Proposta de uma Ferramenta para Elaboração e Monitoração de Aplicações Distribuídas

RONALDO AUGUSTO DE LARA GONÇALVES [1]
DURVAL MAKOTO AKAMATU [2]
ITANA MARIA DE SOUZA GIMENES [1]

[1] DIN / CTC / UEM
Av. Colombo. 5790,
ZIP: 87020-900
Maringá - PR, Brazil
{ronaldo,itana}@din.uem.br

[2] DC / CCT/ UFSCAR
Via Washington Luiz. km 235,
ZIP: 13565-905, P.O. Box 676,
São Carlos - SP, Brazil
ddma@power.ufscar.br

ABSTRACT

The development of distributed applications enables either lightly loaded or idle processors to be grouped together for the execution of the same application, thus increasing applications speed up.

This paper analyses the structures and resources of existing tools for the development of distributed applications in order to extract relevant concepts and techniques. Based on these information, a more simple tool is proposed which offers many interactive facilities to the user.

The main techniques used are described showing the tool architecture and resources provided for distributed application programmers.

KEYWORDS: Distributed Application, Tool, Software Engineering.

1 Introdução

Devido ao crescente aumento da quantidade de informações, ocasionado por diversos motivos tais como aumento populacional e complexidade dos problemas apresentados, uma das principais preocupações em sistemas computacionais é a agilização na execução das tarefas. Nesse sentido, torna-se necessário o aproveitamento máximo dos recursos computacionais disponíveis.

Na tentativa de solucionar este problema, as ferramentas para o desenvolvimento de aplicações distribuídas têm sido eficientes. Além de executarem computação paralela, aproveitando os recursos remotos que poderiam estar ociosos, elas possibilitam transparência para os programadores, com alta taxa de processamento, confiabilidade e segurança.

Compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Neste artigo, é discutido a estrutura e o funcionamento de algumas ferramentas para o desenvolvimento de aplicações distribuídas, abordando as facilidades que cada uma oferece para os programadores. Em seguida, é apresentada uma proposta para o desenvolvimento de uma ferramenta similar simplificada, chamada FEMAD.

2 Ferramenta Para Aplicações Distribuídas

Uma ferramenta para aplicações distribuídas pode fornecer tanto o ambiente para o desenvolvimento das aplicações distribuídas quanto a plataforma para a execução destas.

O ambiente de desenvolvimento quase sempre se compõe de uma interface gráfica com acesso a editores de textos, compiladores, bibliotecas de funções e interface de execução e depuração das aplicações. Em alguns ambientes de desenvolvimento, existe uma interface de monitoração da execução da aplicação distribuída, como no caso da ferramenta aqui proposta.

Uma plataforma para aplicações distribuídas é uma estrutura que suporta a execução destas aplicações. Esta plataforma esconde os detalhes da rede, fornecendo para as aplicações distribuídas um ambiente integrado, homogêneo e transparente. Uma plataforma para aplicações distribuídas pode ser estabelecida previamente ou em tempo de execução [5].

A plataforma estabelecida previamente se comporta como um sistema operacional distribuído, que se instala na rede e espera pela solicitação da execução das aplicações. Já a plataforma estabelecida em tempo de execução, é instalada juntamente com o código das aplicações distribuídas.

Antes da apresentação da ferramenta aqui proposta, serão descritas brevemente algumas ferramentas existentes, de onde serão utilizados alguns conceitos e técnicas. Serão abordados a estrutura, o funcionamento e as facilidades de programação de cada uma delas.

2.1 Arquitetura ANSA

ANSA ou Arquitetura Avançada para Sistemas de Rede [2], é uma arquitetura que pode suportar a construção de aplicações distribuídas sobre uma rede de *workstations*, de modo que a distribuição seja transparente para os usuários e programadores.

Os principais componentes da arquitetura ANSA são: os Núcleos, o *Trader* e o Gerente de Configuração. A maneira como os componentes da arquitetura ANSA são interligados pode ser vista na figura 1, que mostra uma cooperação entre dois sistemas ANSA.

Nesta figura, sobre cada plataforma ANSA estão sendo executadas várias aplicações, que interagem com a rede de hospedeiros através dos Núcleos.

Os Núcleos ampliam os recursos da infra-estrutura local, fornecendo para cada hospedeiro um ambiente computacional distribuído comum. Estes Núcleos são hábeis para trabalhar em

conjunto com os processos especiais *Trader* e Gerente de Configuração, formando uma plataforma básica para computação distribuída.

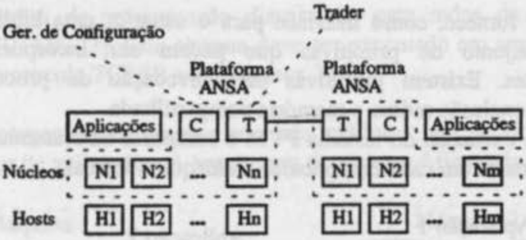

Figura 1 Uma Cooperação entre Sistemas ANSA.

