# P-RIO : Construção Gráfica e Modular de Programas Paralelos e Distribuídos

Enrique Vinicio Carrera E.
Dpto. Eng. Elétrica - PUC/Rio
Cx. Postal 38063 - CEP 22453-900,
Rio de Janeiro - RJ
vinicio@ele.puc-rio.br

Orlando Loques Julius Leite
Pós-Graduação em Computação Aplicada e Automação
e Dpto. de Eng. Elétrica - UFF
Rua Passo da Pátria, 156 - CEP 24210-240, Niterói-RJ
{loques, julius}@caa.uff.br

## Resumo

O presente trabalho descreve a metodologia de construção de *software* embutida no ambiente P-RIO do ponto de vista da modularização, paralelização e configuração gráfica de aplicações. É mostrada, a título de exemplo, a paralelização de um sistema de visão artificial que utiliza redes neurais artificiais para o reconhecimento de imagens. Esta aplicação, que tem grandes requerimentos de tempo de máquina, teve seu desempenho otimizado aproveitando as abstrações nos níveis de programação e configuração embutidas no ambiente. Apresenta-se também a interface gráfica, integrada ao ambiente, que provê facilidades para a configuração, depuração e gerenciamento, propiciando grande eficiência no processo de desenvolvimento de aplicações concorrentes.

## Abstract

This paper describes the software construction methodology embedded in the P-RIO environment, from the point of view of modularity, parallelization and graphic configuration of applications. It is presented, as an example, the parallelization of an artificial vision system that uses artificial neural networks for image recognition. This application, that requires a lot of CPU time, had its performance optimized by the use of the programming and configuration level abstractions embbeded in the environment. It is also shown, the graphic interface, integrated to the environment, that provides facilities for configuration, debugging and management, allowing high efficiency in the development process of concurrent applications.

## 1. INTRODUÇÃO

O Ambiente ***P-RIO*** (*Parallel, Reconfigurable Interconnectable Objects*)[1], em desenvolvimento por um grupo de pesquisadores e alunos de pós-graduação no DEE-PUC/RJ e no CAA-UFF, oferece facilidades para a programação e a configuração de componentes numa arquitetura distribuída. Este ambiente é centrado em uma metodologia que separa a construção dos componentes de uma aplicação, através da programação, da construção da própria aplicação, através da configuração. A metodologia por si só não impõe restrições sobre a semântica ou a linguagem utilizada

na construção dos componentes, assim como sobre o aspecto distribuído ou não da aplicação.

Os componentes do sistema são denominados módulos e são compostos por uma parte de código e dados, e um ou mais pontos de conexão externos chamados de conectores. Os módulos podem ser, por sua vez, componentes de outros módulos e não há restrição quanto à quantidade de módulos componentes de um módulo composto e, nem mesmo, quanto à quantidade de níveis de aninhamento dos módulos. A programação dos módulos não difere da programação tradicional baseada em bibliotecas externas de funções. Entretanto, a composição e a interligação dos componentes da aplicação são especificadas por uma linguagem especializada de configuração[2].

O mapeamento dos conceitos associados à metodologia em representações gráficas é imediato. Desta forma, nossa proposta inclui uma interface gráfica para a construção e gerenciamento de sistemas paralelos e distribuídos. Esta interface permite que sistemas sejam compostos graficamente em alto nível, através da seleção de seus componentes, definição de suas interligações e alocação física dos mesmos. O acompanhamento do funcionamento para fins de teste e depuração, e controles de operação da aplicação, tais como, parada de funcionamento de componentes, realocação de componentes e reconfigurações diversas, também são suportados.

O Ambiente P-RIO está construído sobre o sistema **PVM** (*Parallel Virtual Machine*)[3], garantindo-se a portabilidade do mesmo a vários sistemas operacionais e a dezenas de arquiteturas diferentes[4]. Em outras palavras, as máquinas individuais envolvidas no ambiente podem ser multiprocessadores de memória compartilhada ou memória local, supercomputadores vetoriais, estações gráficas especializadas, ou estações de trabalho escalares, e podem estar interligadas através de uma variedade de redes, tais como Ethernet, Token Ring, FDDI, etc.

