. 1998.

[BROOKSHEAR, 1998] - Brookshear, J.G. *Ciência da Computação: Uma Visão Abrangente*. Bookman Editora - Porto Alegre, 1999.

[BUYYA, 1998] - Buyya, R. *High-Performance Cluster Computing*. Volume 1: "Architecture and Systems". Prentice Hall Publishing. New Jersey, 1999.

[CÁCERES, 2001] – Cáceres, E. N., Mongelli, H., Song, S. W. *Algoritmos Paralelos usando CGM/PVM/MPI: Uma Introdução*. "Anais do XXI Congresso da Sociedade Brasileira de Computação" - Volume 2: XX Jornada de Atualização em Informática. Fortaleza: 2001.

[CASANOVA, 1998] - Casanova, H. e Dongarra, J. *Netsolve: A Network Solver for Solving Computational Science Problems*. Technical Report CS-95-313. University of Tennesse: 1995.

[CATTKETT, 1998] - Cattlett, C. & Smarr, L. "Metacomputing". In: *"Communications of the ACM. Vol. 35*. 1992.

[CENAPAD-NE, 1997] – CENAPAD-NE. *Introdução à Programação Paralela com PVM – Manual do Curso.* Fortaleza: Centro Nacional de Processamento de Alto Desempenho no Nordeste, 1997.

[_____, 1998] - CENAPAD-NE. *Programação Paralela com MPI – Manual do Curso.* Fortaleza: Centro Nacional de Processamento de Alto Desempenho no Nordeste, 1998.

[_____, 1999] – CENAPAD-NE. *Treinamento em Tecnologias da Computação - Java – Manual do Curso.* Fortaleza: Centro Nacional de Processamento de Alto Desempenho no Nordeste, 1999.

[CHANG, 1996] - Chang, Y., Carpenter, B. e Fox, G. *"MPI Java Wrapper Implementation"* – New York: Syracuse University, Fev. 1996

[CHENN, 1997a] – Chen, Z. *"An Architecture to Support Collaborative Distributed Heterogeneous Computing Applications"* – Old Dominion University, Março 1997.

[_____, 1997b] - Chen, Z., Maly, K., Mehrotra, P. *et al*. *"Web Based Framework for Distributed Computing"* – Old Dominion University, 1997.

[CHRISTIANSEN, 1997] - Christiansen, B.O., Cappello, P., Ionescu, M. F. *et al*. *"Javelin: Internet-Based Parallel Computing Using Java"*. Califórnia: University of Californa, 1997.

[CORMEN, 1996] - Cormen, T.H., Leiserson, C.E., Rivest, R.L. *"Introduction to Algorithms"*. MIT Press, 1996.

[CULLER, 1993] - Culler, D.E, Karp, R.M., Patterson, D. *et al*. "Toward a Realistic Model of Parallel Computation". In: *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Pp.1-12. Maio: 1993.

[DEITEL, 2001] - Deitel, H.M. & Deitel, P.J. *"Java, como programar"*. 3ª edição. Porto Alegre, Janeiro, 2001.

[ECKHOUSE, 1978] - ECKHOUSE. "Panel Discussion on Distributed Processing". In: *National Computer Conference*. Califórnia, Junho, 1978.

[ENSLOW, 1974] - Enslow, P.H. *Multiprocessors and Parallel Processing*. New York: John Wiley & Sons, 1974.

[FARLEY, 1998] - Farley, J. *Java Distributed Computing*. California: O'Reilly & Associates, 1998.

[FISCHER, 1994] - Fischer, M. e Dongarra, J. *Another Architecture: PVM on Windows 95/NT*, 1994.

[FLANAGAN, 1997] - Flanagan, D. *Java in a Nutshell*. 2ª ed. Sebastopol, CA: O'Reilly Associates, 1997.

[FLYNN, 1966] - Flynn, M.J. "Very High Speed Computing Systems". In: *IEEE Proceedings* – 54:12: Dezembro, 1966.

[FORTUNE, 1978] - Fortune, S., Wyllie, J. "Parallellism in Random Access Machine" In: *ACM Symp. – Theory of Computing*, 1978.

[FOSTER, 1995] - Foster, I.T. *Designed and Building Parallel Programs – Concepts ant Tools for Parallel Software Engineering* – Addison Wesley Publishing – USA, 1995.

[_____, 1997] - Foster, I.T. & Globus, C.K. "A Metacomputer Infrastructure Toolkit" In: *Internet Journal of Supercomputer Applications*, 1997.

[_____, 1998] - Foster, I.T. & Kesselman, C. "The Globus Project. A Status Report" In: *Proceeding IPPS/SPDP'98 Heterogeneous Computing Workshop*, 1998.

[_____, 1999] - Foster, I.T. "*The Grid: Blueprint for a New Computing Infrastructure*". Morgan Kauffmann Publishers Inc. – USA, 1999.

[FOX, 1996] - Fox, G.C. *"An application pespective on high-performance computiong and communications*". Technical Report SCCS-757, Syracuse, NY: Abril, 1996.

[_____, 1997a] - Fox, G.C. *High Performance Distributed Computing*. New York: Syracuse University, 1997.

[_____, 1997b] - Fox, G.C. e Furmanski, W. *"Java for Parallel Computing and as a General Language for Scientific and Engineering Simulation and Modelling"* . New York: Syracuse University, 1997.

[_____, 1997c] - Fox, G.C. *"Java and Web Technologies for Simulation and Modelling in Computacional Science and Engineering"*. New York: Syracuse University, 1997.

[GAREY, 1979] - Garey, M. R., and Johnson, D. S. *Computers and Intractability, "A Guide to the Theory of NP-completeness*". W H Freeman & Co. Kauffman. S. Francisco, 1979.

[GEIST, 1993] - Geist, G.A. *et al. PVM3 User's Guide and Reference Manual – Technical Report. ORNL/TM-12187*, Oak Ridge National Laboratory, Tennesse, USA, 1993.

[_____, 1994a] - Geist, G.A., Beguelin, A. *et al. PVM: Parallel Virtual Machine*, MIT Press, 1994.

[_____, 1994b] - Geist, G.A., Kohl, J.A. e Papadopoulos, P.M. *PVM and MPI: a Comparasion of Features.* MIT Press, 1994.

[GENTZSCH, 1999] - Gentzsch, W. *"Metacomputing: from workstation clusters to Internet computing" – Guest Editorial*. In: FGCS – Future Generation Computer Systems. Vol. 15 – ContentsDirect – Elsevier Science B.V. - October, 1999.

[GOLDSTINE, 1963] - Goldstine, S.C. & Neumann, J. von. **"**On the Principles of Large-Scale Computing Machines". In: *Collected Works of John von Neumann* - Pergamon New York, 1963.

[GOSLING, 1996a] - Gosling, J., Yellin, F. *et al*. *The Java Programming Language*. Addison-Wesley Publishing Company, 1996.

[_____, 1996b] – Gosling, J., Joy, B. e Steele, G. *The Java Language Specification*. Massachusets: Addison-Wesley Publishing Company, 1996.

[GRAY, 1997] - Gray, P. A. & Sunderam, V. S. *"IceT: Distributed Computing and Java"*. Atlanta – GA: Emory University, 1997.

[GRIMSHAW, 1997] - Grimshaw, A., Wulf, W. *et al*. **"**The Legion Vision of a Worldwide Virtual Computer". In: *Communications of the ACM*. Vol 40 - Jan. 1997.