O *Trader* gerencia a estrutura de diretório, permitindo que cada programa servidor registre sua presença na rede e que cada cliente localize os servidores. O *Trader* permite a comunicação entre componentes das aplicações. O Gerente de Configuração é utilizado para iniciar novos componentes das aplicações que já estejam sendo executadas no sistema ANSA.

O *TestBench* é uma implementação baseada na arquitetura ANSA que fornece ao programador duas ferramentas principais: um Compilador *Stub* e um Pré-processador C.

O Compilador *Stub* converte uma especificação de interface de serviços em um conjunto de procedimentos *stubs*. A especificação da interface consiste de um arquivo texto escrito em linguagem de definição de interface, que deve definir os tipos de dados e as operações relacionadas com as partes comunicantes da aplicação distribuída.

Devido a complexidade desta interface, o programador deve ter conhecimento profundo sobre a linguagem de definição de interface e sobre os detalhes da rede, para que a aplicação seja desenvolvida de forma correta e confiável.

O Pré-processador C processa os arquivos fontes dos componentes da aplicação distribuída, escritos em uma linguagem C estendida, e gera um outro arquivo fonte escrito em C, para ser posteriormente compilado. Esta linguagem estendida possui declarações que fazem referências aos procedimentos *stubs* gerados anteriormente pelo Compilador *Stub*.

2.2 PVM - Parallel Virtual Machine

PVM ou *Parallel Virtual Machine* [6], é um pacote de *software* que permite a utilização de uma rede heterogênea de computadores como um único recurso computacional, como mostra a figura 2.

O sistema PVM fornece um ambiente para o desenvolvimento e execução de grandes aplicações paralelas e concorrentes, compostas por muitos componentes. Estes componentes são

programas relativamente independentes e de alta granulosidade. Um componente corresponde a um arquivo objeto capaz de ser executado como um processo em nível do usuário.

O sistema PVM fornece, como interface para o usuário, uma biblioteca de funções [4] composta por um conjunto de primitivas que podem ser incorporadas em linguagens procedimentais existentes. Existem primitivas para invocação de processos, transmissão e recepção de mensagens, exclusão mútua e memória compartilhada.

A plataforma de execução do sistema PVM é composta basicamente por um conjunto de *daemons*, que são executados em cada computador da máquina virtual.

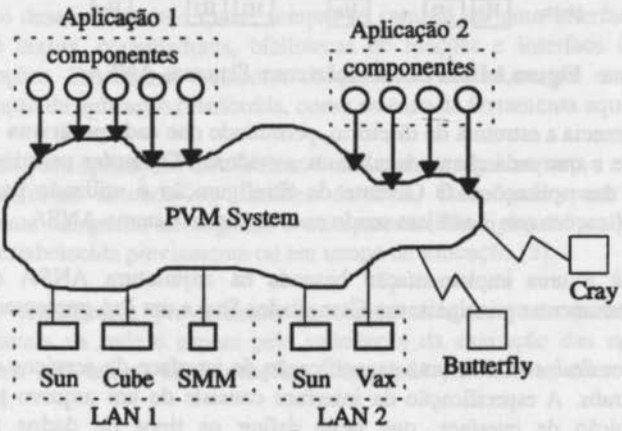

Figura 2 Modelo da Arquitetura PVM

Para o usuário iniciar o *daemon* sobre uma máquina, ele especifica um arquivo de entrada, que deve conter uma lista de estações que comporão a sua máquina virtual paralela. O *daemon* local ativa os demais *daemons* remotos que se comunicam através de *sockets*. Sobre esta plataforma de execução composta por *daemons* serão executadas as aplicações distribuídas.

Durante a execução, múltiplas instâncias de cada componente poderão ser ativadas. Para se tornar parte da máquina virtual, cada componente deverá registrar-se no *daemon*. O *daemon* local avisará os demais *daemons* para que atualizem suas tabelas de locação de componentes.

Todas as primitivas ativadas por um processo da aplicação ativam indiretamente o *daemon* local, repassando para ele as solicitações feitas pelo processo. O envio de mensagem é coordenado pelos diversos *daemons* da máquina virtual. Visto que cada *daemon* possui uma tabela de locação de componentes, o *daemon* local sabe para qual dos outros *daemons* ele deve enviar a mensagem.

Da mesma forma que o envio de mensagens, quando um *daemon* local receber uma mensagem, ele a entregará para o processo local apropriado, desde que este processo execute a primitiva *receive*.

2.3 SPD - Sistema de Programação Distribuída

SPD é um sistema de programação distribuída, para redes de estações de trabalho, desenvolvido no DCC/UFMG [3]. Este sistema deve ser executado em ambientes UNIX, onde se tenham disponíveis o protocolo TCP/IP e a biblioteca TLI.

O sistema é composto basicamente por três módulos: um servidor, uma biblioteca de funções e um disparador de aplicações. A arquitetura do sistema é ilustrada na figura 3.