A aplicação descrita neste trabalho é o reconhecimento das imagens obtidas por uma câmara de vídeo, mediante a utilização de um sistema baseado em duas etapas: Pré-processamento e Classificação por Redes Neurais[5]. A etapa de pré-processamento está baseada em procedimentos comuns da área de processamento digital de sinais (filtros, transformações, convoluções, etc.), o que implica em uma elevada utilização de UCP, assim como uma alta repetição das operações dentro da matriz bi-dimensional que representa a imagem a classificar.

Na seção 2 deste artigo são discutidas as características mais relevantes apresentadas pela metodologia que implementa o Ambiente P-RIO. Na seção 3 são dados alguns detalhes sobre a programação e configuração das aplicações em P-RIO. A seção 4 mostra a estrutura na qual se apoia a aplicação de reconhecimento de imagens e o processo de paralelização da mesma, assim como alguns testes de desempenho. Na seção 5, são descritas as características da interface gráfica que atua como o gerente de configuração do ambiente. Finalmente, na seção 6, são apresentadas algumas conclusões e perspectivas futuras de trabalho.

## 2. METODOLOGIA DE CONSTRUÇÃO DE SOFTWARE

A metodologia que permeia o ambiente incentiva o uso de princípios já consolidados e universalmente aceitos na área de engenharia de sistemas de computação. A desejada característica de modularidade, e assim seus benefícios diretos, dentre eles o potencial de re-utilização, advém naturalmente. De acordo com o modelo utilizado, assume-se que uma aplicação é realizada pela composição de vários módulos ou componentes, onde cada um deles requer recursos ou provê facilidades para outros módulos. Após a fase de concepção, onde os componentes de uma aplicação são inicialmente definidos, os módulos correspondentes podem ser projetados independentemente e posteriormente reunidos para construir o sistema em sua forma final. Os módulos já construídos e testados podem ser armazenados em um repositório (biblioteca de módulos) e re-usados a qualquer tempo na composição de novos sistemas aplicativos.

Sem dúvida, o sucesso desta proposta requer soluções especializadas para cada domínio de aplicação. Embora na prática os domínios possam se entrelaçar, selecionando-se um específico, um conjunto de módulos especializados pode ser construído e usado na composição das aplicações, à medida que estas forem surgindo. Com o decorrer do tempo, em conseqüência de novas necessidades e também da experiência adquirida, módulos antigos podem ser aperfeiçoados e outros módulos construídos, enriquecendo o conjunto disponível e facilitando a composição de novas aplicações.

O modelo adotado permite distinguir as relações de implementação das relações de interligação e interação existentes entre os módulos de um programa. Esta é uma característica fundamental, já que quando as pessoas projetam sistemas elas tipicamente provêem uma descrição arquitetural consistindo de um conjunto de componentes funcionais e um conjunto de conexões que indicam as interações entre aqueles componentes[6]. Desta forma, ao invés de um sistema ter componentes ligados na própria descrição do código, como ocorre na programação tradicional, impõe-se uma fase de descrição da configuração, composta de informação sobre os componentes usados e suas interconexões. Isto cria uma forte distinção e isolamento entre os componentes propriamente ditos e os seus mecanismos de interligação, simplificando a especificação, teste e reutilização dos mesmos.