[GROPP, 1994] - Gropp, W., Lusk, E. e Skjellum, A. *Using MPI, Portable Parallel Programming with the Message-Passing Interface*. MPI Press. 1994.

[HAROLD, 1997] - Harold, E.R. *"Java Networking Programming"*. Sebastopol, CA: O'Really Associates, 1997.

[HARIRI, 1998] - Hariri, S. & Raghavendra, C. S. "Cluster Computing" – Special Issue on *"High Performance Distributed Computing"* – Baltzer Science Publishers – Volume I (1998) No. I – Published May 1998.

[HARRISON, 1995] - Harrison, C.G. *et al*. "Mobile Agents: Are they a good idea?". *Research Report – IBM Research Division*.- Watson Research Center, NY: Março, 1995.

[HAUPT, 1999] - Haupt, T., Akarsu, E., Fox, G. & Furmansky, W. **"**Web based metacomputing". In: FGCS – Future Generation Computer Systems. Vol. 15 – ContentsDirect – Elsevier Science B.V. - October, 1999.

[HAYKIN, 1991] - Haykin, S. **"***Neural Networks. A Comprehensive Foundation*". Macmilian College Publishing Company - 1991.

[HSIEH, 1997] - Hsieh, C.A., Conte, M.T. *et al*. "Optimizing NET Compilets for Improved Java Performance". In: *IEEE Communications - Computational & Engineering*. Junho, 1997.

[HWANG, 1993] - Hwang, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. New York: McGrawHill, 1993.

[_____, 1998] - Hwang, K., Xu, Z. *Scalable Parallel Computing – Technology, Architecture, Programming*. WCB McGrawHill, 1998.

[JÁJÁ, 1993] - Jájá, J. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company. Massachusetts, 1993.

[JAWORSKI, 1998] - Jaworski, J. *Java 1.2 Unleashed*. Sams Publishing. USA - August, 1998.

[_____, 1999] - Jaworski, J. *Java 2 - Certification*. New Riders Publishing Company. Indiana - July, 1999.

[KIRNER, 1989] - Kirner, C. "Sistemas operacionais para ambientes paralelos". In: *IX Congresso da Sociedade Brasileira de Computação / VII Jornada de Atualização em Informática*, São Carlos – SP, 1989.

[KUNG, 1978] - Kung, H.T. & Leiseron, C. E. *Systolic Arrays (for VLSI)": Sparce Matrix Proceedings*. Philadelphia: SIAM, 1978.

[LAUDON, 1999] - Laudon, Kenneth C., Laudon, Jane P.: *Sistemas de Informação (4ª edição)*. Livros Técnicos Científicos, RJ, 1999.

[LONG, 1994] - Long, N., Larry E.: *Introduction to Computers and Information Systems*. Prentice Hall, 1994.

[MANGIONE, 1998] - Mangione, C. "Performance tests show Java as fast as C++". In: *Javaworld Technical Report*, Fevereiro, 1998.

[MARCHIORO, 1997] - Marchioro II, T.L. e Landau, R.H. "Web-Based Education in Computational Science and Engineering". In: *IEEE Communications*, Abril-Junho, 1997.

[MARKUS, 1997] - Markus, S., Weerawarana, S. *et al*. *"Scientific Computing via the Web: The Net Pellpack PSE Server"* – Theme Article - IEEE Computational & Engineering, Julho-Setembro, 1997.

[MPI-FORUM, 1993] - MPI Forum. "Document for a standard message-passing interface". In: "*Techincal Report*". University of Tennesseee, 1993.

[MIGLIARDI, 1998] - Migliardi, M. & Sunderam, V. *"Heterogeneous Distributed Virtual Machines in the Harness Metacomputing Framework"* – Emory University, 1998.

[MIGLIARDI, 1999] - Migliardi, M., Schubiger, S., Sunderam, V. *"A Distributed JavaSpace Implementation for Harness"* – NSF ACI-9872167. Emory University, 1999.

[MURDOCCA, 2000] - Murdocca, Milles J. & Heuring, Vicent P. *"Introdução à Arquitetura de Computadores"* – Editora Campus, Rio de Janeiro, 2000.

[ORFALI, 1998] - Orfali, R., Barkey, D. *Client / Server Programming with Java and Corba*. 2ª ed. Canada: John Wiley & Sons, 1998.

[PACHECO, 1997] - Pacheco, P. S. *Parallel Programming with MPI.*. 5ª ed. USA: Morgan Kaufmann Publishers Inc., 1997.

[PATTERSON, 1997] - Patterson, D.A. & Hennessy, J. L. *Computer Organization and Design: the hardware and software interface*. USA: Morgan Kaufmann Publishers Inc., 1997.

[PHILIPPSEN, 1997] - Philippsen, M. *"Is Java ready for computationa science?"* – University of Karlsruhe, Germany: Draft, 1997

[PRAMANICK, 1998] - Pramanick, I. "Parallel and Distributed Computing Using Java". In: *ICDCS'98 Tutorials* – Mountain View, CA, 1998.

[SEBESTA, 2000] - Sebesta, Robert W.: *Conceitos de Linguagens de Programação*. 4ª edição. Bookman Editora, Porto Alegre, 2000.

[SHOFFNER, 1998] - Shoffner, M. "Write your own MOM!". In: *Javaworld Technical Report*, Abril, 1998.

[SNIR, 1996] - Snir, M. *et al*. *PI: The Complete Reference*. Massachussets: MPI Press. 1996.

[SOARES, 1995] - Soares, L.F., Lemos, G. e Colcher, S. *Redes de Computadores: das LANs, MANs e WANs às redes ATM*. 2ª ed. Rio de Janeiro: Editora Campus, 1995.

[SOUSA, 1999] - Sousa, P.B.M. & Fiallos, M.A. "WVM – Uma Ferramenta para Processamento Distribuído na WWW" – In: *Proceedings of 11th Symposium on Computer Architecture and High Performance Computing*. Brazilian Computer Society: UFRGS. Setembro, 1999.

[SRIDHARAN, 1997] - Sridharan, P. *Advanced Java Networking*. New Jersey: Prentice-Hall Inc., 1997.

[SRINIVAS, 1997] - Srinivas, K., Jagannathan, V.J. *et al*. *"Java and Beyond: Executable Content"*. In: *IEEE Communications - Computational & Engineering.* Junho, 1997.

[STANKOVIE, 1998] - Stankovie, N. e Zhang, N.S.K. *"Java and Parallel Processing on the Internet"* – Macquaric University, Austrália: Draft, 1998.

[SUNDERAM, 1993] - Sunderam, V. "PVM: A framework for parallel distributed computing." In: *"Concurrency: Practice and Distributed Computing"* –Draft, 1993.

[TANENBAUM, 1995] - Tanenbaum, A.S. *Sistemas Operacionais Modernos.* Prentice-Hall do Brasil, Rio de Janeiro, 1995.

[_____, 1997a] - Tanenbaum, A.S. *Distributed Operating Systems*. New Jersey: Prentice Hall, 1997.

[_____, 1997b] - Tanenbaum, A.S. *Redes de Computadores*. 4ª. edição. Editora Campus, Rio de Janeiro, 1997.

