UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Depuração de Programas Paralelos – Projeto de uma interface intuitiva

por

DENISE STRINGHINI

Tese submetida à avaliação, como
requisito parcial para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Philippe O. A. Navaux
Orientador

Porto Alegre, janeiro de 2003.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Stringhini, Denise

Depuração de programas paralelos – Depuração de Programas Paralelos – Projeto de uma interface intuitiva/ por Denise Stringhini. - Porto Alegre: PPGC da UFRGS, 2003.
101 p.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, 2003. Orientador: Navaux, Philippe O. A.

1. Depuração paralela. 2. Visualização de processos paralelos. 3. Seleção de processos paralelos. 4. Programação paralela. 5. *Clusters* de PCs. I. Navaux, Philippe O. A. . II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitor Adjunto de Pós-Graduação: Prof. Jaime Evaldo Fensterseifer
Diretor do Instituto de Informática: Prof. Philippe O. A. Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Meus sinceros agradecimentos ao Prof. Navaux por ter me dado a oportunidade de realizar este curso, pelas palavras de incentivo não só durante o doutorado, mas durante os onze anos em que estive sob sua orientação neste instituto (incluindo o trabalho de mestrado e bolsas de pesquisa) e por ser um professor sempre disposto abrir portas a seus alunos com grande generosidade.

Agradeço também ao Prof. Jacques Chassin-de-Kergommeaux, que me orientou durante o doutorado-sanduíche na cidade de Grenoble, assim como a todos os amigos que encontrei e fiz por lá. Um agradecimento especial ao casal que lá me hospedou: minha grande amiga e colega dos tempos de faculdade Roberta Jungblut Hessel e meu grande amigo Fabiano Hessel. Também agradeço a outra grande amiga e colega de faculdade, além de grande incentivadora, que generosamente me hospedou nas missões em Lisboa: Lisiane Pioner Ramos. Aproveito para agradecer ao pessoal da Universidade Nova de Lisboa, pela cooperação no trabalho realizado.

Agradeço aos meus colegas e amigos de "mesmo barco" pela troca de idéias e de incentivos que sempre ocorreram nos "bastidores", ou seja, nas salas de doutorandos, durante os almoços ou nas memoráveis "horas do café". Foram muitos os colegas que me incentivaram, dos quais posso destacar minha grande amiga Isabel Manssour, Ricardo Dornelles (o Cadinho), Maurício Pilla, Rogério Rizzi, colegas do GPPD e da sala 245.

Agradeço a todos os meu parentes e amigos, em especial ao amor da minha vida Yuri Teixeira, por sempre terem me incentivado mesmo sem entender muito bem no que consistia essa minha por vezes estranha atividade e tema de pesquisa.

Finalmente, agradeço ao Instituto de Informática da UFRGS, na figura de seus professores e em especial de seus funcionários por tornarem este ambiente tão especial e agradável, apesar de tantas dificuldades, e por demonstrarem uma dedicação capaz de tornar injusto todo o preconceito existente contra a figura do funcionário público neste país.

A meu pai José Luiz Stringhini
(in memorian)

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A programação paralela é sem dúvida mais complexa do que a programação seqüencial. O controle de múltiplos processos e de suas interações são as principais razões para tal complexidade. Apesar da existência de algumas ferramentas que atendem à fase de desenvolvimento de programas paralelos, a complexidade é normalmente passada para as ferramentas paralelas, isto é, as ferramentas não são de fácil utilização. Assim, existe uma necessidade de ambientes e ferramentas realmente fáceis de usar no âmbito da programação paralela. Embora existam algumas ferramentas interessantes, inclusive algumas comerciais, seu uso permanece insuficiente, em parte devido à complexidade na utilização de algumas delas, em parte devido ao seu uso específico em determinadas plataformas. Portanto, existe ainda um grande campo de estudo no que diz respeito a melhorias de projeto sobre ferramentas existentes e desenvolvimento de ferramentas com um maior número de recursos.

Provavelmente, a ferramenta paralela mais necessária aos programadores é o depurador paralelo. Por sua vez, ferramentas de depuração paralela estão entre as mais complexas de se desenvolver e talvez isso explique o motivo pelo qual poucas têm sido efetivamente utilizadas. Este trabalho descreve uma contribuição no campo da depuração paralela através da análise de interfaces de depuração paralela e da proposta de um modelo. A partir deste modelo, uma interface de depuração paralela – PADI (*PArallel Debugger Interface*) foi desenvolvida e seu principal objetivo é o de oferecer uma interface intuitiva e de fácil utilização.

O modelo proposto e conseqüentemente a ferramenta PADI tratam da depuração paralela simbólica *on-line*. A depuração *on-line* trata do oferecimento de acesso aos símbolos do programa, como variáveis e registradores. A depuração *on-line* diferencia-se da *off-line* pelo tipo de interação com a execução do programa. A depuração *on-line* oferece interação direta com a aplicação, enquanto que a *off-line* interage com um arquivo de monitoração gravado durante a execução da aplicação paralela. A depuração *on-line* é similar à depuração seqüencial tradicional e, conseqüentemente, é de mais fácil utilização por parte da maioria dos programadores.

Palavras-chave: 1. Depuração paralela. 2. Visualização de processos paralelos. 3. Seleção de processos paralelos. 4. Programação paralela. 5. *Clusters* de PCs.

TITLE: “PARALLEL PROGRAM DEBUGGING – DEVELOPING AN INTUITIVE INTERFACE”

Abstract

Parallel programming is undoubtedly more complex than serial programming. The control of multiple processes and their interactions are the main reasons for such complexity. Although there are a certain number of tools that address the development phase of parallel programs, the complexity of such parallel structure is often passed on parallel tools. Thus there is a need for specific easy-to-use environments and tools for parallel programming. In spite of the existence of interesting tools, including a number of commercial ones, their use remains insufficient, partly because of the complexity of utilizing some of them, partly because some of them are constructed for specific platforms. Therefore, there remains a lot of room for improvements of existing tools or development of more supportive ones.

Probably, the most required parallel tool by programmers is a parallel debugger. In turn, parallel debugging tools are among the most complex to develop and this explains perhaps why few of such tools have been commonly used so far. This work describes a contribution to the field of parallel debugging through the analysis of parallel debugging interfaces and the proposition of a model. From this model, a parallel debugger interface – PADI (PArallel Debugger Interface), was developed and its main goal is to provide an intuitive and easy-to-use interface.

The proposed model and consequently the PADI tool address parallel on-line symbolic debugger. On-line debugging is concerned with providing access to program symbols, like variables and registers. On-line debugging differs from off-line debugging by the interaction mode with the program execution. On-line debugging has a direct interaction with the application while off-line debugging interacts with a trace file recorded during the original execution of the parallel application. On-line debugging is similar to traditional serial debugging and, consequently, easier to use for the majority of programmers.

Keywords: 1. Parallel debugging. 2. Parallel processes visualization. 3. Parallel processes selection. 4. Parallel programming. 5. Clusters of PCs.

1 Introdução

O desenvolvimento do paralelismo como um dos meios mais convenientes para o aumento de desempenho de aplicações computacionalmente intensivas trouxe consigo diversos desafios. O projeto de ambientes e ferramentas para programação paralela é um desses desafios. Este trabalho aborda a depuração de programas paralelos e descreve o projeto de uma ferramenta que atende a uma das fases mais importantes do desenvolvimento de aplicações paralelas: a depuração de tais programas visando a detecção de erros.

O objetivo é, partindo de uma análise aprofundada do estado da arte, apontar as principais características desejáveis na interface dos depuradores paralelos. O resultado desta análise é aplicado no desenvolvimento de um projeto e de um protótipo para interfaces de depuração paralela. Uma das contribuições que se pretende alcançar é a proposta de um modelo padronizado para tais interfaces.

Num primeiro momento, a pesquisa na área de paralelismo se concentrou principalmente em desenvolver *hardware* paralelo. A seguir, ambientes de programação, como o PVM [GEI 94] e o MPI [PAC 97], foram e ainda são alguns dos principais objetos de pesquisa nesta área. O aperfeiçoamento dos agregados (*clusters*) de estações de trabalho (ou PCs), através de novas e mais rápidas tecnologias de interconexões (como ATM, SCI e Myrinet) barateou o custo do hardware paralelo eficiente. Hoje em dia, o uso de agregados, devido às suas inúmeras vantagens, tem sido largamente difundido. Assim, um dos modelos de programação paralela que tem se firmado é o modelo de memória distribuída, onde os processos localizados em diferentes processadores se comunicam através de troca de mensagens.

O conjunto de processos compreende as diversas tarefas de uma aplicação paralela que possuem um certo grau de independência, já que não compartilham espaço de endereçamento e normalmente também não compartilham o mesmo processador. Apenas o algoritmo é capaz de restringir o paralelismo através do grau de dependência de dados existente entre as tarefas.

Alternativas que visam amenizar as dificuldades do paralelismo explícito existente no paradigma de memória distribuída têm sido propostos. Atualmente, a mais visada é a simulação de memória compartilhada em arquiteturas distribuídas (*Distributed Shared Memory – DSM*). Entretanto, a programação paralela explícita mantém sua importância, dado que sistemas como os de DSM necessitam de suporte para a troca de mensagens entre os processadores e muitas vezes esse suporte é realizado via software.

À medida em que ambientes explícitos se tornam mais comuns, percebe-se a falta de ferramentas de suporte à programação, como depuradores paralelos. O tema da pesquisa para esta tese de doutorado é justamente a depuração de programas paralelos e os aspectos que a envolvem, principalmente os que dizem respeito à interação com o usuário. Assim, são abordadas as principais características de depuradores paralelos com o intuito de definir a melhor maneira de interação com uma ferramenta deste tipo.

Entre os assuntos tratados estão a depuração simbólica e os reflexos do paralelismo neste tipo de depuração, o controle de diferentes fluxos de execução e a necessidade de ferramentas gráficas. É importante salientar que um depurador não é uma ferramenta automática de detecção de erros, mas que auxilia na detecção uma vez que oferece meios para que o programador verifique estados do sistema.

Em relação à depuração seqüencial, a paralela é significativamente mais complexa. Um depurador paralelo deve coordenar os diferentes fluxos de controle (processos e *threads*) que fazem parte da aplicação. Ou seja, a cada momento haverá diversas instruções sendo executadas, assim como eventos de comunicação. Enquanto isso, um depurador seqüencial controla apenas um fluxo principal, que até pode conter várias *threads* locais, mas ainda assim haverá apenas uma instrução sendo executada a cada momento.

Depuradores paralelos devem se preocupar com o fator de escalabilidade das aplicações paralelas, ou seja, com o tratamento de um número possivelmente grande e indefinido de processos. A complexidade dos programas paralelos pode crescer de acordo com seu tamanho e variedade de tarefas, o que dificulta a análise de seu comportamento, assim como o controle e verificação de estado. Entre as técnicas que têm sido desenvolvidas para amenizar este fator estão os mais diversos tipos de filtros. A filtragem de informações tem como objetivo ou eliminar informações redundantes de forma automática ou focar a depuração nos aspectos definidos pelo usuário.

Basicamente, existem dois tipos de depuradores paralelos, se considerarmos seu modo de execução: *on-line* e *of-fline*. Depuradores *on-line* são similares aos depuradores seqüenciais tradicionais, onde informações simbólicas são verificadas durante a execução do programa. Já depuradores *off-line* são baseados em técnicas de re-execução (*replay*) para permitir a depuração cíclica (dificultada pelo não-determinismo inerente às aplicações paralelas). O objetivo dos depuradores *on-line* é a detecção de erros relativos ao estado dos processos, enquanto que os depuradores *off-line* são voltados à detecção de erros advindos do paralelismo (basicamente de sincronização).

Segundo [BUH 96], os erros concorrentes ocorrem com **freqüência decrescente** na seguinte ordem: erros seqüenciais tradicionais, erros no projeto de algoritmos, *deadlocks*[1] e condições de corrida (*race conditions*[2]). Entretanto, a **dificuldade** em encontrar e corrigir estes erros cresce exponencialmente dos erros seqüenciais às condições de corrida. Isto quer dizer que a tradicional depuração simbólica dos processos (nível de variáveis e pilha) continua importante, mas que além disso existe um outro nível de dificuldade que é a depuração visando os problemas relativos ao paralelismo. Este tipo de erro, que como mencionado é menos freqüente, pode ter facilitada sua depuração pela inclusão no projeto de mecanismos de

---

[1] Situação onde um processo entra em estado de espera por um evento que deveria ser gerado por um outro processo que está neste mesmo estado.

[2] Situação onde uma mensagem inesperada chega ao seu destino antes da mensagem correta, causando um erro. O atraso pode ser causado por diversos fatores independentes do programa, como sobrecarga do processador ou da rede de comunicação.

visualização. Cabe salientar que estes dois tipos de ferramenta não são exclusivos, podendo ser utilizados de forma complementar.

O modelo proposto neste trabalho é adequado à depuração *online*, cujo principal objetivo é a detecção de erros simbólicos. Entretanto, existe uma preocupação especial com a coordenação e o gerenciamento de processos do ponto de vista do depurador através da aplicação de técnicas de **visualização gráfica** e **seleção de processos**. Algumas ferramentas existentes, como demonstrado, apresentam uma ou outra técnica, mas são poucas as que apresentam as duas de forma coordenada. A ferramenta desenvolvida com base no modelo proposto neste trabalho chama-se PADI (*PArallel Debugger Interface*) e trata-se, portanto, de uma interface para um depurador paralelo *online*.

Esquemas de visualização são mais comumente utilizados em ferramentas de análise de desempenho e em depuradores *offline* para ajudar na detecção de "gargalos" de desempenho e erros de sincronização. Em depuradores *online*, a visualização é utilizada para a demonstração dos estados de cada processo, mas ainda não foi completamente explorada. Esta tese defende o uso da visualização como um meio para que se entenda o funcionamento geral da aplicação paralela durante sua execução, assim como para que se possa acompanhar mais facilmente o processo de depuração.

Além disso, a visualização é combinada com um mecanismo de seleção de processos, que possibilita o controle e a visualização somente dos processos pertencentes a um determinado grupo selecionado pelo usuário. Este mecanismo oferece um meio para que o usuário escolha entre grupos de processos pré-definidos ou defina um grupo próprio para ser utilizado a qualquer momento.

Outros conceitos e técnicas de depuração paralela também são apresentados e implementados. O esquema de navegação da ferramenta é essencial e considera os níveis de abstração possíveis. O nível de **coordenação** é um deles e é o ponto de partida para que o usuário possa acessar o nível inferior, que é o de **processos**. Este, funciona como um depurador seqüencial para o processo em questão e propicia a inspeção do estado da aplicação a partir da demonstração de linhas de código, variáveis, *threads*, etc.

A PADI se insere no projeto de cooperação entre o Grupo de Processamento Paralelo e Distribuído da UFRGS e o grupo equivalente do ID-IMAG (França). A equipe francesa, por sua vez mantém uma cooperação com uma equipe da Universidade Nova de Lisboa, que desenvolve uma plataforma de depuração chamada FIDDLE (anteriormente conhecida como PDBG [CUN 98]), que serve de base para o desenvolvimento da PADI. Assim, a ferramenta se insere em cooperações existentes entre estas três instituições. No âmbito da UFRGS, oferecerá suporte à biblioteca de programação paralela DECK [BAR 2000].

1.1 Descrição dos capítulos

Com o intuito de descrever o trabalho realizado, este texto está dividido da seguinte forma:

- **Capítulo 2**: realiza uma revisão de conceitos básicos de paralelismo relevantes para que se compreenda e se contextualize o trabalho realizado, que pertence a esta área da computação.

- **Capítulo 3**: aprofunda aspectos relativos à depuração de programas paralelos e que foram importantes na definição do modelo e no projeto da ferramenta PADI. Trata tópicos como os modos de operação de um depurador, a estrutura básica de um programa paralelo, a visualização aplicada à depuração paralela e descreve um padrão desenvolvido por um grupo de pesquisadores da área.

- **Capítulo 4**: apresenta uma série de ferramentas de depuração presentes na literatura e que são utilizadas para a análise do estado da arte na área da depuração paralela, assim como para comparação com o trabalho realizado.

- **Capítulo 5**: apresenta o desenvolvimento de uma interface intuitiva para depuração paralela através da descrição das principais características do modelo proposto e da ferramenta PADI, como as decisões de projeto, os mecanismos de seleção e visualização e as principais funcionalidades dos dois níveis da interface.

- **Capítulo 6**: descreve a arquitetura e a implementação do protótipo PADI. Trata de tópicos como a estrutura básica da ferramenta, como se dá a interação com o usuário, quais as características da implementação da seleção de processos, como se dá a integração com o depurador de mais baixo nível e apresenta um exemplo do fluxo de execução de um comando dentro do ptorótipo PADI.

- **Capítulo 7**: apresenta testes realizados, assim como a validação da ferramenta a partir da descrição da interação com o usuário, comparação entre interfaces e apresentação de uma proposta preliminar de um modelo para interfaces de depuração paralela baseado na validação das características do protótipo PADI.

- **Capítulo 8**: aponta novos caminhos e possibilidades que podem ser exploradas a partir do protótipo PADI. A descrição de trabalhos futuros é suficientemente detalhada para que possa servir de ponto de partida para novos trabalhos.

- **Conclusão**: resume os problemas encontrados e as soluções alcançadas através do projeto e do desenvolvimento do protótipo da ferramenta PADI.

2 Desenvolvimento de aplicações paralelas e distribuídas

O desenvolvimento de software paralelo e distribuído envolve uma série de aspectos que precisam de definição. Entre esses aspectos, estão a arquitetura alvo, o modelo de programação e os ambientes e ferramentas disponíveis. Este capítulo aborda os principais problemas relacionados à depuração de programas paralelos, que é o foco deste trabalho. O objetivo é apresentar esses tópicos e, ao mesmo tempo, situar o presente trabalho no contexto do processamento paralelo e distribuído.

2.1 Revisão de arquiteturas e modelos de programação

De uma forma geral, existem duas maneiras de se alcançar o paralelismo no que diz respeito à programação: implícita e explícita. Na forma implícita, o paralelismo propriamente dito é transparente ao usuário, enquanto que na explícita ele deve descrever o paralelismo através de algum mecanismo de linguagem de programação ou afim.

Além disso, características da arquitetura alvo podem influenciar diretamente no modelo de programação. Neste nível, destacam-se dois paradigmas básicos: o de memória compartilhada e o de memória distribuída.

A idéia aqui não é fazer um estudo aprofundado sobre estes temas, já amplamente discutidos na literatura. Ao invés disso, esta seção faz uma rápida revisão sobre estes quatro aspectos e, no final, indica o contexto do modelo proposto e do protótipo desenvolvido.

2.1.1 Arquiteturas de memória compartilhada

No paradigma de memória compartilhada, os diversos processos que compõem uma aplicação paralela podem se comunicar e trocar informações através de uma memória comum. Isto é possível porque os diversos processadores que fazem parte da máquina paralela possuem algum tipo de conexão com toda a memória presente na máquina. Este fator permite que os processos distribuídos nos diferentes processadores compartilhem o mesmo espaço de endereçamento. Máquinas desse tipo são comumente conhecidas como multiprocessadores.

Uma de suas principais vantagens é que todo o estudo existente em programação concorrente pôde ser reaproveitado no uso deste paradigma. Entretanto, os problemas com este tipo de sistema estão no nível do hardware, onde a principal causa é o fator de *escalabilidade*, que é muito fraco. Por este motivo, uma das abordagens que mais tem sido foco de estudos é a simulação de memória compartilhada sobre arquiteturas com memória distribuída.

2.1.2 Arquiteturas de memória distribuída

No paradigma de memória distribuída, que abrange as máquinas com memória distribuída, os processos da aplicação paralela necessitam de comunicação via troca de mensagens. Isto é necessário porque as máquinas deste tipo possuem diversos processadores onde cada um possui uma memória local. Estas máquinas são comumente conhecidas como multicomputadores e os agregados (*clusters*) de computadores se encaixam nesta categoria.

Assim, para que haja comunicação, é necessário que os processos/processadores utilizem uma rede de interconexão para terem acesso às informações locais uns dos outros. Este paradigma possui a vantagem da escalabilidade, no entanto perde no nível da complexidade necessária para o desenvolvimento dos programas paralelos.

2.1.3 Modelo de programação com paralelismo implícito

Nesta abordagem, em geral tem-se um compilador capaz de detectar paralelismo num programa escrito numa linguagem seqüencial existente. Este compilador é responsável por detectar quais as partes do código seqüencial que podem ser executadas em paralelo.

A complexidade do compilador é o maior problema desta abordagem. Entre as principais dificuldades estão a decomposição do problema em processos paralelos e a identificação e gerenciamento da comunicação e sincronização destes processos [PER 96].

Assim, tem-se que as principais vantagens do paralelismo implícito estão na transparência ao programador, que pode continuar a programar de forma seqüencial, e no reaproveitamento de algoritmos seqüenciais. As desvantagens estão na complexidade do compilador e na própria dependência que o programa terá deste compilador, no sentido de que somente os tipos de paralelismo previstos por ele é que serão aplicados no momento da paralelização automática.

2.1.4 Modelo de programação com paralelismo explícito

Nesta abordagem, o programador é responsável por expressar o paralelismo de seu programa através de algum tipo de construção específica. Estas construções são oferecidas através de três formas: linguagens paralelas, linguagens de coordenação e extensões de linguagens [PER 96].

As linguagens paralelas são as linguagens projetadas com o principal objetivo de expressar o paralelismo em forma de comandos específicos. No projeto destas linguagens não são consideradas outras linguagens existentes e nem suas correspondentes aplicações. O principal objetivo é a transposição do paralelismo para os comandos da linguagem de forma a facilitar a programação paralela explícita. Exemplos

de linguagens paralelas são o SR (*Syncronizing Resources*) [AND 93] e a Occam [QUI 94].

As extensões de linguagens já existentes são uma abordagem bastante difundida hoje em dia. Consistem em acrescentar mecanismos que possibilitem paralelismo a linguagens já existentes e amplamente utilizadas. Vários paradigmas têm sido propostos, entre os quais destacam-se a memória compartilhada distribuída e as bibliotecas de comunicação.

A memória compartilhada distribuída (ou DSM – *Distributed Shared Memory*) consiste na programação através de variáveis compartilhadas que na verdade estão distribuídas nos diferentes processadores. Esta abordagem possui a vantagem da facilidade de programação aliada à flexibilidade da arquitetura com memória distribuída. Em geral, existe uma camada de software que gerencia o compartilhamento de variáveis e "esconde" a arquitetura distribuída do programador.

As bibliotecas de comunicação, em sua maioria, acrescentam mecanismos que possibilitam a passagem de mensagens e o gerenciamento de múltiplos processos às suas linguagens de base, normalmente linguagens seqüenciais existentes. Existem desde bibliotecas específicas a determinadas máquinas paralelas até bibliotecas portáveis e heterogêneas. Neste contexto, se destacam o PVM [GEI 94], que é uma biblioteca utilizável inclusive sobre redes de estações de trabalho heterogêneas, e o MPI [PAC 97], semelhante ao PVM, mas que na verdade é um padrão que tem o objetivo de definir um modelo para diferentes implementações.

2.1.5 Arquitetura e modelo de programação PADI

O modelo de interface proposto nesta tese, assim como a própria ferramenta PADI, se encaixam no paradigma de memória distribuída com modelo de programação explícito. Como será visto mais adiante, a interface proposta trata da depuração de processos paralelos distribuídos em diversos processadores sem memória comum. Assim sendo, esses processos se comunicam através de troca de mensagens, o que implica na descrição do paralelismo sendo realizada de forma explícita. Mais especificamente, essa descrição do paralelismo é realizada através de bibliotecas de programação paralela como a PVM e a MPI.

Atualmente, o paradigma de memória distribuída é uma das alternativas mais populares em termos de programação paralela, visto que os agregados de máquinas (principalmente PC´s) vêm alcançando uma melhor relação custo-benefício no mercado das máquinas paralelas. No caso específico da PADI, o objetivo é adaptá-la também ao uso com o ambiente DECK - *Distributed Execution and Communication Kernel* [BAR 98, BAR 2000], desenvolvido no Grupo de Processamento Paralelo e Distribuído (GPPD) do Instituto de Informática da UFRGS. O DECK é um ambiente distribuído que fornece primitivas e serviços próprios para programação em agregados, como *threads*, caixas postais e nomeação de objetos.

A figura 2.1 mostra um resumo da classificação básica adotada nesta seção para situar o modelo de arquitetura e programação atendidos pela ferramenta PADI.

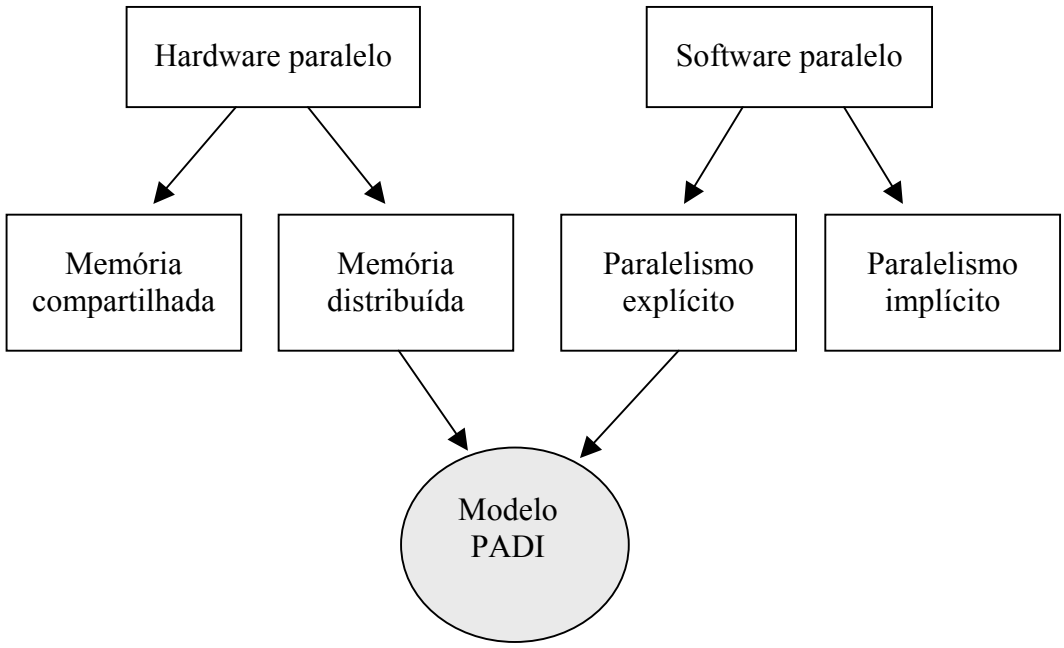

FIGURA 2.1 – Contexto do modelo proposto e da ferramenta PADI

2.2 Revisão de ambientes e ferramentas de desenvolvimento

Esta seção traça um rápido panorama dos tipos de ferramentas e ambientes existentes para auxiliar na programação paralela e distribuída. Para tornar o texto mais objetivo, serão considerados apenas ambientes que tratam de programação explícita e comunicação via troca de mensagens (memória distribuída).

2.2.1 Programação visual

A programação visual em geral e no nível comercial é utilizada para solucionar problemas bem específicos de ambientes de produção com características bem definidas (dificilmente é encontrada como linguagem genérica). A descrição do paralelismo pode ser considerada como um problema específico, onde se pode definir componentes e meios de interação entre eles. Assim sendo, a programação paralela se torna naturalmente um alvo de investigação adequado à introdução de técnicas de programação visual.

Basicamente, uma linguagem visual paralela permite que se estruture a aplicação e seus componentes e que se estabeleçam relações entre eles de forma gráfica. A idéia é reduzir ao máximo o envolvimento do programador com definições textuais. Além da parte gráfica, outro componente essencial é o gerador de código, que se encarrega de traduzir a definição gráfica do programa em código textual (executável ou numa linguagem intermediária).

Nas linguagens gráficas baseadas em processos o próprio programador é quem define a estrutura e as operações de troca de mensagens de forma explícita (outros paradigmas, como o *dataflow*, por exemplo, requerem menos envolvimento por parte do programador). A estrutura de processos de um programa paralelo é definida por um grafo de processos onde os nodos representam os próprios processos e os arcos representam a comunicação entre eles (normalmente). No código de cada processo, as operações de interação com a biblioteca de programação são realizadas através de chamadas (texto) ou através de algum tipo de representação gráfica. A vantagem de se utilizar linguagens visuais no paradigma baseado em processos é o grande espectro de aplicações que nele se encaixam [DOZ 2000].

Entre os ambientes e ferramentas que se utilizam de programação visual baseada em processos destacam-se o TRAPPER [SCH 93], o Meander [GIE 97] e o EDPEPPS/PVMGraph [DEL 97].