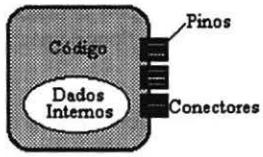


*Figura 1. Módulo Básico*

Conforme descrito anteriormente, a metodologia P-RIO define e se baseia em dois conceitos fundamentais para a construção de um sistema: os módulos e os conectores[7]. A estrutura básica de construção de um sistema é o módulo. O conceito de módulo se

divide, por sua vez, em dois, dependendo se este é uma unidade operacional (instância) ou se é um tipo (classe) usado como fôrma para a criação das unidades operacionais. Uma classe de módulo é uma entidade que contém uma parte de código e dados e um ou mais pontos de interconexão externa. Cada instância de um módulo é uma imagem de uma classe que executa o código e possui o seu próprio estado e dados internos (Figura 1). Esta abordagem, baseada em objetos, compreende o módulo como um objeto onde uma instância do módulo é equivalente a uma instância de um objeto.

Uma instância sofre uma ação importante para a arquitetura: a configuração. A configuração consiste na disposição e organização das instâncias e suas relações, ou seja, sua interconexão. Em um modelo tradicional de programação as ligações entre os elementos são estáticas, sendo fixadas durante a programação. Numa metodologia baseada em objetos tais ligações são potencialmente dinâmicas, realizadas em uma fase posterior à criação do sistema, ou mesmo mutáveis durante a sua operação. Esta estrutura exige um método bem definido para expressar as interações entre os módulos, que vai ser realizada através dos pontos de interconexão ou conectores.

A função dos conectores é organizar e estruturar a interface do módulo. Estes elementos concentram as atividades de comunicação e são fundamentais à configuração, pois são os pontos de "colagem" de um módulo com outro. Em outros termos, os conectores são os meios de interação entre um módulo e o seu mundo exterior. Todos os serviços usados pelo módulo ou oferecidos por ele ao sistema são acessíveis, unicamente, através dos seus conectores. Cada conector contém um ou mais pinos de ligação. Um pino é o elemento básico por onde se realiza a comunicação, pois é a entidade através da qual os métodos ou procedimentos de um módulo irão se comunicar externamente. Pode-se visualizar os pinos como os elementos primários ativos da comunicação num módulo, enquanto que os conectores tem a função de agrupar os pinos para fins de configuração. Um conector possui um tipo ou classe que depende dos pinos nele contidos. Por sua vez, cada pino está caracterizado pelo tipo de dados e direção das mensagens, e pelo tipo de transação associada.

Desta forma, a configuração é a ligação de dois ou mais conectores, realizando, assim, a interconexão dos módulos a eles associados. Um módulo somente pode interagir diretamente com outro módulo ao qual esteja conectado. Além de expressar a interação potencial entre dois ou mais módulos, o conceito de conector pode ser usado para verificar a consistência da configuração, pois uma conexão só pode ser realizada entre conectores que contenham assinaturas de tipos compatíveis para uma interseção total ou parcial dos seus pinos.

Um conceito também associado ao de conexão é o de "grupo". Uma conexão de um conector para um grupo implica que ele está ligado a todos os outros conectores que também pertençam ou estão conectados a esse grupo (Figura 2). Uma mensagem enviada por um membro do grupo é recebida por todos os seus membros que queiram recebê-la. Esta conexão especializada tem uso em diversas aplicações que requerem comunicação via disseminação de mensagens tipo *broadcast* ou *multicast*, sincronização mediante barreiras ou encontros (*rendezvous*), etc. Deve ser notado que o protocolo usado para a disseminação pode ser especificado no domínio da configuração. Além

disso, os grupos podem ser explicitamente nomeados, facilitando o gerenciamento dinâmico dos mesmos.

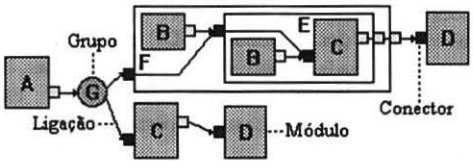


*Figura 2. Grupos e Encapsulamento de Módulos*

Um objetivo da metodologia implementada pelo ambiente P-RIO é fornecer um ambiente não restritivo para a modelagem. Desta forma, utiliza-se um conceito genérico que permite diversas abordagens mais específicas. Este conceito genérico é a composição de módulos, que permite a criação de configurações parciais, denominadas classes compostas, a partir de classes básicas anteriormente definidas, para fins de encapsulamento e reutilização[8] (Figura 2). Todas as classes definidas (básicas ou compostas) podem ser instanciadas e utilizadas separadamente, podendo ser executadas concorrentemente com as demais.