[_____, 1999] - Tanenbaum, A.S. *Organização Estruturada dos Computadores*. 4ª edição. LTC Editora, 1999.

[TRYSTRAM, 1999] - Trystram, D. "Scheduling for Parallel and Distributed Systems". In: *International School on Advanced Algorithmic Techniques for Parallel Computation with Applications – Mini-course Draft*. Natal, Setembro: 1999.

[VALIANT, 1990] - Valiant, L.G. "A Bridging Model for Parallel Computation". In: *Communications of ACM, 33(8): 103-111*. Agosto, 1990.

[VANHELSUWE, 1997] - Vanhelsuwe, L. "Create your own supercomputer with Java". In: *Javaworld Technical Report*, Janeiro, 1997

[VIANA, 1998] - Viana, V. *Meta Heurísticas e Programação Paralela em Otimização Combinatória*. Fortaleza – CE: Edições UFC, 1998.

[MELCÍADES, 1999] – Melcíades, W., Fiallos, M., Pimentel C., *"Paralelização do Algoritmo do "Backpropagation" em Clusters de Estações de Trabalho"* in: IV Congresso Brasileiro de Redes Neurais, São José dos Campos, SP, 1999.

[YELLIN, 1996] - Yellin, F. *The JIT Compiler API*. Sun Microsystems. October, 1996.

[ZOMAYA, 1996] - Zomaya, A.Y. *Parallel & Distributed Computing Handbook*. McGraw Hill Publishing. New Jersey: USA, 1996.

II. Fontes Auxiliares

[BRASIL, 1999a] - BRASIL, Governo do. "*Ciência e Tecnologia para a Construção da Sociedade da Informação*". Ministério da Ciência e Tecnologia. Conselho Nacional de Ciência e Tecnologia (CCT). Brasília, Outubro: 1999.

[_____, 1999b] - BRASIL, Governo do. "*Programa Sociedade da Informação*". Ministério da Ciência e Tecnologia. Conselho Nacional de Ciência e Tecnologia (CCT). Brasília, Novembro: 1999.

[_____, 1999c] - BRASIL, Governo do. "*Bases de um Programa Brasileiro para a Sociedade da Informação*". Ministério da Ciência e Tecnologia. Conselho Nacional de Ciência e Tecnologia (CCT). Brasília, Novembro: 1999.

[_____, 2000a] - BRASIL, Governo do. **"***Indicadores de Crescimento da Internet no Mundo 1998 e 1999; Indicadores de Crescimento da Internet no Brasil 1995/1997*. Comitê Gestor da Internet no Brasil. Brasília, Janeiro de 2000.

[_____, 2000b] - **"***Estatísticas sobre a Quantidade de Nomes de Domínios Registrados no Brasil*". Fundação de Amparo a Pesquisa do Estado de São Paulo (FAPESP) - **Fonte de Consulta:** `http://registro.fapesp.br/estatisticas.html`. São Paulo, Janeiro de 2000.

[EXAME, 2000] - **Info EXAME,** Ano 15, no. 177, edição Dezembro 2000.

Reportagem 1: LOPES, Airton, FORTES, Débora *et al*. "Eles arrasaram em 2000". Pág. 72: "*Linguagem do ano*".

Reportagem 2: MOREIRA, Maria Isabel. "Só é preciso um Browser". Pág. 210-213.

[SBC, 1999] - **SBAC-PAD'99.** Proceedings of 11th Symposium on Computer Architecture and High Performance Computing**.** Editors: NAVAUX, Philippe O. A., DIVERIO, T. A. & GEYER, C. R. – Porto Alegre, Instituto de Informática da UFRGS, 1999.

[USA, 1991] - **UNITED STATES, Government.** "High Performance Computing Act of 1991". In: *United States Code – Title 15: Commerce and Trade – Chapter 81: High Performance Computing*. Washington, December: 1991.

APÊNDICES

Apêndice A: Siglas e Abreviaturas
Apêndice B: Marcas Registradas
Apêndice C: Anexos

A. Siglas e Abreviaturas

AAL	*ATM Adaptation Layer*
API	*Application Programming Interface*
ATM	*Assyncronous Transmission Mode*
AWT	*Abstract Window Toolkit*
BD	Banco de Dados
BFS	*Basic Fortran Support*
C_{WVM}	Cliente WVM
CDCE	*Collaborative Distributed Computing Environment*
CENAPAD-NE	Centro Nac. de Processamento de Alto Desempenho no Nordeste
CGI	*Common Gateway Interface*
CIP	*Classic IP*
CORBA	*Common Object Request Broker Architecture*
CPU	*Central Processing Unit*
C/S	Cliente / Servidor
DAG	*Directed Acyclic Graph*
DCE	*Distributed Computing Environment*
DNS	*Domain Name System*
DAP	Domínio de Aplicações e Processamento
DP	Domínio de Processamento
EP	Elemento Processador
FTP	*File Transfer Protocol*
GASS	*Global Access to Secondary Storage*
GMT	*Globus Metacomputing Toolkit*
GRAM	*Globus Resource Allocation Manager*
GSI	*Generic Service Security*
GUI	*Graphic User Interface*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HPCC	*High Performance Cluster Computing*
HPC	*High Performance Computing*
IBM	*International Business Machine*
IDE	*Integrated Development Environment*
IDL	*Interface Definition Language*
IFC	*Internet Foundation Classes*
IIOP	*Internet Inter-Orb Protocol*

IP[118]	*Internet Protocol*
IWAY	*Information Wide Area Year*
JAVADC	*Java for Distributed Computing*
JIT	*Just In Time compilation*
JLS	*Java Language Specification*
JRE	*Java Runtime Environment*
JVM	*Java Virtual Machine*
LAN	*Local Area Network*
LANE	*LAN Emulation*
LP	Linguagem de Programação
MC	Metacomputação
MDS	*Metacomputing Directory Service*
MIMD	*Multiple Instruction stream over Multiple Data stream*
MIME	*Multipurpose Internet Mail Extensions*
MISD	*Multiple Instruction stream and Single Data stream*
MIPS	Milhões de Instruções Por Segundo
MOM	*Message-Oriented Middleware*
ML	Metalinguagem
MP	*Message Passing*
MPI	*Message Passing Interface*
MPL[119]	*Message Passing Library*
MPL	*Mentat Programming Language*
NAR	Núcleo de Atendimento Remoto
NASA	*National Aeronautics and Space Administration*
NC	*Network Computer*
NCSA	*National Center for Supercomputer Application*
NET	*Native Executable Translation*
NFS[120]	*Network File System*
NFS	*Number Field Sieve*
NIS	*Network Information Service*
NIST	*National Institute of Standards and Technology*
NOPC	*Network Of Personal Computers*
NOW	*Network Of Workstations*
NPD	Núcleo de Processamento de Dados (UFC)
NT	*(Windows) New Tecnology*

[118] Comumente, o termo IP é usado para designar o **endereço** `TCP/IP` de uma determinada máquina na Internet.

[119] MPL no sentido mais utilizado no texto.

[120] NFS como sigla mais comumente utiliza no texto.