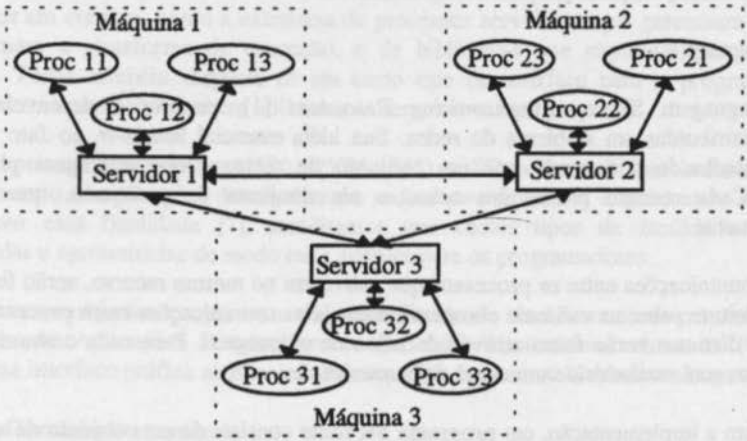

Figura 3 A Arquitetura do SPD

Um aplicação no SPD é um conjunto de programas (processos) que cooperam entre si para a solução de um dado problema. Os programas que compõe a aplicação trocam informações entre si através de canais de comunicação.

A distribuição da aplicação entre as máquinas que compõem a rede, bem como o modo de conexão dos diversos canais dos vários programas, devem ser definidos em um arquivo de configuração da aplicação, escrito pelo programador.

O servidor é o principal elemento do SPD e deve haver uma instância dele em cada máquina da rede, conectadas entre si. Pode-se dividir as funções do servidor em quatro: disparo dos processos, gerenciamento dos processos componentes da aplicação, comunicação entre processos e roteamento de mensagens.

A biblioteca de funções deve ser utilizada pelas aplicações e serve basicamente para o envio e recebimento de mensagens de forma síncrona ou assíncrona.

Licensed to UFSCAR Conta Avulsa

O disparador de aplicações é um programa que deverá ser executado pelo usuário para disparar a execução de uma aplicação no SPD.

A função básica do disparador de aplicações é interpretar as informações do arquivo de configurações e enviá-las para o servidor mestre da aplicação, que será o servidor local a mesma. Este servidor local passará a acompanhar, gerenciar e centralizar todas as informações referentes a execução da aplicação.

Quando a aplicação terminar, o servidor mestre enviará de volta ao disparador, os resultados da execução, para que possam ser retornados ao usuário

2.4 Linguagem SR

A linguagem SR ou *Synchronizing Resources* [1], permite o desenvolvimento de aplicações distribuídas em ambiente de redes. Sua idéia essencial baseia-se no fato de que um programa distribuído é formado por um conjunto de recursos que interagem por meio de operações. Cada recurso possui um nome e um conjunto de processos, que executarão concorrentemente.

As comunicações entre os processos que estiverem no mesmo recurso, serão feitas através da escrita e leitura sobre as variáveis compartilhadas. Já as comunicações entre processos situados em recursos distintos serão feitas através de troca de mensagens. Para cada comunicação entre dois processos será estabelecido um canal de comunicação.

Quanto a implementação, um programa SR fonte consiste de um conjunto de arquivos, no qual cada elemento deverá conter o código fonte para um ou mais recursos. Estes códigos fontes deverão ser compilados separadamente e carregados em alguma estação da rede de computadores.

Os arquivos objetos dos recursos, que deverão ser carregados sobre o mesmo processador, serão ligados juntos com um *software* suporte para formar um Módulo de Carga. Este *software* fornecerá suporte para as primitivas de troca de mensagens, além de fornecer a infra-estrutura básica para as comunicações entre processadores.

Nessa infra-estrutura, existe um par de processos especiais que tratam das operações remotas: o processo *In*, que recebe as mensagens remotas e o processo *Out*, que envia as mensagens remotas. Estes processos serão utilizados pelas primitivas de comunicação fornecidas pelo núcleo. Para cada canal de comunicação será utilizado um par de processos *In* e *Out*.

O programador deve ter habilidade para desenvolver os módulos componentes de sua aplicação distribuída e para utilizar as primitivas fornecidas pelo software de suporte.

Quando o usuário solicita a execução de um programa, os módulos de carga são carregados sobre os processadores da rede virtual, formando uma plataforma em tempo de

execução. A execução de um programa SR deverá começar quando todos os módulos de carga entrarem em execução em suas respectivas estações.

Todos os processos de cada módulo de carga estarão em uma lista local. Quando um processo terminar, ele chamará a primitiva *quit* do software de suporte, que o removerá desta lista. O término do programa é detectado quando cada módulo de carga terminar localmente.

3 Proposta de uma Ferramenta Para Aplicações Distribuídas

Pode-se notar que todas as ferramentas apresentadas possuem várias características arquiteturais em comum, como a existência de processos servidores, que gerenciam as aplicações e que formam a plataforma de execução, e de bibliotecas que exportam primitivas para as aplicações. Todas também dispõem de um certo tipo de interface para o programador. Estas características comuns são também utilizadas na ferramenta aqui proposta.