2.2.2 Depuração

Este é o principal tópico tratado neste trabalho e, a fim de introduzi-lo no contexto dos ambientes e ferramentas de programação paralela, ele é brevemente tratado nesta seção.

A depuração paralela baseada em processos basicamente consiste no tratamento de dois tipos de erro:

- **Erros seqüenciais**: a depuração de tais erros consiste em tratar os erros locais de cada processo. Os depuradores simbólicos tradicionais são adaptados para trabalharem com processos seqüenciais. As ferramentas normalmente possuem mecanismos para lidarem com uma quantidade indefinida de janelas, de acordo com o número de processos da aplicação. Técnicas para controlar diversos fluxos de depuração e reduzir a quantidade de informação a ser controlada são o principal foco da pesquisa em tais depuradores.

- **Erros de sincronização**: a depuração de tais erros consiste em detectar erros decorrentes do paralelismo explícito, ou seja, da comunicação entre os processos. Ferramentas de visualização e análise de dados monitorados são mais adequadas a esse tipo de depuração. Técnicas de re-execução de programas e visualização são os principais focos de pesquisa em tais depuradores.

Os erros seqüenciais ocorrem em maior quantidade, enquanto que os erros de sincronização possuem um grau mais alto de dificuldade de detecção. Assim sendo, os dois tipos de depuração são de grande auxílio ao programador e as ferramentas podem se utilizar dos dois tipos de técnicas.

2.2.3 Análise de desempenho

A análise de desempenho de programas paralelos também é um fator importante no contexto da programação paralela. O objetivo é refinar o programa de tal forma que ele represente um ganho efetivo de desempenho principalmente em relação à versão seqüencial ou até mesmo em relação a versões anteriores. Cabe ressaltar que as ferramentas em questão são para análise de programas e não de hardware.

No caso da programação baseada em processos podem existir vários fatores de inibição de desempenho, tais como a comunicação e sincronização entre os processos, criação e término de processos, escalonamento, ociosidade, etc. As principais técnicas de análise de desempenho incluem a monitoração do programa em tempo de execução, onde são gravadas as informações para a análise e a apresentação destas informações de forma gráfica ao usuário.

Os tipos de informação gravada durante a execução recaem em duas classes principais [STE 00]: tempos de execução e taxas de utilização de recursos. Os tempos de execução podem ser decompostos em várias medidas que representam o tempo gasto pela execução das diversas partes do programa (procedimentos, protocolos de comunicação, etc). As taxas de utilização de recursos podem ser relacionadas aos processadores (percentual gasto com sincronização, término de processos, comunicação, etc) ou podem ser globais, indicando problemas como uma pequena utilização dos processadores ou uma grande taxa de ociosidade. Os dados detalhados podem indicar gargalos como falta de paralelismo no programa, fraco desempenho do escalonador, uso excessivo de primitivas de sincronização, etc.

2.2.4 Ambientes completos e integração de ferramentas

Como mencionado, ferramentas de programação visual, depuração e análise de desempenho são fundamentais no desenvolvimento de aplicações paralelas. Ainda poderíamos incluir neste contexto ferramentas de monitoração, visualização, teste e simulação de programas. Assim, muitos pesquisadores tem trabalhado no desenvolvimento de ferramentas completas ou na integração de ferramentas existentes.

Ambientes como o Annai [CLE 96a], que inclui suporte a programação, depuração, análise de desempenho e ainda visualização de dados paralelos para HPF (*High Performance Fortran*) são ferramentas poderosas. O Annai, cujos ambientes de depuração e visualização figuram na lista de ferramentas abordadas no capítulo 2, é um exemplo de ambiente completo (apesar de não possuir programação visual). Neste sentido pode-se citar ainda:

- o SkIE (*Skeleton Integrated Environment*) [BAC 99] que é um ambiente que possibilita o desenvolvimento de aplicações de maneira parcialmente visual, assim como a integração de ferramentas como bibliotecas de programação, ferramentas de monitoração e depuração, entre outros.

- o ambiente GRADE (*Graphical Application Development Environment*) [KAC 97]: possui linguagem de programação visual – a GRAPNEL – depuração e análise de desempenho,

- o EDPEPPS [DEL 97]: programação visual, simulação, depuração e análise de desempenho para programação PVM e

- o TRAPPER [SCH 93]: programação visual, depuração, monitoração e visualização para programas PVM/MPI em Windows NT.

A integração de ferramentas é um caminho natural, visto que há trabalhos sendo desenvolvidos nestas áreas nos mais diversos centros de pesquisa. Os próprios ambientes GRADE e EDPEPPS estão envolvidos num projeto europeu (SEPP/HPCTI [CUN 2000]), que define a integração de ferramentas de projeto, depuração, monitoração, visualização, análise, simulação e teste desenvolvidas em centros europeus. De fato esta é uma tendência e o próprio protótipo PADI se utiliza deste recurso, visto que é integrado a uma outra ferramenta de depuração, como é descrito mais adiante neste texto.

2.2.5 Contexto do modelo proposto e da ferramenta PADI

A PADI, como já mencionado, é uma interface para uma ferramenta mais básica de depuração, cujo projeto foi baseado no estudo sobre interfaces apresentado no decorrer deste trabalho. A figura 2.2 ilustra a posição do modelo da PADI no contexto básico dos ambientes de desenvolvimento de programas paralelos. Além desses ambientes básicos, ferramentas auxiliares como as de monitoração, visualização de dados e as de teste também podem ser incluídas no contexto.

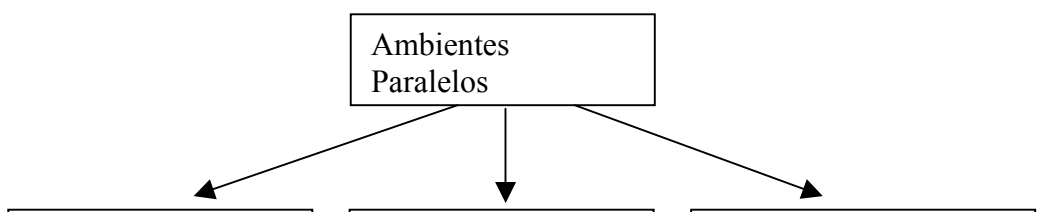

FIGURA 2.2 – Modelo da PADI no contexto dos ambientes de desenvolvimento

2.3 Problemas da depuração paralela

São vários os fatores que levam a depuração paralela a ser mais complexa do que a seqüencial. A depuração paralela baseada em processos paralelos possui esta complexidade devido aos diversos fluxos de execução de uma aplicação paralela, que podem estar sendo executados em diversos processadores. Estes, podem ser heterogêneos, podem possuir cargas diferentes e normalmente não possuem nem memória nem relógio comum (paradigma de memória distribuída). Os problemas decorrentes desta situação em relação à depuração paralela são analisados nos tópicos dessa seção.

2.3.1 Grande número de processadores e processos

Como trata-se aqui de aplicações que se encaixam no paradigma de memória distribuída, deve-se considerar que uma das vantagens de tal paradigma é justamente a facilidade de se acrescentar novos componentes: processadores e/ou processos. Este fator é o de *escalabilidade*, que pode se tornar crítico em se tratando da depuração de uma aplicação paralela.

Pode-se dizer que este foi o fator que determinou o impulso da pesquisa na área da depuração paralela. Inicialmente, visto que um programa paralelo é basicamente uma coleção de processos seqüenciais, associava-se um depurador seqüencial a cada processo e controlava-se cada um individualmente. Entretanto, esta prática pode se tornar insustentável quando o número de processos é muito grande ou cresce por algum motivo.

O grande número de processos, desta forma, torna a quantidade de informações a serem tratadas praticamente insuportável caso não exista um tratamento adequado dessas informações. Técnicas de eliminação de redundâncias nas informações, assim como a visualização estão entre as mais adequadas ao tratamento desta grande quantidade de informações. Além disso, as experiências no controle individual de várias janelas ao mesmo tempo mostram que um controle central é necessário, de forma a expandir as operações normais de um depurador simbólico a um grupos de processos.

O modelo de interface proposto neste trabalho, busca atender justamente este tipo de problema, ou seja, basicamente oferecer:

- um controle central para todos os comandos de depuração paralelos de uma forma clara e intuitiva,
- mecanismos de formação de grupos de processos (seleção),
- mecanismos de visualização de processos, com o objetivo de diminuir a quantidade de informações apresentadas de forma textual.

Estas e outras características deste modelo são aprofundadas mais adiante neste texto. Cabe salientar que os problemas tratados e que foram descritos até aqui (grande quantidade de informações, escalabilidade, visualização e seleção de processos), em sua maioria são típicos de depuradores simbólicos paralelos *on-line*, categoria na qual o presente trabalho se encaixa. O desafio do modelo PADI, pode-se dizer, está em "trivializar" a depuração, ou seja, criar uma interface que atenda a estes requisitos e que seja facilmente utilizável (intuitivamente).

2.3.2 Depuração cíclica

A depuração cíclica é uma característica básica da depuração de programas seqüenciais. Dado um conjunto de dados de entrada, um programa seqüencial se comportará da mesma maneira sempre que for executado. Este comportamento define um programa seqüencial como sendo determinista. A depuração cíclica consiste em encontrar o possível local do erro através da execução do programa junto a um depurador simbólico normal, fazer uma alteração no código e verificar se o erro foi corrigido através de uma nova execução. Este ciclo se repete até que não haja mais erros.

Já num programa paralelo existem fatores que podem alterar o comportamento do programa, independentemente dos dados de entrada. Entre estes fatores estão a heterogeneidade dos processadores, as diferentes cargas de processamento, os atrasos de comunicação e sincronização entre processos (o que ocasionam as condições de corrida), e a intrusão (ou *probe-effect*) devido a ferramentas de monitoração ou até mesmo depuração [MAR 90]. Assim sendo, programas paralelos são considerados não-deterministas.

Este não-determinismo característico dos programas paralelos impede que a depuração cíclica seja realizada sem que algum mecanismo próprio para isso seja empregado. Técnicas de re-execução como a descrita por Leblanc [LEB 87] são aplicadas em depuradores *off-line* e permitem a depuração cíclica baseada na gravação de eventos durante a execução e posterior re-execução do programa baseada nos dados gravados.

O modelo PADI, desenvolvido para depuradores simbólicos *on-line*, não trata o problema da depuração cíclica. Um paralelo mais aprofundado sobre os tipos de depuração (*on-line x off-line*) é realizado no capítulo 3, que trata da depuração com mais detalhes.

2.4 Considerações finais

O objetivo desta introdução foi o de situar este trabalho e a ferramenta desenvolvida no contexto do processamento paralelo e distribuído. Assim, tópicos relacionados, que circundam aqueles aplicados neste trabalho, foram descritos e o trabalho foi devidamente contextualizado. Pode-se resumir a situação do modelo proposto e conseqüentemente da ferramenta PADI da seguinte forma:

- Arquitetura alvo e modelo de programação das aplicações atendidas pelo modelo:
  - Arquitetura de memória distribuída (*clusters* Linux);
  - Programação explícita e baseada em processos;
  - Comunicação via troca de mensagens.
- Tipo de ferramenta:
  - Depurador visando a detecção de erros de programação, principalmente os erros locais (depuração simbólica *on-line*).
  - Interface gráfica integrada à um depurador de mais baixo nível que lhe acrescenta mecanismos tais como os de seleção e visualização de processos.
- Classe de problemas de depuração atendidos pelo modelo:

- Problemas relacionados à grande quantidade de informações geradas pela depuração de um grande número de processos paralelos;

- Depuração intuitiva.

3 Depuração de aplicações paralelas e distribuídas

O desafio da depuração de programas paralelos está em adaptar algumas das características tradicionais das ferramentas seqüenciais que já são de domínio dos programadores em geral, assim como em desenvolver técnicas específicas. Com isso, pretende-se facilitar a programação paralela e, assim, difundi-la cada vez mais.

A própria complexidade inerente ao paralelismo, principalmente o de memória distribuída, é um fator que compromete o desenvolvimento de ferramentas de depuração. Além disso, outros fatores acabam por causar uma certa resistência por parte dos usuários, tais como a complexidade na utilização de tais ferramentas juntamente com o tempo que demanda seu aprendizado e o fato de a maioria existente ser de uso específico de determinadas plataformas.

Este capítulo aborda as principais características de depuradores paralelos, assim como apresenta algumas experiências já realizadas.

3.1 Tipos de execução dos depuradores paralelos

Ainda que métodos estáticos possam ser utilizados na análise de um programa paralelo, a análise dinâmica, que tem como objeto de observação a execução do programa, é o principal meio de detecção de erros e de gargalos de desempenho. Esta análise requer, portanto, uma certa interação com a execução do programa. Neste contexto, dois tipos de interações podem ser distinguidas:

- **Interação direta** (*on-line*): é caracterizada pelo controle explícito da execução por meio de um depurador simbólico. Neste tipo de interação, a execução do depurador ocorre ao mesmo tempo em que a execução da aplicação. Na verdade, o depurador tem a habilidade de controlar a execução da aplicação por meio de comandos específicos, assim como mostrar o conteúdo de variáveis, registradores e pilha.

- **Interação monitorada** (*off-line* ou *post-mortem*): caracterizada pela gravação de uma certa quantidade de informações (eventos) durante a execução (geralmente armazenadas num arquivo de *trace*), sendo estas informações, num segundo estágio, interpretadas e apresentadas ao usuário de diversas formas (estatísticas, animações gráficas, etc).

As próximas seções analisam estas duas abordagens para a depuração de programas paralelos.

3.1.1 Interação direta ou *on-line*

Possibilita a depuração simbólica tradicional de uma aplicação. A depuração simbólica, que acontece durante a execução do programa, é praticamente o único método que permite a análise dos estados específicos do programa em qualquer nível de detalhe (estado das variáveis e instruções executadas).

Os depuradores mais simples oferecem funcionalidades de um depurador seqüencial para cada um dos processos do programa. Os mais primitivos, exigem que cada processo seja aberto numa janela diferente com controles individuais para cada um (por exemplo, o *dbxtool*, da Sun). Já os mais sofisticados, mantêm um controle central único para todos os processos, ainda que alguns não tenham interface com gráficos de visualização de processos (e.g., TotalView [ETN 01] e DDBG [KAC 97]).

Os depuradores paralelos, em primeiro lugar, devem tratar o programa como uma entidade única. Uma característica desejável é a possibilidade de se poder parar todo ou parte do programa em função do estado de um ou de vários processos. Dois problemas devem ser tratados para que isso seja possível [LEU 92]:

- a detecção de pontos de parada (*breakpoints*) distribuídos, e
- a parada de vários processos.

A detecção de *breakpoints* distribuídos é um extenso assunto de pesquisa. Trata-se de definir pontos de parada no programa a partir de determinadas condições de estado dos processos ou da ocorrência de eventos específicos, como por exemplo a execução de uma instrução particular. Um *breakpoint* pode ser definido como um estado global onde se deseja que o programa seja parado para que as variáveis possam ser examinadas. Alguns algoritmos para detecção de *breakpoints* podem ser encontrados em [BAR 96] e em [PIN 97], que apresenta uma abordagem prática, onde quatro algoritmos encontrados na literatura são experimentados através da construção de uma ferramenta.

Um outro problema a ser considerado é a parada de todo ou parte do programa. Após detectado o ponto de parada, os processos que fazem parte deste *breakpoint* devem ser paralisados para que possam ser examinados. No caso de arquiteturas com memória distribuída, é praticamente impossível realizar esta parada de forma simultânea. Sempre ocorrerá um lapso de tempo não desprezível entre o momento de parada do primeiro ao último processo paralisado.

Este trabalho, entretanto, não trata da detecção de *breakpoints* baseada em condições, o que pode ser encontrado na literatura supra citada. Em contrapartida, examina e propõe outros mecanismos relativos à distribuição de comandos de depuração (abordados mais adiante) que se fazem necessários para oferecer um controle distribuído sobre a aplicação paralela sendo depurada. Os *breakpoints* são abordados em sua forma mais comum, que é aquela baseada na paralização através da determinação de posições específicas no código fonte.

3.1.2 Interação indireta ou *off-line*

A interação indireta (*off-line* ou ainda *post-mortem*), caracterizada pela gravação de informações durante a execução, tem como objetivo fornecer ao usuário uma representação esquemática da execução. Este processo pode ser dividido em duas partes: gravação dos eventos durante a execução e interpretação das informações após a execução. A gravação dos eventos é realizada num arquivo conhecido como *trace file*. As duas fases deste processo são fontes importantes de pesquisa.

A gravação de informações ou eventos necessita de técnicas similares às de definição de *breakpoints* para a especificação dos pontos de gravação. Normalmente, estes pontos são aqueles de ocorrências de eventos, como por exemplo o envio de uma mensagem. Um problema desta fase é reduzir ao máximo a quantidade de informações gravadas para evitar a geração de arquivos de *trace* excessivamente grandes.

Após a geração do arquivo de *trace*, deve-se empregar alguma técnica de análise deste arquivo, pois sua simples leitura através de um editor de texto torna-se praticamente impossível, dado o número de informações gravadas. Apesar de técnicas de análise através de consultas a uma base de dados relacional já ter sido proposta, a apresentação dos dados de forma gráfica tem sido o meio mais viável de análise dos *traces*.

Entretanto, ainda há uma outra forma de utilização do arquivo de *trace* voltada principalmente à depuração de programas paralelos: a re-execução [LEB 87]. A re-execução de programas paralelos surgiu da necessidade de se evitar os problemas do não-determinismo e possibilitar a depuração cíclica. Em [LEU 92] são apresentados três tipos de re-execução:

- **Re-execução guiada pelos dados**: é baseada na gravação de todas as informações trocadas entre os diferentes processos.

- **Re-execução guiada pelo controle**: o objetivo deste método é a redução da quantidade de informações armazenadas. Toda a informação trocada pelos processos é gerada novamente durante a re-execução, ao invés de ser extraída do arquivo de *trace*.

- **Re-execução especulativa**: consiste em re-executar o programa comparando a ordem de ocorrência dos eventos com a ordem gravada durante a execução inicial.

Note-se que, ainda que necessitem de uma fase de gravação durante a execução inicial, as duas últimas técnicas pressupõem uma interação direta com a execução da aplicação, visto que durante a re-execução o programa realmente **executa** e a gravação apenas tenta manter a **ordem** da primeira execução. Isto permite uma análise cíclica e ao mesmo tempo uma depuração simbólica da aplicação.

Já as técnicas de análise de desempenho e comunicação entre processos baseadas em gráficos, mencionada acima, não exige uma interação direta e se constitui no tipo monitorado ou *post-mortem* legítimo.

Uma re-execução de programa paralelo visando depuração completamente *post-mortem* exigiria que um número excessivo de informações fossem armazenadas. Por este mesmo motivo, a depuração simbólica inteiramente *post-mortem* pode ser descartada.

Este trabalho aborda a depuração direta (*on-line*). Entretanto, vale salientar que os dois tipos de interação não são exclusivos e mecanismos podem ser implementados para fazer com que um depurador *post-mortem* (*off-line*) execute de forma complementar a um depurador *on-line*.

3.2 Estrutura básica de um depurador paralelo *on-line*

O paralelismo introduziu um nível a mais de abstração em relação à depuração tradicional. Este, é o **nível de coordenação** que oferece um controle geral sobre os componentes da aplicação paralela. Assim, mecanismos que dêem suporte a este novo nível são necessários e ainda não foram totalmente explorados.

O nível imediatamente inferior ao de coordenação é o de controle de processos individuais, cuja depuração simbólica possui grande semelhança com a depuração seqüencial tradicional. Esta semelhança existe porque individualmente um processo pode ser considerado um programa seqüencial. As *threads*, neste contexto, seriam sub-componentes dos processos. *Threads* remotas não serão aqui abordadas. A figura 3.1 demonstra esta estrutura hierárquica básica.

O nível de coordenação é o mais amplamente discutido neste trabalho, mesmo porque o controle individual dos processos herdou as características existentes nos depuradores seqüenciais tradicionais. As principais características de um depurador paralelo relativas ao nível de coordenação são:

- **Controle geral da aplicação**. O nível de coordenação deve prover mecanismos que possibilitem o controle geral da aplicação a partir de um elemento central da interface. Assim, esse elemento é capaz de controlar a depuração através de mecanismos que possibilitem comandos de depuração distribuídos e seleção de processos.

- **Observação geral do estado da aplicação**. O nível de coordenação deve oferecer meios que possibilitem a visualização da aplicação, de forma que o usuário possa reconhecer seus elementos distribuídos assim como seus estados de processamento. A visualização gráfica pode ser explorada nesse nível de forma a oferecer informações que facilitem a compreensão da aplicação por parte do usuário.

- **Acesso aos níveis inferiores**. O nível de coordenação serve para dar acesso ao nível imediatamente inferior, que é o de processos individuais. Assim, funciona como um ponto de partida para que o usuário tenha acesso a informações mais detalhadas sobre os diversos elementos de sua aplicação.

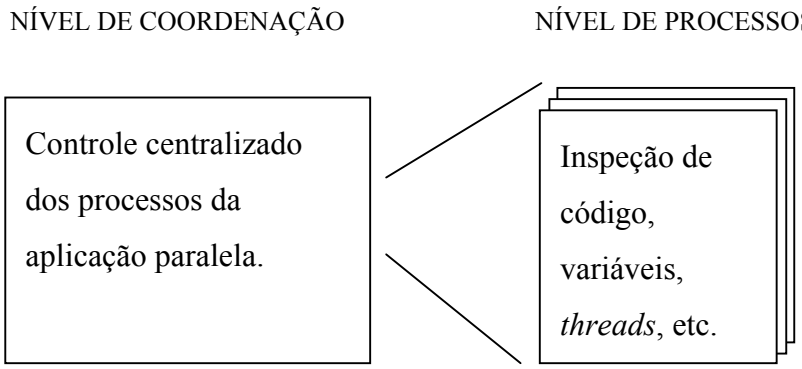

FIGURA 3.1 – Estrutura hierárquica básica de um depurador simbólico paralelo.

3.3 Visualização aplicada à depuração paralela

Tradicionalmente, a visualização gráfica é utilizada principalmente para análise de desempenho de programas paralelos. Por este motivo e por serem áreas afins, esta seção traça um paralelo entre a visualização voltada à análise de desempenho e a voltada à depuração.

A ferramenta ParaGraph [HEA 91], onde uma série de gráficos animados são utilizados para a demonstração de diversos aspectos do programa, foi um marco significativo na área de análise de desempenho. Já no que diz respeito à depuração paralela, a utilização de visualização gráfica ainda necessita de uma maior exploração. Basicamente, os tipos de visualização mais comumente encontrados em ferramentas de depuração paralela são os seguintes:

- **Visualização de dados distribuídos**: é o tipo de representação gráfica mais comum nos depuradores paralelos, principalmente a visualização de vetores e matrizes (e.g., TotalView [ETN 01], Annai [CLÉ 96a]).

- **Visualização de processos distribuídos**: esta representação pode se dar no nível de coordenação do depurador (e.g., p2d2 [HOO 96]). Esta abordagem é a adotada neste trabalho.

De uma maneira geral, a principal diferença entre os diversos tipos de representações gráficas já propostos, principalmente para análise de desempenho, é o tipo de informação apresentada. Neste contexto, existem ferramentas que demonstram gráficos dependentes da aplicação ou permitem que o usuário crie sua própria representação [STA 93]. Entretanto, o mais comum é que gráficos genéricos que demonstram determinadas características da aplicação sejam oferecidos, como os descritos por Heath [HEA 96]. Quando o objetivo é a análise do comportamento de um programa paralelo, a apresentação da interação entre os processos (comunicação,

sincronização) é o tipo de informação mais freqüentemente apresentada, juntamente com gráficos representando a análise de desempenho dos programas.

A experiência com o desenvolvimento de ferramentas de análise de desempenho demonstrou que os usuários têm resistência em aprender ambientes muito complexos [PAN 99]. Neste caso da análise de desempenho, onde os gráficos representam aspectos diferentes da aplicação baseados num grande volume de dados, sente-se a falta de mecanismos que indiquem ao usuário onde podem estar os gargalos do sistema.

Do ponto de vista do usuário, os aspectos importantes numa ferramenta de auxílio ao desenvolvimento de programas envolvem a eficiência e a facilidade de uso, assim como funcionalidades, funções suportadas e a quantidade de controle que se tem sobre as operações da ferramenta. Neste sentido, os principais aspectos de uma ferramenta de visualização, segundo Casavant [CAS 92] são:

- personalização, flexibilidade, poder de expressão;
- eficiência e facilidade de uso;
- controle da múltipla *glanularidade*;
- controle da velocidade de visualização;
- controle da direção de visualização;
- níveis de abstração;
- tamanho do *display*;
- animações;
- assistência extra na análise.

Estes aspectos foram referenciados visando ferramentas de análise de desempenho, mas podem ser adaptados, na medida do possível, para as ferramentas de depuração paralela. Da mesma forma, o trabalho de Leroux e Exton [LER 2001] no desenvolvimento de uma ferramenta de visualização para programas concorrentes orientados a objetos, aponta quatro pilares para as ferramentas de visualização que podem ser de grande utilidade no projeto de uma:

- **Abstração**: possibilita que se evite as limitações de uma linguagem de implementação. Entretanto, o nível de abstração empregado deve ser muito bem avaliado, pois os extremos podem causar dificuldades. Por exemplo, a exclusão de muitos detalhes pode esconder aspectos importantes da execução, simplificando-a, assim como a inclusão de muitos detalhes pode atrapalhar o acompanhamento da execução. O uso de filtros é aconselhado para flexibilizar e personalizar a visualização, mas requer uma certa especialização por parte do usuário.

- **Ênfase**: as mesmas informações podem ser mostradas no mesmo nível de abstração, mas com diferentes ênfases. Por exemplo, anomalias como *deadlocks* ou condições de corrida podem ser enfatizadas de formas diferentes numa mesma visualização.

- **Representação**: a escolha de um meio de representação atrativo pode ter um sólido impacto na compreensão, motivação e entusiasmo e pode gerar um maior interesse na ferramenta. A representação é parcialmente determinada pelos níveis de abstração e ênfase da ferramenta.

- **Navegação**: permite que o usuário explore e interaja com o ambiente de visualização e oferece meios de redução ou aumento da complexidade visual através da variação dos níveis de abstração, ênfase e representação. A habilidade de navegar num potencialmente grande e complicado ambiente visual é geralmente uma parte essencial de qualquer ferramenta de visualização.

Além desses aspectos gerais de visualização, alguns específicos para ferramentas de depuração paralela *on-line* podem ser definidos, tendo como base o objetivo da ferramenta, os tipos de informações disponíveis e os trabalhos já realizados. Para a depuração, o essencial é que a visualização ajude a controlar uma grande quantidade de objetos de depuração. O usuário deve acompanhar o comportamento geral do sistema de forma visual para que consiga ter uma percepção macroscópica do mesmo. Assim, pode-se definir as seguintes características desejáveis para a visualização gráfica de um depurador paralelo:

- **Visualização geral**. A ferramenta de depuração deve possibilitar que o usuário possa visualizar todos os elementos paralelos (*e.g.*, processos) que compõem sua aplicação.

- **Visualização dos estados de cada elemento**. A ferramenta de depuração deve possibilitar a distinção dos estados pelos quais cada elemento passa durante sua execução.

- **Visualização "filtrada"**. A ferramenta de depuração deve oferecer mecanismos de filtragem, que coloquem elementos da aplicação não interessantes ao usuário à parte, ou seja, momentaneamente de fora do processo de depuração. Esta filtragem deve ocorrer tanto visualmente quanto no que diz respeito ao controle distribuído da depuração. Esta característica está diretamente relacionada ao fator de *escalabilidade* da ferramenta, visto que pode reduzir o número de elementos a serem visualizados e controlados.

Pode-se dizer que a implementação de mecanismos de filtragem muito provavelmente se firme como uma tendência em ferramentas de auxílio à programação paralela e distribuída. Tal afirmação se deve ao fato de tais mecanismos possibilitarem uma abstração da aplicação, o que se mostra muito conveniente em aplicações paralelas

com grande número de elementos distribuídos. Um exemplo de ferramenta com sofisticado sistema de filtragem é a PAJÉ [STE 98], que é principalmente voltada à análise de desempenho de programas.

3.4 O padrão HPD

Apesar de não estar mais ativo (pelo menos aparentemente), o *High Performance Debugging Fórum* (HPDF [FRA 99, HPD 99]) contribuiu na área da depuração paralela com uma série de definições com o intuito de estabelecer um padrão na área. Um dos principais objetivos deste padrão é o de proporcionar a consistência entre plataformas de tal forma que usuários que tenham que trabalhar em diferentes ambientes consigam fazê-lo com o mínimo esforço de aprendizagem de um novo depurador paralelo. Entre os participantes do fórum estão pesquisadores da área, desenvolvedores de depuradores comerciais e representantes de usuários HPC (*High Performance Computing*).