## 3. PROGRAMAÇÃO EM P-RIO

A interface de programação disponível para a construção de módulos, segundo o padrão requerido pela metodologia P-RIO, é implementada através de uma biblioteca de funções. O código das primitivas é simples e eficiente, de modo a não afetar o desempenho oferecido pela plataforma PVM original. Medidas efetuadas demonstraram que o *overhead* introduzido é desprezível, sendo que em alguns casos, para funções semelhantes, o P-RIO apresenta melhor desempenho[1]. Dentro do conjunto de primitivas de programação não existem chamadas restritivas e, como um auxílio à depuração[9], se um erro ocorre em qualquer destas chamadas, uma mensagem auto-explicativa é gerada. Adicionalmente, um pequeno conjunto de primitivas é disponível para definir os conceitos de modularização e interfaceamento, no nível da programação.

Para fins de comunicação, o ambiente P-RIO suporta duas primitivas básicas: síncrona (tipo RPC) e assíncrona, com funções auxiliares de temporização (*timeout*) e propagação de exceções. Além disso, são disponíveis, como em outras metodologias de programação concorrente contemporâneas, construções do tipo *select* com guardas, baseados em expressões lógicas. Deve ser ressaltado que estas primitivas permitem que a programação seja realizada em um nível mais alto que o PVM básico. Desta forma, o programador não precisa se preocupar com detalhes, tais como a alocação de *buffers* e gerenciamento (armação e teste) de *flags*, podendo concentrar esforços na resolução de seu problema específico. Estas primitivas estão documentadas em estilo semelhante as do PVM, facilitando a transição de projetistas já familiarizados com este sistema.

No nível de suporte de operação toda atividade de comunicação é implementada utilizando-se as facilidades fornecidas pelo sistema PVM, dentro do objetivo de prover máxima portabilidade ao ambiente P-RIO. Uma das características relevantes, no nível de configuração, é possibilitar a seleção do protocolo de comunicação entre dois pinos de uma conexão[10], podendo ser usados vários protocolos padronizados ou específicos. Atualmente, permite-se o uso dos protocolos UDP e TCP, além de disseminação não confiável opcionalmente associada à sincronização por barreiras. Em máquinas usando arquiteturas especializadas de interconexão, como, por exemplo, o sistema multiprocessador SP/2 da IBM, protocolos particulares serão transparentemente suportados.

A interface apresentada ao usuário pelo ambiente é tão simples quanto possível, de forma a permitir que engenheiros e cientistas, sem um grande treinamento prévio, possam construir sistemas computacionais relativamente complexos. Além disso, a concepção do suporte de operação do ambiente P-RIO permite que todas as ferramentas de monitoração e depuração originais do PVM, e outras específicas ao ambiente, funcionem como ferramentas auxiliares para este sistema.

Como descrito, a metodologia requer um conjunto de primitivas para programação, modularização e configuração. Este conjunto permite expressar propriedades equivalentes às obtidas através de linguagens orientadas a objetos[7]. No nível da programação, primitivas para a definição de classes de módulos e conectores são disponíveis[11]. No nível da configuração, uma linguagem suporta a composição e o gerenciamento de aplicações, facilitando a construção de sistemas e a reutilização de módulos. Em especial, as facilidades de configuração simplificam a obtenção de requisitos típicos das aplicações, tais como distribuição de carga, tolerância a falhas, paralelização dinâmica, etc. Na próxima seção, mostra-se o uso da linguagem de configuração. Na seção seguinte apresenta-se a interface gráfica disponível para a construção, gerenciamento e depuração de aplicações.