OO	Orientação a Objetos
ORB	*Object Request Architecture*
ORNL	*Oak Ridge National Laboratory*
OSF	*Open Software Foundation*
PAD	Processamento de Alto Desempenho
PC	*Personal Computer*
PHP	*Hypertext Preprocessor*
PVM	*Parallel Virtual Machine*
RISC	*Reduced Instruction Set Computer*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
S_{WVM}	Servidor WVM
SA	*Site* de Aplicações
SIMD	*Single Instruction stream over Multiple Data stream*
SISD	*Single Instruction stream and Single Data stream*
SO	Sistema Operacional
SOCINFO	Sociedade Brasileira da Informação
SSH	*Security Shell*
TCP	*Transmission Control Protocol*
UC	Unidade de Controle
UDP	*User Datagram Protocol*
UECE	Universidade Estadual do Ceará
UFC	Universidade Federal do Ceará
ULA	Unidade Lógica e Aritmética
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
URN	*Uniform Resource Name*
VM	*Virtual Machine*
VPL	*Virtual Programming Laboratories*
VRML	*Virtual Reality Markup Language*
WVM	*Web Virtual Machine*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*

B. Marcas Registradas

IBM©, AIX©, PS/2© e PC© são marcas registradas da International Business Machine Corporation.

Java©, JIT©, JRE©, JVM©, Solaris©, JavaSoft©, JavaScript© e HotJava© ão marcas registradas da Sun Microsystems Corporation.

Windows©, Windows95©, WindowsNT©, COM©, DCOM© e OLE© e Internet Explorer© são marcas registradas da Microsoft Corporation.

Netscape Communicator© e Netscape Navegator© são marca registrada da Netscape Corporation.

C. Anexos

ANEXO C.1

TEMA (cf. seção 1.2.1.B)**:** DETALHES DA CLASSIFICAÇÃO MIMD

Os sistemas MIMD, conforme mencionados, englobam arquiteturas com memória compartilhada e distribuída. No primeiro caso – os multiprocessadores – encontramos 3 classificações básicas relativas à organização e utilização da memória (cf. figura 1.5):

- **Máquinas UMA (*Uniform Memory Access*):** Os processadores percebem todos os módulos de memória, apresentando mesmo tempo de acesso para eles, permitindo prever desempenhos para acesso aos dados. Podem funcionar em módulos interconectados de duas formas:
  - ***Barramento:*** Mecanismo mais simples, onde um canal é igualmente compartilhado por todos os processadores. Por questões de desempenho, pode-se inserir *caches* ou mesmo memórias locais (com cópias de alguns dados da memória compartilhada) em cada processador, o que acarreta uma considerável economia de acesso ao barramento e à memória central. Eventualmente, políticas para controle da **coerência da memória** devem ser empregadas[121];
  - ***Comutador:*** Permite estender a quantidade de processadores em um sistema uniforme, graças ao chaveamento de regiões de memória (semelhante ao que ocorre em sistemas de telefonia para conectar linhas de entrada a saídas), criando uma rede de interconexão *não-bloqueadora*, isto é: vários processadores podem simultaneamente fazer acesso a várias regiões de memória[122].

[121] Neste contexto, destacam-se protocolos como o MESI ([Tanenbaum *apud* Papamarcus & Patel, 1984]), que se baseia na utilização de 4 bits para descrição de estados: "M" (*modified*), "E" (*exclusive*), "S" (*shared*) e "I" (*invalid*). A combinação deles permite criar mecanismos eficientes para consistência, utilizando uma política *write-back* de escrita da memória (onde um dado somente é escrito na memória no momento em que deve ser retirado da *cache*, o que o torna mais eficiente do que a política de escrita imediata – chamada de *write-through*).

[122] Algumas vezes, os projetos de redes chaveadas permitem criar modelos tão eficientes quanto complexos, baseados em múltiplos estágios, que criam diferentes combinações com grande economia (chegando à ordem de

- **Máquinas NUMA (*Non-Uniform Memory Access*):** Neste caso, os processadores não necessitam respeitar a uniformidade de acesso à memória compartilhada (de forma que, normalmente, memórias mais próximas são consideravelmente mais rapidamente acessadas). Em outras palavras, os locais de armazenamento dos dados influem consideravelmente na análise do programa. Vale notar que, neste caso, diferentemente do que ocorria no modelo UMA, as memórias remotas são visíveis por todos os demais processadores e podem ser por eles acessadas através de instruções do tipo `LOAD` e `STORE`:
  - ***CC-NUMA:*** Engloba máquinas que trabalham com **coerência de *cache*** a fim de manter a integridade e consistência dos dados. Utilizam o conceito de multiprocessador **baseado em diretório**, pois mantém em nível de *hardware*, uma base de dados informando sobre o estado de cada informação na *cache*[123].
  - ***NC-NUMA (NCC-NUMA):*** Trabalham sem a necessidade de *cache* (o que torna dispensável a utilização de políticas para manutenção de coerência dos dados) e apresentam um tempo de acesso às memórias remotas conhecido, mas consideravelmente mais lento.
- **Máquinas COMA (*Cache Only Memory Access*):** Cada processador tenta utilizar a memória dos demais como se fosse seu próprio *cache*, de forma que busca-se obter uma velocidade de acesso bem superior, graças à elaboração de *hardware* especializado para a transferência de dados. Contudo, vários problemas relativos à localização e remoção dos dados, tornam a gerência da memória bem mais complexa, tornando necessários sistemas de coerência de cache.

No caso dos multicomputadores, vale destacar que os computadores de memória distribuída não implementam em nível de *hardware*, nenhuma política para acesso às memórias remotas, de forma que tais EPs também são chamados de máquinas NORMA (*NO Remote Memory Access*). Globalmente, classificam-se em:

- **MPP (*Massively Parallel Processors*):** Computadores de memória distribuída formados por EPs independentes interligados por uma rede de

---

`|log n|` estágios, com `|n/2|` comutadores cada): é o caso das **redes ômega** (cf. [Tanenbaum *apud* Adams et al, 1987].

[123] As políticas de gerência de dados em sistemas CC-NUMA devem levar em conta três estados: `uncached` (quando o dado se encontra fora da *cache*), `shared` (memória em uso por várias *caches*) e `modified` (quando se constata inconsistência). Este modelo também é conhecido com DSM de *hardware*, já que, para todos os efeitos, implementa uma memória distribuída compartilhada pelos demais EPs.

interconexão (normalmente proprietária) extremamente rápida[124]. Dentre as características dos MPPs, encontramos [Hwang, ibid.]: alta escalabilidade (atingindo patamares de até milhares de processadores); funcionamento assíncrono em nível de *hardware*, com mecanismos de sincronização por *software* (através de barreiras, por exemplo); comunicação via passagem de mensagens (a ser explicado adiante).

- **COW (*Cluster Of Workstations*):** Neste caso, cada EP é uma estação de trabalho independente (inclusive em nível de *software*), com memória e discos próprios e conectando-se a outros EPs através de uma rede de baixo custo (como Ethernet, FDDI ou ATM)[125].

ANEXO C.2

TEMA (cf. seção 2.2.2)**:** OUTROS TIPOS DE PARALELISMO QUANTO AO PARTICIONAMENTO

Na literatura, pode-se encontrar um modelo mais amplo de classificação para aplicações concorrentes, introduzido por [Fox, 1996] e valendo-se das novas linguagens e tecnologias empregadas para a resolução de problemas por processamento distribuído:

- **Paralelismo de Objetos:** Embora similar ao paralelismo de dados, este modelo apresenta uma complexidade maior que o anterior, devido à própria mudança de paradigma do modelo imperativo para o modelo OO (confira detalhes tabela 1.5) ocorrendo em simulações e modelagens, nos quais objetos com atributos e métodos aparecem. Note que, sendo um esta modalidade de paralelismo se enquadra perfeitamente em sistemas de simulação, onde os objetos devem apresentar dados específicos e responder a eventos (figura C.1).