Embora as ferramentas ANSA, PVM, SPD e SR sejam bastante sofisticadas para o desenvolvimento de aplicações distribuídas em rede de *workstations*, e que outras ferramentas já existam com essa finalidade [7], acreditamos que certos tipos de facilidades possam ser desenvolvidas e apresentadas de modo mais simples para os programadores.

Com esse objetivo, propomos o desenvolvimento de uma Ferramenta para a Elaboração e Monitoração de Aplicações Distribuídas em Rede de *Workstations*, chamada FEMAD, que possuirá uma interface gráfica altamente interativa. Esta ferramenta é descrita a seguir.

3.1 Visão Geral da Ferramenta

A ferramenta FEMAD permitirá ao programador executar aplicações de forma distribuída, sobre uma rede de *workstations*. Cada aplicação será subdividida em módulos menores e distribuída pelas estações da rede, de forma balanceada, conforme mostra a figura 4.

A execução das aplicações será suportada por uma plataforma estabelecida em tempo de execução. Não haverá necessidade de definição de interface entre os módulos de cada aplicação, sendo a distribuição e o gerenciamento feitos de forma bastante transparente para os programadores.

Através desta ferramenta, o programador poderá configurar o ambiente de execução em alguns parâmetros tais como: quais estações farão parte da rede, fator de carga a ser usado no balanceamento e *time-slice* dos processos. O programador poderá também, através de uma interface gráfica, visualizar e interagir com a execução das aplicações.

Em sua estrutura, a FEMAD terá três módulos principais: uma Interface Gráfica do Usuário (IGU), um Supervisor de Execução Distribuída (SED) e um Núcleo Distribuído (ND).

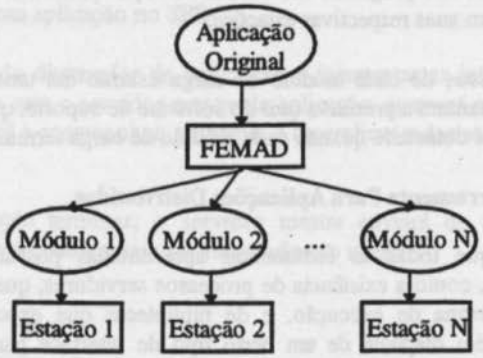

FIGURA 4 Visão Geral da Ferramenta FEMAD

3.2 Interface Gráfica IGU

A ferramenta FEMAD fornecerá uma interface gráfica para o usuário, chamada IGU, que deverá ser ativada em alguma estação da rede de *workstations* para que o usuário possa dispor das facilidades da ferramenta. Através desta interface, o usuário poderá ativar as seguintes funções: configuração, compilação, execução e visualização.

3.2.1 Módulo de Configuração

Através deste módulo, o programador poderá configurar alguns parâmetros do sistema. Isto permitirá que o usuário encontre o melhor desempenho para cada tipo de aplicação. Entre as principais características que poderão ser configuradas, destacamos:

- o número de estações ativas da rede, permitindo assim, a existência de uma rede virtual sobre a rede física existente. Uma vantagem é a possibilidade de deixar uma ou mais estações reservadas para outras atividades;
- o índice de carga* utilizado para a distribuição dos processos;
- a política de escalonamento da CPU. Conforme a política selecionada, o usuário poderá também definir a prioridade e o *time-slice* para os componentes das aplicações;
- o tipo de balanceamento, que poderá ser automático ou manual, onde o usuário deverá informar quais os processos que cada estação deverá executar. No balanceamento automático, a própria ferramenta distribuirá os processos de acordo com o índice de carga utilizado.

3.2.2 Módulo de Compilação

Através deste módulo, o usuário poderá compilar a sua aplicação para posterior execução, conforme a configuração preestabelecida ou conforme os valores *defaults*.

O programador deverá desenvolver sua aplicação de forma procedimental, conforme o modelo abaixo, tendo em mente que estes procedimentos (processos) serão executados de forma distribuída através da rede. Os processos que compartilharem a mesma *workstation* serão executados de forma multiprogramada.

```
proc_A(void)
{ ... envia_msg(msg,proc_B); ... recebe_msg(msg); ...}

proc_B(void)
{ ... recebe_msg(msg); ... envia_msg(msg,proc_A); ... }
```

Durante esta operação, o sistema criará para cada estação uma sub-aplicação. Cada sub-aplicação será composta por um conjunto de procedimentos da aplicação original, de acordo com a política de balanceamento configurada previamente, mais uma estrutura de controle fornecida pelo núcleo ND, que inclui estruturas de dados e códigos de processos especiais.

A idéia de dividir uma aplicação em sub-aplicações foi inspirada na linguagem SR, que utiliza o módulo de carga de forma semelhante.