## 4. EXEMPLO DE PARALELIZAÇÃO

O uso de redes neurais em aplicações de reconhecimento e classificação de padrões tem revolucionado a área de visão por computador, devido à capacidade que estas estruturas têem de extrair informações de conjuntos de dados complexos. Nesta seção, utilizamos um classificador "inteligente" baseado em redes neurais invariante à translação, rotação e ao escalonamento da imagem a reconhecer, para demonstrar as capacidades de modularização e paralelização do ambiente P-RIO.

O sistema pode ser dividido em dois blocos: o Pré-processamento e a Classificação (Figura 3). A função de pré-processamento, que foi modularizada com a utilização do ambiente P-RIO, é apresentada em detalhes na Figura 4. O processo parte de uma imagem em formato *raster* de 256x256 pixels com 256 níveis de cinza. Nesta imagem é aplicado um filtro de mediana para atenuar o ruído espúrio, e é feita a detecção das bordas através do Algoritmo de Sobel para obter a posição do objeto no campo de visão. Paralelamente, aplica-se à imagem inicial um processamento homomórfico por multiplicação para eliminar distorções causadas pela fonte de iluminação. Após esta

etapa, e levando-se em conta os resultados obtidos na etapa de detecção de bordas, é feita a transformação da imagem para coordenadas polares, de modo a facilitar a obtenção dos coeficientes harmônicos circulares[12] da imagem. Finalmente, aplica-se a Transformada de Mellin[13] aos coeficientes harmônicos circulares, obtendo-se uma matriz de 64 K coeficientes complexos que serão apresentados ao classificador. O classificador, por sua vez, é uma rede neural que utiliza o modelo Back-Propagation[14] com uma taxa de aprendizado variável ao longo do treinamento.

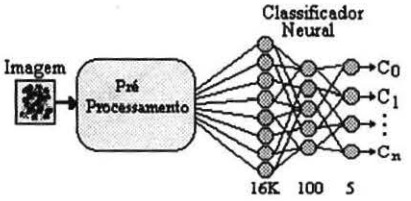


*Figura 3. O Classificador de Imagens*

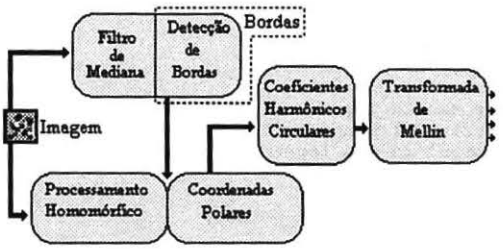


*Figura 4. A Etapa de Pré-processamento*

Em uma primeira versão desta aplicação, pode-se verificar que na etapa de pré-processamento a tarefa de detecção de bordas (marcada por linha pontilhada) consome aproximadamente 80% do tempo total de máquina. Visando otimizar o desempenho, foi investigada a paralelização desta etapa através das facilidades do ambiente P-RIO. Com este objetivo, uma primeira classe de módulo (*pré-processamento*) que executa todo o pré-processamento requerido para o reconhecimento de imagens, com exceção da tarefa de detecção de bordas, foi definida. A tarefa de detecção de bordas, em uma parte da matriz original, foi então delegada a uma segunda classe (*bordas*). Uma instância desta classe pode ser replicada em cada processador disponível no sistema, permitindo a execução concorrente da aplicação com poucas mudanças no código original. A Figura 5, apresenta a configuração da aplicação em um sistema com cinco processadores. A listagem do arquivo de configuração para esta aplicação, escrito na respectiva linguagem de configuração[2], é mostrada a seguir:

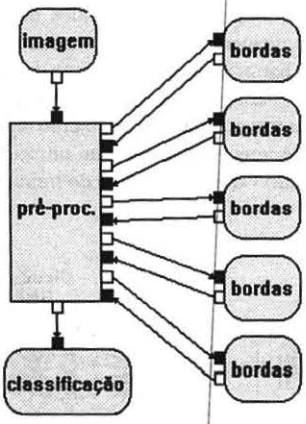


*Figura 5. Reconhecimento de Imagens Utilizando R.N.A.*

```
class Artificial_Vision {
  station EP[6] = {
    "Hydra", "Cygnus", "Gemini", "Orion", "Octans", "Hydrus"
  };
  connector Image_Type {
    pin Image out pin_output;
  };
  connector Matrix_Type {
    pin Matrix out pin_output;
    pin Control in pin_input;
  };
  connector Coef_Type {
    pin Coeficient out pin_output;
  };
  class Vision_Module {
    class Image {
      connector       Image_Type channel_out;
      extern code C   "imagem";
    } module_i;
    class Pre_processing {
      pair connector Image_Type  channel_in;
      connector      Matrix_Type channel_out[5];
      connector      Coef_Type   channel_sol;
      extern code C  "pre_processamento";
    } module_p;
    class Edges {
      pair connector Matrix_Type channel_in;
      extern code C  "bordas";
    } module_e[5];
    class Classifier {
      pair connector Coef_Type channel_in;
      extern code C  "classificador";
    } module_c;
    instantiate module_i at EP[5];
    instantiate module_p at EP[5];
    for i = 0 to 4
      instantiate module_e[i] at EP[i];
    instantiate module_c at EP[5];
    link module i.channel out module p.channel in;
```

```
    for i = 0 to 4
      link module_p.channel_out[i] module_e[i].channel_in;
    link module_p.channel_sol module_c.channel_in;
  } application;
  instantiate application;
};
```

Medidas quantitativas que mostram o aumento de desempenho alcançado através da paralelização, são apresentadas a seguir. Os tempos mostrados são os correspondentes à fase de pré-processamento exclusivamente. Atualmente, a fase de classificação precisa de cerca de 4 minutos de execução. Uma melhora, proposta para o futuro, é a paralelização da rede neural, de modo a reduzir ainda mais os tempos de treinamento e reconhecimento do sistema de classificação de imagens.

| No. Processadores | Tempo | *Aceleração* |
|---|---|---|
| 1 | 2'48" | 1.0 |
| 2 | 1'25" | 1.9 |
| 3 | 1'02" | 2.7 |
| 4 | 0'44" | 3.8 |
| 5 | 0'36" | 4.6 |

*Tabela 1. Desempenho*

O tempo de pré-processamento de 2'48" foi reduzido a 36" utilizando-se 5 estações de trabalho SUN 2. Devido às características da aplicação, que executa um processamento bi-dimensional isolado sobre uma matriz que representa a imagem a reconhecer, e considerando que o custo de comunicação é relativamente baixo, o aumento de desempenho é quase linear. Pode-se facilmente ampliar o número de módulos (e processadores) exclusivamente no nível da configuração, sem nenhuma mudança no código das classes componentes do sistema. Cabe ressaltar que, além do menor tempo de execução, a quantidade de memória requerida numa estação é menor, devido à distribuição do código e dados.

Outros exemplos de programas concorrentes que já foram implementados utilizando o ambiente P-RIO são, entre outros: cálculo de $\pi$, resolução da equação de Poisson e o problema dos filósofos comilões.

## 5. A INTERFACE GRÁFICA

Para a configuração de uma aplicação, o ambiente tem um módulo gerente de configuração que atua como o console do sistema. Este console provê a interface com o usuário permitindo a configuração, monitoração e depuração das aplicações. Além de iniciar a execução dos *daemons* que implementam o suporte de operação do sistema, este módulo tem que traduzir os comandos do usuário em chamadas P-RIO[15]. Uma vez configurada e em execução a aplicação, o console deve cuidar da captura das mensagens provenientes do sistema, podendo também executar atividades de reconfiguração dinâmica.