---

[124] Alguns autores usualmente rotulam tais máquinas como "supercomputadores", mas esta definição está fortemente sujeita às restrições temporais. Uma das arquiteturas a ser utilizada na elaboração da ferramenta (o SP/2 da IBM) enquadra-se nesta categoria.

[125] Alguns autores identificam as COWs indistintamente com as NOWs (*Network Of Workstations*), principalmente levando-se em conta o crescente poder computacional dos microcomputadores. [Tanenbaum, *ibid.*] observa que a "diferença entre um sistema COW descentralizado e uma rede local repleta de máquinas de usuários baseia-se única e exclusivamente no software e na utilização do ambiente". Conforme salienta [Hwang, *ibid.*], a separação entre sistemas MPPs e COWs está ficando cada vez mais sutil nos dias de hoje. Por exemplo, a arquitetura SP/2 pode ser vista como um híbrido das duas alternativas.

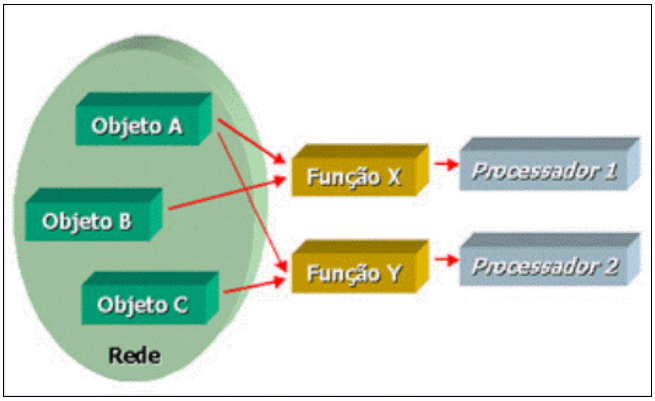


Figura C.1: Diagrama de um Paralelismo de Objetos

- **Paralelismo por Metaproblemas:** Quando o paralelismo funcional é visto como algo restrito a poucas instruções escalonadas para a otimização entre ciclos de comunicação e processamento – como em sistemas de memória compartilhada – certos autores optam por uma generalização do modelo de concorrência funcional (figura C.2). Neste contexto, [Fox, 1996] apresenta a idéia **metaproblemas:** "conjuntos de problemas conjugados em que cada unidade é, ela mesma um problema independente". Desta forma, possuem uma granulosidade grossa e baixíssima necessidade de comunicação.

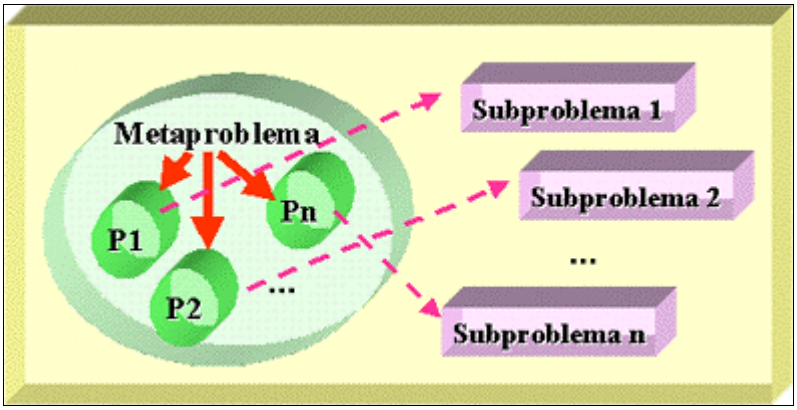


Figura C.2: Diagrama de um Paralelismo por Metaproblemas

ANEXO C.3

TEMA (cf. seção 5.2.1)**:** ORGANIZAÇÃO DO SISTEMA NACIONAL DE PROCESSAMENTO DE ALTO DESEMPENHO E DAS REDES METROPOLITANAS DE ALTA VELOCIDADE

♦ Centros Nacionais de Processamento

Assim como tantos outros países que se seguiram à iniciativa americana (a qual tornou a chamada "superestrada da informação" uma área prioritária para o desenvolvimento da ciência e tecnologia), o Brasil, na tentativa de acompanhar o acelerado ritmo de crescimento da Internet e da Informática, com todo o seu impacto na sociedade, tem procurado definir novos rumos da política gestora, a ser definida pelo próprio Governo Federal. Historicamente, merece destaque o desenvolvimento do Sistema Nacional de Processamento de Alto Desempenho (SINAPAD), surgido em meados de 1990 e formado por vários Centros de Processamento (CENAPADs) e Núcleos de Atendimento Remotos (NARs) – vide figura C.3.

O modelo, que, por um lado, visava baratear os custos para instalação de parques tecnológicos e o atendimento aos pesquisadores, por outro, permitiu uma integração entre grandes centros de processamento. Juntamente com a Rede Nacional de Pesquisa (RNP), o SINAPAD forma o eixo básico de sustentação e suporte à pesquisa no Brasil. Contudo, enquanto a primeira se preocupa com questões de comunicação e conectividade, o segundo visa apenas fornecer recursos computacionais para processamento.

Figura C.3: Organização do SINAPAD

♦ Redes Metropolitanas de Alta Velocidade

Apesar da grande utilização de processamento, o sistema como um todo apresentava uma séria deficiência: a velocidade de comunicação. Assim, tornou-se necessária a implantação de um *backbone* (coluna vertebral básica para a criação de uma rede de computadores geograficamente distribuídos) de alta velocidade.