3.2.3 Módulo de Execução

Para melhor esclarecer este módulo, vamos supor que uma aplicação seja composta pelos procedimentos A, B, C, D, e E, e que a rede virtual possua três estações. Suponhamos também que os procedimentos A, C e E devam ser distribuídos para a estação nº 2 e os procedimentos B e D para a estação nº 3, e que a interface IGU seja executada na estação nº 1. Assim, na fase de compilação, duas sub-aplicações serão geradas.

Após a solicitação de execução da aplicação, o supervisor SED deverá ser disparado localmente e as sub-aplicações deverão ser disparadas em suas respectivas estações. Quando as sub-aplicações entrarem em execução, o núcleo ND correspondente tomará conta da execução de cada sub-aplicação.

Todas as sub-aplicações deverão comunicar-se entre si e com o supervisor SED, a fim de gerenciar a execução da aplicação de uma forma global. O sistema ficará como na figura 5, na qual PU representa um processo de um outro usuário que poderia estar na estação remota.

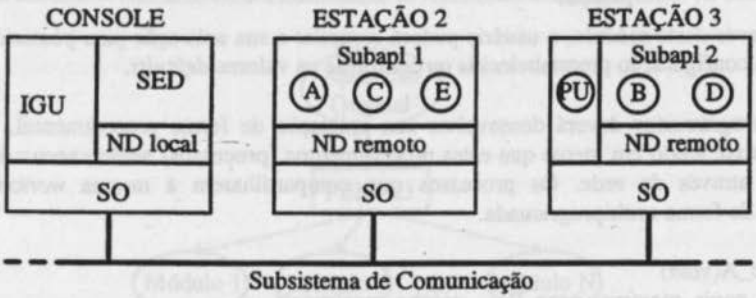

FIGURA 5 Aplicação Distribuída com FEMAD

3.2.4 Módulo de Visualização

Este módulo permitirá ao usuário visualizar a execução de sua aplicação distribuída, através de dois tipos de janelas: uma com o *lay-out* da rede de *workstations* e outra com as entradas e saídas de dados remotos.

Na janela com o *lay-out* da rede, o usuário poderá tanto interagir com a aplicação em tempo de execução, através de funções interativas para destruir, bloquear e acordar um processo, quanto utilizar uma primitiva para cancelar a execução da aplicação.

Durante o desenvolvimento de aplicações tolerantes a falhas, a operação de destruição será interessante para se analisar o desempenho do sistema. Assim, em sistemas como o de controle de processos, a destruição de um processo poderá simular uma falha, auxiliando no desenvolvimento de sistemas mais confiáveis.

A operação de bloqueio de um processo será especialmente adequada quando se estudar o comportamento de um sistema de tempo real. Isto permitirá a análise da conseqüência da demora em um dos processos vitais do sistema, ou ainda, qual a tolerância em termos de atraso, que o sistema poderá ter em cada um de seus processos.

A operação de cancelamento da aplicação será especialmente adequada quando a aplicação toda estiver comprometida, por ter sido mal projetada. Isto evitará que seus códigos distribuídos fiquem abandonados entre as estações, consumindo processamento desnecessário, e agilizará os testes com as aplicações.

3.3 O Supervisor SED

O supervisor SED será um programa executável que deverá ser ativado pela interface IGU, quando for solicitado a execução de uma aplicação distribuída pelo usuário. Ele será composto por um corpo principal adicionado das funções do núcleo distribuído ND.

Sua função principal será disparar as sub-aplicações através das estações correspondentes (assim como o disparador SPD) e monitorar as suas execuções. O relacionamento global deste supervisor com as outras entidades que participarão do sistema pode ser visto na figura 6.

O supervisor SED receberá as instruções da interface IGU para gerenciar as sub-aplicações, conforme o usuário desejar. Para permitir a monitoração da aplicação pelo usuário, o supervisor SED constantemente enviará informações atualizadas sobre a execução das aplicações para a interface IGU. Quando todos os processos de cada estação remota tiverem suas execuções finalizadas, cada sub-aplicação deverá avisar o supervisor SED, para que posteriormente o supervisor SED possa se desativar também.

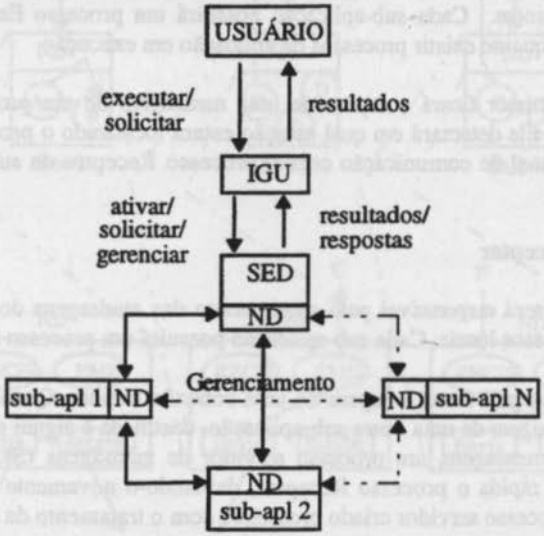

FIGURA 6 Relacionamento Global entre os Módulos da FEMAD

A existência de um processo supervisor para gerenciar a execução distribuída, é comprovada em outros sistemas: para o sistema PVM existe o *daemon* PVM; para o sistema SPD existe o servidor SPD e para a linguagem SR existe o software de suporte do módulo de carga.