O modelo de programação adotado pelo fórum é explícito. Compreende programas compostos por várias *threads*, que dividem o mesmo espaço de endereçamento, e com memória distribuída, através de processos que se comunicam por troca de mensagens. A sintaxe elaborada considera principalmente o uso das linguagens C, C++ e Fortran, que são as mais comuns entre programadores de aplicações de alto desempenho. Além disso, considera que os depuradores são capazes de interagir com ambientes paralelos tais como PVM [GEI 94] e MPI [PAC 97].

Assim, são três os modelos de paralelismo considerados no desenvolvimento do padrão: aplicações paralelas compostas somente de processos, aplicações compostas somente de *threads* e aplicações multiníveis (vários processos e *threads*). Desta forma, o padrão se aplica a depuradores preocupados em suportar estes três modelos.

Num primeiro momento, o padrão não considera interfaces gráficas (GUI´s – *Graphical User Interfaces*). Inicialmente, foi definida uma linguagem de comandos com o objetivo de servir como base para uma futura definição de uma GUI padronizada. A primeira versão do padrão foi lançada no final de 1998 e pode ser encontrada em [HPD 99].

Entre as principais definições realizadas pelo fórum estão a definição do conceito de **conjuntos de processos/*threads***, a definição de **conjuntos pré-definidos de processos/*threads*** e o estabelecimento de um **modelo para início e parada de processos/*threads***. Assim, pode-se dizer que o mérito deste fórum está em definir um modelo consistente de depuração paralela.

3.4.1 Conceito de conjunto de processos/*threads*

Este conceito, segundo o padrão, fornece a base para que se estenda a semântica de operações de depuração serial para uma forma que se aplique à depuração paralela. Este conceito permite que um comando de depuração possa ser aplicado a todo

um conjunto de processos/*threads*, ao invés de apenas a um processo por vez. Foram definidos três tipos de conjuntos de processos/*threads* (p/t):

- **Conjunto p/t destino**. O conceito de conjunto p/t destino é usado para restringir a amplitude de um comando do depurador tal que ele se aplique a um, muitos ou todos os elementos da aplicação paralela sendo depurada. Em outras palavras, esse conjunto define qual ou quais serão os processos/*threads* destino dos comandos distribuídos de depuração.

- **Conjunto p/t corrente**. O padrão especifica uma série de comandos e a sintaxe necessária para referenciar os conjuntos p/t. Assim, o usuário pode referenciar um determinado conjunto a qualquer momento. Entretanto, sempre haverá um conjunto p/t corrente. Inicialmente, o conjunto p/t corrente corresponde a todos os processos e *threads* envolvidos na execução da aplicação. A mudança deste conjunto se dá através de um comando definido na linguagem especificada no padrão.

- **Conjunto p/t afetado**. Um conjunto de critérios, que dependerá do comando a ser executado, deve ser utilizado para determinar se um determinado comando pode ou não ser aplicado a determinados membros do conjunto p/t corrente. O conjunto afetado consiste em todos os processos e *threads* para os quais o comando é válido. Por exemplo, um comando que reativa um processo parado não se aplica a um processo que já está em execução.

3.4.2 Conjuntos pré-definidos

O depurador deve criar e manter **conjuntos pré-definidos** de processos/*threads* (p/t), tal que possam ser referenciados pelo nome. A versão 1 do padrão define seis conjuntos, a saber:

- **all**: todos os p/t associados ao programa sendo depurado.

- **running**: todos os p/t no estado *running* (em execução).

- **stopped**: todos os p/t no estado *stopped* (parados).

- **runnable**: todos os p/t no estado *stopped*/*runnable* (pronto para executar).

- **held**: todos os p/t no estado *stopped/held* (em espera por algum evento específico que independe do usuário).

- **exec** (executável): todos os p/t associados a um determinado executável.

Além dos conjuntos pré-definidos do depurador, os usuários podem criar um número indefinido de **conjuntos definidos pelo usuário**, que podem refletir a lógica do programa. Por exemplo, conjuntos podem ser estabelecidos para processos clientes e outros para servidores.

3.4.3 Modelo de início e parada de processos/*threads*

Numa depuração serial, como só há um fluxo de controle, a ocorrência de um evento como a chegada a um *breakpoint* tem um tratamento trivial: a *thread* única é parada. Já numa depuração paralela, deve-se definir o que acontecerá aos demais processos e *threads* quando da ocorrência de um evento deste tipo.

Normalmente, o modelo adotado em depuradores que suportam múltiplas *threads* é o denominado *stop-the-world*. Neste modelo, sempre que uma *thread* do processo em execução chegar a um ponto de parada, todas as demais também serão paralisadas. Ao mesmo tempo em que possui a vantagem de permitir que o usuário examine todas as *threads*, este modelo possui a desvantagem de ser intrusivo.

Depuradores que suportam múltiplos processos adotam um modelo diferente. Neste caso, somente o processo que encontrou o ponto de parada é paralisado, os demais continuam normalmente. Este modelo é referido pelo padrão como *leave-others-alone*.

O HPDF adotou um modelo híbrido na definição do padrão HPD. Ele define a utilização do modelo *stop-the-world* para as *threads* de um mesmo processo e a utilização do modelo *leave-others-alone* para os processos de uma maneira geral. A vantagem é que este modelo pode ser implementado em todos os sistemas HPC.

3.5 Considerações finais

Existem basicamente dois tipos de ferramentas de detecção de erros em programas paralelos. A primeira, que visa detectar erros simbólicos, permite a análise dos estados específicos do programa em qualquer nível de detalhe (estado das variáveis e instruções executadas). A segunda, visa a detecção de erros relativos ao paralelismo e sincronização entre processos paralelos.

Este trabalho tem como objetivo analisar o desenvolvimento de interfaces e propor um modelo de ferramenta que atenda ao primeiro tipo de depuração paralela, a simbólica. Este tipo de ferramenta é fundamental, já que a maioria dos erros de um programa pode ser detectada através da depuração simbólica. Assim, o projeto da ferramenta de depuração apresentada neste trabalho definiu o tipo de execução como sendo *on-line*. Com isso, o objetivo é disponibilizar, em tempo de execução, o maior número possível de informações ao usuário relativas a cada processo da aplicação.

Outra funcionalidade do modelo diz respeito à seleção de grupos de processos. Como definido no HPDF, uma ferramenta de depuração deve prover um mecanismo de seleção de processos/*threads*, tal que cada comando de depuração seja distribuído apenas aos processos/*threads* selecionados (grupo). Além disso, grupos pré-

definidos e grupos definidos pelo usuário também fazem parte da definição HPD e do modelo que será descrito nos próximos capítulos. Este tipo de abordagem facilita o controle de processos interessantes ao usuário, agindo como uma espécie de filtro.

4 Ferramentas de depuração

Este capítulo tem por objetivo traçar um panorama dos trabalhos mais significantes na área atualmente, apresentando uma série de depuradores e suas principais características. No final, algumas considerações são apresentadas com o objetivo de relacionar as ferramentas apresentadas ao presente trabalho.

Um aspecto importante a ser considerado na depuração paralela é a interface do depurador. Esta interface irá influenciar diretamente no grau de complexidade do processo de depuração. Esta seção aborda algumas ferramentas de depuração existentes, principalmente no que diz respeito às suas interfaces.

Em geral, as interfaces podem ser texto-intensivas ou gráfico-intensivas, dependendo do tipo de depuração para o qual foram projetadas. As interfaces texto-intensivas apresentam as informações utilizando-se quase que exclusivamente de texto. Os comandos podem ser enviados ao depurador através de uma linguagem ou através de botões específicos. O resultado do comando é puramente textual. Este tipo de interface é o mais utilizado para a depuração simbólica tradicional, pois elementos como código fonte e conteúdo de variáveis são informações essencialmente textuais.

Já as interfaces gráfico-intensivas são adequadas à análise dos programas no que se relaciona aos aspectos do paralelismo, como sincronização, detecção de *deadlock*, condições de corrida, etc. Este tipo de interface implementa tipos de visualização diferentes para a demonstração de dados e eventos relativos à execução da aplicação. O uso de gráficos e animações também pode ser de grande auxílio à depuração simbólica principalmente para a demonstração do estado geral da aplicação.

De uma maneira geral, a distinção entre estes dois tipos pode ser controversa, pois ferramentas textuais podem se utilizar de gráficos, ainda que não de forma intensiva. As ferramentas de depuração apresentadas no decorrer deste capítulo são as seguintes: *gdb* [STA 94], TotalView [ETN 01], p2d2 [HOO 96], DETOP [WIS 96], Prism [SIS 94], MAD [KRA 97], Annai [CLÉ 96a], Fiddle [CUN 98], Panorama [MAY 96], DCDB [WUX 99] e Net-dbx [NEO 2001]. As três últimas não tiveram influência direta sobre o modelo proposto, por isso são mais rapidamente descritas numa seção específica.

A análise foi realizada considerando aspectos relevantes para uma futura comparação com o protótipo PADI, utilizado para a validação do modelo desenvolvido. Estes aspectos são a existência ou não de visualização de processos, o tipo de mecanismo de seleção oferecido (quando presente), a existência ou não de grupos pré-definidos e a arquitetura alvo.

4.1 GDB

O *gdb* [STA 94] é um depurador simbólico que atua no nível de processos que podem ou não pertencer a uma aplicação paralela. O *gdb* foi

desenvolvido para depurar programas em plataformas do tipo Unix e seu funcionamento se dá a partir do interpretador de comandos.

A possibilidade de se utilizar o *gdb* para a depuração de aplicações paralelas advém do fato de que pode-se associar um depurador a cada processo de uma mesma aplicação. Desta forma, é possível que se controle a partir de interpretadores de comandos diferentes quantos forem os processos da aplicação. Obviamente que se o número de processos for muito grande, haverá uma certa dificuldade em se controlar todas as janelas abertas. A complexidade de uma operação como esta pode se tornar intolerável.

Ao longo do tempo, o *gdb* tem sido usado como base para os mais diversos tipos de interface e ferramentas de depuração (e.g., p2d2, Net-dbx e Fiddle). Além de ser uma ferramenta do tipo *free*, protegida pela *GNU General Public License* (GPL), o *gdb* implementa toda a interação de baixo nível necessária entre um depurador e a execução de um processo. Depuradores como o GDB são comumente referenciados no contexto da depuração paralela como *depuradores de processo*.

4.2 TotalView

O TotalView [ETN 01] é um depurador para programas paralelos escritos em C, C++, Fortran e ainda PVM e MPI. A apresentação da depuração é textual e dividida em várias janelas, cujas principais são:

- *Root Window*: lista todos os processos sob o controle do depurador.
- *Process Window*: permite a depuração simbólica do processo selecionado para executar nesta janela. Pode-se abrir janelas diferentes para cada processo que se deseja acompanhar (figura 4.1). É a partir desta janela que se tem acesso aos comandos de depuração.
- *Variable*: mostra o endereço, o tipo e o valor de uma variável local, registrador ou variável global.

Para depuração de programas paralelos, o TotalView apresenta a definição de grupo de processos e *breakpoints* flexíveis. Um grupo de processos no TotalView é, por padrão[3], constituído por todos os processos de um programa alvo. Assim, pode-se escolher iniciar, parar ou executar passo-a-passo todos os processos de uma aplicação ou apenas um.

Para pontos de parada, o TotalView oferece os *breakpoints* flexíveis (*Flexible Multi-Process Breakpoints*), que permitem que os pontos de parada, mesmo condicionais, sejam compartilhados entre processos relacionados que foram criados com o mesmo executável. Isto significa que se for definido um *breakpoint* na linha cinco da

[3] O manual, que pode ser encontrado em [ETN 01], descreve uma forma de alterar este padrão. Na versão de avaliação da ferramenta utilizada para testes este mecanismo não foi encontrado.

cópia de um executável, ele pode ser automaticamente compartilhado por todas as demais cópias deste executável, nesta aplicação.

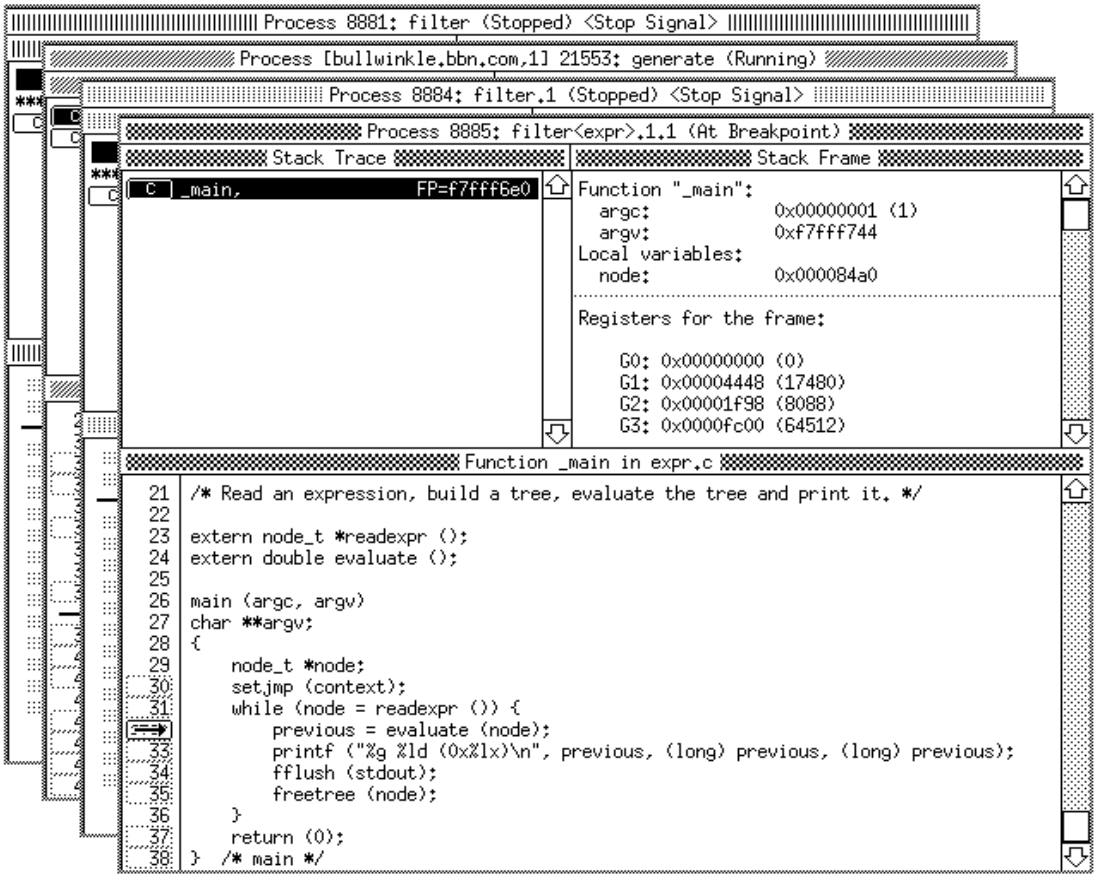

FIGURA 4.1 – *Process Window* do TotalView

Na depuração PVM e MPI existem outras janelas, baseadas nas características de cada biblioteca de comunicação. Para o PVM existe a *PVM Tasks and Configuration Window*, que apresenta a configuração do sistema PVM e uma lista das tarefas em execução.

Para o MPI, foi criada a *Message State Window*, que apresenta o estado de todas as comunicações existentes num processo MPI. Uma vantagem desta janela é que ela permite uma espécie de "navegação" entre o processo fonte e o destino da mensagem, fazendo com que o processo relevante fique sempre visível.

Atualmente, o TotalView é visto como uma espécie de padrão para depuradores paralelos. Possui um esquema de navegação *top-down*, onde se pode partir de uma visualização geral e chegar aos elementos mais básicos da aplicação para realizar a inspeção (níveis de coordenação e de processos). Entretanto, não existe uma separação clara e definida no nível da interface entre os níveis de abstração. Este fator, apesar de não comprometer o funcionamento da ferramenta, interfere na sua

intuitividade e facilidade de uso. Além disso, não há mecanismos de filtragem específicos.

Outra restrição ao ambiente se dá justamente no aspecto extremamente textual do mesmo, principalmente na inexistência de uma visualização gráfica de processos para o nível de coordenação que é representado na ferramenta pela *Root window*. O único recurso presente é a visualização de dados.

4.3 p2d2

O p2d2 (***p**ortable **p**arallel/**d**istributed **d**ebugger*) [HOO 96] é um projeto da NASA para o desenvolvimento de um depurador de aplicações paralelas com um alto número de processos. O objetivo, além da portabilidade, é a possibilidade de depurar programas com até 256 processos sem a necessidade de abrir uma janela para cada um.

O ambiente possui mecanismos de visualização geral e individual de processos, assim como controle de depuração distribuído para os processos selecionados. A visualização geral é viabilizada através de um elemento gráfico da interface, que é uma grade de processos (*process grid*). Nesta grade, cada processo da aplicação paralela corresponde a uma célula. A identificação de cada processo se dá basicamente através do posicionamento do mesmo na grade. Além disso, é possível selecionar qualquer coluna tal que a área da interface imediatamente ao lado da grade forneça informações adicionais de forma textual. Entre estas informações estão o identificador do processo (*pid*), o nome do executável e seu estado atual.

Uma das vantagens da grade é a possibilidade de representação dos processos através de ícones previamente escolhidos pelo usuário que indicam o estado de cada processo. Não menos importante é a possibilidade de se definir, através da grade, grupos de processos habilitados a receberem os comandos de depuração (*control set*). Assim, os comandos de controle tais como *continue* e *step* são sempre enviados a todos os processos que fazem parte do *control set*.

A visualização de código se dá apenas com um processo de cada vez, o chamado *focus process*. Este, tem seu código demonstrado na área de código da interface, o mesmo acontecendo com sua pilha de chamadas (*stack trace*). Através da grade pode-se selecionar até quatro *focus processes*, entretanto, apenas um de cada vez terá seu código demonstrado na área de código.

A figura 4.2 apresenta a interface do p2d2 na qual podem ser identificadas as áreas descritas brevemente acima. O p2d2 é um dos depuradores simbólicos paralelos mais poderosos e com um dos projetos de interface mais interessantes dentre os estudados. Ainda assim, pode-se identificar algumas desvantagens.

A primeira delas, relativa à grade de processos é a dificuldade de identificação dos processos. O posicionamento na grade é complementado pela janela textual posicionada ao seu lado, que funciona como uma espécie de legenda. É preciso que se selecione uma coluna e se faça uma contagem para relacionar a célula desejada com sua informação textual correspondente.

Note-se que os processos na grade estão agrupados por tipo de executável, onde cada coluna possui um tipo diferente (a não ser que o número máximo de processos por coluna seja extrapolado - neste caso os processo adicionais são colocados na coluna imediatamente posterior). Em [HOO 96] são apresentadas algumas alternativas a esse modelo, mas que aparentemente ainda não foram implementadas

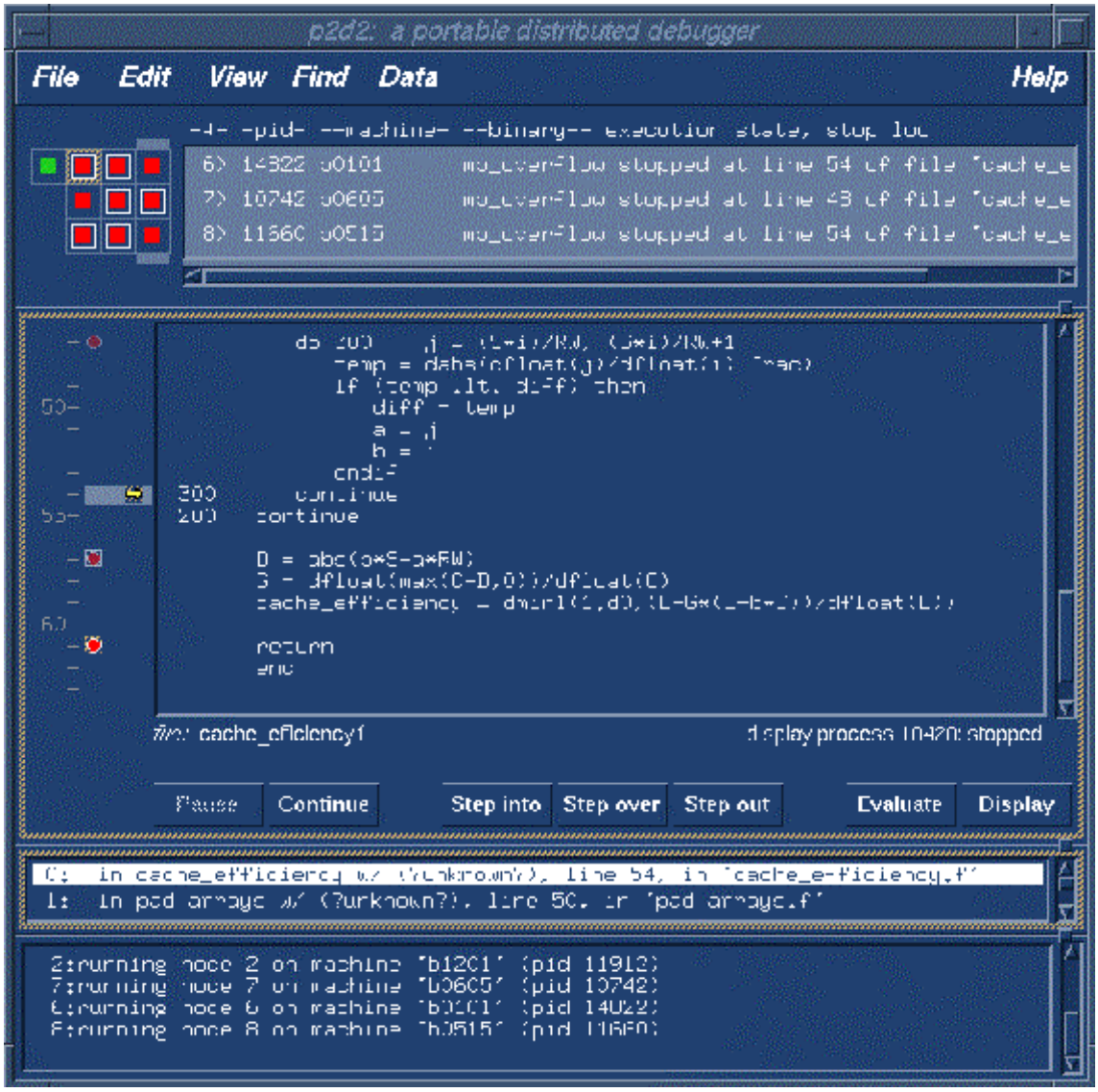

FIGURA 4.2 – Interface do p2d2

Quanto à área para código fonte, foi adotada uma abordagem que pode se tornar insatisfatória para alguns usuários. A ferramenta apresenta apenas o código fonte de **um** dos processos (o *focus process*), o que pode ser insuficiente caso o usuário deseje fazer um acompanhamento detalhado de mais de um processo ao mesmo tempo. Apesar disso, é possível que se acompanhe o estado de variáveis de mais de um processo.

4.4 DETOP

O DETOP [WIS 96] é um depurador *on-line* cuja principal vantagem é a de estar inserido num ambiente que integra uma ferramenta de avaliação de desempenho (PATOP) e uma ferramenta de monitoração de baixo nível para máquinas PowerPC. Assim, o DETOP pode oferecer, entre outras coisas, uma depuração baseada no paradigma de orientação a eventos. Neste esquema, basicamente o depurador é avisado pelo monitor do acontecimento de algum evento previamente definido como sendo de interesse do usuário. Outra vantagem do DETOP é o gerenciamento da criação dinâmica de *threads*, o que também é disponibilizado com o auxílio do monitor. A desvantagem de se ter um monitor deste tipo é dependência de arquitetura.

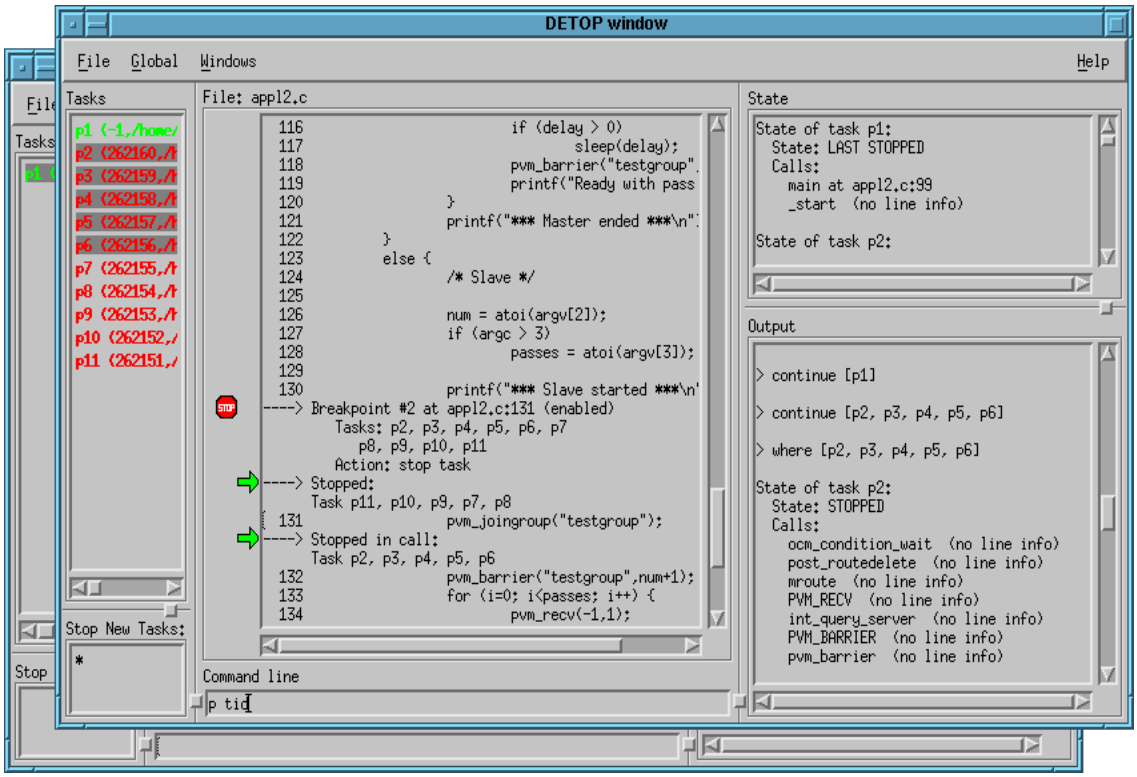

FIGURA 4.3 – Interface do DETOP

O *trace* gerado pelo monitor pode ainda ser usado para o desenvolvimento de animações, mas a interface do DETOP não oferece nenhum tipo de visualização gráfica de processos. Esta interface pode ser visualizada na figura 4.3.

No caso de *threads* que executam o mesmo código, o DETOP pode ser usado com apenas uma janela. Caso contrário, permite que múltiplas janelas sejam abertas para a depuração. A ação de parada quando um *breakpoint* é alcançado por uma *thread* pode se dar:

- na própria *thread*,

- em todas as *threads* do processo (nodo) a qual ela pertence
- ou em toda a aplicação.

Desta forma, o conceito de grupo processos/*threads* também é aplicado no DETOP.

4.5 Node Prism

O Node Prism [SIS 94] foi originalmente desenvolvido para a Connection Machine (CM5). Sua principal característica é a disponibilidade de uma linguagem especial para a definição de grupos de processos. Além da especificação através de listas e intervalos de identificadores, a definição de um grupo pode ser baseada em expressões envolvendo os dados da aplicação. É possível formar grupos de processos a partir da definição de expressões, tal que um processo fará parte do grupo se determinada expressão for válida em seu contexto. Além disso, o Node Prism mantém uma janela com todos os grupos definidos e que podem ser escolhidos a qualquer momento para que sejam o grupo corrente (atualmente selecionado).

Com relação à depuração propriamente dita, o Node Prism oferece mensagens textuais advindas dos depuradores e um mecanismo para filtrar mensagens redundantes. Além disso, existe a *where tree*, que é uma generalização de um *back trace* para multiprocessadores. Esta árvore agrupa *traces* de diversos processadores que fazem chamadas às mesmas funções a partir do mesmo ponto num código SPMD (*Single Program Multiple Data*). Também oferece visualização de dados através de representações gráficas de vetores e expressões.