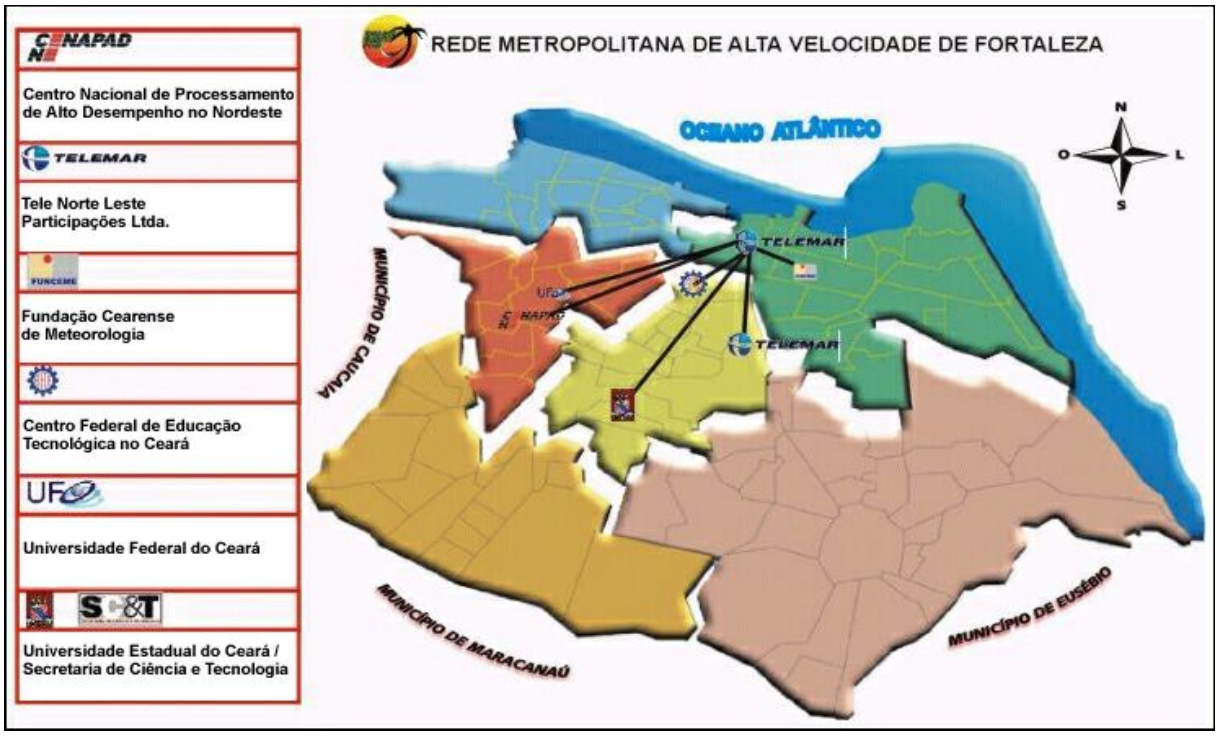


Figura C.4: Organização da RMAV-FOR

Para isso, a partir de 1998, começou a ser implantado a nível nacional, redes metropolitanas baseadas em tecnologia ATM com altas taxas de transferência e controle de qualidade de tráfego, no chamado projeto RMAV (Redes Metropolitanas de Alta Velocidade) e que dará origem à chamada Internet-2 no país.

Inicialmente englobando até 14 experimentos dentro de uma mesma localidade, várias instituições (num total de 79) receberam equipamentos e treinamento de pessoal a fim de elaborarem projetos pilotos e construírem redes metropolitanas. Passada a fase experimental, os atuais centros (presentes em todas as grandes metrópoles) entraram no processo de integração, visando atingir patamares de velocidade de ordem que varia entre 34 e 155 Mbps, expansível até a capacidade de alguns Gbps. Em Fortaleza, o projeto-piloto abrangeu uma boa parcela da área metropolitana, envolvendo as seguintes instituições: CENAPAD/NE, Telemar, Funceme, CEFET-CE, UFC e Secitece. A figura C.4 ilustra a organização da RMAV-FOR.

- Perspectivas Futuras

Um passo importante foi dado no ano passado (1999), com a publicação de um documento emitido pelo Conselho Nacional de Ciência e Tecnologia (CCT), a partir de um Relatório original feito ainda em 1997 e que visa elaborar uma estratégia de Ciência e Tecnologia para a construção da assim chamada **Sociedade Brasileira da Informação** (SOCINFO), definida como um "um novo ambiente global baseado em comunicação e informação, cujas regras e modos de operação estão sendo construídos em todo o mundo" [Brasil, 1999a].

As palavras iniciais do texto ilustram bem o grau de importância conferido a este projeto: "A concepção de uma estratégia nacional para estimular a adequada inserção da sociedade brasileira na Sociedade Global da Informação é prioritária." O objetivo visa, em última instância, elaborar um projeto de amplitude nacional, "para estabelecer e testar no Brasil a necessária infra-estrutura, os futuros serviços e as típicas aplicações da Sociedade da Informação, tendo como base o desenvolvimento de uma nova geração de redes Internet, com benefícios estendidos a toda a sociedade brasileira."

O projeto, porém, não é uma iniciativa única do Governo, mas consiste em um esforço conjunto de cooperações entre parceiros e a sociedade. O modelo proposto para o seu desenvolvimento segue o diagrama de evolução em espiral (figura C.5).

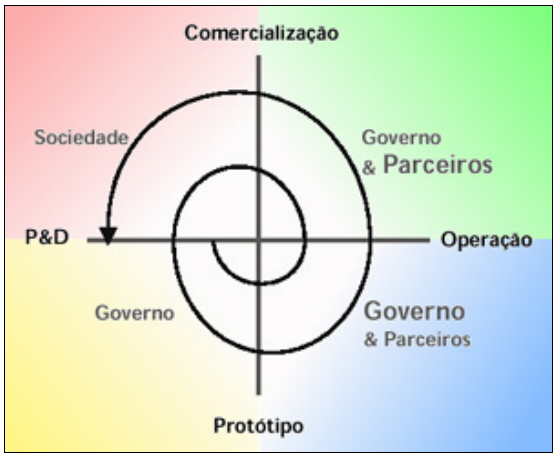


Figura C.5: Ciclo de Utilização e Desenvolvimento de Novas Tecnologias

Um ponto importante a ser ressaltado é o de que o papel fundamental do Governo "deve ser o de universalizar as oportunidades individuais, institucionais e regionais" e "dar suporte e incentivar o desenvolvimento tecnológico" a fim de evitar "o aprofundamento da desigualdade social que já existe entre os brasileiros" no chamado ***apartheid* digital** [Brasil, 1999b].

Dez são os objetivo setoriais a serem alcançados, merecendo destaque o enfoque de Ciência e Tecnologia (experimentos cooperativos e disseminação de resultados) e Educação (massificação da educação a distância). Além destas áreas, mencionam-se campos como Saúde, Meio-Ambiente e Agricultura, Empresa (Indústria e Comércio), Cultura, Transporte, Trabalho, Governo e Relações Internacionais [Brasil, 1999c].

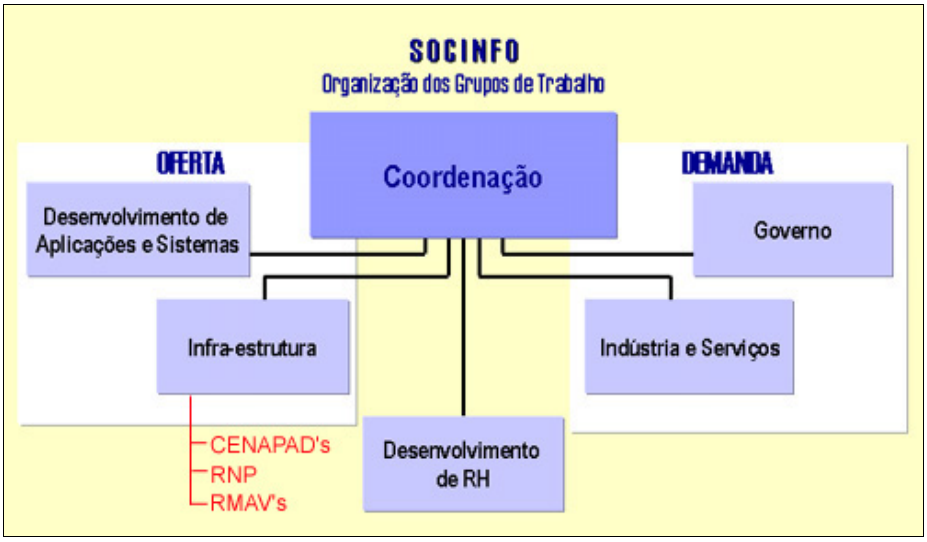


Figura C.6: Organização dos GTs da SOCINFO

O estado atual do Projeto compreende uma fase de organização da SOCINFO em Grupos de Trabalhos (GTs) distintos, preocupados especificamente com a oferta ou demanda de serviços e suporte à comunicação/processamento. Os atuais CENAPADs, juntamente com a RNP e as RMAVs constituem o fundamento da Infra-estrutura do novo sistema, conforme ilustrado na figura C.6.