3.4 O Núcleo Distribuído ND

Este módulo será constituído por todas as funções que fornecerão suporte para a criação e comunicação de processos, multiprogramação e todas as funções de gerenciamento da execução das aplicações. Ele será o responsável pela formação da plataforma em tempo de execução.

Basicamente, o núcleo ND será composto por um conjunto de códigos de processos especiais do sistema e um conjunto de primitivas, que serão discutidos em maiores detalhes nos itens seguintes.

3.4.1 O Processo Escalonador

Durante a execução de cada sub-aplicação, após terem sido ativados todos os processos do usuário e do sistema, o controle da execução será entregue para o processo escalonador. Este processo do sistema será responsável pelo escalonamento dos processos que estiverem em multiprogramação na sub-aplicação.

3.4.2 O Processo Emissor

Este processo do sistema será responsável pelo envio das mensagens dos processos locais para os processos remotos. Cada sub-aplicação possuirá um processo Emissor. Este processo ficará em execução enquanto existir processos da aplicação em execução.

O processo Emissor ficará a espera de uma mensagem de um processo local, para ser enviada remotamente. Ele detectará em qual estação estará localizado o processo destino. Então, ele estabelecerá um canal de comunicação com o processo Receptor da sub-aplicação destino e enviará a mensagem.

3.4.3 O Processo Receptor

Este processo será responsável pelo recebimento das mensagens dos processos remotos, enviadas para os processos locais. Cada sub-aplicação possuirá um processo Receptor.

O processo Receptor ficará a espera de uma conexão remota. Quando esta for efetivada, ele receberá uma mensagem de uma outra sub-aplicação, destinada à algum processo local. Então, ele criará, para cada mensagem, um processo servidor de mensagens (S). Isto será feito para liberar de forma mais rápida o processo Receptor, deixando-o novamente à espera de conexão remota, enquanto o processo servidor criado prossegue com o tratamento da mensagem.

A utilização dos processos Emissor e Receptor foi inspirada nos processos *In* e *Out* do Módulo de Carga gerado pela linguagem SR.

3.4.4 As Primitivas de Comunicação entre Processos

Para que os processos da aplicação distribuída possam se comunicar, o núcleo distribuído exportará duas primitivas básicas de comunicação: Envia_Msg( ) e Recebe_Msg( ). A figura 7 mostra uma visão geral da distribuição e execução de uma suposta aplicação composta pelos procedimentos A, B, C, D, E, F e G, enfatizando as trocas de mensagens.

A primitiva Envia_Msg() colocará a mensagem em um *buffer* local, caso o processo destino esteja executando localmente, ou repassará para o processo Emissor, que tratará das comunicações remotas de envio. A primitiva Recebe_Msg() poderá ser utilizada por um processo da aplicação, para receber uma mensagem de um outro processo. A mensagem será sempre obtida

de um *buffer* local, podendo ter sido colocada pelo próprio processo emissor, quando este for local, ou pelo processo Receptor, quando a comunicação for remota.