4.6 MAD

O ambiente MAD (*Monitoring And Debugging environment*) [KRA 97] possui uma interface gráfica voltada à demonstração de eventos de comunicação e sincronização de processos. Isso se deve ao fato de a interface trabalhar com um arquivo de *trace* (execução *off-line* ou monitorada), de onde são extraídos os eventos mostrados na interface através de um diagrama espaço-tempo (figura 4.4).

Este diagrama, demonstra todos os processos em atividade através de linhas paralelas. A comunicação entre eles é apresentada através de linhas ligando os processos. Os eventos podem ser inspecionados através das janelas de eventos, localizadas logo abaixo do diagrama.

Outra característica do ambiente é a possibilidade de manipulação dos eventos. O usuário pode, por exemplo, alterar a ordem de chegada das mensagens e re-executar a aplicação. Isto pode ser realizado graficamente através do próprio diagrama espaço-tempo, responsável por mostrar os eventos.

O ambiente MAD é muito poderoso no sentido de facilitar a busca de erros de comunicação e sincronização. A manipulação de eventos também é uma vantagem deste ambiente, pois permite que se experimente com o programa. A única desvantagem do ambiente MAD é intrínseca ao seu modo de execução, que dificulta a depuração simbólica. Assim sendo, a aplicação paralela é considerada uma entidade única onde o tipo de informação que se pode conseguir é toda relativa aos eventos que foram armazenados durante a execução original.

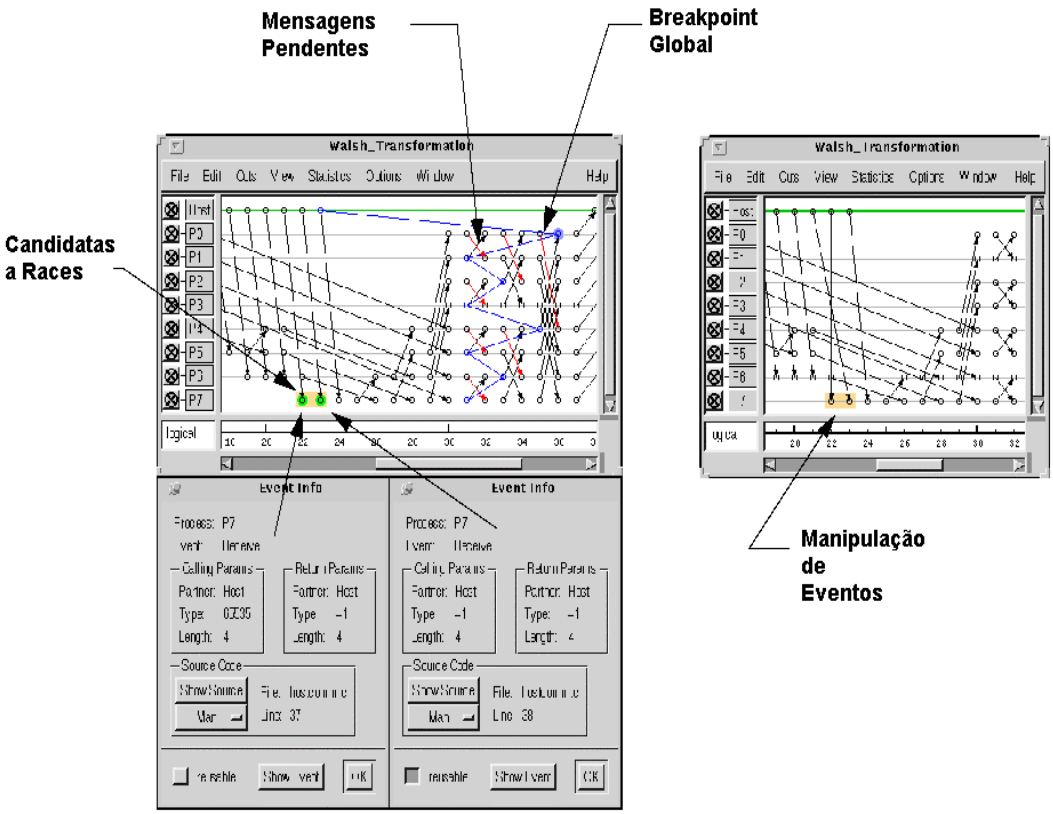

FIGURA 4.4 – Interface do ambiente MAD

4.7 Annai

O ambiente Annai [CLÉ 96a, CLÉ 96b] integra ferramentas para paralelização, depuração, monitoração e análise de desempenho de programas HPF (*High Peformance Fortran*) com passagem de mensagens através da biblioteca de comunicação MPI [PAC 97]. As três ferramentas básicas que compõem o ambiente Annai são: PST (*Parallelization Support Tool*), PDT (*Parallel Debugging Tool*) e PMA (*Performance Monitor and Analyzer*).

A ferramenta **PDT** é um depurador simbólico convencional, mas que suporta diferentes níveis de abstração. No nível de dados paralelos, oferece representações gráficas coerentes de grandes conjuntos de dados distribuídos (mostra tanto os valores dos dados quanto sua distribuição), assim como *breakpoints* com

condições globais de parada. No nível de passagem de mensagens, ajuda o programador na detecção de *deadlocks* e detecção de condições de corrida.

Assim, o ambiente Annai oferece basicamente duas opções: visualização do código fonte e visualização gráfica dos dados distribuídos. A visualização do código se dá em dois níveis. O primeiro mostra o código fonte completo e o segundo, que é complementar, mostra apenas os principais comandos no nível de estrutura do código fonte (por exemplo, *loops* e rotinas).

Já a visualização gráfica dos dados distribuídos é realizada através da reconstituição dos fragmentos do vetor distribuído, que se encontram em diferentes processadores. Esta reconstituição é realizada através de informações disponíveis graças à biblioteca de tempo de execução PST.

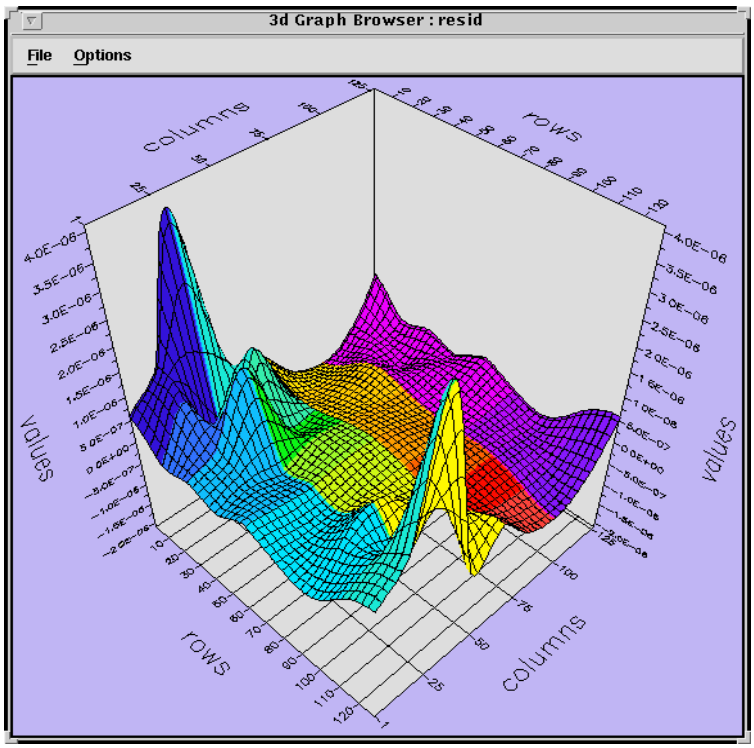

FIGURA 4.5 – Visualização de dados distribuídos no Annai

Breakpoints baseados em dados, ou *watchpoints*, especificam uma condição de parada em termos do estado da memória do programa, como por exemplo a modificação de um vetor.

O PDT também possui um cuidado extra em se tratando de “corridas” (*races*) entre mensagens. A biblioteca PDT utiliza um vetor de *timestamps* enviado juntamente com as mensagens para determinar, durante a recepção, se ocorreu uma corrida. Neste caso, esta ocorrência é anotada para que o usuário seja informado. Numa subseqüente re-execução, esta “anotação” pode ser utilizada para forçar um *replay*

determinístico da execução anterior. Além disso, permite que novos caminhos sejam escolhidos pelo usuário.

Nota-se que, graficamente, a preocupação do Annai é a demonstração da distribuição dos dados entre os processadores (figura 4.5). A detecção de problemas de comunicação e sincronização não é realizada de modo visual. Ainda assim, por ser do tipo de execução monitorada, o Annai oferece facilidades de re-execução para facilitar a solução deste tipo de problema.

4.8 Fiddle

O Fiddle [CUN 98] (cujo projeto foi anteriormente denominado PDBG) é uma infraestrutura de software que suporta depuração paralela e distribuída. Oferece um ambiente de depuração onde um cliente pode realizar chamadas a fim de solicitar praticamente qualquer tipo de comando de depuração sobre a aplicação que estiver sob o controle do ambiente.

Em outras palavras, consiste num conjunto de funções que implementam serviços relativos a depuração de programas paralelos. Entre as funções do Fiddle destaca-se o controle centralizado sobre um conjunto de elementos de depuração. Esses elementos são depuradores do nível de processos, onde cada um tem sob controle um processo ou um conjunto de *threads*. Assim, o Fiddle provê a interface entre um cliente (que pode ser uma interface gráfica de depuração) e a depuração paralela de uma aplicação.

O Fiddle possui uma série de primitivas que correspondem aos comandos de depuração a serem enviados aos depuradores do nível de processos. Estas primitivas podem ser divididas em classes:

- **Controle da depuração**. Contém primitivas para que o cliente possa controlar uma sessão de depuração, tais como colocar um processo sob o controle do depurador (carregar) e remover ou matar um processo.

- **Controle da execução**. Controla diretamente o caminho de execução seguido por um processo individual, uma vez que este é conhecido pelo depurador. São primitivas para iniciar a execução de um processo e dar continuidade à execução de um processo parado.

- **Inspeção e modificação**. Primitivas tradicionais para inspeção e modificação do estado de um processo em pontos determinados (o processo deve estar parado).

Assim, o Fiddle oferece um conjunto de rotinas básicas de depuração, assim como controla os processos paralelos sendo depurados. Entretanto, o Fiddle não oferece nenhum tipo de visualização ou seleção de processos, deixando esta implementação para o nível do cliente. O protótipo da ferramenta de depuração

apresentada nesta tese de doutorado utiliza o Fiddle como infraestrutura básica. Assim sendo, o Fiddle será abordado novamente mais adiante neste texto.

4.9 Outras

Esta seção aborda algumas ferramentas que não tiveram influência direta no projeto da PADI (a maioria são ferramentas recentes), mas merecem menção justamente por serem mais recentes e refletirem, portanto, algumas tendências na área (muitas das quais implementadas de alguma forma no modelo PADI). Estas ferramentas serão rapidamente abordadas nesta seção: Panorama [MAY 96], DCDB [WUX 99] e Net-dbx [NEO 2001].

O **Panorama** é um depurador simbólico híbrido, ou seja, possui depuração *on-line*, mas implementa re-execução baseada na ordem das mensagens gravadas num arquivo de *trace* durante uma execução específica da aplicação. Além disso, outra entre as principais características da ferramenta é a *extensibilidade*, ou seja, provê a capacidade de um usuário acrescentar suas próprias visualizações ao ambiente.

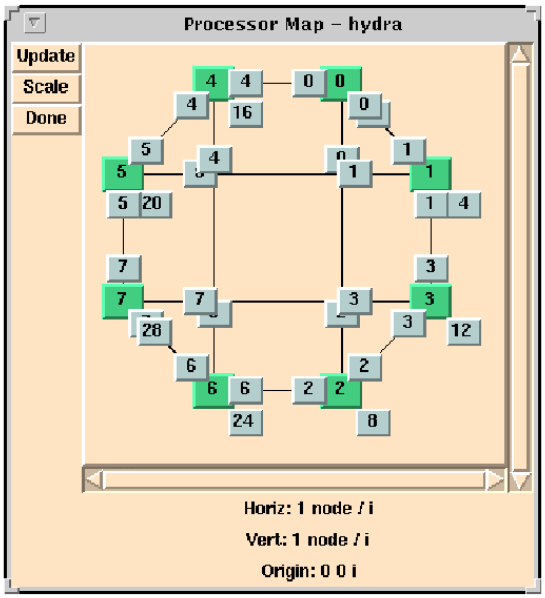

FIGURA 4.6 – *Processor map* do Panorama

Do ponto de vista da interface de depuração, o Panorama oferece quatro tipos de visualizações: a *Processor map*, a *Time line,* a *Array map* e a *Base debugger view*. A primeira (figura 4.6), oferece uma visualização dos processos distribuídos e da comunicação entre eles. Neste sentido, se aproxima bastante da PADI, mas acrescenta funcionalidades como a visualização do número de mensagens pendentes entre os processos. Em contra-partida, não há seleção de processos e o controle de depuração é feito através de janelas individuais, ou seja, não há grupos de processos e nem a possibilidade de se enviar comandos distribuídos. A segunda janela é do tipo espaço-tempo e funciona tanto no modo *on-line* quanto no *post-mortem*. A terceira provê

visualização de dados e a última é a janela individual que pode ser aberta para cada processo.

O **DCDB** (*Dawning Cluster DeBugging*) é um depurador paralelo *on-line* baseado num modelo cliente-servidor que interage com processos PVM e MPI a partir de clientes conectados a depuradores de processos, que podem ser o *gdb* ou o *dbx*. É baseado, portanto, no uso de depuradores seqüenciais e utiliza Java para a implementação de outras funções e interface. Neste sentido, assemelha-se bastante à PADI, pois também visa a portabilidade da ferramenta. Entretanto, as semelhanças terminam neste ponto.

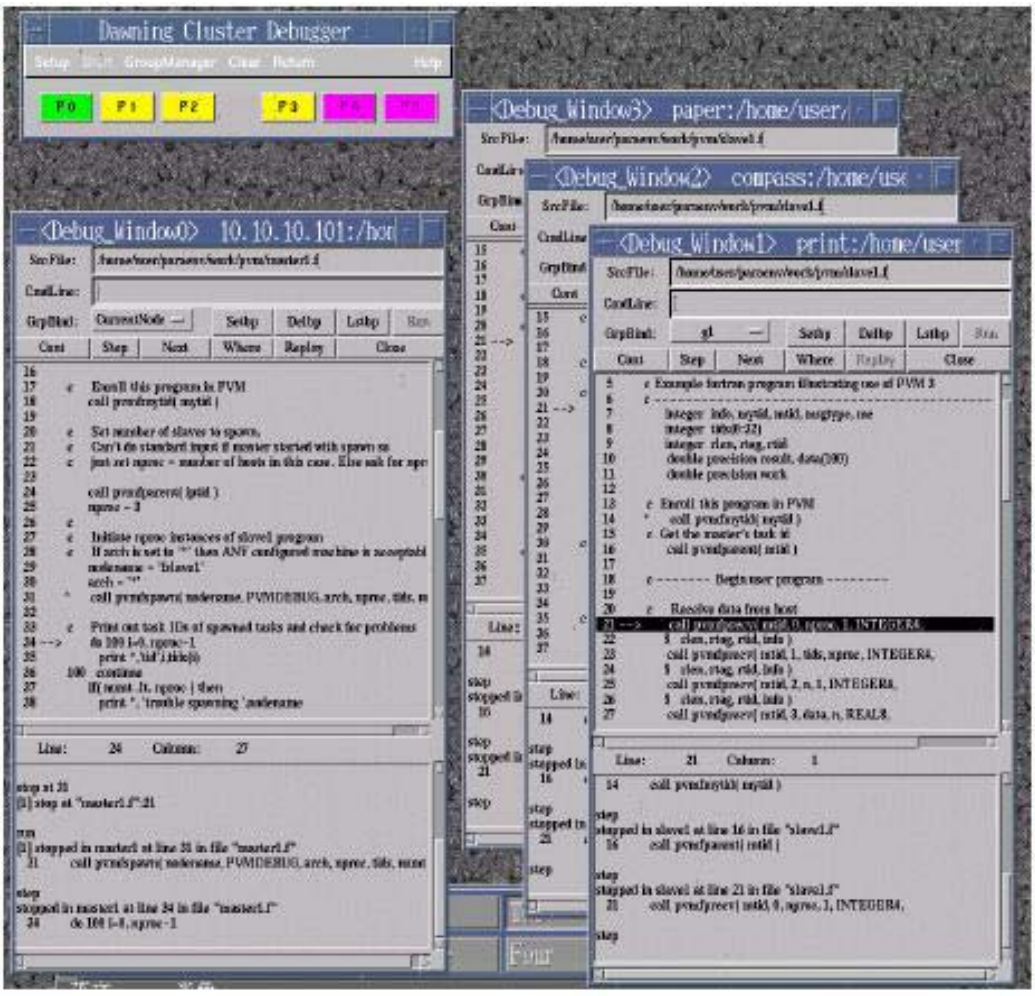

FIGURA 4.7 – Múltiplas janelas do DCDB

O DCDB possui uma interface inteiramente textual (figura 4.7), onde o código fonte de cada processo é mostrado numa janela interna semelhante ao depurador de processo escolhido (*gdb* ou *dbx*). O artigo de Wu *et al*. [WUX 99] não deixa claro como se dá o gerenciamento de grupos. A princípio, sub-processos criados a partir de um mesmo "pai" fazem parte de um grupo. O artigo citado também não faz referência à existência de um local na interface que centralize os comandos de depuração distribuídos. Ao invés disso, percebe-se que cada janela aberta de um processo possui os

comandos de depuração do depurador de processos seqüencial, o que não deixa claro qual é a semântica da ferramenta em relação aos comandos distribuídos. Enfim, o DCDB se limita a trazer janelas de depuração seqüencial remotas para o seu ambiente e possui um sistema de seleção de grupo bastante simples.

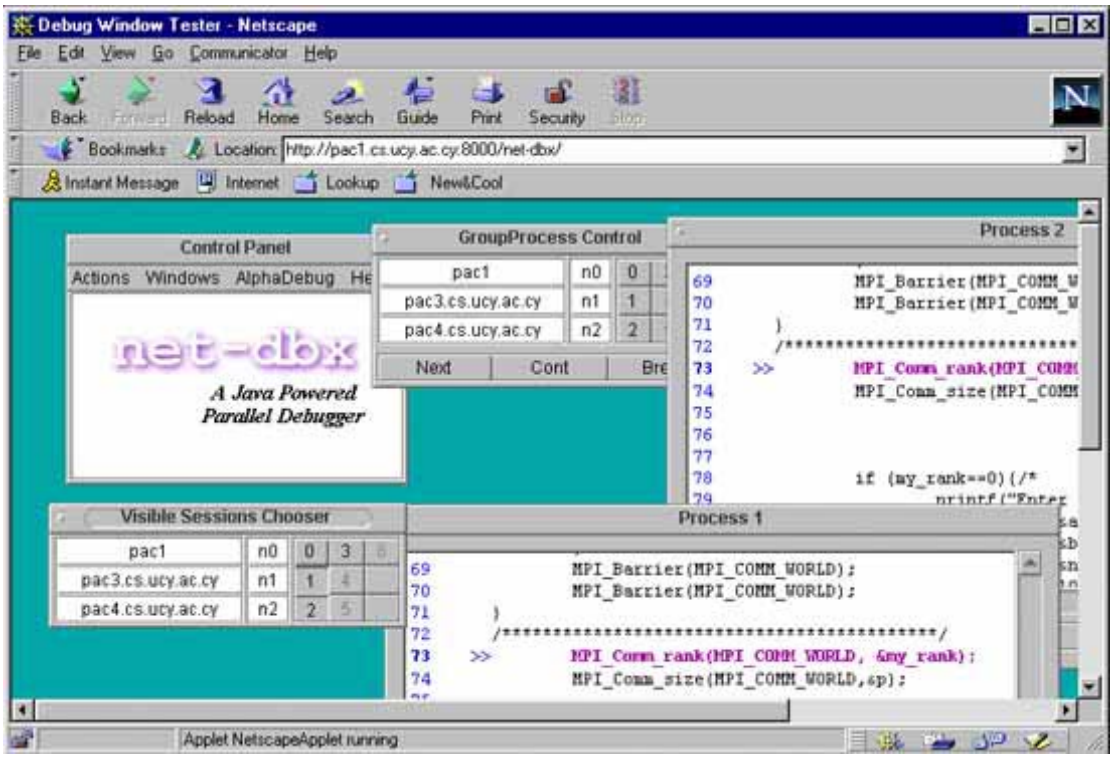

FIGURA 4.8 – Interface da ferramenta Net-dbx

O **Net-dbx** é uma ferramenta de depuração paralela *on-line* que pode ser utilizada via Internet através de um navegador. Isto é possível pois sua interface foi implementada utilizando-se Java *applets* e a comunicação com o ambiente paralelo (aplicações MPI) é realizada através de *login* remoto via *telnet/rsh*. Maiores detalhes sobre sua implementação, que utiliza o *gdb* como depurador local de processos, podem ser encontrados no artigo de Neophytou [NEO 2001] e na *homepage* da ferramenta ([NET 2002]).

No que diz respeito à interface e à interação com o usuário, que são os focos de interesse dessa análise, a Net-dbx possui preocupações semelhantes às do modelo PADI. Uma delas é a presença de um elemento de coordenação (*Process Coordinator*) responsável pela seleção de processos e envio de comandos distribuídos. Entretanto, o mecanismo de seleção é bem mais primitivo, pois não possui opção de grupos pré-definidos, apenas seleção de usuário. Além disso, não há visualização gráfica de processos. Um exemplo da interface da Net-dbx que é acessada via navegador pode ser observado na figura 4.8.

4.10 Considerações finais

Das ferramentas apresentadas neste capítulo, e que são as mais significativas atualmente, a que mais se aproxima do enfoque deste trabalho é o depurador p2d2. O TotalView não possui elementos que possibilitem visualização gráfica de processos e nem a seleção de processos. O MAD não possui inspeção simbólica, apesar de apresentar um sofisticado mecanismo de detecção de erros de sincronização. O Annai é um ambiente complexo e seu uso envolve a opção por um ambiente pesado e completo de visualização, depuração e análise de desempenho.

TABELA 4.1 – Sumário das principais ferramentas apresentadas

Características	TV	p2d2	DETOP	NP	MAD	Annai
Depuração simbólica	Sim	Sim	Sim	Sim	Não	Sim
Representação gráfica de processos	Não	Sim	Não	Sim	Sim	Não
Grupos pré-definidos	Não	Sim	Sim	Sim	Não	Não
Mecanismo de seleção de	Não	Sim	Sim	Sim	Não	Não
Níveis de abstração	Sim	Não	Sim	Sim	Sim	Sim
Boa identificação dos processos	Não	Não	Sim	Não	Sim	-
Disponível para *clusters* Linux/PC	Sim	Não	Não	Não	Sim	-

Já o p2d2 oferece visualização gráfica e seleção de processos. Entretanto, a visualização apresenta a dificuldade de se identificar os elementos paralelos da aplicação e a impossibilidade de se reduzir a quantidade destes elementos (não há mecanismos de "filtragem" para a quantidade de processos visíveis). Além disso, o p2d2 não permite que mais de uma janela de visualização de código seja aberta, o que poderia ser deixado à escolha do usuário (ter ou não um grande número de janelas abertas – que é o caso do TotalView). Um resumo das características abordadas é apresentado na tabela 4.1.

A seguir, serão descritos os principais conceitos e definições considerados no desenvolvimento do modelo proposto para interfaces intuitivas de depuração paralela, baseados na análise das interfaces apresentadas, além das características relevantes descritas no capítulo anterior. Algumas dessas definições são demonstradas através da própria apresentação da ferramenta PADI (*PArallel Debugger Interface*).

5 Desenvolvimento de uma interface intuitiva

Até aqui o trabalho realizado tem sido referenciado como *modelo* e/ou *ferramenta* PADI a fim de simplificar a compreensão. Entretanto, na realidade o trabalho pode ser dividido em três partes: projeto, *prototipação* e modelo resultante.

O projeto engloba o trabalho descrito até aqui de estudo e análise de interfaces, além da definição das características desejáveis que são descritas neste capítulo. A *prototipação* é a implementação do protótipo propriamente dito para a ferramenta PADI, onde foram aplicados os conceitos e definições de projeto. Esta implementação é descrita no próximo capítulo. O modelo resultante são as características do protótipo que podem ser aplicadas a qualquer projeto de interface de depuração com o intuito de deixar tais interfaces mais intuitivas e com um certo grau de padronização. Este modelo é descrito no decorrer do capítulo 7.

Assim, este capítulo descreve características desejáveis numa interface intuitiva. De acordo com o projeto aqui descrito, esta interface é voltada à depuração simbólica direta de programas paralelos e tem condições de controlar a depuração de forma distribuída. Quanto a este controle, o projeto da interface propõe meios para que o usuário estabeleça a amplitude dos comandos de depuração distribuídos através da seleção de processos. Além disso, apresenta um esquema de visualização dos elementos paralelos da aplicação indicando quais estão selecionados e quais são seus estados. Estas duas características desejadas são oferecidas através de duas funcionalidades: o **mecanismo de seleção de processos** e sua **área de visualização**, detalhados a seguir.

Este capítulo descreve as decisões de projeto, as principais funcionalidades, define o funcionamento do mecanismo de seleção de processos e como se dá a visualização dos mesmos e, por fim, apresenta o funcionamento básico do sistema, com vistas à posterior descrição da implementação do protótipo.

5.1 Decisões de projeto

O projeto PADI teve como principal objetivo tornar o uso da ferramenta nele baseada o mais intuitivo possível. Entre as decisões de projeto tomadas, destacam-se as seguintes:

Similaridade. Apesar de possuir características especialmente desenvolvidas para suportar o paralelismo, optou-se no projeto PADI por um ambiente de depuração similar aos ambientes das ferramentas seqüenciais, que já são de domínio da maioria dos depuradores, tornando a ferramenta, portanto, mais intuitiva.

Visualização. A visualização de processos é realizada através de unidades facilmente identificáveis por ícones numa área de visualização específica. Através desses ícones é possível:

- identificar o estado do processo através de sua cor;

- acessar o seu conteúdo (depuração simbólica de cada processo);
- selecioná-lo.

Filtragem. Somente processos selecionados são visualizados. O objetivo é possibilitar a redução de processos na área de visualização e tornar a depuração mais intuitiva, já que somente aqueles visualizados é que estão aptos a receberem os comandos de depuração.

Hierarquia. Os dois principais níveis hierárquicos numa ferramenta de depuração são claramente distinguidos:

- Coordenação: agrupa todas as funcionalidades que suportam o paralelismo, representado pela janela principal da ferramenta;
- Processo: agrupa as funcionalidades individuais de cada processo, representado pelas janelas de processo, onde tem-se uma instância para cada processo o qual se deseja aplicar a depuração simbólica.

Acessibilidade. A maioria dos comando básicos da interface são acessíveis diretamente através de botões, minimizando o uso de menus do tipo *pop-up*, muito comuns em ferramentas desse tipo. Ao invés disso, optou-se por utilizar ícones e recursos de *tooltips*.

Assim, no projeto PADI, essas foram as características desejáveis julgadas como as mais importantes e que devem estar presentes no projeto de qualquer depurador paralelo. Desta forma, uma das principais contribuições deste trabalho é a de apresentar um modelo básico que pode servir como proposta de um padrão para o projeto de interfaces de depuradores paralelos.

5.2 Mecanismos de seleção e visualização de processos

Os mecanismos de seleção e visualização de processos do nível de coordenação são os principais focos de atenção deste trabalho. O mecanismo de seleção tem o objetivo de agrupar e definir quais processos estarão sob controle do depurador a partir dos comandos de depuração distribuídos. A visualização gráfica permite que estes processos tenham seu desenvolvimento acompanhado numa área da interface específica para isso.

5.2.1 Seleção de processos

O mecanismo de seleção de processos tem como objetivo principal oferecer meios para que se possa trabalhar com a depuração de tal forma que seja possível recriar a lógica da aplicação. Assim, processos com funções semelhantes podem ser agrupados e manipulados separadamente de outros que tenham diferentes funções. O objetivo do projeto PADI é tornar este mecanismo o mais intuitivo possível, uma vez que o próprio processo de depuração de tais aplicações já é intrinsecamente complexo.

Assim, optou-se pela criação de grupos pré-definidos e pela acessibilidade a partir de botões na própria janela principal de depuração (nível de coordenação). Na verdade, os grupos podem ser divididos em três tipos diferentes (além daquele que engloba **todos** os processos), a saber: grupo definido pelo **estado** dos processos, grupo definido pelo **usuário** e grupo definido pelo **executável**.