ANEXO C.4

TEMA (cf. seção 4.2.2)**:** CURSOS DE PARALELISMO MINISTRADOS NO CENAPAD-NE ENTRE 1996 E 1999

Para auxiliar na absorção da alternativa do processamento paralelo-distribuído por parte da comunidade de pesquisa, vários cursos foram desenvolvidos no CENAPAD-NE, incluindo o desenvolvimento de *Workshops Virtuais*, para a educação a distância, realizados via Internet. O quadro a seguir ilustra os números de inscritos e concludentes destes eventos no quatriênio de 1996 a 1999.

O quadro indica que dos 17 treinamentos ministrados nos últimos 4 anos, das aproximadamente 370 pessoas inscritas, 222 (2/3) concluíram os cursos e treinamentos (os últimos realizados completamente via Internet em *Workshops Virtuais*)[126]. A figura C.7 ilustra, a partir dos dados anteriores, uma organização da distribuição de alunos inicialmente inscritos e efetivamente treinados pelos cursos de paralelismo no Centro. O que pode se observar é que, além de uma considerável defasagem entre esses números, houve uma considerável queda de

[126] A conclusão de um curso – mesmo via Internet - de acordo com a política adotada no CENAPAD-NE, implica na obtenção de uma nota final acima da média (7,0) para os exercícios e projetos desenvolvidos por cada aluno individualmente.

participação nos últimos cursos (dos quais apenas uma taxa inferior a 1/3 dos participantes realmente têm concluído os cursos).

Título do Curso	Período	Inscritos ([127])	Treinados
Programação com PVM	25/06 a 17/07/96	25	25
Introdução à Prog. Paralela com PVM	08 a 12/04/96	25	10
Introdução à Prog. Paralela com PVM	08/08 a 12/10/96	25	15
Curso de PVM – Graduação	26 a 30/08/96	20	20
Programação Paralela com MPI	31/03 a 04/04/97	25	13
Introdução à Prog. Paralela com PVM	23 a 27/06/97	15	05
Programação Paralela com MPI	24 a 28/11/97	20	15
Programação Paralela com MPI	1° a 12/12/97	25	06
Introdução à Prog. Paralela com PVM	13 a 17/01/97	20	16
Programação Paralela com MPI	09 a 27/02/98	15	08
Introdução à Prog. Paralela com MPI	18 a 22/05/98	25	24
Programação Paralela com MPI	29/06 a 10/07/98	10	04
Introdução à Prog. Paralela com MPI	03 a 08/11/98	15	10
Programação Paralela com MPI	16/11 a 04/12/98	45	38
Programação Paralela com MPI	05 a 16/07/99	15	02
Programação Paralela com MPI	30/08 a 11/09/99	15	05
Programação Paralela com MPI	03 a 14/11/99	15	06

Tabela C.1: Quadro de cursos realizados no CENAPAD-NE de 1996 a 1999

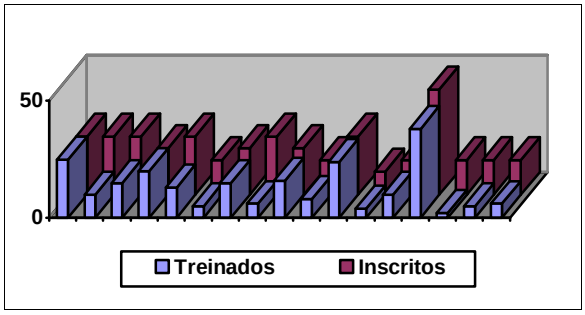


Figura C.7: Distribuição dos resultados nos cursos do CENAPAD-NE (1995-1999)

ANEXO C.5

TEMA (cf. seção 7.1.1)**:** MECANISMO DE VALIDAÇÃO DOS CLIENTES

Uma observação que deve ser feita nessa discussão reside no mecanismo de validação dos clientes, isto é, na forma de como permitir o seu funcionamento na prática. Observe que, quando executam sob a forma de **aplicação**, as máquinas virtuais não apresentam quaisquer restrições com relação à segurança, já que tais programas são chamados nativamente e executam a partir de uma fonte local. No entanto, sob a forma de *applet*, isto é, na situação em que o código provém de uma fonte externa (o que não implica obrigatoriamente em origem confiável), as pesadas barreiras de segurança surgem e inibem fortemente as funções possíveis.

[127] Número oficialmente cadastrado de participantes (normalmente limitados a um limite de 25 por curso).

Note que, a partir da interface cliente, comandos são enviados ao servidor e, entre estes, destacam-se funções para criação de diretórios, remoção de arquivos e execução de aplicações (vide, por exemplo, a seção 5.3.1). Além disso, a própria edição de arquivos locais se torna problemática, já que como o código é alienígena, isso pode apresentar uma brecha de segurança. A questão central reside, portanto, em como utilizar a *applet* com todas as funcionalidades. Este problema não é particular da ferramenta, mas comum às especificações de Java para tal situação. Dessa forma, a solução se torna igualmente usual: validar a *applet*, no processo chamado comumente como **assinatura**. Resumidamente, assinar uma *applets* significa basicamente que o usuário reconhece a aplicação como confiável (normalmente porque provém de uma fonte conhecida) e que, portanto, possui o direito de executar com certas prioridades.

O modelo atual[128] (JDK 1.2), o mecanismo de validação passa pelo crivo do administrador de segurança (Security Manager), que, por definição, restringe o espaço de atuação de uma *applet* a um nível elementar de segurança (o *sendbox*).

No entanto, este tratamento pode ser mudado com a utilização de uma política de segurança (Security Policy), que pode ajustar tanto objetos locais quanto remotos, para executarem tarefas que extrapolam os limites preestabelecidos. Será através dos ajustes neste administrador, que todo o esquema de funcionamento da WVM se tornará possível. A figura C.8 ilustra esquematicamente o modelo funcional da segurança na versão 1.2 do Java.

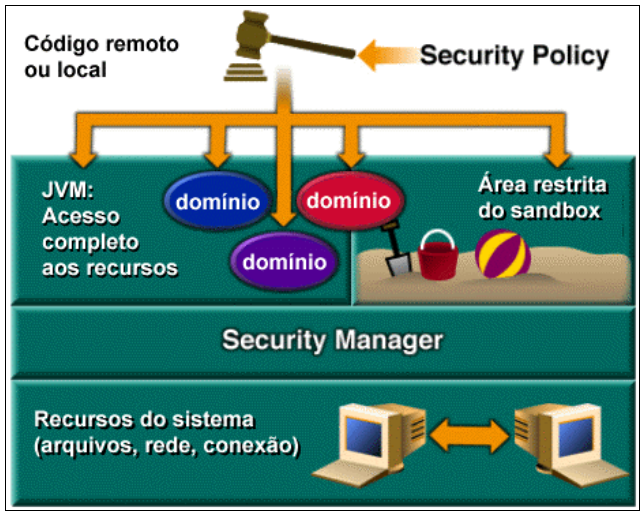


Figura C.8: Esquema de segurança em Java 2.