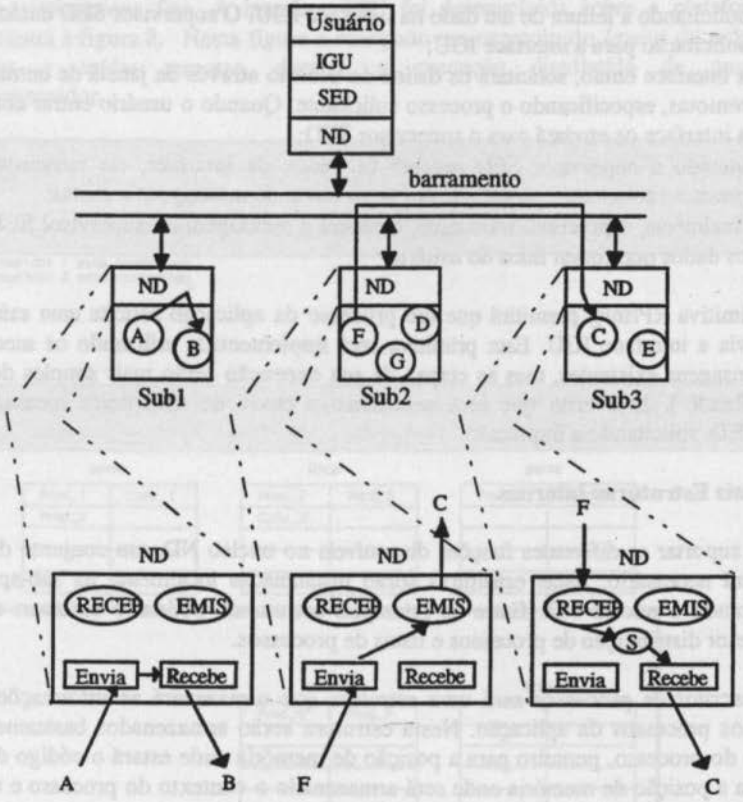

FIGURA 7 Visão Interna do Núcleo ND nas Trocas de Mensagens

3.4.5 As Primitivas de Entradas e Saídas de Dados Remotos

Para permitir que o usuário da aplicação possa ter o controle das entradas e saídas de dados manipuladas pelos processos distribuídos, será necessário a existência de funções apropriadas que permitam esse tratamento através da interface gráfica do usuário. Assim, o núcleo distribuído fornecerá para a aplicação duas funções para entrada e saída de dados: RRead( ) e RPrint( ), respectivamente.

A primitiva RRead() permitirá que um processo da aplicação solicite uma entrada de dados do usuário, via a interface IGU. Esta primitiva será implementada utilizando os mecanismos de troca de mensagens existentes, da seguinte maneira:

Unregistered Version of CVISION PDFCompressor

- o processo solicitante executará basicamente duas operações: um envio e um recebimento de mensagens;
- inicialmente, o processo solicitante enviará uma mensagem para o supervisor SED, solicitando a leitura de um dado na interface IGU. O supervisor SED então repassará a solicitação para a interface IGU;
- a interface então, solicitará os dados do usuário através da janela de entradas e saídas remotas, especificando o processo solicitante. Quando o usuário entrar com os dados, a interface os enviará para o supervisor SED;
- quando o supervisor SED receber os dados da interface, ele retransmitirá para o processo solicitante, através de um novo envio de mensagens remotas;
- finalmente, o processo solicitante receberá a mensagem do supervisor SED, contendo os dados que foram lidos do usuário.

A primitiva RPrint() permitirá que um processo da aplicação solicite uma saída de dados do usuário via a interface IGU. Esta primitiva será implementada utilizando os mecanismos de troca de mensagens existentes, mas as etapas de sua execução serão mais simples do que as da primitiva RRead( ), haja visto que será necessário o envio de uma única mensagem para o supervisor SED, solicitando a impressão.

3.5 Principais Estruturas Internas

Para suportar as diferentes funções disponíveis no núcleo ND, um conjunto de estruturas de dados será necessário. Estas estruturas serão instanciadas localmente na sub-aplicação que estiver utilizando o núcleo ND. Entre as principais estruturas, podemos destacar: descritor de processos, vetor distribuição de processos e listas de processos.

O descritor de processos será uma estrutura que armazenará as informações principais associadas aos processos da aplicação. Nesta estrutura serão armazenados basicamente: espaço para o nome do processo, ponteiro para a posição de memória onde estará o código do processo, ponteiro para a posição de memória onde será armazenado o contexto do processo e um ponteiro para mensagens recebidas. Todo processo possuirá um descritor próprio, que será alocado no momento de sua criação.

Como a distribuição lógica dos processos entre as sub-aplicações será feita de forma antecipada, será montada uma tabela que associará todos os processos e suas respectivas localizações na rede. Essa tabela será incluída no código de cada sub-aplicação, e será utilizada nas questões de roteamento.

Para gerenciar a execução multiprogramada dos processos em cada estação, existirão basicamente três tipos de listas de processos: uma lista de processos prontos, uma lista de processos suspensos por envio de mensagens e uma lista de processos suspensos a espera de mensagens. Estas listas serão estruturas encadeadas com ponteiros para os descritores dos processos.

copy of CVISION PDFCompressor

3.6 Implementação da Ferramenta

A ferramenta FEMAD está em fase de implementação no DCC/UFSCar, em um ambiente da rede de *workstations* Sun. A interface IGU foi desenvolvida sobre a plataforma XView, conforme mostra a figura 8. Nesta figura é mostrado um exemplo do *layout* da rede e da janela de entradas e saídas remotas, durante a execução distribuída de uma aplicação produtor/consumidor.