A maioria dos grupos pré-definidos se baseia no estado de cada processo no momento da seleção. Quando selecionado, o conjunto de processos com o mesmo estado passa a ser o **conjunto corrente** e permanece o mesmo ainda que os estados dos processos selecionados se alterem (o que normalmente acontece no decorrer da depuração). Este é também um motivo pelo qual o mecanismo de seleção deve ser imediatamente acessível na interface, pois torna a depuração mais dinâmica. Estes grupos pré-definidos são baseados nos seguintes estados dos processos relativos à depuração: **pronto**, **em execução**, **parado** e **finalizado**.

TABELA 5.1 – Resumo dos grupos pré-definidos no projeto PADI

Grupo	**Tipo**	**Descrição**
Todos	Todos	Todos os processos da aplicação.
Pronto	Estado	Processos prontos para iniciarem sua execução sob o comando do depurador.
Executando	Estado	Processos em execução.
Parado	Estado	Processos parados pelo depurador, esperando comando.
Finalizado	Estado	Processos que terminaram sua execução.
Usuário	Usuário	Processos escolhidos pelo usuário.
Não usuário	Usuário	Inverte a seleção do usuário.
Executável	Executável	Baseado no nome do arquivo executável dos processos.

Os demais grupos são baseados em parâmetros diferenciados. O primeiro é o grupo de processos que compõe toda a aplicação (todos). Ainda existe a opção de seleção de todos os processos com o mesmo executável e o grupo de usuário. Este último tem a finalidade de permitir uma seleção mais de acordo com a funcionalidade de cada processo e de acordo com a visão do usuário sobre a aplicação. Permite que se realize uma operação dualizada, onde é possível selecionar os processos que fazem parte do grupo do usuário (pré-estabelecido por ele) ou os que não fazem parte do mesmo (inversão). Os grupos e suas funções são resumidos na tabela 5.1.

Como um dos principais objetivos é a simplicidade no uso das ferramentas de depuração paralela, optou-se por um mecanismo de seleção que pudesse estar imediatamente acessível na sua interface e que a ação correspondente à seleção fosse imediata e sem necessidade de aprendizado prévio. Desta forma, foram descartadas técnicas como a do Node Prism, que possibilita a seleção através da construção de expressões lógicas através de uma linguagem própria. Apesar de este ser um mecanismo poderoso em termos de seleção, vai de encontro ao objetivo do projeto PADI de facilidade de uso e *intuitividade*.

Em contrapartida, mecanismos demasiadamente simplificados como o do TotalView perdem em flexibilidade. O TotalView possui apenas dois mecanismos de seleção: um tipo de grupo pré-definido que são os *breakpoints* compartilhados entre processos de mesmo executável (e de mesmo "pai") e o passo-a-passo em grupo, onde a própria definição de grupo é um tanto obscura.

Os mecanismos de seleção do Net-dbx e do p2d2 se aproximam mais do propósito de aliar a facilidade de uso à flexibilidade na seleção. O Net-dbx apresenta uma lista de processos implementada através de botões, os quais servem para realizar a seleção de usuário. Este é um mecanismo muito simples, porém flexível. O protótipo PADI implementa um mecanismo semelhante através da seleção de usuário e acrescenta a possibilidade de inversão desta seleção. Esta abordagem é sugerida como sendo uma característica básica do modelo de interfaces que é descrito no capítulo 7, ou seja, pode ser implementada em qualquer ferramenta de depuração e deixa espaço para que novos esquemas opcionais também sejam implementados.

Já o p2d2, que possui uma grade (matriz) de ícones de processos para representar cada um, possui um editor que permite a configuração dos ícones, ou seja, é possível definir uma condição de estado do processo e associá-la à imagem do ícone. Possui um conjunto básico de ícones correspondentes aos estados dos processos e o usuário pode adicionar estados através do editor (tarefa, esta, que é opcional). A seleção se dá através da escolha dos processos através dos ícones (seleção de usuário) ou através da escolha de um estado.

O protótipo PADI possui um conjunto básico de seleção semelhante ao conjunto base do p2d2. A opção de adicionar condições de estado através de um editor *user-friendly* é uma proposta de trabalho futuro para o protótipo PADI, pois é um recurso que atende aos requisitos de facilidade de uso com maior poder de seleção. Entretanto, o protótipo PADI tem a vantagem de oferecer botões de seleção diretamente na interface e de oferecer uma operação dualizada através da inversão da seleção de usuário, opção esta inexistente no p2d2.

5.2.2 Visualização de processos

O mecanismo de visualização de processos tem como principal objetivo auxiliar no controle da depuração de uma aplicação paralela. Ela diferencia-se da visualização de dados, presente em algumas ferramentas de depuração, justamente por este motivo. A visualização de dados oferece meios para que se possa acompanhar visualmente as alterações que ocorrem principalmente com dados de vetores e matrizes distribuídos, estruturas muito comuns nesse tipo de aplicação. Já a visualização de processos provê meios para que se possa acompanhar o estado dos processos no decorrer da depuração e ainda oferece mecanismos para acesso e controle desses processos.

No projeto PADI, optou-se por aliar a visualização de processos ao mecanismo de seleção de processos de forma direta. Assim, somente os processos selecionados são visualizados. Pode-se dizer que os objetivos para se utilizar esses dois mecanismos em conjunto são a *intuitividade* e, de certa forma, a *escalabilidade*.

A *intuitividade* da ferramenta é alcançada em grande parte pela visualização de processos, pois possibilita que se tenha uma idéia geral da aplicação como um todo. O fato deste mecanismo estar diretamente associado ao da seleção de processos causa um efeito visual inteiramente coerente com a semântica da ferramenta. Em primeiro lugar, a seleção de processos restringe o alvo dos comandos de depuração. Assim, nem todos os processos estarão recebendo determinado comando – vai depender de estarem ou não selecionados. Desta forma, é mais adequado e lógico que se visualize apenas os processos que fazem parte da seleção.

Este aspecto de visualização seletiva está intimamente ligado ao outro objetivo mencionado, que é o da *escalabilidade*. O fato de que uma aplicação paralela pode ter um grande número de processos faz com que se tenha uma preocupação com o crescimento da visualização. Embora este não seja o principal foco de estudo desta ferramenta, a *escalabilidade* é atendida pela seleção de processos aliada à visualização. Isto acontece porque o mecanismo de seleção permite que se reduza o número de processos atingidos e, portanto, visualizados. Em casos onde isso não afeta a execução da aplicação, esta técnica pode ser utilizada para reduzir o número de processos na área de visualização, atuando como um mecanismo de filtragem. Por exemplo, uma alternativa a esse esquema seria apenas marcar os processos selecionados de alguma forma particular, como ocorre no p2d2, por exemplo. Entretanto, isto poderia causar uma confusão visual, na medida em que o número de processos pode ser elevado.

Pode-se dizer ainda que os dois mecanismos atuando em conjunto fazem com que se tenha a possibilidade de várias visões da aplicação a qualquer momento. Essas visões correspondem aos diferentes grupos de processos que podem ser selecionados. Como a seleção de processos é realizada através de um componente da interface que é um seletor, é possível que se altere diretamente a visualização um número indeterminado de vezes e a qualquer momento durante a execução da aplicação.

Entre as ferramentas descritas no capítulo anterior, que formam um conjunto significativo de ferramentas de depuração paralela encontradas atualmente, nenhuma possui uma abordagem como a do projeto PADI, que faz uma associação entre três aspectos importantes:

- visualização gráfica de processos;
- apresentação de ícones representativos;
- seleção que atua como um filtro para a visualização.

A proposta PADI com respeito à visualização é justamente associar estas três características com o objetivo de auxiliar na capacidade de compreensão e acompanhamento da aplicação através da visualização dos processos, incluir informações de forma clara e intuitiva sobre os estados dos processos através dos ícones representativos e permitir a redução de processos na janela através da visualização seletiva. Estes aspectos são encontrados individualmente em ferramentas como:

- Net-dbx, que possui um conjunto de botões que pode ser considerado como visualização de processos (primeiro item);

- Panorama, que também possui apenas visualização de processos (primeiro item);

- TotalView, que embora não possua visualização gráfica de processos, possui uma lista textual que pretende um efeito semelhante e diferencia os estados dos processos por meio de indicação textual e cores de texto diferentes (segundo item).

A exceção entre as ferramentas apresentadas é o p2d2 que possui os dois primeiros aspectos, se aproximando mais da proposta PADI. No p2d2, a grade onde os processos estão representados permanece demonstrando todos eles (o máximo é de 256), mas agrupa os que possuem semelhança de estado em colunas subseqüentes dentro da própria grade de visualização. A confusão visual pode se estabelecer na medida em que os processos são representados por ícones muito pequenos e que não há forma de se reduzir o número de ícones na visualização (não há mecanismo que exerça o papel de filtro, ou seja, a ferramenta não inclui o terceiro item citado).

Assim, um dos objetivos do projeto PADI com a visualização é o de demonstrar a importância de cada um destes três aspectos não apenas individualmente, mas em conjunto. A partir daí, a estrutura da PADI pode ser utilizada como um modelo para projetos de futuros depuradores, como proposto mais adiante neste texto.

5.3 Descrição da interface

O principal objetivo do projeto PADI é o de desenvolver um modelo para interfaces simples e capaz de atender aos principais requisitos de interfaces de depuração paralela. Como mostrado no capítulo anterior, algum trabalho no que diz respeito a interfaces de depuração paralela de uma maneira geral já foi realizado e merece ser considerado como ponto de partida. A estrutura do modelo proposto e do protótipo PADI é semelhante a dos depuradores simbólicos abordados na primeira parte do trabalho. Assim, o projeto de sua interface reflete esta estrutura básica, possuindo um nível de coordenação e um de processos, como demonstrado inicialmente na figura 3.1 e reproduzido na figura 5.1.

As seções subseqüentes descrevem as principais funcionalidades de cada nível no protótipo PADI, a fim de exemplificar a aplicação dos conceitos e definições do projeto aqui descrito. É importante salientar que o controle distribuído do protótipo é centralizado numa única janela, chamada *Main View*. É possível, entretanto, que cada processo seja controlado por uma janela especificamente aberta para isso. Esta janela é a *Process View*.

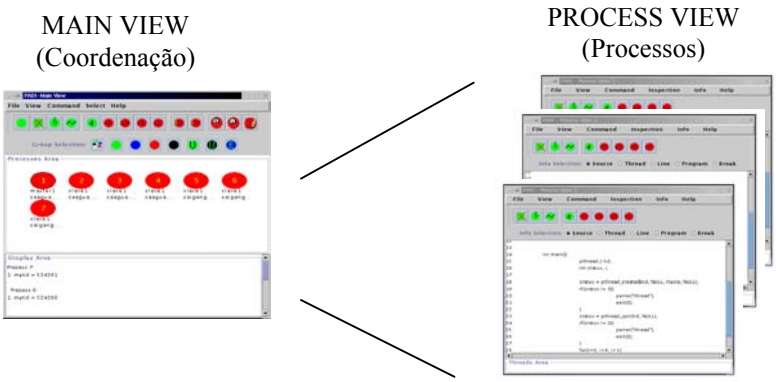

FIGURA 5.1 – Estrutura básica do protótipo PADI

5.3.1 Nível de coordenação

O nível de coordenação é o responsável por implementar as funcionalidades distribuídas no projeto PADI, incluindo seus mecanismos de seleção e visualização. A partir do nível de coordenação é possível acessar todos os processos que estiverem sob controle da depuração. O projeto da interface visa oferecer acesso simples e imediato aos mecanismos de seleção e visualização, assim como ao nível de processos. A simplicidade de operação é essencial para o sucesso deste tipo de interface, já que o entendimento e acompanhamento de um programa paralelo é intrinsecamente complexo.

Os comandos de depuração neste nível são, em sua maioria, distribuídos, ou seja, terão efeito sobre todos os processos selecionados. Os comandos propostos e que estão atualmente disponíveis no protótipo são:

- *Load*: carrega um executável, que pode ser o programa principal de uma aplicação distribuída.
- *Kill*: mata um ou mais processos.
- *Attach*: carrega um ou mais processos já criados. Estes processos podem ser escolhidos através de uma lista genérica de processos de uma máquina específica ou de uma busca baseada no nome do executável num conjunto de máquinas. No primeiro caso, o *attach* é individual. No segundo, todos os processos com determinado executável serão carregados.

- *Detach*: retira o processo do comando do depurador.

- *Run*: executa processos que estejam prontos (*ready*).

- *Step*: executa passo-a-passo os processos que estejam selecionados no momento e que estejam parados (*stopped*).

- *Next*: executa passo-a-passo sem entrar nas funções os processos que estejam selecionados no momento e que estejam parados (*stopped*).

- *Continue*: executa os processos selecionados no momento e que estejam parados (*stopped*) até o final ou até o próximo ponto de parada.

- *Finish*: finaliza chamada de função atual dos processos selecionados e que estejam parados (*stopped*).

- *Set breakpoint*: coloca um mesmo ponto de parada nos processos selecionados.

- *Delete breakpoint*: remove um ponto de parada de todos os processos selecionados.

- *Display variable*: mostra o conteúdo de determinada variável em todos os processos selecionados na área denominada *Watch*.

- *Undisplay variable*: retira a variável da lista de variáveis a serem visualizadas de todos os processos selecionados.

- *Set variable*: altera o conteúdo de determinada variável em todos os processos selecionados.

Além desses comandos tradicionais, que foram estendidos para depuração distribuída, a interface disponibiliza grupos pré-definidos para a seleção de processos (foram resumidos na tabela 5.1). Os comandos de seleção de grupos (*Group Selection*) que correspondem aos tipos de grupos pré-definidos, são os seguintes:

- *All*: corresponde a **todos** os processos sob o controle do depurador.

- *Ready*: processos no estado **pronto** no momento da seleção.

- *Running*: processos no estado **em execução** no momento da seleção.

- *Stopped*: processos no estado **parado** no momento da seleção.

- *Terminated*: processos no estado **finalizado** no momento da seleção.

- *User*: grupo definido pelo **usuário**.
- *Not user*: **inverte** a seleção de usuário.
- *Exec*: baseada nos nomes dos arquivos **executáveis** de cada processo.

Os componentes do grupo *user* são fixos, ou seja, aqueles definidos pelo usuário, possibilitando o estabelecimento de uma organização estática (alterada apenas pelo usuário) que reflita a semântica da aplicação paralela. Já os grupos pré-definidos baseados nos estados dos processos são dependentes destes estados, assim seus componentes são definidos no momento da seleção e portanto podem variar a cada seleção.

O mecanismo de seleção é propositalmente simplificado e acessível. Não há a necessidade de longas especificações baseadas em inúmeros estados de programação pelo qual cada processo pode passar. Tais especificações podem ser poderosas, mas muitas vezes são subutilizadas pela própria necessidade de longos períodos de aprendizado da ferramenta.

Quanto à visualização, o nível de coordenação proposto pelo projeto PADI inclui uma área central para visualização de processos. O objetivo é oferecer uma espécie de "painel de controle" da aplicação paralela, a partir do qual se pode ter acesso aos seus componentes e acompanhar o processo de depuração. Os processos estão dispostos nesta área como ícones (elipses, no protótipo atual) cujas cores representam seus estados. O acesso ao nível de processos se dá através dos próprios ícones. O mecanismo de seleção se insere neste contexto, pois limita o número de processos visíveis, facilitando o controle.

5.3.2 Nível de processos

O nível de processos é acessível a partir do nível de coordenação. Assim, é possível que se abra uma janela de controle para cada processo presente na área de visualização da janela principal. O objetivo deste nível é oferecer mecanismos de inspeção simbólica a cada processo sendo depurado.

Assim sendo, a interface do nível de processos assemelha-se à de qualquer depurador simbólico tradicional (seqüencial). Além dos comandos tradicionais há um seletor que permite escolher o tipo de informação que será visualizada (por exemplo, o código fonte do processo). Além disso, existe a área de *Watch* que serve para a visualização das variáveis.

A interface do nível de processos repete os comandos de depuração existentes na janela principal. Os comandos no nível de processos possuem um nível maior de prioridade, ou seja, quando invocados a partir deste nível, ignoram a seleção de processos do nível de coordenação. Assim, é possível que se depure cada processo individualmente.

No protótipo atual, é necessário que se abra uma janela para cada processo que se deseje depurar individualmente através da visualização do código fonte. Uma alternativa para que se evite um grande número de janelas abertas seria implementar uma abordagem como a do TotalView. Nesta ferramenta, janelas abertas são reaproveitadas para a visualização de processos, mas o usuário pode abrir novas janelas sempre que quiser. Esta abordagem é mais flexível que a do p2d2, por exemplo, que obriga ao uso de uma janela apenas.

5.4 Considerações finais

O desenvolvimento de uma interface intuitiva para depuradores de programas paralelos passa pela definição de características desejáveis e pela análise de ferramentas já existentes. O projeto da PADI foi baseado nas características de **similaridade**, **visualização**, **filtragem**, **hierarquia** e **acessibilidade**, como descrito nesse capítulo.

A partir dessas definições básicas de projeto, baseadas na análise do estado da arte, o protótipo da PADI foi estruturado. Os mecanismos de seleção e visualização surgiram da comunhão de duas dessas características: visualização e filtragem. A estruturação em dois níveis também resultou da união de duas características: similaridade e hierarquia. Por fim, o *layout* da interface, assim como tipos de comandos disponíveis, baseou-se também em duas dessas definições: similaridade e acessibilidade.

Assim, foi desenvolvido o projeto e o protótipo da ferramenta PADI. Aspectos da arquitetura e da implementação do protótipo são descritos no capítulo a seguir. Os capítulos subseqüentes apresentam as suas contribuições mais importantes.

6 Implementação e arquitetura do protótipo PADI

A ferramenta PADI é uma interface de depuração que se utiliza de uma infra-estrutura de depuração paralela. O nível de coordenação com características gráficas é o nível de abstração mais alto de um depurador paralelo. Por este motivo, optou-se por utilizar como camada inferior um ambiente que já realiza o trabalho de comunicação distribuída com o nível de processos. Este ambiente funciona como um servidor que dá acesso a este nível de processos. Ele oferece um nível de coordenação básico, ao qual a PADI acrescenta uma interface gráfica (GUI) e os mecanismos citados acima.

A decisão de se utilizar uma infra-estrutura pronta veio da complexidade de se desenvolver um sistema completo, que iria do projeto da interface às camadas mais baixas de software. Junto a essa complexidade, existe o fator tempo de desenvolvimento, que poderia se tornar muito extenso. O *framework* que se encaixou no esquema pretendido para a PADI foi o PDBG [CUN 98], recentemente renomeado para **Fiddle**. Entretanto, salienta-se que a arquitetura modular da PADI permitirá seu acoplamento a sistemas distintos, bastando para isso que se desenvolvam novos módulos de interação.

O Fiddle basicamente é o responsável por controlar de forma centralizada os depuradores remotos ou locais que agem no nível de processos. A PADI utiliza esta ferramenta no seu modo servidor, ou seja, envia os comandos de depuração como uma requisição para que sejam executados e recebe como resposta os resultados daquele comando. Esta interação, assim como a arquitetura da PADI e alguns detalhes de implementação, são descritos nas seções a seguir.

6.1 Arquitetura básica do protótipo PADI

A arquitetura básica da PADI compreende três módulos principais: interação, seleção e comunicação (figura 6.1). Estes módulos são os responsáveis pelo funcionamento básico da PADI.

Cada módulo tem uma funcionalidade específica, a saber:

- **Interação**. Este é o componente que implementa a interação com o usuário. Consiste numa GUI (*Graphical User Interface*) onde os comandos de depuração ficam disponíveis, assim como os resultados.

- **Seleção**. O módulo de seleção é responsável pelo suporte ao mecanismo de seleção de processos. Recebe os comandos do usuário, verifica seu escopo e envia para o módulo de comunicação.

- **Comunicação**. Este módulo realiza os passos necessários para a comunicação entre a PADI e o Fiddle. A princípio, esta

comunicação é feita via *pipe*, onde *strings* de comando são trocadas entre os dois ambientes.

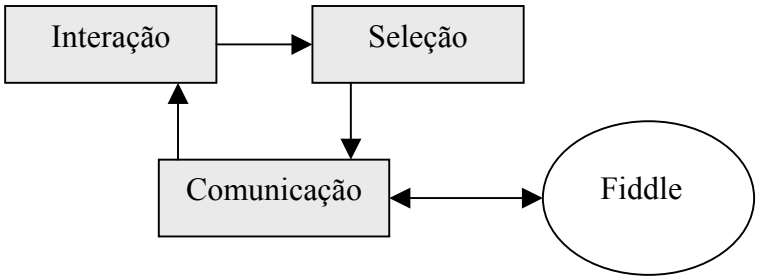

FIGURA 6.1 – Arquitetura básica da PADI

A implementação do protótipo da ferramenta PADI foi realizada através da linguagem Java 2 [HOR 99], cujos recursos de orientação a objetos e interface orientada a eventos foram de grande auxílio. As próximas seções apresentam detalhes e opções de implementação de cada módulo.

6.2 Interação com o usuário

O módulo de interação é composto basicamente por duas janelas que implementam os dois níveis de abstração do depurador: *Main View* e *Process View*. A *Main View* controla todos os aspectos distribuídos da depuração (nível de coordenação) e possui a área de visualização de processos. A *Process View* controla o nível de processo. O módulo de interação é responsável basicamente por receber os comandos do usuário e apresentar os resultados.

A implementação deste módulo foi realizada através do pacote Java Swing [SAT 99]. Assim, cada elemento da interface é um objeto e todos eles tem seu funcionamento orientado a eventos, incluindo aqueles que representam os processos, o que facilitou sua implementação. Nenhuma ferramenta visual de construção de interface foi utilizada, apesar de terem sido cogitadas. Na época do início da construção do protótipo, as ferramentas existentes ainda não haviam sido atualizadas para utilizarem o Swing. Além disso, o código gerado por essas ferramentas pode ser mais pesado.

O módulo de interação, portanto, é composto pelos objetos da interface, que são notificados sempre que algum evento ocorre (e.g., clique de *mouse* ou atualização de informações). Para cada elemento (objeto) da interface que corresponde a um comando de depuração, existe uma ação relacionada que vai ativar os objetos responsáveis pela sua execução, de forma que a requisição correta chegue ao módulo de comunicação com o Fiddle. Da mesma forma, os resultados retornados a partir do módulo de comunicação acionam os objetos responsáveis pela atualização da interface, como por exemplo a troca de estado (cor) de um processo ou a apresentação do conteúdo de uma ou mais variáveis, etc.

Com relação aos comandos de depuração realizados sobre os processos, o sincronismo com o restante do ambiente é conseguido basicamente através de um recurso de implementação da PADI: a criação de um estado intermediário chamado *debugging* (“em depuração”) que impede a requisição de novos comandos para os processos afetados[4]. Quando um comando é acionado, os processos no grupo corrente têm seu estado modificado para *debugging*. Visualmente, estes processos têm sua cor alterada para refletir este estado indefinido. A cor que indica o estado *debugging* no protótipo atual é a cinza. Sempre que um comando é finalizado, o estado e por conseguinte as cores dos processos são atualizados. Processos não afetados têm sua cor restaurada assim que o estado de invalidez é constatado.

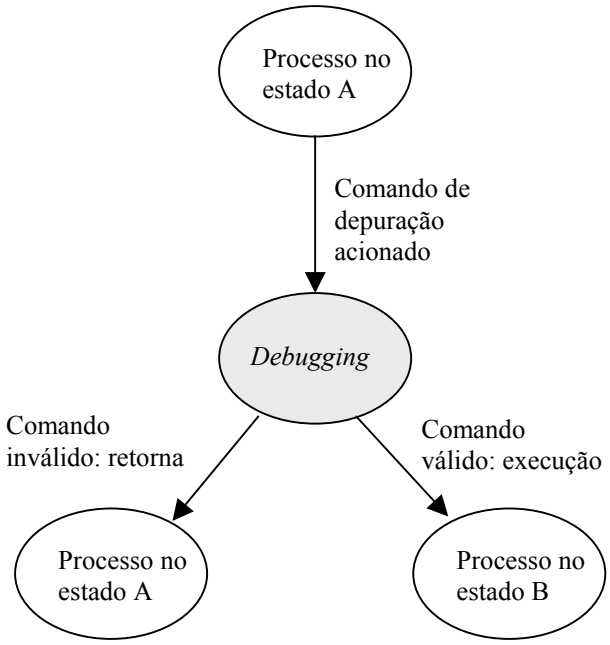

FIGURA 6.2 – Esquema de transição do estado intermediário *debugging*.

Esse mecanismo é interessante por vários motivos. Sempre que um comando for requisitado, os processos afetados devem ficar bloqueados até a finalização da execução do comando. O bloqueio é feito porque o estado do processo pode ser alterado a cada comando, portanto não se pode enviar um novo sem que o anterior tenha sido finalizado e que o estado do processo tenha sido atualizado.

Como mencionado, nem sempre todos os processos selecionados serão afetados por determinado comando. Esta diferença pode se dar por dois motivos:

---

[4] Processos *afetados* são os que estão no grupo corrente **e** num estado válido para execução do comando de depuração. O grupo *corrente* contém **todos** os processos selecionados. Estes conceitos serão retomados com maior profundidade na próxima seção.

- processos possuem um estado inválido para determinado comando ou

- o comando foi originado de uma janela de processo (*Process View*), o que faz com que a seleção seja ignorada e somente o processo dono da janela seja afetado pelo comando.

Assim, o recurso de estado intermediário também indica se o comando é distribuído ou individual, pois ou apenas um ou todos os processos passarão pelo estado *debugging*. A figura 6.2 demonstra o esquema de funcionamento e de transição dos outros estados para o estado intermediário *debugging*.

A implementação dos processos, estados e condições de seleção serão abordados mais adiante. Entretanto, cabe salientar que a solução adotada para bloquear novos comandos é eficiente e tem efeito visual, o que atende ao requisito de *intuitividade* da ferramenta.

6.3 Seleção de processos

O módulo de seleção mantém a estrutura que suporta o mecanismo de seleção de processos. Os principais elementos deste módulo são as classes que implementam as ações relativas aos comandos de depuração (as chamadas classes *Action*) e a classe que implementa o processo (classe *Proc*). Durante a execução, cada processo será representado por um objeto do tipo *Proc* e para cada comando de depuração um objeto do tipo *Action* será criado.

A classe *Proc* mantém vários atributos que são responsáveis por armazenar informações do processo que serão úteis quando da execução dos comandos. Entre estes atributos destacam-se:

- Identificador do processo;

- Nome da máquina;

- Nome do executável;

- Estado atual;

- Estado anterior (para restauração em caso de comando inválido – vide figura 6.2);

- Seleção (indica se o processo está selecionado ou não);

- Usuário (indica se faz parte do grupo do usuário).

A classe *Proc* também tem um papel importante no retorno dos comandos. É ela que recebe os resultados e ativa os comandos da interface necessários à sua atualização. Para isso, o módulo de comunicação com o Fiddle ativa o método desta classe correspondente ao resultado do comando. Portanto, além de conter todos os

atributos de cada processo, a classe *Proc* também implementa os métodos necessários para sua atualização, tanto interna (atributos) quanto externa (ativação dos métodos de atualização da interface, no módulo de interação).

As classes *Action* são acionadas pelos objetos da interface e são responsáveis por verificar duas condições relativas aos processos do grupo corrente. O teste das condições é feito através de consulta aos atributos da classe *Proc*. O comando é enviado somente se o(s) processos(s) alvo satisfizer(em) estas duas condições, especificadas abaixo:

- **Condição de seleção**. Verifica quais processos estão selecionados para receberem os comandos de depuração distribuídos. A classe *Proc* mantém um atributo que indica se o processo o qual representa está ou não selecionado. Esta condição não é verificada caso o comando tenha sido originado da janela individual de um processo (*Process View*), pois esta tem prioridade sobre a seleção atual.

- **Condição de estado**. Verifica, para cada processo do grupo corrente, seu estado de depuração atual e permite que o comando seja enviado somente se o processo estiver num estado válido para que seja executado (de acordo com o diagrama de estados demonstrado mais adiante na figura 7.2).

As condições de seleção e de estado diferenciam o grupo corrente do grupo afetado (correspondentes ao *current p/t set* e ao *affected p/t set*, definidos no padrão HPD – ver capítulo 3). O grupo corrente é o grupo de processos selecionados no momento. O grupo afetado é um subgrupo do grupo corrente cujos processos estão num estado válido para receberem determinado comando. Se todos os processos selecionados estiverem num estado válido, o grupo corrente será igual ao grupo afetado.

Como exemplo de verificação de estado, podemos citar o comando ***run***. Ele será enviado somente aos processos do grupo corrente que estiverem no estado *ready* ou *terminated*. Isto também funciona como um mecanismo de consistência e otimização, pois evita que comandos desnecessários sejam enviados ao próximo módulo e, por conseqüência, ao Fiddle (que não o executaria de qualquer forma).