Atualmente existem duas classes de console utilizáveis no ambiente P-RIO. A primeira é um console de baixo-nível que traduz cada linha de comandos, expressa em uma linguagem intermediária de configuração[16], em primitivas P-RIO enviadas ao suporte de operação. Esta linguagem intermediária é gerada, interativamente, a partir de um preprocessamento de descrições de configuração expressas em alto nível, como no exemplo apresentado na seção 4. Esta facilidade, embora bastante útil, dificulta o aproveitamendo do potencial oferecido pela metodologia. Como uma segunda opção, foi desenvolvida uma interface gráfica, que permite a visualização completa do sistema, facilitando as tarefas de configuração, monitoração e depuração dos componentes da aplicação. Um detalhe desta interface é apresentado na Figura 6.

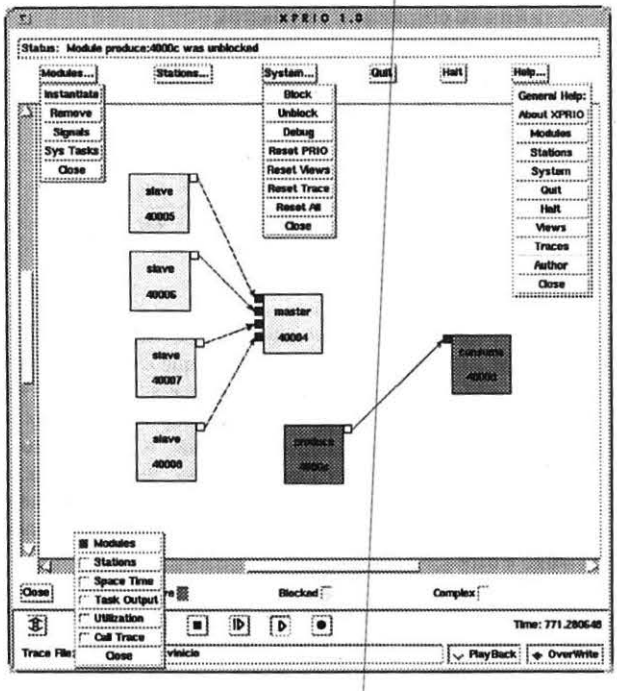


*Figura 6. Detalhe da Interface Gráfica P-RIO*

A interface gráfica permite a instanciação e alocação física, o controle da execução e a remoção de módulos, assim como também a interligação de conectores e pinos com simples comandos de *mouse*. Para fim de configuração e re-utilização, os módulos podem ser previamente programados e armazenados em uma biblioteca personalizável. A interface suporta o conceito de encapsulamento permitindo criar graficamente classes simples e compostas. A compactação e descompactação de classes compostas é suportada, permitindo que o projetista inspecione a constituição interna das classes e simplificando a exposição de sistemas complexos. A operação das instâncias de módulos, incluindo suas interações e saídas do tipo *print-like*, podem ser visualizadas em tempo de execução. Mediante a integração de várias características do XPVM[3] , ferramenta de

depuração embutida no sistema PVM, à nossa interface gráfica, é também possível depurar o ambiente através da visualização da troca de mensagens entre os módulos e do estado de execução de cada um deles. Esta interface foi implementada através da ferramenta Tcl/Tk[17], sendo também amplamente portável.

## 6. CONCLUSÃO

Neste artigo foram apresentadas as características mais relevantes do Ambiente P-RIO. Este ambiente é centrado em uma metodologia de *software* que privilegia a construção de sistemas concorrentes através da composição de módulos, estando disponível em um grande número de arquiteturas de *hardware*. Ele pretende tornar a programação paralela e distribuída mais acessível a programadores não especialistas e, também, oferecer conceitos e mecanismos de construção de *software* baseados no paradigma de configuração, que incentivam a modularidade e reusabilidade do *software*.