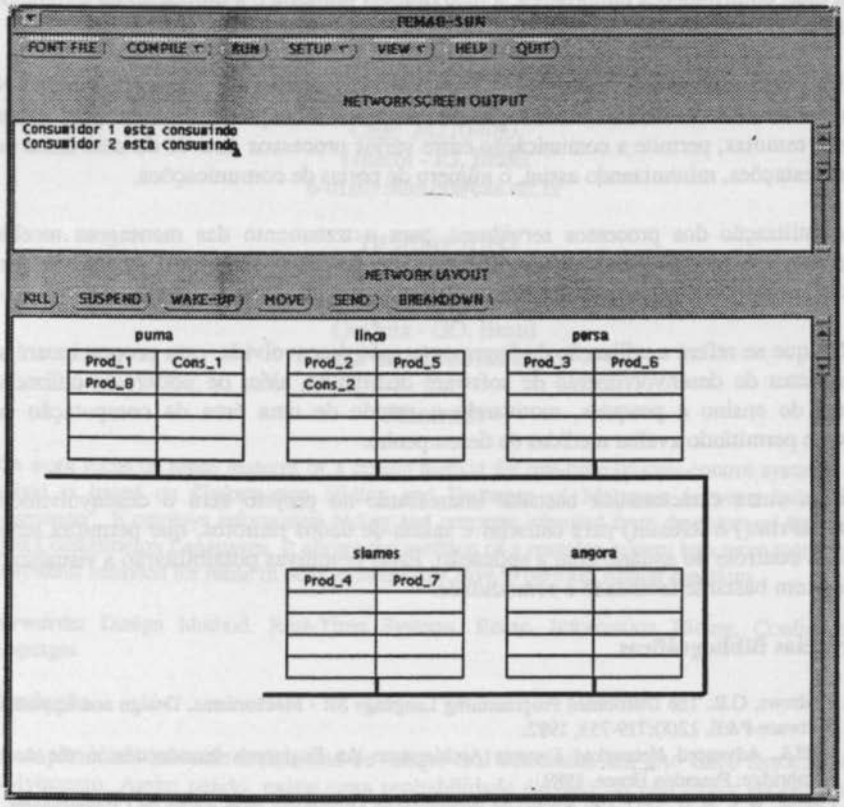

FIGURA 8 Interface Gráfica IGU Xview

4 Conclusão

A maioria das ferramentas existentes para o desenvolvimento de aplicações distribuídas implementa uma estrutura de grande porte, capaz de suportar o desenvolvimento de grandes aplicações distribuídas simultaneamente. Nestas plataformas, uma aplicação distribuída é composta por um conjunto de aplicações individuais que o programador deve desenvolver. A interação entre estas aplicações individuais é definida através de uma interface de configuração, que o

Evaluation Version DVISION PDFCompressor

programador também deve desenvolver. Este nível de complexidade é transportado para o programador durante o desenvolvimento de suas aplicações.

A ferramenta FEMAD possibilitará o desenvolvimento de aplicações de forma bastante simples, onde o programador desenvolverá uma única aplicação fonte, sem se preocupar com detalhes específicos do sistema. A proposta de desenvolvimento desta ferramenta permite-nos observar duas contribuições importantes: a metodologia utilizada e a utilização da ferramenta após desenvolvida.

No que se refere a metodologia, a forma de estruturação do núcleo distribuído ND é o principal ponto a ser destacado. A utilização dos processos Receptor e Emissor, para as trocas de mensagens remotas, permite a comunicação entre vários processos através de uma única conexão entre duas estações, minimizando assim, o número de portas de comunicações.

A utilização dos processos servidores, para o tratamento das mensagens recebidas de estações remotas, permite uma rápida liberação do processo Receptor, agilizando assim, as comunicações remotas, já que estas podem se tornar o principal gargalo do sistema.

No que se refere a utilização da ferramenta após desenvolvida, esta proporcionará suporte para ambientes de desenvolvimento de software distribuído, além de poder ser utilizada como ferramenta de ensino e pesquisa, motivando o estudo de uma área da computação bastante complexa, e permitindo avaliar medidas de desempenho.

Uma outra característica bastante interessante no projeto será o desenvolvimento das primitivas RPrint() e RRead() para entradas e saídas de dados remotos, que permitirá uma maior interação e controle do usuário com a aplicação. Estas primitivas possibilitarão a visualização dos resultados com bastante facilidade e comodidade.

5 Referências Bibliográficas

[1] Andrews, G.R. The Distributed Programming Language SR - Mechanisms, Design and Implementation. Software-P&E, 12(8):719-753, 1982.

[2] ANSA. Advanced Networked Systems Archicteture: An Engineer's Introduction to the Architeture. Cambridge: Poseidon House, 1989.

[3] Caldas, W.S. et al. SPD: Um Núcleo de Programação Distribuída para Redes de Computadores. In: IV-SBAC-PAD, 1992, São Paulo:1992, p.444-457.

[4] Geist, G.A. & Sunderam, V.S. Network-Based Concurrent Computing on the PVM System. *Concurrency-P&E*, 4(4):293-311, 1992.

[5] Gonçalves, R.A.L, Ferramenta para Elaboração e Monitoração de Aplicações Distribuídas em Rede de *Workstations* SUN. São Carlos, SP: UFSCar, 1994, 119p. Dissertação (Mestrado em Sistemas Distribuídos) - Programa de Pós-Graduação em Ciência da Computação, UFSCar, 1994.

[6] Sunderam, V.S. PVM: A Framework for Parallel Distributed Computing. Concurrent: P&E, 2(4):315-339, 1990.

[7] Turcotte, L.H. A Survey of Software Enviroments for Exploiting Network Computing Resources. Mississippi: Center for Computational Field Simulation, 1993.