Assim, um objeto *Action* é responsável por analisar todos os processos do grupo corrente e repassar o comando de depuração aos próximos módulos cada vez que verificar que um processo faz parte do grupo afetado. Quando termina a verificação, monta uma requisição com os dados necessários à execução do comando e a envia ao módulo de comunicação.

6.4 Comunicação: integração com o Fiddle

O Fiddle é o ambiente de depuração que dá suporte ao protótipo da PADI e, portanto, pode-se dizer que todas as funcionalidades da PADI, principalmente de seleção e visualização gráfica foram acrescentadas a este ambiente base. O Fiddle, realiza o trabalho de nível mais baixo do ambiente integrado, que é o controle dos

vários depuradores de processo que estão distribuídos. Estes depuradores de processo, como o próprio nome diz, são depuradores seqüenciais que ficam conectados aos processos da aplicação paralela durante a depuração. Atualmente, o Fiddle utiliza como depurador de processo o *gdb*, o depurador da GNU.

O Fiddle não é, a princípio, um depurador para o usuário final, pois não possui uma GUI. Além disso, ele provê apenas uma interação individual com cada processo remoto, ou seja, cada comando de depuração deve ser enviado individualmente para cada processo, o que pode ser muito trabalhoso se realizado manualmente. A PADI realiza este processo automaticamente através de sua janela principal, a *Main View*, deixando este procedimento seqüencial transparente ao usuário. Além disso, o mecanismo de seleção permite que o usuário especifique o escopo de atuação do comando. Enfim, a PADI oferece meios mais amigáveis de visualização dos resultadosda depuração, não apenas textuais, mas gráficos também.

6.4.1 Descrição do FIDDLE

O Fiddle proporciona um ambiente de depuração cujo objetivo principal é oferecer o controle centralizado sobre um conjunto de elementos de depuração. Esses elementos são depuradores do nível de processos, onde cada um tem sob controle um processo ou um conjunto de *threads*. Assim, o Fiddle provê a interface entre um cliente (que pode ser uma interface gráfica de depuração) e um ambiente de depuração paralela.

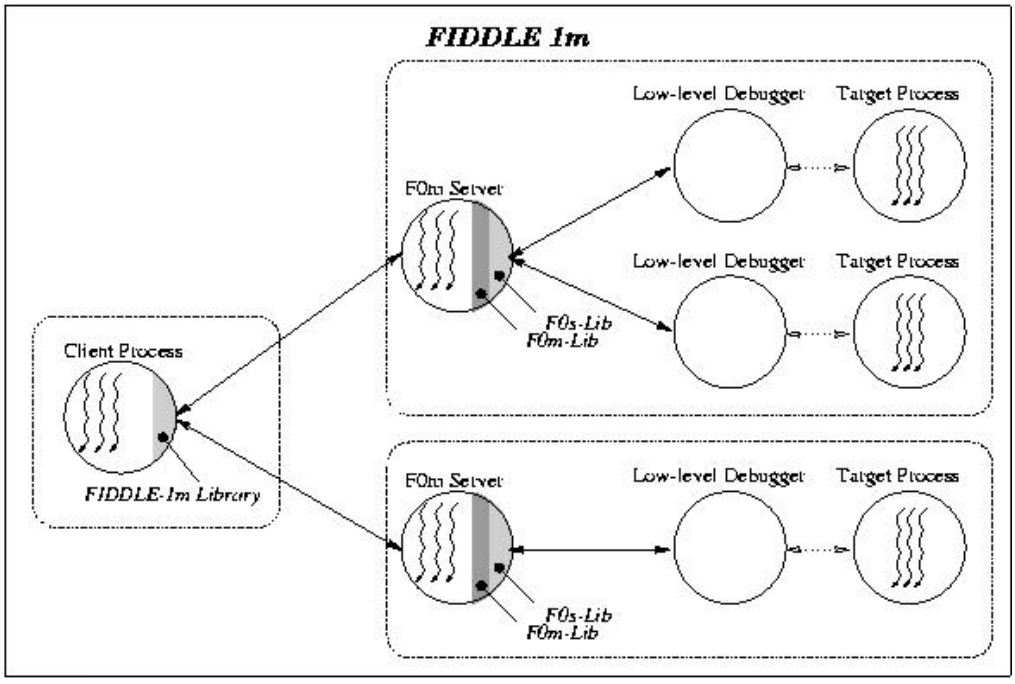

FIGURA 6.3 – Arquitetura do Fiddle

A figura 6.3 mostra um dos níveis de implementação do Fiddle e sua interação com o nível de processos. Nesta figura, podemos identificar os seguintes elementos:

- ***Target process***: processo da aplicação que está sendo depurada.

- ***Low-level debugger***: é o depurador do nível de processos que está controlando o *target process.* Atualmente o Fiddle trabalha com o *gdb*.

- ***F0m server***: é o servidor do Fiddle para coordenar a depuração local de um ou mais depuradores do nível de processos.

- ***Fiddle-1m library***: é a biblioteca que coordena a depuração remota de vários elementos *F0m server* (os coordenadores locais). Existe uma versão onde este controle remoto é realizado por um servidor, ao invés de ser uma biblioteca. Isso possibilita seu uso por parte de programas cliente não escritos na linguagem C/C++, utilizada no desenvolvimento desta biblioteca.

- ***Client***: é o programa que implementa a interface do usuário com o ambiente de depuração. A PADI se encaixa neste contexto como cliente do servidor Fiddle-1m.

O Fiddle possui uma série de primitivas que correspondem aos comandos de depuração a serem enviados aos depuradores do nível de processos. Estas primitivas podem ser divididas em classes [CUN 98]:

- **Controle da depuração**. Contém primitivas para que o cliente possa controlar uma sessão de depuração, tais como colocar um processo sob o controle do depurador (carregar) e remover ou matar um processo.

- **Controle da execução**. Controla diretamente o caminho de execução seguido por um processo individual, uma vez que este é conhecido pelo depurador. São primitivas para iniciar a execução de um processo e dar continuidade à execução de um processo previamente parado.

- **Inspeção e modificação**. Primitivas tradicionais para inspeção e modificação do estado de um processo em pontos determinados (o processo deve estar parado).

Assim, o cliente tem um conjunto variado de comandos de depuração que podem ser utilizados para o controle dos processos. O Fiddle se comunica com estes processos e possui um poderoso interpretador para as repostas devolvidas pelos depuradores de processos. Cabe salientar que esta comunicação é feita via caracteres e, portanto, cada resposta deve ser devidamente interpretada, assim com a *string* de entrada (comando) deve ser exata. O módulo de comunicação do protótipo da PADI realiza atualmente esta tarefa.

6.4.2 Interface PADI-Fiddle

A PADI está num nível acima do Fiddle, pois executa como um cliente. Considerando a figura 6.3, a PADI seria o elemento *client*, com a ressalva de que a biblioteca Fiddle não é ligada à PADI. Na verdade, o Fiddle é usado pelo protótipo atual na sua versão servidor e a comunicação é realizada por caracteres via *pipe*.

Em resumo, a PADI captura os comandos dos usuários, verifica sua validade e escopo, repassa estes comandos para o Fiddle para que sejam executados e reflete os resultados obtidos. A figura 6.4 demonstra esse relacionamento, omitindo a arquitetura do Fiddle, para simplificar.

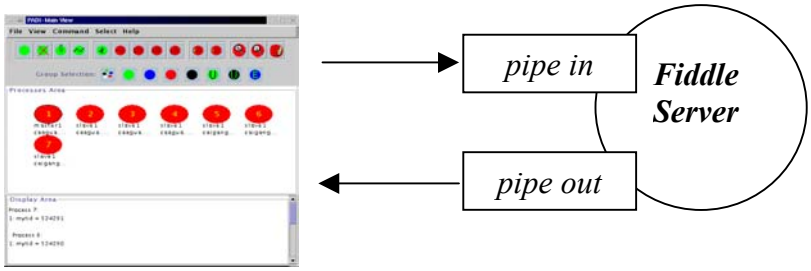

FIGURA 6.4 – Interface PADI – Fiddle (servidor)

O mecanismo de *pipe* foi utilizado para tornar as ferramentas independentes de linguagem. A biblioteca Fiddle foi desenvolvida para programas em C e C++, enquanto a PADI está sendo desenvolvida em Java. Foram realizados testes para analisar a possibilidade de utilização do Java - JNI, que permite a interação entre Java e código nativo (C/C++). Apesar de o Fiddle trabalhar com estruturas muito complexas, verificou-se a possibilidade de efetivar esta interação. Entretanto, como o processo poderia se tornar muito demorado, optou-se, num primeiro momento, pela comunicação via *pipe*. A interface com JNI está sendo desenvolvida em paralelo e poderá ser utilizada numa futura versão da PADI.

Através do *pipe* o módulo de comunicação da PADI envia as *strings* formatadas que contêm os parâmetros necessários para a execução do comando desejado. O formato dessa string é definido pelo Fiddle e corresponde às suas estruturas internas, uma para cada comando de depuração. Portanto, é necessário conhecer o formato exato para que o servidor do Fiddle interprete corretamente a requisição. Entretanto, no caso de se utilizar diretamente a biblioteca, isso não é necessário, pois todas as requisições, neste caso, seriam feitas através de chamadas de funções (C/C++). No caso da versão Java JNI, chamada Fiddle-j, essa comunicação poderá ser realizada através da chamada de métodos nativos.

A resposta, recebida via *pipe*, também é uma *string* formatada que é interpretada pela PADI a fim de que seja refletida no elemento de interface adequado. A *string* é recebida pelo módulo de comunicação e sofre um processo de *parsing* para que

as principais informações relativas ao resultado do comando de depuração que originou esta resposta sejam retiradas. Entre as principais informações retiradas dessa *string*, tem-se:

- **tid**: identificador do processo que originou a resposta;
- **ação**: tipo de procedimento que deverá ser tomado para a atualização do estado da depuração e da interface.
- **dados**: informações relativas ao resultado de determinado comando.

O *tid* é utilizado para que se acesse o objeto *Proc* correspondente (que representa o processo no ambiente PADI). Para tanto, existe um servidor responsável por armazenar as referências de todos os objetos *Proc* ativos e por devolver estas referências quando requisitado. A consulta, portanto, é realizada através do *tid* informado no resultado do comando de depuração.

A *ação* a ser tomada depende do tipo e/ou do formato da resposta. Por exemplo, se a resposta for um evento de chegada a um *breakpoint*, o processo correspondente terá seu estado atualizado para *stopped* e, se sua janela de processo estiver aberta, terá o ícone que indica a localização da execução atualizado para a linha corrente do arquivo fonte atual. A ação é realizada dentro do contexto de *Proc*, por isso é necessário obter sua referência antes.

Os *dados* indicam o novo conteúdo dos componentes da interface. Segundo o mesmo exemplo de uma chegada a um *breakpoint*, os dados seriam o valor da linha corrente e o nome do arquivo fonte corrente. Estes dados, portanto, são utilizados para a atualização da interface.

Basicamente as interações entre clientes e o Fiddle podem ser de dois tipos [CUN 98]: síncronas ou assíncronas. Estes tipos de interação correspondem principalmente às chamadas aos serviços da biblioteca Fiddle, mas se refletem também na interação do cliente com o servidor Fiddle:

- **Interação síncrona**. Boa parte dos comandos de depuração são síncronos, pois o invocador espera até que um valor seja retornado (indicando sucesso ou falha). Por exemplo, uma inspeção de variável pode ter sido requisitada e o seu valor é retornado para apresentação ao usuário. Uma falha pode ocorrer se a variável não existe.
- **Interação assíncrona**. Alguns comandos não têm resultado imediato e podem acontecer a qualquer momento. O Fiddle captura estes resultados assíncronos e os envia para o cliente. Como exemplo, temos a chegada de um processo ou *thread* a um *breakpoint* previamente colocado.

Como já mencionado, a comunicação do protótipo PADI atual com o Fiddle é realizada via *pipe*: um para envio das requisições e outro para recebimento de

resultados. Para simplificar a implementação, o módulo de comunicação da PADI trata estes dois *pipes* de forma independente (assíncrona). Assim, todos os comandos, inclusive os síncronos, são tratados de forma assíncrona, ou seja, as requisições são feitas sem espera por resposta. O *pipe* de recebimento captura todos os resultados e os passa para o módulo de interpretação. A implementação deste *assincronismo* é conseguida com o uso de *threads* no módulo de comunicação – uma para cada *pipe*. Além dos *pipes* de entrada e saída, a PADI possui outro conectado à saída de erro do servidor Fiddle. Sempre que ocorrer um erro de comunicação, este será capturado por este *pipe* e o tratamento adequado será dado.

6.4.3 Integração com outras ferramentas

O módulo de comunicação do protótipo foi desenvolvido para trabalhar com o Fiddle, mas é praticamente independente dos demais (interação e seleção). Existe um protocolo de comunicação entre os módulos que permite que o de comunicação seja reescrito sem que os demais precisem ser alterados. Apenas no caso de alguma alteração na demonstração de resultados é que uma alteração no módulo de interação seria necessária.

O protocolo de comunicação entre os módulos de seleção e o de comunicação é baseado numa classe denominada *TkOut* (nome baseado numa estrutura do Fiddle), cujo conteúdo são informações referentes ao comando e ao processo a ser executado. Uma classe *Action*, sempre que identifica um processo a ser afetado por um determinado comando, preenche as informações sobre este comando num objeto *TkOut* e o envia ao módulo de comunicação para que seja montada a mensagem à ferramenta de depuração utilizada pela PADI.

Assim, um novo módulo de comunicação poderia ser escrito para uma outra ferramenta de depuração tendo como base as informações contidas nesta classe. Estas informações são acessadas através de métodos do tipo *get()*.

Já a demonstração de novos resultados na ferramenta implicaria num trabalho um pouco maior. As informações sobre os estados dos processos, assim como os métodos de atualização da interface estão contidas na classe *Proc*. Assim, essa classe deveria ser alterada para conter as novas funcionalidades. Apesar de se tratar de alteração de código, o trabalho não seria muito complexo. Se fosse desejável, a integração da PADI com um outra ferramenta de baixo nível seria perfeitamente viável devido à sua estrutura modular e à sua implementação orientada a objetos.

6.5 Fluxo de execução

O fluxo de execução entre os objetos da PADI é resumidamente demonstrado no exemplo da figura 6.5. Os objetos e métodos foram simplificados, o objetivo é simplesmente mostrar a estrutura do algoritmo principal.

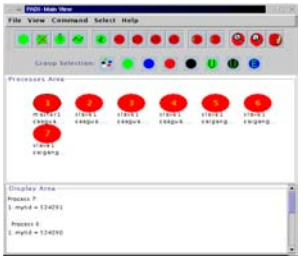

FIGURA 6.5 – Fluxo de execução de um comando *run* na PADI (pseudo-código)

O exemplo mostra um comando **run** sendo ativado na interface e o fluxo de processamento até sua chegada no módulo de comunicação com o Fiddle. Da mesma forma, o exemplo demonstra o retorno na forma de uma chegada do ou dos processos a um ponto de parada (*breakpoint hit*).

Primeiramente, os objetos que compõem o módulo de interação com o usuário, no exemplo a *Main View*, recebem o comando *run* originado de um evento num botão da interface. A seguir, um objeto do módulo de seleção do tipo *Action* é criado

par atender a essa requisição. No exemplo, o objeto criado é o *RunAction*. Este objeto realiza, então, todas as verificações de seleção e de estado relativas ao comando *run*.

Para cada processo que passa pelas duas verificações, um objeto do tipo *Send* é criado. Este objeto, que faz parte do módulo de comunicação do protótipo da PADI, monta a mensagem com todas as informações necessárias para que o Fiddle consiga enviar o comando com os parâmetros corretos para os processos afetados. No caso do exemplo, essas informações seriam o *tid* do processo e os argumentos do comando *run*. Assim que a mensagem é montada, é enviada via *pipe* ao Fiddle. No caso da utilização de Java JNI, as informações devem ser enviadas via parâmetros na chamada do método correspondente ao *run* do Fiddle.

No modelo atual, o *pipe* de recepção da PADI, que é conectado à saída do Fiddle, está ininterruptamente recebendo eventos através de uma *thread* específica que faz parte do módulo de comunicação. No exemplo, onde um comando *run* foi enviado, diferentes eventos assíncronos podem ocorrer, como a chegada a um ponto de parada ou ao final (término) do processo. A figura 6.5 também mostra o fluxo inverso de execução caso uma chegada a um ponto de parada ocorra.

Assim que a *thread* de recepção lê uma mensagem do *pipe*, ela imediatamente dá início ao seu processo de decodificação para saber que tipo de evento foi recebido. No caso, como o evento é a chegada a um *breakpoint*, um objeto apropriado é criado (*Break*) para tratar desse evento. Este objeto deve procurar a referência ao objeto do tipo *Proc* que representa o processo que originou o evento. Assim que essa referência é obtida junto ao servidor de processos através do seu *tid*, o método adequado, no caso o *break()*, é invocado em *Proc*. Esse método fará as atualizações de estado (no caso o processo passará de *running* para *stopped*) e de interface (mudança de cor e alteração na posição do código, caso a *Process View* desse processo esteja aberta).

6.6 Considerações finais

Do ponto de vista da arquitetura, o protótipo PADI pode ser dividida em três módulos: interação, seleção e comunicação. O módulo de interação implementa a interface com o usuário, o de seleção é responsável pelas rotinas de verificação de estado e grupo e o de comunicação implementa a integração com outras ferramentas, no caso do protótipo, com o ambiente Fiddle.

O módulo de interação foi projetado tendo em vista a facilidade de operação. Assim, praticamente todos os comandos de depuração podem ser acessados diretamente através de botões, sem a necessidade de se abrir menus. Esta configuração é bastante comum nos depuradores seqüenciais, mas não está sendo aplicada com freqüência nos projetos de depuradores paralelos.

Neste sentido, outro aspecto que facilita a interação com a ferramenta é a divisão clara dos níveis hierárquicos. Na PADI, o usuário sabe que todos os comandos que partem da *Main View* são distribuídos e os que partem de uma *Process View* são relativos ao processo dono da janela. Além disso, sabe-se que os comandos distribuídos

serão aplicados exclusivamente nos processos selecionados e que estão visíveis na janela de visualização da *Main View*.

Outro recurso visual que facilita a interação com a ferramenta é o *estado intermediário*, chamado *debugging*. Quando um comando de depuração é acionado em qualquer janela da interface, os processos alvo daquele comando são colocados no estado *debugging* até que o comando esteja completo. Este recurso trouxe vários benefícios ao projeto da PADI. Um exemplo, é a visualização da hierarquia do comando, ou seja, se apenas um processo entrar no estado *debugging*, sabe-se que o comando originou-se da *Process View* daquele processo, já se todos entrarem neste estado, sabe-se que originou-se da janela principal.

Outro exemplo de vantagem desta abordagem é o conhecimento, por parte do usuário, de que os processos neste estado ainda não estão aptos a receberem novos comandos, ou seja, estão esperando resultados de uma requisição anterior. Isto é importante na medida em que após a realização de cada comando o estado da aplicação se altera e comandos enviados antes desta alteração podem perder o sentido.

O módulo de seleção é composto basicamente por duas classes principais: classes do tipo *Action* e classe *Proc*. Existe praticamente uma classe do tipo *Action* para cada comando de depuração. Essas classes são responsáveis pelas verificações de seleção relativas ao comando e à cada processo do grupo corrente. São essas classes que verificam se os processos do grupo corrente (que são os selecionados) também fazem parte do grupo afetado, isto é, se estão num estado válido para receberem determinado comando. Isto, feito no nível da PADI, funciona como uma espécie de otimização, já que evita que um comando seja enviado remotamente sem necessidade, pois a verificação feita pela PADI é local.

A classe *Proc* é uma das mais importantes, pois representa cada processo sob o controle do depurador. É tanto uma representação visual, já que é o próprio objeto que é mostrado na janela de visualização, quanto de implementação, pois contém todas as informações sobre o processo assim como os métodos que o manipulam. A vantagem desta abordagem é que esta classe contém praticamente todas as informações e código necessário para a manipulação dos processos, o que facilita sua manutenção e alteração, quando necessário.

O módulo de comunicação é responsável pela integração da PADI com outras ferramentas. Para o desenvolvimento do protótipo foi escolhida a ferramenta Fiddle para realizar as operações de depuração de mais baixo nível para a PADI. A vantagem da integração com o Fiddle é, naturalmente, o aproveitamento de um trabalho cujas funcionalidades são indispensáveis ao funcionamento da PADI. Desde o princípio, o principal objetivo com o projeto da PADI foi o estudo de interfaces de depuração, não havendo interesse, portanto, no desenvolvimento de toda uma infra-estrutura de depuração de baixo nível. Assim, o Fiddle se encaixou perfeitamente no projeto.

Apesar disso, como mencionado no decorrer deste capítulo, o módulo de comunicação pode ser reescrito para que a PADI seja utilizada com outras ferramentas que venham a surgir. Além disso, também há a possibilidade de se altera-lo para que possa ler eventos de uma ferramenta de monitoração *on-line*, por exemplo, como será abordado mais adiante neste trabalho.

7 Teste e validação da ferramenta PADI

Com o objetivo de testar e validar o trabalho, foi desenvolvido um protótipo para a ferramenta PADI. Este protótipo possui todas as funcionalidades descritas nas seções anteriores e abre espaço para a implementação de outras, propostas como trabalhos futuros mais adiante neste texto. Para validar o trabalho até aqui realizado, este capítulo descreve como se dá a interação com o usuário através da depuração de um programa exemplo simples. Além disso, realiza uma comparação entre a depuração no ambiente PADI e no ambiente TotalView e apresenta alguns aspectos que deveriam ser considerados no desenvolvimento de um modelo para interfaces de depuração paralela.

7.1 Interação com o usuário

A interface básica é composta das janelas *Main View* e *Process View* (resumidas na figura 5.1). A *Main View* corresponde ao nível de coordenação e oferece os mecanismos básicos de seleção e visualização. As janelas apresentadas demonstram o *layout* básico, porém não necessariamente definitivo da ferramenta.

A figura 7.1 demonstra um estado de execução de um programa com sete processos PVM sendo executados (o exemplo é o programa *master/slave* que acompanha sua distribuição). O processo *master* foi carregado e executado até a criação dos *slaves*. No momento, ele está no estado *stopped*, alcançado quando chegou num *breakpoint* previamente estabelecido. Os processos do tipo *slave* foram trazidos para o controle do depurador através do comando *attach*. Os números dentro dos ícones de cada processo são seus identificadores no ambiente PADI. Abaixo de cada ícone tem-se o nome do executável e o nome da máquina na qual o processo está sendo executado.

O comando *attach* funciona de duas maneiras: processo a processo ou através do nome do executável de um grupo de processos. No primeiro método, o usuário é apresentado a uma lista de processos onde é possível selecionar um deles para que seja trazido para o depurador. No segundo método, é possível fornecer ao depurador um nome de executável que será utilizado para uma busca realizada em todas as máquinas listadas num arquivo chamado *padi_hosts*. Durante esta busca, sempre que um processo com este nome de executável é encontrado, o mesmo é trazido para o comando do depurador.

A figura 7.1 apresenta todos os processos da aplicação na área de visualização, pois a seleção de processos está em *all* (todos os processos visíveis). Por este mesmo motivo, se algum comando distribuído for acionado ele terá efeito sobre todos os processos selecionados que estiverem aptos a receber este comando, ou seja, que estejam num estado de execução aceitável. A figura 7.2 demonstra o diagrama de transição de estados do Fiddle [CUN 98], que se adapta bem à ferramenta PADI.

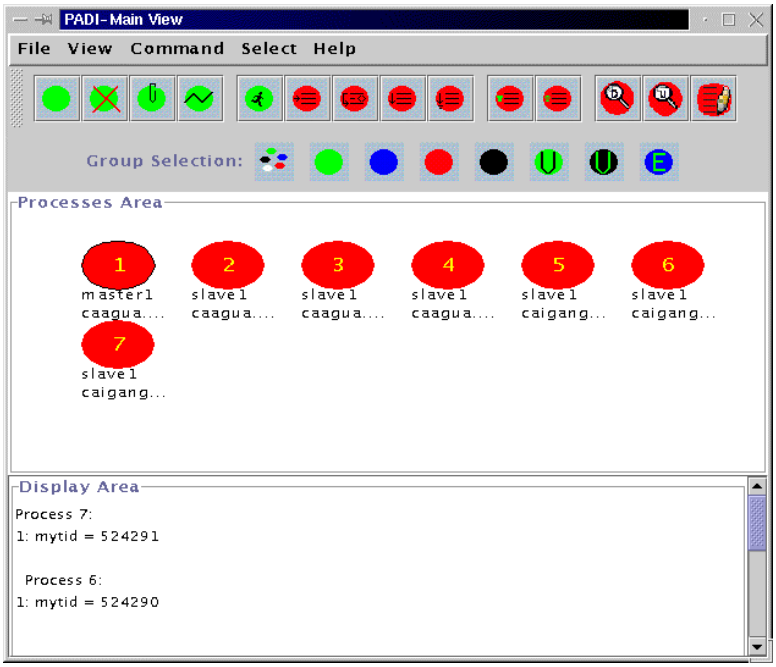

FIGURA 7.1 – *Main View* do protótipo PADI

Para se depurar uma aplicação paralela, a primeira coisa a fazer é carregar seu executável principal (**load**) e/ou utilizar o comando **attach** para buscar processos que já tiveram sua execução iniciada. Uma vez que o programa foi carregado a partir de um *load*, ele estará no estado *ready*, pronto para iniciar a execução através de um comando **run**. É interessante utilizar um comando de **set breakpoint** antes do comando *run*, a fim de que a execução pare (estado *stopped*) num ponto onde se deseja realizar uma inspeção. Entretanto, se o comando utilizado para buscar o(s) processo(s) foi o *attach*, este(s) estará (estarão) no estado *stopped*, esperando um comando de depuração. Neste caso, deve-se utilizar o comando **continue** para continuar sua execução.

Se um comando **set breakpoint** da *Main View* da PADI (correspondente ao *break* da figura 7.2) for acionado e a linha escolhida para fixar o *breakpoint* for válida em todos os processos, todos fixarão o *breakpoint* naquela linha. Entretanto, se o programa for composto por códigos diferentes, isto pode não ser desejável. No exemplo, existe uma instância do processo *master* e seis do processo *slave*, com códigos fonte diferentes para *master* e *slave*. Neste caso, o ideal é separar a depuração em dois grupos: um composto pelo processo *master* e outro composto pelos processos *slave*.

Vale lembrar que a PADI oferece três formas básicas de seleção de grupos: uma baseada no estado do processo, outra baseada na seleção feita pelo usuário e outra no nome do executável. No momento da captura da imagem da figura 7.1 todos os processos se encontram no mesmo estado (*stopped*), o que impossibilita a utilização do método de seleção por estado.

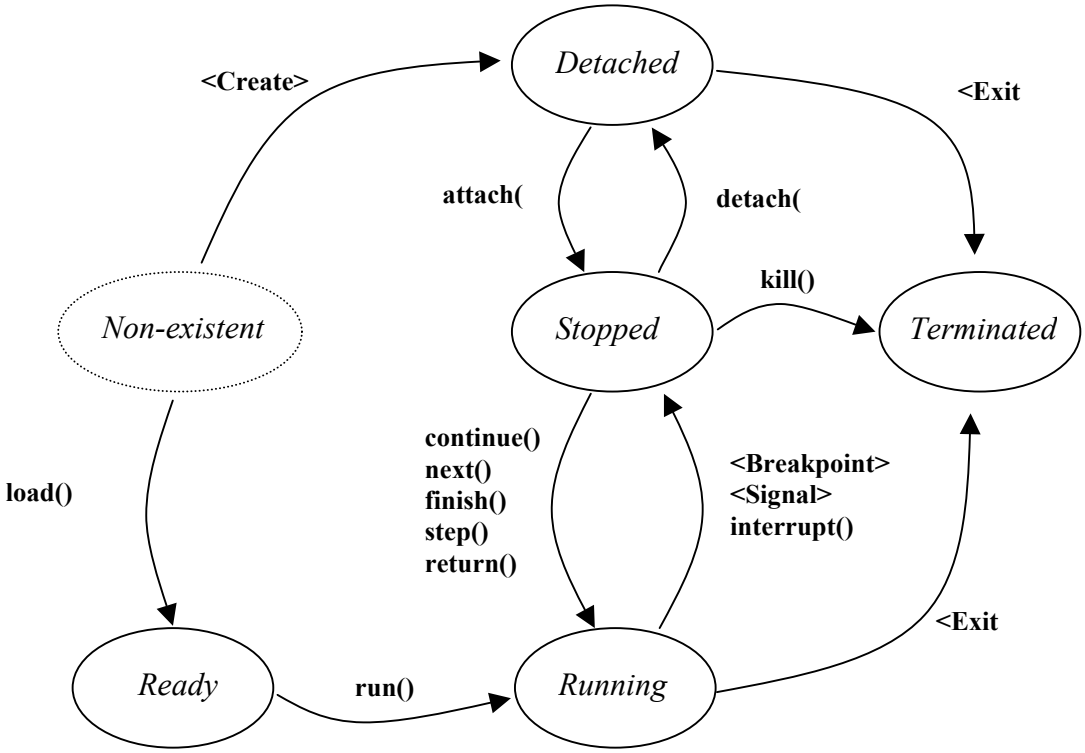

FIGURA 7.2 – Diagrama de transição de estados PADI/Fiddle

Assim, para que seja possível separar a depuração do *master* da depuração dos *slaves*, pode-se utilizar a seleção via grupo do usuário ou via executável. Na primeira, basta selecionar o processo *master* como sendo do usuário (operação feita com o *mouse*) e selecionar o grupo correspondente na interface (*Group Selection* igual a U). O segundo método consiste em entrar com o nome do executável e selecionar o grupo executável (*Group Selection* igual a E).