Foi apresentado, em forma de exemplo, um sistema de reconhecimento de imagens utilizando redes neurais baseado na metodologia P-RIO. Este sistema aproveita as facilidades de distribuição e modularização do ambiente para executar em paralelo os diferentes módulos da aplicação. Os resultados de um teste de desempenho confirmam o aumento de rendimento que pode ser alcançado.

Nossa experiência demonstra que o ambiente P-RIO facilita a construção e o suporte de aplicações paralelas e distribuídas, além de prover uma metodologia de programação concorrente simples e flexível. Sua aplicação seria particularmente útil na área de processamento científico onde paralelismo e facilidades de modularização sejam requeridos. Dentro de um domínio específico de aplicação, bibliotecas de módulos poderiam ser desenvolvidas facilitando a realização de experimentos e testes. O mapeamento dos conceitos associados à metodologia em representações gráficas simplifica a utilização do ambiente. Neste sentido, sua interface gráfica permite que sistemas sejam compostos, depurados e gerenciados em alto nível através do uso de recursos visuais. Esta interface está operacional, provendo atualmente todas as facilidades descritas neste artigo. O ambiente está disponível para distribuição a pessoas interessadas em usá-lo para fins de ensino e pesquisa.

## 7. BIBLIOGRAFIA

[1] Carrera E., Loques O., Leite J., *P-RIO: A Metodologia RIO sobre PVM*, 13° ,Simpósio Brasileiro de Redes de Computadores, Belo Horizonte, maio 1995.

[2] Malucelli V., *Ambiente Rio - Linguagem de Configuração*, Relatório Técnico DEE-PUC/RJ, Rio de Janeiro, maio 1994.

[3] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V., *PVM: Paralell Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press, agosto 1994.

[4] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V., *PVM 3 Users's Guide and Reference Manual*, ORNL/TM-12187, EUA, setembro 1994.

[5] Carrera E., Perelmuter G., Vellasco M., Pacheco M., *Image Classification Using Artificial Neural Networks*, International Conference on Education, Practice, and Promotion on Computational Methods in Engineering Using Small Computers (EPMESC V), abril 1995.

[6] Allen R., Garlan D., *Beyond Definition/Use: Architectural Interconnection*, Workshop on Interface Definition Languages, Portland - EUA, janeiro 1994.

[7] Werner J., Loques O., *Ambiente RIO: Metodologia e Suporte para Sistemas Configuráveis*, XX SEMISH da Reunião Anual da Sociedade Brasileira de Computação, Florianápolis, setembro 1993.

[8] Ning J., Miriyala K., Kozaczynski W., *An Architecture-driven, Business-specific, and Component-based Approach to Software Engineering*, III International Conference of Software Reuse, Rio de Janeiro, novembro 1994.

[9] Hong W., Black J., Manning E., *A Framework for Distributed Debugging*, IEEE Software, janeiro 1990.

[10] Sztajnberg A., Loques O., *O Sistema de Comunicação Multi-Protocolo do Ambiente RIO*, V Simpósio Brasileiro de Computadores Tolerantes a Falhas, São José dos Campos, outubro 1993.

[11] Carrera E., *P-RIO: Manual do Usuário*, Relatório Técnico DEE-PUC/RJ, Rio de Janeiro, dezembro 1994.

[12] Oppenheim A., Schafer R., *Discrete-Time Signal Processing*, Prentice Hall, 1989.

[13] Wu R., Hwang R., Stark H., *Pattern Recognition in Practice II*, Proceedings of the Eighth International Conference on Pattern Recognition, 1986.

[14] Rumelhart D., McClelland J., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, The MIT Press, 1988.

[15] Carrera E., *Metodologia para a Construção de Sistemas Configuráveis Acima de PVM*, Dissertação de Mestrado DEE-PUC/RJ, Rio de Janeiro, 1995.

[16] Sztajnberg A., Malucelli V., *Manual do Usuário do Ambiente RIO*, Relatório Técnico DEE-PUC/RJ, Rio de Janeiro, abril 1994.

[17] Ousterhout J., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.