Existem duas formas para uma *applet* ser considerada confiável: quando ela está instalada em um disco local, num diretório sinalizado pela variável CLASSPATH[129], ou quando esta se encontra assinada e sua identidade existe na base de dados local. Em todos os casos, o esquema de funcionamento (figura C.9) consiste em analisar os *bytecodes* em 3 níveis: por um verificador (Class Verifier), um carregador de classes (Class Loader) e, finalmente, pelo administrador de segurança. Tais refinamentos tratam de resolver

[128] O processo para autenticação de *applets* tem sofrido contínuas modificações nas sucessivas especificações da Sun e nas várias distribuições do ambiente de desenvolvimento JDK.

[129] O CLASSPATH indica caminhos para a busca de classes. Para as primeiras versões do JDK, tal definição era imprescindível. Atualmente, sua principal função consiste em indicar que as classes ali existentes são confiáveis.

sucessivamente os seguintes problemas: se o código representa uma classe válida, se ela pode ser carregada e, finalmente, executada.

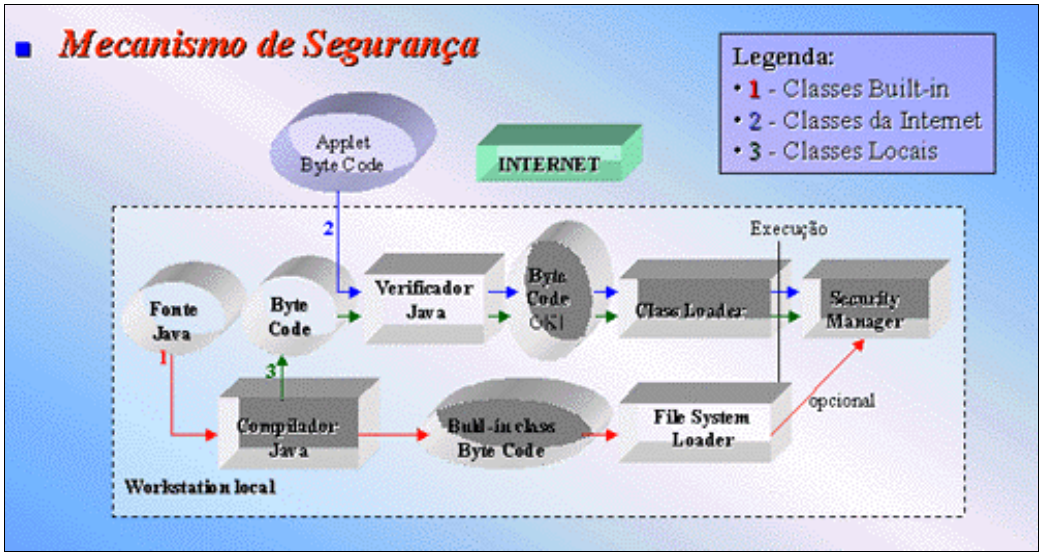


Figura C.9: Mecanismo de execução de classes em Java.

O processo para a assinatura da *applet* consiste basicamente de duas etapas: uma feita no desenvolvedor para gerar as informações e arquivos necessários para a certificação (passos 1, 2 e 3) e outra feita nos clientes, a fim de importar e ajustar o ambiente de execução (passos 4, 5 e 6), conforme detalhado a seguir.

Etapa I: Assinatura digital

(1) Criação de uma identidade e geração das suas chaves públicas e privadas DSA para a entidade (através do utilitário `javakey`);

(2) Geração de um certificado com diretivas que especificam as informações da aplicação (*subject*, *date*, *serial number*);

(3) Geração de um arquivo de diretivas com informações sobre a assinatura (*singer*, *cert*, *chain*)

Etapa II: Carga da *applet* assinada

(4) Cadastro da entidade que assinou a *applet* e identificação da mesma como confiável (novamente, através do `javakey`, desta vez no lado do cliente);

(5) Importação do certificado da entidade assinante para a base da entidade[130];

(6) Finalmente, carregar os *bytecodes* em *browsers* previamente configurados para suportar as *applets* assinadas[131].

[130] A base dos registros (`identitydb.obj`) deve ficar em um local acessível à autenticação do usuário, como em seu diretório *home* ou o arquivo *profile* no `Windows`.

Vale destacar que a necessidade de ajustes nos *browsers* representa um impecílio considerável para o que deveria ser uma carga transparente de *applets* com funcionalidades superiores às permitidas pelo *sandbox*. Contudo, considerando que este é um fator basicamente tecnológico, todos estes problemas tendem a ser resolvidos mais facilmente com o passar do tempo e com o advento de novas ferramentas. As novas especificações JRE têm evoluído de forma a possibilitar maiores poderes às *applets* assinadas; contudo, o procedimento atual ainda continua consideravelmente complexo.

---

[131] Estritamente falando, um código `HTML` bastaria conter a referência ao arquivo do formato `jar`, no qual se encontram os *bytecodes* e os certificados. No caso dos navegadores, contudo, isso significa que devem possuir *plugins* previamente instalados para dar suporte à execução das classes mais novas. Por exemplo, o `Netscape` possui a necessidade de se utilizar a ferramenta `HTML Converter`, para que este *browser* possa tratar a *applet* assinada, ao apontar para um código adaptado.

D. Documentações

D.1 – Relatório dos arquivos fontes (`Javadoc`)

OBS : A DOCUMENTAÇÃO FOI GERADA PELO SISTEMA DE DOCUMENTAÇÃO DA LINGUAGEM JAVA.

A partir da próxima página, são descritas informações sobre as classes utilizadas tanto no cliente, quanto no servidor, seja em nível de serviços, seja em nível de componentes gráficos. A organização é feita através das seguintes partes, de acordo com o padrão da ferramenta `Javadoc`:

1. Classes internas (*inner classes*): que são classes embutidas dentro do código da classe principal;
2. Campos (*fields*): nos quais são registrados as principais variáveis, constantes ou estruturas da classe;
3. Método construtor (*constructor*): Rotina principal da classe, responsável pela ativação do objeto, permitindo polimorfismo e a simulação de heranças múltiplas;
4. Métodos: Conjunto de ações ou comportamentos interentes à classe.

As seguintes classes foram documentadas:

- `Class wvmcli`: Responsável pela classe principal da aplicação cliente;
- `Class wvmclidialog`: Ambiente para criação de caixas de diálogos no cliente;
- `Class wvmcliframe`: Janela principal da aplicação cliente;
- `Class wvmcligraf`: Telas para exibição de saídas gráficas;
- `Class wvmclimac`: Janela de configuração do ambiente virtual (máquinas);
- `Class wvmcliwindow`: Interface gráfica para suporte às janelas no cliente;
- `Class wvmloadimag`: Classe para exibição de imagens nas múltiplas janelas;
- `Class wvmser`: Classe principal da aplicação servidora;
- `Class wvmserdialog`: Ambiente para exibição de caixas de diálogo no servidor;
- `Class wvmserframe`: Janela principal da aplicação servidora;
- `Class wvmserwindow`: Interface gráfica para suporte às janelas no servidor.

Maiores informações podem ser obtidas em: `http://www.cenapadne.br/~benicio/wvm/docs/`.

Última atualização: terça-feira, 26 de agosto de 2003
A.M.D.G.