No exemplo, como se deseja um tipo de depuração *dual*, a alternativa mais adequada é a de grupo de usuário, pois conta com o mecanismo de inversão. Assim, basta selecionar o processo *master* como sendo de usuário que se terá, em contrapartida, os processos *slave* como não-usuários. Na figura 7.1 isso já foi feito, pois processo *master* possui uma borda realçada, o que significa que faz parte do grupo de usuário. A seleção, a partir daí, se dá através da escolha, em *Group Selection*, de U (usuário - *master*) ou U (inversão - *slaves*).

É importante salientar que na seleção de processos feita pelo usuário o grupo definido pelo usuário é fixo, até que ele mesmo faça uma alteração em sua configuração. Assim, a qualquer momento pode-se escolher um outro grupo, baseado no estado dos processos, por exemplo, e retornar ao grupo de usuário previamente configurado simplesmente selecionando-se o grupo correspondente na interface (em *Group Selection* na *Main View*). Os estados dos processos não influem na seleção de usuário.

Já os grupos pré-definidos baseados nos estados são dinâmicos, ou seja, quando um processo tem seu estado alterado, ele deve passar de um grupo a outro. Por exemplo, se um processo no estado *ready* é posto em execução, ele imediatamente sai do grupo *ready* e entra no grupo *running*. Entretanto, para facilitar o acompanhamento por parte do usuário, esta mudança só se refletirá na janela de visualização se o usuário refizer a seleção (o que significaria uma ação de *refresh*). Em outras palavras, um processo selecionado por estado não perde a seleção automaticamente por mudar de estado – ele só perde se o usuário refizer a seleção.

Como já mencionado, a PADI possibilita também o controle individual de cada processo através da *Process View*. Assim, é possível abrir uma janela individual para cada processo e aplicar comandos de depuração. A figura 7.3 apresenta esta janela para o processo *master*. Ainda que a seleção esteja em *all*, como na figura 7.1, todos os comandos que partirem desta janela serão válidos somente para o processo *master*. A figura 7.3 apresenta também uma linha com o *breakpoint* identificado pelo número 1 e a linha em execução.

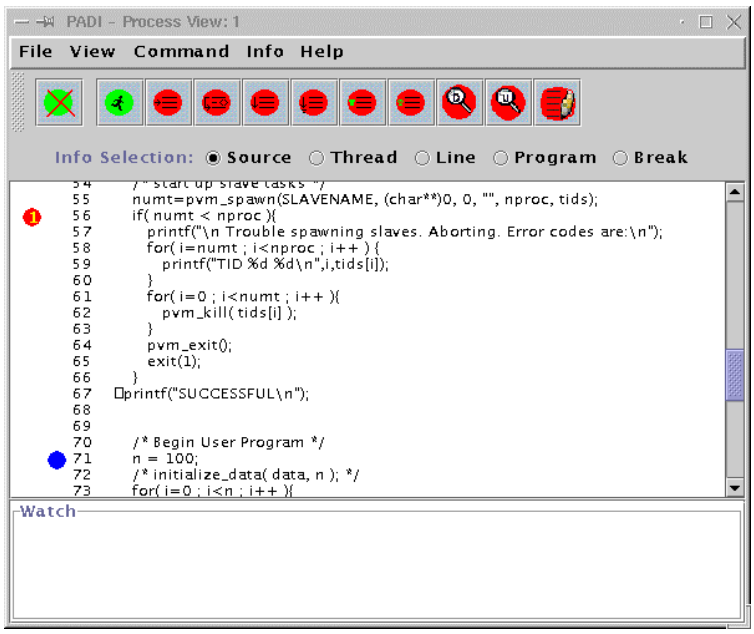

FIGURA 7.3 – *Process View* do protótipo PADI

Como exemplo de uso para o mecanismo de seleção pode ser usada a aplicação de um *breakpoint* somente nos processos *slave* selecionados. Neste caso, teríamos a área de visualização (*Processes Area*) apenas com os processos selecionados sendo visualizados. O comando **set breakpoint** seria acionado e uma linha de código poderia ser escolhida para a fixação do *breakpoint*. Com apenas um comando, o *breakpoint* seria fixado em todos processos selecionados. Assim sendo, este comando não seria aplicado ao processo *master*.

Além do comando *break*, os outros comandos distribuídos que fossem acionados com esta seleção não teriam efeito sobre o processo *master*, o único que ficaria de fora. Por exemplo, os processos *slave* poderiam ser executados passo-a-passo até o momento em que retornassem os dados ao *master*. O mecanismo de seleção evita que comandos indesejáveis, como o próprio *break*, sejam enviados ao *master* enquanto o que se deseja é acompanhar a execução dos *slaves*.

A visualização aliada ao mecanismo de seleção ajuda visualmente nesta focalização. Há uma espécie de filtragem e apenas os processos selecionados são demonstrados na área de visualização. Isto mostra a possibilidade de redução da quantidade de processos a ser visualizada. Entretanto, esta não é a principal função da seleção e a redução do número de processos visualizados depende bastante da estrutura da aplicação e da manipulação deste mecanismo por parte do usuário.

7.2 Comparação entre interfaces

O desenvolvimento do protótipo da PADI serviu para possibilitar o teste das principais funcionalidades definidas nas decisões de projeto, como abordado na seção anterior. Para fins de comparação com outras ferramentas é necessário definir o escopo de comparação, que contenha as principais características e os principais objetivos da PADI, que estão presentes no protótipo. Assim, do ponto de vista prático, uma comparação pode ser feita considerando-se os seguintes aspectos: intuitividade e facilidade de uso, modelos de programação aceitos e plataforma de execução.

Neste sentido, os objetivos da PADI são o de ser uma ferramenta intuitiva e de fácil utilização, disponível para uso com as mais populares bibliotecas de programação paralela (PVM e MPI) em *clusters* Linux. Além disso, as experiências realizadas com o protótipo mostram que é possível desenvolver um modelo para depuradores paralelos a partir de sua estrutura. Os aspectos que levam a essa conclusão são abordados na próxima seção.

Algumas considerações sobre outras ferramentas já foram feitas no capítulo 2. O objetivo desta seção é realizar uma comparação direta e prática entre a PADI e outra ferramenta. Entre as mais conhecidas, já descritas no texto, destacam-se a p2d2, a Node Prism e a Totalview. Estas, são depuradores paralelos simbólicos e *on-line*, como a PADI. Entretanto, do ponto de vista prático, as duas primeiras possuem alguns empecilhos. A p2d2 não está disponível para *download* e seu uso é restrito à NASA (ver o site do p2d2 em [HOO 99]). A Node Prism foi originalmente concebida pela *Thinking Machine Corporation* e atualmente possui uma versão pertencente ao ambiente *Sun HPC* [SUN 01] e não está disponível para *clusters* Linux. Ambas aceitam as bibliotecas PVM e MPI.

Já a TotalView, apesar de ser uma ferramenta comercial, possui uma distribuição de avaliação para *clusters* Linux e outros ambientes (disponível em [ETN 01]). Como é uma ferramenta comercial altamente utilizada, a TotalView possui um grande conjunto de funcionalidades implementado. Entretanto, uma comparação pode ser realizada no nível de manipulação de processos e *intuitividade* da interface, a fim de validação do protótipo da ferramenta PADI.

O mesmo exemplo descrito na seção anterior pode ser utilizado para uma comparação prática entre as duas ferramentas. Como descrito anteriormente, a aplicação possui uma divisão lógica entre dois tipos de componentes: *master* e *slaves*. A PADI oferece diferentes maneiras de se depurar estes dois componentes como dois grupos diferentes, a fim de aplicar diferentes comandos de depuração para cada um. Como exemplo, foi utilizado o método de seleção de usuário, onde o processo *master* foi marcado como sendo de usuário e utilizou-se os seletores U (*User*) e U (*Not user*) de *Group Selection* para possibilitar a alternação entre eles. Dessa forma, é possível aplicar-se comandos distribuídos a partir da *Main View* somente aos processos pertencentes ao grupo selecionado.

Já a noção de grupo do TotalView é menos flexível. Existem apenas dois tipos de grupos: o grupo de controle e o grupo compartilhado. O grupo de controle inclui o processo pai e todos os processos a ele relacionados. O grupo compartilhado é o conjunto de processos dentro de um grupo de controle que compartilha o mesmo código fonte.

Ao se depurar o mesmo exemplo (*master*/*slave*) com o TotalView é possível experimentar o efeito de um grupo de controle através da depuração passo-a-passo. Na verdade o próprio comando **step** é subdividido em *step* simples e de grupo. Se a opção *Step group* do menu *pop-up* for escolhida, todos os processos realizarão este *step*. Isto porque este *step* atua sobre o grupo de controle, que inclui o pai e todos os relacionados. Como o processo *master* é o criador dos *slaves*, todos fazem parte do grupo de controle. Assim, diferentemente da PADI, o *step* no TotalView só pode ser realizado individualmente ou por todos os processos neste exemplo. Assim sendo, não há como depurar somente os *slaves*, como foi feito na PADI.

A semântica padrão de manipulação de *breakpoints* também é diferente no TotalView. Quando um *breakpoint* é definido para um processo no TotalView, ele é compartilhado entre todos os processos com o mesmo código fonte. Por padrão, os *breakpoints* seguem o modelo de grupo compartilhado e não de grupo de controle como o comando *step*. Assim, os *breakpoints* serão compartilhados pelos processos *slave*, o que neste caso é perfeitamente adequado.

Já a PADI, através dos mecanismos de seleção, permite que isso seja facilmente definido pelo usuário de forma mais flexível. A mesma operação já descrita para o comando *step* (definição de grupo de usuário) pode ser utilizada aqui para definir *breakpoints* ou no *master* ou nos *slaves*, de forma absolutamente independente.

Apesar de o compartilhamento de *breakpoints* entre processos de mesmo código ser adequado a um bom número de aplicações, ele pode ser indesejado para outras. Como exemplo, pode-se citar uma aplicação do tipo SPMD (*Single Program Multiple Data*), onde todos os processos possuem o mesmo código fonte, mas, de acordo com sua tarefa, podem estar executando trechos diferentes do código. Ainda que definir um mesmo *breakpoint* para todos eles não necessariamente incorra em erro, algumas dessas operações podem ser desnecessárias e diminuir a eficiência da ferramenta, assim como aumentar seu nível de intrusão.

Além dos aspectos de seleção, a visualização também é um diferencial entre as duas ferramentas. A PADI possui visualização gráfica de processos, enquanto a

TotalView possibilita apenas a visualização de dados. Os processos na TotalView são apresentados de forma textual, através de uma lista pertencente à sua janela principal (*Root window*) que contém todos os processos da aplicação, estejam ou não sob o comando do depurador. Esta janela principal serve apenas para dar acesso aos processos da aplicação, nenhum comando específico de depuração parte dela.

A PADI possui uma abordagem estruturada com seus dois tipos de janelas, onde a principal (*Main* View) centraliza todos os comandos de depuração distribuídos. Esta estrutura torna mais clara a semântica do depurador. Através da visualização, é possível saber quais processos serão afetados por um comando distribuído antes mesmo que este seja efetivamente enviado aos depuradores de processo. Toda esta informação é centralizada na janela principal, enquanto que no TotalView isto não acontece. Além disso, os comandos no TotalView são acessados por listas de opções por vezes extensas em menus do tipo *pop-up*. Na PADI, os principais comandos estão presentes diretamente na própria interface. Todas essas opções de projeto colaboram para que a PADI seja uma ferramenta intuitiva.

7.3 Modelo para interfaces de depuradores paralelos

Observando-se o conjunto de ferramentas de depuração apresentadas no capítulo 4, nota-se que existe uma grande diversidade de interfaces e modos de interação com o usuário. Apesar de a diversidade oferecer um bom campo de análise, a grande variedade de interfaces pode ser considerada uma barreira ao uso de tais ferramentas, já que demanda um tempo de aprendizado indesejável a cada vez que o usuário se depara com uma nova.

A idéia de se desenvolver padrões vem justamente atender a esse problema de sobrecarga de diversificação. Neste sentido, o próprio padrão HPD, já brevemente descrito no capítulo 3, foi um esforço no sentido de se criar um padrão para ferramentas de depuração paralela. Entretanto, a definição do padrão ficou apenas na estrutura e interface textual dos depuradores paralelos. Nada foi definido com relação às interfaces gráficas.

Através das análise realizadas e do desenvolvimento do projeto e do protótipo PADI, é possível chegar a algumas conclusões que levem à proposta de um modelo para interfaces de depuradores paralelos. Um modelo baseado na PADI seria coerente com o padrão HPD para interfaces textuais, já que o próprio projeto da ferramenta considerou os princípios estabelecidos no padrão. A idéia desta seção não é a de descrever um padrão completo, mas extrapolar a PADI e apontar suas contribuições neste sentido.

A grande maioria dos princípios e características da PADI já foram mencionados ao longo do texto. A partir deles, é possível formular uma lista de aspectos que podem ser utilizados na formulação de um modelo ou padrão. Assim, um padrão baseado no projeto da PADI deve conter os seguintes aspectos:

1. **Níveis de abstração**: um depurador paralelo deve estar estruturado em dois níveis hierárquicos de abstração, os quais devem estar evidenciados na interface de depuração.

- Nível de coordenação: deve conter todos os comandos distribuídos permitidos pelo depurador. Estes comandos devem estar visivelmente disponíveis no elemento da interface que representa este nível.

- Nível de processos/*threads*: responsável pela depuração simbólica de cada processo e/ou conjunto de *threads* da aplicação paralela. Possui interface semelhante à dos depuradores tradicionais (seqüenciais).

2. **Seleção de processos**: um depurador paralelo deve oferecer mecanismos de seleção de processos. Um mecanismo básico, baseado na seleção de usuário deve ser oferecido e deve estar contido de forma visível na unidade da interface que implementa o nível de coordenação. Outros mecanismos de seleção mais sofisticados podem ser implementados (recomendado). É interessante que se tenha um tipo de seleção básica presente em todos os depuradores que seguirem o padrão. A idéia é oferecer um modelo básico para que o usuário esteja familiarizado com pelo menos um tipo de seleção sempre que se deparar com uma nova ferramenta de depuração. A seleção de usuário é a mais simples e flexível, por isso foi escolhida aqui para ser o mecanismo básico de seleção de processos.

3. **Grupos pré-definidos**: uma característica desejável é a existência de grupos de processos pré-definidos. Eles podem ser baseados nos estados dos processos ou serem automáticos (por exemplo, processos filhos de um mesmo pai).

4. **Visualização de processos**: um depurador paralelo deve oferecer um meio para que o usuário possa visualizar todos os processos sob o controle do depurador, assim como seus estados. Esta visualização deve ser preferencialmente gráfica, embora a visualização textual também seja aceita. O importante é que essa visualização seja demonstrada na unidade da interface que implementa o nível de coordenação e que todos os processos possam ser facilmente identificados. O acesso aos processos deve ser realizado através de sua representação.

5. **Filtragem na visualização**: um depurador paralelo deve oferecer meios para que se possa reduzir o número de processos na área de visualização.

Apesar de algumas dessas características estarem presentes já em algumas ferramentas existentes, pode-se dizer que a PADI as apresentou e as organizou de forma coerente, com o objetivo de simplificar e tornar o ambiente de depuração paralela mais amigável e intuitivo. Pode-se dizer que esta é uma das maiores contribuições da PADI.

A tabela 7.1 relaciona as ferramentas *on-line* apresentadas no capítulo 4 e as características acima descritas. Pode-se perceber que apenas a PADI reúne todos os aspectos apresentados. Salienta-se que a relação foi realizada levando-se em consideração os aspectos acima descritos **como foram descritos**.

TABELA 7.1 – Depuradores paralelos *on-line* x modelo da PADI

Ferramentas	Níveis de abstração	Seleção de processos	Grupos pré-definidos	Visualização de processos	Filtragem
PADI	Sim	Sim	Sim	Sim	Sim
P2d2	Não	Sim	Sim	Sim	Não
Totalview	Não	Não	Sim	Sim	Não
Detop	Não	Sim	Sim	Sim	Não
Node Prism	Não	Sim	Não	Não	Não
Panorama	Não	Não	Não	Sim	Não
DCDB	Não	Não	Sim	Não	Não
Netdbx	Sim	Sim	Não	Sim	Não

7.4 Considerações finais

Ao longo deste capítulo, as principais características da PADI foram apresentadas e por vezes confrontadas com características semelhantes de outras ferramentas. Nota-se uma grande diversidade de abordagem para cada aspecto presente nas mais diversas ferramentas. Isto é um fato natural, se for considerado que a maioria delas foi desenvolvida como trabalho de pesquisa e poucas vezes seu uso se tornou extensivamente comercial. Para que isso seja possível, é necessário que se estabeleça uma padronização básica para a interface dos depuradores paralelos.

Pode-se citar, por exemplo, o caso dos depuradores seqüenciais. A interface de tais depuradores é semelhante, independentemente do ambiente e do fabricante. Alguns possuem mais funcionalidades e são mais poderosos do que outros, mas a essência é a mesma. Um programador que mude de ambiente saberá utilizar um e outro de forma intuitiva devido à semelhança entre eles.

O projeto PADI foi desenvolvido com o objetivo de oferecer a mesma *intuitividade* e facilidade de uso. Assim, foram considerados aspectos funcionais de presença obrigatória em depuradores paralelos e sua organização na interface foi realizada de forma que se possa estabelecer um modelo flexível para depuradores paralelos.

Testes preliminares com o protótipo da PADI demonstraram a facilidade de uso da ferramenta e de seus mecanismos de seleção e visualização. A separação em níveis hierárquicos destaca-se como um dos fatores que mais contribuem para que a ferramenta seja intuitiva e fácil de usar. Apesar de ser uma solução simples, poucas ferramentas possuem essa estrutura de forma clara, como demonstrou a tabela 7.1.

Outro mecanismo de destaque é a união entre a seleção de processos e a visualização gráfica. Esta união resultou num mecanismo de filtragem da visualização, cuja vantagem é reduzir o número de processos visualizados e auxiliar na focalização de um grupo específico. Apesar de a filtragem ser comprovadamente um mecanismo de auxílio na visualização de processos, como já demonstrado em ferramentas de análise de desempenho, poucas ferramentas de depuração adotam mecanismos desse tipo, como também demonstrou a tabela 7.1.

8 Contribuições a curto e médio prazo

Além das contribuições já apresentadas, podemos acrescentar aquelas possibilitadas pelo trabalho já realizado, ou seja, aquelas que poderão vir a ser implementadas a partir do protótipo desenvolvido. Essas contribuições foram divididas em duas categorias: médio prazo e longo prazo.

Este capítulo descreve os trabalhos futuros com um nível de detalhamento suficiente para que possam ser realizados por demais pesquisadores da área que sejam interessados.

8.1 Trabalhos futuros – curto prazo

Esta seção propõe alguns trabalhos que podem ser realizados para expandir o uso do protótipo PADI e concretizar uma versão totalmente operacional. Entre estes trabalhos estão alterações no módulo de comunicação para que o protótipo utilize a versão Java do Fiddle, testes com a utilização da PADI para a depuração de aplicações paralelas reais (em fase de desenvolvimento) e ajustes para a efetiva utilização da PADI para depurar aplicações desenvolvidas com o DECK.

8.1.1 Alteração no módulo de comunicação

Como descrito no capítulo 6, que aborda a implementação do protótipo, atualmente a PADI conta com o ambiente Fiddle como infraestrutura básica de depuração. Devido a uma incompatibilidade de linguagem (PADI é escrita em Java e Fiddle é uma biblioteca C), foi desenvolvido um servidor para o Fiddle que possibilitou a comunicação via *pipe*, através de cadeias de caracteres formatadas.

Esta solução foi a mais satisfatória na época em que a cooperação iniciou, mas ao mesmo tempo verificou-se que seria possível a utilização da biblioteca de métodos nativos do Java (JNI – *Java Native Interface*) para realizar esta comunicação. Assim, os desenvolvedores do Fiddle passaram a desenvolver sua versão Java, o Fiddle-j.

Uma versão inicial do Fiddle-j já se encontra disponível para testes. A eliminação do *pipe* tornaria o código da PADI mais "limpo", pois evitaria o tratamento de *strings* de entrada e saída presente na versão atual. Além disso, a própria manutenção da PADI seria simplificada, pois atualmente é necessário que se conheça os formatos de *string* para cada comando enviado/recebido do Fiddle. Caso a versão Fiddle-j fosse utilizada, seria necessário apenas conhecer os métodos disponíveis e seus argumentos.

Para realizar esta tarefa, apenas o módulo de comunicação deve ser reescrito, ou seja, os módulos de interação e seleção, cujas funções foram descritas no capítulo 6, não devem sofrer alterações.

8.1.2 Depuração de aplicações em fase de desenvolvimento

O protótipo da PADI teve como principal objetivo o de validar o projeto e arquitetura da interface idealizada para um depurador paralelo. A partir do protótipo foi possível realizar comparações desta interface com outras existentes. Como visto no capítulo 5, a PADI reúne em seu projeto características que tornam a estrutura de sua interface ao mesmo tempo intuitiva e com recursos capazes de facilitar a depuração de aplicações paralelas. A partir desta estrutura, foram apresentadas características passíveis de serem aplicadas a um modelo de interface para depuradores paralelos.

Os recursos da PADI já foram testados através da depuração de programas paralelos prontos e se mostraram satisfatórios. A curto prazo, entretanto, a PADI deve ser apresentada a usuários que desenvolvam aplicações paralelas reais. O objetivo é observar o comportamento dos usuários perante a interface e realizar a validação da ferramenta através de sua utilização prática real. Além disso, as contribuições dos usuários poderão ser absorvidas e incluídas nas próximas versões da ferramenta.

8.1.3 Utilização com diferentes ambientes de desenvolvimento

Atualmente, o protótipo da PADI é capaz de depurar programas paralelos desenvolvidos com as bibliotecas PVM e MPICH. A idéia é tornar a PADI compatível com o maior número de bibliotecas e versões possível. Num primeiro momento, testes e ajustes de compatibilidade com o ambiente DECK serão prioritários, visto que este é um ambiente desenvolvido no âmbito desta universidade.

Inicialmente, o PVM e o MPICH foram utilizados para testes devido ao grande número de aplicações existentes e sendo desenvolvidas com estas bibliotecas. Assim, optou-se para que o protótipo fosse desenvolvido tendo em vista esses ambientes. Entretanto, apenas pequenos ajustes serão necessários para uso com os demais ambientes, visto que a comunicação da PADI é essencialmente realizada com o ambiente de depuração de baixo nível, no caso o Fiddle, e não com a biblioteca utilizada.

Apesar disso, algumas informações de sistema são necessárias na inicialização e finalização de processos, principalmente nas rotinas de *load*, *attachment* e *kill*. Portanto, alguns testes e ajustes visando diferentes bibliotecas, tais como DECK, LAM-MPI, Athapascan e outras, devem ser realizados nesse sentido.

8.2 Trabalhos futuros – médio prazo

A partir do protótipo da interface, alguns recursos adicionais podem ser testados para inclusão no projeto final da ferramenta PADI. Estes recursos terão o objetivo principal de consolidar tanto o protótipo quanto o modelo resultante da PADI como extensíveis.

Apesar do objetivo de projetar uma ferramenta simples, estes mecanismos de extensão tornarão modelo e ferramenta mais adaptáveis a novas experiências e novos modelos de programação. A idéia é fazer com que a ferramenta seja um ponto de partida para o estudo de novos mecanismos que facilitem a depuração paralela.

Assim, ao mesmo tempo em que é uma ferramenta de depuração paralela, a PADI deixa o caminho aberto para novas experiências nesta área. O objetivo é deixá-la modular de tal forma que o pesquisador que deseje implementar mecanismos como estes não precise construir um depurador paralelo inteiramente novo.

8.2.1 Monitoração voltada à depuração de programas paralelos

A idéia aqui é possibilitar a inclusão de um módulo de comunicação para ferramentas de monitoração. O objetivo é permitir a conexão entre estes dois tipos de ferramentas possibilitando a visualização de outros tipos de eventos. Atualmente, a PADI só é capaz de demonstrar eventos gerados pelos depuradores de processo, como por exemplo a chegada de um ou mais deles num ponto de parada (*breakpoint*). Esse tipo de evento é sinalizado na interface através da troca de cor dos ícones de cada processo que recebe um evento deste tipo.

Assim como os eventos característicos de depuração, existem outros relacionados ao paralelismo que podem ser sinalizados na interface. Um bom exemplo são os eventos relacionados à comunicação e sincronização de processos. Além dos estados de execução atuais (*ready, running, stopped, terminated*), poderiam ser acrescentados novos estados relativos à comunicação, como por exemplo *sending* e *receiving*.

Para que isso fosse possível, na configuração atual do protótipo PADI, seria necessário utilizar um mecanismo de instrumentação do código fonte. No caso das ferramentas PVM e MPI estes mecanismos já existem embutidos nas bibliotecas de programação. No caso do ambiente DECK, um mecanismo deste tipo poderia ser desenvolvido. A pesquisa para incluir um módulo leitor de eventos de comunicação e gerenciamento de processos na PADI incluiria os seguintes passos básicos:

- Estudo dos mecanismos de instrumentação e geração de *traces* da(s) biblioteca(s) alvo. Cabe salientar que os eventos deverão ser capturados em tempo de execução, ou seja, *on-line*.

- Desenvolvimento de um módulo de leitura de eventos para a PADI, que deverá recebê-los, interpreta-los e ativar o método correspondente da interface.

- Incluir, na PADI, um método para cada tipo de novo evento e criar a animação correspondente na área de visualização.

Problemas relacionados à monitoração *on-line* devem ser levados em consideração, como a intrusão que tal mecanismo pode ocasionar no sistema. Uma idéia

para tentar minimizar a intrusão seria a criação de máscaras que permitissem a monitoração exclusivamente de eventos escolhidos pelo usuário. Da mesma forma, o tipo de informação gerada também poderia ser classificada em níveis, onde o nível mínimo seria aquele que ocasionaria a menor intrusão.

Outro problema a ser solucionado seria a identificação dos processos no protótipo, pois esse mecanismo monitoração funcionaria "por fora" do ambiente Fiddle, que possui seu próprio mecanismo de identificação de processos. Seria necessário realizar um mapeamento entre, por exemplo, os identificadores de processo no sistema operacional e os identificadores de processo do Fiddle (que são os mesmos utilizados na PADI).

Além disso, pode-se dizer que um leitor de eventos *on-line* seria o primeiro passo para a integração da PADI com uma ferramenta de depuração ou visualização *off-line*. A grosso modo, bastaria que o mecanismo de leitura de *traces* da ferramenta *off-line* fosse redirecionado para receber os eventos *on-line*. A coordenação e sincronização entre elas também seria um bom objeto de pesquisa.

8.2.2 Novos modelos de visualização

O trabalho teria como objetivo possibilitar diferentes disposições iniciais dos processos na área de visualização. Atualmente, os ícones dos processos são distribuídos de forma seqüencial, da esquerda para a direita, formando linhas e colunas. A idéia é oferecer novos modelos de visualização que possibilitem uma maior compreensão do comportamento da aplicação paralela. Além disso, estes novos tipos de visualização também poderiam ser idealizados visando a demonstração de novos eventos, como os de comunicação, descritos no tópico anterior.

A seguir, são descritos três exemplos de disposições de ícones de processos que poderiam ser estudadas para inclusão na PADI. Dois dos modelos apresentados a seguir poderiam utilizar o mesmo tipo de elipse do atual protótipo da PADI.

Disposição circular

A disposição circular seria adequada no caso de se implementar a monitoração *on-line* visando a captura de eventos de comunicação, descrita no tópico anterior. Este tipo de disposição foi utilizado como um dos tipos de visualização da ferramenta Paragraph [HEA 91], que é uma ferramenta *off-line* para depuração visando análise de desempenho de programas paralelos e que foi uma das pioneiras nesta área.

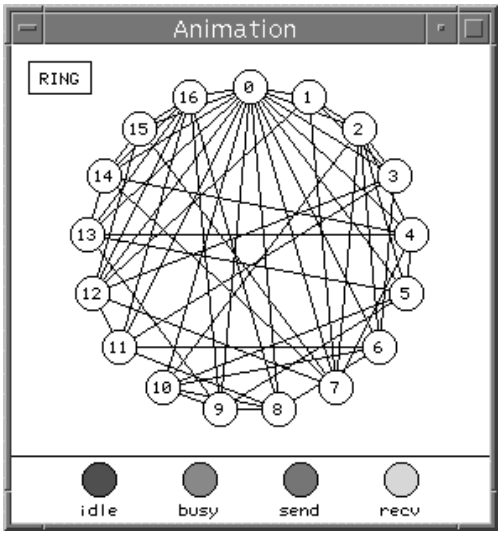

FIGURA 8.1 – Disposição circular de processos no Paragraph

Consiste em dispor os processos (elipses, no caso) num círculo e representar a comunicação entre estes processos através de linhas que ligam aqueles que estão se comunicando durante a animação. As cores das elipses, a cada momento, indicam os estados possíveis dos processos. No caso da PADI, mais dois tipos de estado poderiam ser incluídos: enviando (*sending*) e recebendo (*receiving*).

A figura 8.1 exemplifica este tipo de disposição através da janela *Animation/Ring* do Paragraph. O exemplo demonstra o círculo ou anel após o término da animação: as linhas representam que houve no mínimo uma comunicação entre os processos interligados.

Este tipo de representação, apesar de simples, seria adequado para demonstrar a comunicação entre os processos. Entretanto, este tipo de representação sofre um pouco no aspecto de escalabilidade, já que quanto maior o número de processos, mais difícil se torna a visualização de todos eles.

Apesar disso, o mecanismo de seleção da PADI poderia auxiliar a diminuir o número de processos na visualização. Por exemplo, ao invés de se criar dois novos estados (enviando e recebendo), poderia ser criado um de **comunicação**. Assim, bastaria selecionar este estado para se ter no círculo apenas os processos comunicantes, o que poderia (ou não) diminuir o número de processos no círculo. Ao invés de linhas, os processos seriam interligados por arcos dirigidos, que indicariam o emissor e o receptor.

Diagrama espaço-tempo

O diagrama espaço tempo é bastante utilizado em ferramentas de depuração *off-line*, principalmente naquelas voltadas à avaliação de desempenho de programas paralelos. O diagrama, ao longo do tempo, foi se tornando uma ferramenta cada vez mais sofisticada, como se pode verificar no ambiente MAD [KRA 97] e PAJÉ [STE 98].

Entretanto, o objetivo de tal representação na PADI seria principalmente o de demonstrar a comunicação entre os processos e os estados de cada um em cada ponto do programa. Neste tipo de diagrama, como é possível observar na figura, cada processo é representado por uma linha horizontal (tempo), que indica também o seu estado. A comunicação entre os processos é representada através de uma linha entre os processos comunicantes.

Este tipo de representação demandaria um pouco mais de trabalho na PADI em relação à descrita no tópico anterior, visto que o tipo de representação dos processos se alteraria completamente em relação ao tipo do protótipo atual: de elipse para uma linha que varia ao longo do tempo. As próprias informações sobre o tempo deveriam ser retiradas da ferramenta de monitoração *on-line*, o que representaria dados a mais a serem processados e controlados através de um mecanismo de relógios lógicos. Na verdade, este mecanismo já existe em algumas ferramentas de monitoração. Um estudo sobre a integração de uma dessas ferramentas ao ambiente poderia ser realizado a fim de verificação de viabilidade. Como exemplo de ferramenta de monitoração para programas PVM pode-se citar o Tape/PVM [MAI 95].

Grafo de processos

A idéia de que grafos são estruturas adequadas para a representação de programas paralelos baseados em processos e troca de mensagens não é absolutamente nova e seu principal uso até agora foi no desenvolvimento de linguagens visuais de programação paralela. Segundo Norman [NOR 93], existem duas classes de modelos computacionais que incorporam a comunicação entre processos de diferentes maneiras e podem ser representados através de grafos: os modelos baseados em tarefas e os baseados em processos. Os baseados em tarefas são geralmente utilizados por pesquisadores interessados em problemas de escalonamento e tendem a ter base em multiprocessadores. Já os baseados em processos são geralmente utilizados para mapeamento explícito de programas paralelos e tendem a ter base em sistemas distribuídos.

Nos modelos baseados em processos, que é o caso da PADI, o problema consiste em arranjar estes processos em grafos não dirigidos. Um arco, neste modelo, pode corresponder ao volume de comunicação entre os processos, além do próprio evento de comunicação associado.

Entretanto, a representação por meio de um grafo de uma aplicação que foi desenvolvida da forma tradicional (textual) não é nada trivial. Deve-se conhecer todos os nós, no caso os processos, da aplicação, assim como o caminho ou padrão de comunicação entre eles. Feito isso, deve-se desenhar este grafo e apresenta-lo ao usuário. A vantagem desta representação está em oferecer uma idéia geral justamente do padrão de comunicação entre os processos e facilitar a visualização desta comunicação durante a execução da aplicação, visto que os processos que se comunicam estarão próximos uns dos outros na representação gráfica.

Silva [SIL 94] apresenta um trabalho de comparação e implementação de desenho de diferentes tipos de grafos genéricos, através de diversos algoritmos encontrados na literatura para tal. Entre os algoritmos para desenho de grafos não-

orientados, foram estudados o *Simulated Annealing* e o *Random Search.* Segundo a autora, apesar de ambos os algoritmos gerarem figuras satisfatórias, não se mostraram adequados a ferramentas interativas. Esse problema decorre do fato de consumirem muito tempo na geração das figuras e necessitarem de um grande número de parâmetros, que devem ser iniciados a cada execução.

Justamente este problema motivou os autores do algoritmo *Force-directed Placement* [FRU 91], que desenvolveram uma ferramenta de geração de grafos (*Nature*) cuja entrada é simplesmente uma lista de adjacências. Esta ferramenta foi usada no trabalho de Stringhini [STR 97a, STR97b] para desenhar um grafo baseado numa aplicação paralela. A título de ilustração, o desenho resultante é mostrado na figura 8.3.

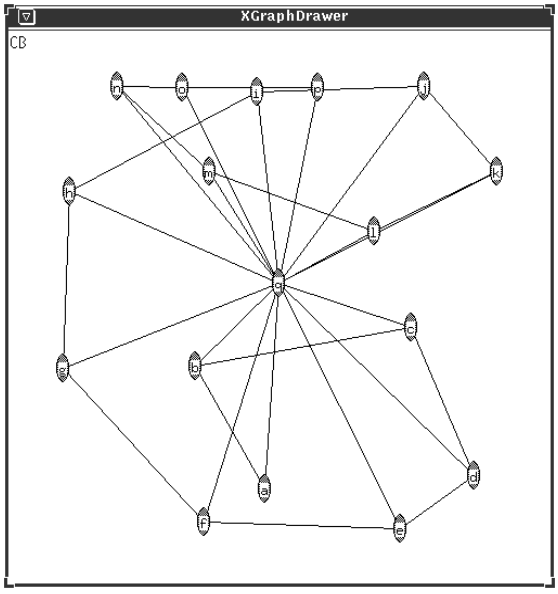

FIGURA 8.3 – Exemplo de grafo de processos

A inclusão de uma visualização deste tipo na PADI necessitaria que fossem realizados os seguintes passos básicos:

- monitoração *on-line*, com o objetivo de gravar um arquivo que contenha informações suficientes para a geração da lista de adjacências;

- módulo que implemente um algoritmo de desenho de grafos, como o de Früchtermann, mencionado acima.

Uma outra abordagem seria realizar o desenho baseado em topologias conhecidas de programas paralelos, tais como *pipeline*, anel, hipercubo, árvore, etc. Huband e McDonald [HUB 2001] descrevem uma metodologia de reconhecimento de

padrões de grafos, usada em seu depurador paralelo *off-line* chamado DEPICT, que se mostrou eficiente para alguns tipos de topologias.

8.2.3 Outros

Outros aspectos da depuração paralela ainda poderiam ser alvo de estudos mais aprofundados para implementação na PADI. O objetivo seria tornar a ferramenta mais poderosa, mas não necessariamente propor algo absolutamente novo. Entre os assuntos que se encaixam nesta categoria estão a depuração de *threads* e a implementação de algum algoritmo que permita a utilização de *breakpoints* condicionais.

Depuração de *threads*

A PADI originalmente iria dar uma maior atenção às *threads*, mas a pesquisa acabou sendo focada em técnicas de depuração de processos, mais especificamente em seleção e visualização de processos. Entretanto, a depuração de *threads* na PADI é possível de ser implementada e alguns testes neste sentido foram realizados.

O ambiente *Fiddle*, utilizado atualmente como infra-estrutura de depuração do protótipo PADI, oferece algumas primitivas em sua biblioteca para o tratamento de *threads*. Estas primitivas podem ser utilizadas pela PADI no sentido de oferecer uma interface confortável para a manipulação de *threads*. Por exemplo, um tratamento visual semelhante ao dado aos processos poderia ser aplicado às *threads* dos processos. A janela *Process View* da PADI, poderia ter uma área de visualização de *threads* e a manipulação dessas *threads* poderia ser realizada nessa mesma janela.

No protótipo da PADI já foram realizados testes no sentido de visualização de algumas informações textuais sobre *threads*. A janela da *Process View* que normalmente é utilizada para visualização do código fonte pode, ao invés disso, ser selecionada para mostrar informações sobre as *threads* do processo dono da janela em questão.

***Breakpoints* condicionais**

A definição de *breakpoints* condicionais trata de indicar pontos de parada no programa a partir de determinadas condições de estado dos processos ou da ocorrência de eventos específicos, como por exemplo a execução de uma instrução particular [LEU 92]. Diversos algoritmos já foram desenvolvidos e testados ([BAR 96] e [PIN 97]) e a proposta aqui seria a de implementar um algoritmo de detecção da condição de parada, assim como estudar a efetiva realização da "parada" de todos os processos. Atualmente, a parada dos processos na PADI se dá através da designação de uma mesma linha de execução ou do nome de uma função em comum.

8.3 Considerações finais

O projeto da ferramenta PADI não trouxe apenas o benefício da implementação de um protótipo para uma interface de depuração paralela e da proposta de um modelo. O estudo que envolveu seu desenvolvimento permitiu a realização de uma análise profunda de todo o trabalho que tem sido feito na área de interfaces de depuração paralela. Um dos grandes méritos do projeto PADI está em absorver características vantajosas existentes em outras ferramentas e aglutiná-las de forma organizada e facilmente utilizável: a chamada *intuitividade*, objetivo maior do projeto da interface PADI.

O projeto PADI resultou num protótipo que já pode ser utilizado na depuração de programas paralelos, assim como numa ferramenta aberta a novos experimentos. Um modelo básico para interfaces de depuração paralela também foi proposto e pode servir como base de padronização de tais interfaces, facilitando assim seu uso. Este capítulo descreveu uma série de trabalhos que podem ser realizados a partir do protótipo e podem ser considerados como contribuições do projeto. Além disso, até o momento o projeto obteve duas publicações ([STR 2000] e [STR 2001]).

9 Conclusão

O rápido crescimento do processamento paralelo, impulsionado pelas novas tecnologias que possibilitam a criação eficiente de *clusters* de estações de trabalho, salientou a falta de ferramentas e ambientes que auxiliem o desenvolvimento de aplicações paralelas. Entre esses ambientes, destacam-se os depuradores paralelos.

A depuração paralela ainda está se adaptando ao desafio do paralelismo. Num primeiro momento, notou-se que uma das principais características da depuração serial havia se perdido: o determinismo, que possibilita a depuração cíclica. Assim, técnicas de depuração *off-line* para permitir a re-execução dos programas foram desenvolvidas. Entretanto, percebeu-se que os mecanismos de monitoração necessários podiam ser intrusivos e causar o mal-funcionamento da aplicação. A pesquisa nesta área se dedicou então a aperfeiçoar estes mecanismos.

Por outro lado, a depuração cíclica pode ser preterida em nome de um controle completo sobre a depuração e acesso total ao programa em tempo de execução. A depuração *on-line* é baseada em técnicas tradicionais de depuração e permite este tipo de controle.

Os depuradores paralelos *on-line* possuem, basicamente, uma estrutura comum, embora nem sempre bem caracterizada: um nível de coordenação, que controla os processos distribuídos da aplicação, e um nível de processos, que corresponde ao mesmo tipo de controle disponibilizado por um depurador serial tradicional. Identificou-se o nível de coordenação como um importante objeto de pesquisa, já que este é que foi introduzido a partir do paralelismo. Este nível, apesar de presente na maioria dos depuradores existentes, ainda não foi suficientemente explorado.

O padrão HPD foi utilizado como uma das principais bases para o presente trabalho, no sentido em que define como este novo nível de abstração pode ser estruturado e organizado. O conceito de conjunto de processos/*threads* foi empregado aqui para definir o mecanismo de seleção de processos. Este mecanismo é o principal empregado no desenvolvimento do modelo proposto para interfaces de depuração, assim como no protótipo da ferramenta apresentada neste trabalho chamada PADI *(PArallel Debugger Interface*). Esta ferramenta, além de dar acesso a um ambiente de depuração paralela *on-line*, lhe acrescenta funcionalidades como a seleção de processos, os comandos distribuídos e a visualização.

Assim, a Tese de Doutorado aqui descrita consistiu no projeto e desenvolvimento de um modelo para uma interface gráfica completa para um ambiente de depuração simbólico *on-line*, tendo como principal objeto de pesquisa o nível de coordenação. A este nível, foram acrescentados os mecanismos de seleção de processos e visualização, cujos principais objetivos são o de facilitar a interação com a ferramenta e diminuir a quantidade de informação textual relativa à depuração.

No projeto da PADI foi realizada uma espécie de reorganização de características existentes em algumas ferramentas, que acabou por formular novos mecanismos e evidenciar tendências. Pode-se citar, por exemplo, a própria utilização

em conjunto dos mecanismos de seleção e visualização. Esta união resultou num mecanismo de filtragem para a visualização de processos que ainda não é comum em ferramentas de depuração paralela.

Um protótipo foi desenvolvido e demonstrou que os mecanismos de seleção e visualização são úteis na medida em que tornam este tipo de ferramenta mais poderosa, principalmente no que diz respeito ao controle da depuração distribuída. Ao mesmo tempo, a visualização torna a ferramenta mais fácil de usar, atingindo um dos principais objetivos, que é o de tornar a depuração uma tarefa quase intuitiva.

Testes realizados demonstraram a flexibilidade do modelo, que permite que determinados processos sejam selecionados para receberem comandos de depuração distribuídos. O mecanismo de seleção serve como um filtro para reduzir o número de processos na janela de visualização, característica desejável em aplicações com grande número de processos. A partir dos testes, foi possível definir as principais características, que além de tornarem a interface mais intuitiva, são genéricas o suficiente para descreverem um modelo sobre o qual podem se basear diferentes interfaces de depuração paralela. A vantagem de um modelo básico como este é o de oferecer um determinado grau de padronização das ferramentas, o que diminuiria o impacto inicial de dificuldade de utilização por parte dos usuários.

Outra tendência em ambientes de programação paralela que foi seguida pela PADI é a integração de ferramentas. Além da integração realizada com sucesso com o ambiente Fiddle, a PADI se propõe a ser uma ferramenta aberta a experiências como novos tipos de integração. Naturalmente que integrações desse tipo oferecem algumas dificuldades, pois são trabalhos realizados paralelamente. Entretanto, trata-se de um tipo de cooperação que pode ser muito vantajosa.

Bibliografia

[AND 93] ANDREWS, G.; OLSSON, Ronald A. **The SR Programming Language**. [S.l.]: Benjamin/Cummings, 1993.

[BAR 96] BARBOSA, Valmir C. **An Introduction to Distributed Algorithms.** [S. l.]: The MIT Press, 1996

[BAR 98] BARRETO, M.; NAVAUX, P.; RIVIÉRE, M. DECK: a new model for a distributed executive kernel integrating communication and multithreading for support of distributed object oriented application with fault tolerance support. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN, 4., 1998, Neuquén, AR. **Anales. . .** Neuquén: Universidad Nacional de Comahue, Facultad de Economia y Administración, Departamento de Informática y Estadística, 1998. v.2,p.623–637.

[BAR 2000] BARRETO, M. **DECK:** um ambiente para programação paralela em agregados de multiprocessadores. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[BRI 97] BRIAT, J. et al. Athapascan Runtime: efficiency for irregular problems. In: INTERNATIONAL EURO-PAR CONFERENCE ON PARALLEL PROCESSING, EURO-PAR, 3., 1997, Passan. **Parallel Processing**: proceedings. Berlin: Springer-Verlag, 1997. p. 591-600. (Lecture Notes in Computer Science, v. 1300).

[BUH 96] BUHR, P. et al. KDB: A Multi-threaded Debugger for Multi-threaded Applications. In: SYMPOSIUM ON PARALLEL AND DISTRIBUTED TOOLS, 1996. **Proceedings...** Philadelphia: [s.n.], 1996. p. 80-87.

[CAS 92] CASAVANT, T. L.; KOHL, J. A; PAPELIS, Y. E. Practical Use of Visualization for Parallel Systems. In: EUROPEAN WORKSHOP ON PARALLEL COMPUTERS (Invited Keynote Address Text), 1992. **Proceedings…** Barcelona: [s.n.], 1992.

[CLÉ 96a] CLÉMENÇON, C. et al. Annai Scalable Run-time Support for Interactive Debugging and Performance Analysis of Large Scale Parallel Programs. In: INTERNATIONAL EURO-PAR CONFERENCE ON PARALLEL PROCESSING, EURO-PAR, 1996, Lyon. **Parallel Processing:** proceedings. Berlin: Springer-Verlag, 1996. p.64-69. (Lecture Notes in Computing Science, v. 1123).

[CLÉ 96b] CLÉMENÇON, C. et al. **Annai Scalable Run-time Support for Interactive Debugging and Performance Analysis of Large Scale Parallel Programs.** Manno, Suíça: Centro Svizzero di Calcolo

Scientifico, 1996. (Technical Report n. CSCS-TR-96-04, CH-6928). Disponível em: <ftp://ftp.cscs.ch/pub/CSCS/ techreports/1996/TR-96-04.ps.gz>. Acesso em: abr. 1996.

[CUN 98] CUNHA, J. C.; LOURENÇO, J. et al . A Framework to Support Parallel and Distributed Debugging. In: INTERNATIONAL CONFERENCE AND EXHIBITION ON HIGH-PERFORMANCE COMPUTING AND NETWORKING, HPCN EUROPE, 1998, Amsterdam. **High-performance computing and networking:** proceedings. Berlin: Springer-Verlag, 1998. p.708-717. (Lecture Notes on Computer Science, v. 1401).

[CUN 2000] CUNHA, J.; KACSUK, P.; WINTER, S. (Ed.). **Parallel Program Development for Cluster Computing**: methodology, tools and integrated environments. Nota: documento obtido junto aos autores, a ser publicado.

[DEL 97] DELAITRE, T. et al. A graphical toolset for simulation modelling of parallel systems. **Parallel Computing**, [S. l.], v. 8, p. 57-84, 1997.

[DOZ 2000] DÓZSA, G. **Visual Programming to Support Parallel Program Design**. Nota: documento obtido junto aos autores, a ser publicado.

[ETN 2001] ETNUS INC. **TotalView Debugger**. Disponível em: <http://www.etnus.com/products/totalview>. Acesso em: abr. 2001.

[FRA 99] FRANCIONI, J. M.; PANCAKE, C. **High Performance Debugging Standards Effort.** Disponível em: <http://www.ptools.org/hpdf/article>. Acesso em: ago. 1999.

[FRU 91] FRUCHTERMAN, T.; REINGOLD, E. Graph Drawing by Forced Directed Placement. **Soft. Practice and Experience**, London, v. 21, n. 11, p. 1129-1164, Nov. 1991.

[GEI 94] GEIST, Al et al. **PVM**: Parallel and Virtual Machine- A User s Guide and Tutorial for Networked Parallel Computing. London: MIT, 1994.

[GIE 97] GIESE, H.; WIRTZ, G. Modular development of correct Meander programs. In: INTERNETIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, PDPTA, 1997. **Proceedings…** Las Vegas: [s.n.], 1997. 4p.

[HEA 91] HEATH, M.; ETHERIDGE, J. Visualizing the Performance of Parallel Programs. **IEEE Software**, New York, v. 8, n. 5, p. 29-39, May 1991.

[HEA 96] HEATH, M. Visualization of parallel and distributed systems. In: **Parallel and Distributed Computing Handbook**. New York: McGraw-Hill, 1996. p. 897-916.

[HOO 96] HOOD, R. The p2d2 Project: Building a Portable Distributed Debugger. In: SIGMETRICS SYMPOSIUM ON PARALLEL AND DISTRIBUTED TOOLS, SPDT, 1996, Philadelphia. **Proceedings...** New York: ACM, 1996. Homepage do p2d2. Disponível em: <http://www.nas.nasa.gov/Groups/Tools/p2d2>. Acesso em: mar. 2002.

[HPD 99] HIGH PERFORMANCE DEBUGGING FORUM. **HPD Version 1 Standard:** Command Interface for Parallel Debuggers. Disponível em <http://www.ptools.org/hpdf/draft>. Acesso em: dez. 1999.

[HUB 2001] HUBAND, S.; MCDONALD, C. DEPICT: A topology-based debugger for MPI programs. In: HIGH-LEVEL PARALLEL PROGRAMMING MODELS AND SUPPORTIVE ENVIRONMENTS, HIPS, 6., 2001. **Proceedings…** Berlin: Springer-Verlag, 2001. p. 109-121.

[KAC 97] KACSUK, P. et al. A Graphical Development and Debugging Environment for Parallel Programs. **Parallel Computing**, [S. l.], v. 22, p. 1747-1170, Jan. 1997.

[KRA 97] KRANZMÜLLER, D.; GRABNER, S.; VOLKERT, J. Debugging with the MAD Environment. **Parallel Computing**, [S. l.], v. 23, p. 199-217, Feb. 1997.

[LEB 87] LEBLANC, T.; MELLOR-CRUMMEY, J. Debugging Parallel Programs with Instant Replay. **IEEE Transactions on Computers**, [S.l.], v. C-36, n. 4, p. 471 - 481, 1987.

[LER 2001] LEROUX, H.; EXTON, C. COOPE: A tool for representing concurrent object-oriented program execution through visualization. In: EUROMICRO WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING, EURO-PDP, 9., 2001. **Proceedings…** Los Alamitos: IEEE Computer Society, 2001.

[LEU 92] LEU, E. **La Réexécution Pièrre Angulaire de la Mise au Point des Programmes Parallèlles**. 1992. Tese (Doutorado em Ciência da Computação) - École Polytechnique Federale de Lausanne, Lausanne.

[MAI 95] MAILLET, E. Issues in performance tracing with Tape/PVM. In: EUROPEAN PVM USERS' GROUP MEETING, EuroPVM, 1995. **Proceedings…** Lyon: [s.n.], 1995. p. 143-148.

[MAY 96] MAY, J.; BERMAN, F. Retargetability and extensibility in a parallel debugger. **Journal of Parallel and Distributed Computing**, Orlando, v. 35, n. 2, p. 142-155, June 1996.

[NEO 2001] NEOPHYTOU, N.; EVRIPIDOU, P. Net-debx: a web-based debugger of MPI programs over low-bandwidth lines. **IEEE Transactions on Parallel and Distributed Systems**, Los Alamitos, v. 12, n. 9, p. 986-995, Sept. 2001.

[NET 2002] NET-DBX Home Page. Disponível em: <http://www.cs.ucy.ac.cy/~net-dbx>. Acesso em: mar. 2001.

[NOR 93] NORMAN, M. G.; THANISH, P. Models of Machines and Computation for Mapping in Multicomputers. **ACM Computing Surveys**, [S. l.], v. 25, n. 3, p. 263-302, Sept. 1993.

[PAC 97] PACHECO, P. **Parallel Programming with MPI**. San Francisco: Morgan Kaufmann, 1997.

[PAN 99] PANCAKE, C. Applying Human Factors to the Design of Performance Tools. In: INTERNATIONAL EURO-PAR CONFERENCE ON PARALLEL PROCESSING, EURO-PAR, 5.,1999. **Parallel processing:** proceedings. Berlin: Springer-Verlag, 1999. p. 44-60. (Lecture Notes in Computing Science, v. 1685).

[PER 96] PERROT, R. Parallel Languages. In: **Parallel & Distributed Computing Handbook.** New York: McGraw-Hill, 1996. p. 843-864.

[PIN 97] PINHEIRO, M.; DRUMMOND, L. Detecção de *Breakpoints* em Programas MPI. In: SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES E PROCESSAMENTO DE ALTO DESEMPENHO, 9., 1997, Campos do Jordão, SP. **Anais...** Campos do Jordão: Laboratório de Sistemas Integráveis/Escola Politécnica da USP, 1997. p. 153-166.

[QUI 94] QUINN, M. **Parallel Computing – Theory and Practice**. New York: McGraw Hill, 1994.

[SCH 93] SCHEIDLER, C.; SCHAFERS, L.; TRAPPER: A Graphical Programming Environment for Industrial High-Performance Applications, In: PARALLEL ARCHITECTURES AND LANGUAGES EUROPE, PARLE, 1996. **Proceedings…** Munich: [s.n.], 1993. p. 403-413.

[SIL 94] SILVA, M. **Desenho Automático de Diagramas.** 1994. Dissertação (Mestrado em CIência da Computação) – Departamento de Ciência da Computação, Unicamp, Campinas.

[SIS 94] SISTARE, S. et al. A Scalable Debugger for Massively Parallel Message-passing Programs. **IEEE Parallel and Distributed Technology**, [S.l.], v. 2, n. 2, p. 50-56, 1994.

[STA 93] STASKO, J.; KRAEMER, E. A Methodology for Building Aplicattion-specific Visualizations of Parallel Programs. **Journal of Parallel and Distributed Computing**, Orlando, v. 18, n. 2, p. 258-264, June 1993.

[STA 94] STALLMAN, R. **Debugging with GDB**. Boston: Free Software Foundation, 1994.

[STE 98] STEIN, B. O.; KERGOMMEAUX, J. C. Interactive Visualization Environment of Multi-threaded Parallel Programs. In: PARALLEL

COMPUTING: FUNDAMENTALS, APPLICATIONS AND NEW DIRECTIONS, PARCO, 1998. **Proceedings...** Bonn: [s.n.], 1998. p. 311-318.

[STR 97a] STRINGHINI, D. **TFPS**: Um Sistema de Pré-processamento de *Traces* para Auxiliar na Visualização de Programas Paralelos. 1997. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[STR 97b] STRINGHINI, D.; NAVAUX, P. **TFPS:** Um Sistema de Pré-processamento de *Traces* para Auxiliar a Visualização de Programas Paralelos. In: SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES E PROCESSAMENTO DE ALTO DESEMPENHO, 9., 1997, Campos do Jordão, SP. **Anais...** Campos do Jordão: Laboratório de Sistemas Integráveis/Escola Politécnica da USP, 1997. p. 533-537.

[STR 2000] STRINGHINI, D.; NAVAUX, P.; KERGOMMEAUX, J. C. A Selection Mechanism to Group Processes in a Parallel Debugger. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, PDPTA, 2000. **Proceedings...** Las Vegas: [s.n.], 2000. p. 2575-2581.

[STR 2001] STRINGHINI, D., NAVAUX, P., KERGOMMEAUX, J. C. A Debugger Interface for Parallel Programs In: SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES E PROCESSAMENTO DE ALTO DESEMPENHO, 13., 2001, Pirenópolis, GO. **Anais...** Brasília: Biblioteca Imediata do Departamento de Ciência da Computação da UNB, 2001. p.214 – 221.

[WIS 96] WISMULLER, R. et al. Interactive debugging and performance analysis of massively parallel applications. **Parallel Computing**, Netherlands, v. 22, n. 3, p. 415-442, Mar. 1996.

[WUX 99] WU, X. et al. Design and implementation of a Java-based distributed debugger supporting PVM and MPI. **Parallel and Distributed Computing and Practice**, [S.l.], v. 2, n. 4, Dec. 1999.