FD-1262

Jaime Simão Sichman

UMA FERRAMENTA DE MONITORAÇÃO PARA UM NÚCLEO DE RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS ORIENTADO A OBJETOS

Dissertação apresentada à Escola Politécnica da USP para a obtenção do título de Mestre em Engenharia de Eletricidade

(PCS)

Orientador: Prof. Dr. Eleri Cardozo

São Paulo, 1991

AGRADECIMENTOS

Ao Centro de Pesquisas em Engenharia Elétrica (CEPEL), Instituto Tecnológico da Aeronáutica (ITA), Universidade de Carnegie Mellon e ao Laboratório de Sistemas Integráveis (LSI) da EPUSP por terem possibilitado, em tempos distintos, a utilização de estações de trabalho nas quais a ferramenta proposta foi implementada. No caso do LSI/EPUSP, gostaria de ressaltar o apoio que recebi do Eng. Casimiro de Almeida Barreto, que muito me auxiliou nos meus primeiros passos com o sistema X WINDOWS v. 11.

Ao Prof. Antonio Marcos de Aguirra Massola, por ter me propiciado a chance de ingressar na vida acadêmica.

À Prof. Edith Ranzini, por se mostrar sempre disponível a resolver os meus problemas cotidianos.

Ao Prof. Francisco José de Oliveira Dias, chefe do Departamento de Engenharia de Computação e Sistemas Digitais (PCS), por ter permitido minha estada no ITA nos últimos meses de realização do trabalho.

Aos meus colegas do Laboratório de Sistemas Digitais (LSD) da EPUSP, pelo apoio e troca de idéias durante estes anos de convivência. Gostaria de agradecer particularmente aos meus colegas da sala 29, Eng. Wladimir Roberto Esposito e Eng. Luiz Alberto Costa Ortiz, pelo insubstituível auxílio na utilização da rede Novell e do processador de textos WORD v. 5.0, e ao Eng. Italo Santiago Vega, pelas valorosas sugestões e leituras preliminares deste material.

A Silvia M. F. Carvalho, por auxiliar-me em esclarecer, durante todos estes anos, o meu intrincado processo de compreensão da aventura humana.

Ao Nei, Ronald, Marcos, Guga e ao outro habitante "fantasma" da casa onde me hospedei em São José dos Campos, o meu muito obrigado pelo carinho e hospitalidade com que fui recebido.

A todos os meus amigos, que se privaram de minha convivência durante este árduo, porém profícuo período. De uma forma ou de outra, estiveram sempre presentes.

Ao Prof. Eleri Cardozo, orientador deste trabalho, cabe um agradecimento especial. Além de seu inestimável auxílio na parte técnica, engajou-se pessoalmente no sentido de providenciar os recursos computacionais necessários para a implementação prática deste trabalho. Tal fato lhe custou algumas viagens ao Rio de Janeiro, bem como alguns processos burocráticos para possibilitar a utilização das máquinas do ITA. Nos últimos meses, quando permaneci em São José dos Campos, mostrou-se solícito para atender até as minhas dúvidas mais banais. Neste período, tive a sorte de poder tê-lo conhecido melhor e descoberto sua grandeza como ser humano. Tal convivência, certamente, jamais se apagará de minha memória.

Nada do que posso me alucina
Tanto quanto o que não fiz
Nada que eu quero me suprime
De que por não saber inda não quis

Jura Secreta (Sueli Costa e Abel Silva)

Estamos atrás de descobertas
Não somos profissionais do espanto
Para achar a água é preciso descer terra adentro
Encharcar-se no lodo
Mas há os que preferem olhar os céus
Esperar pelas chuvas

Somos Profissionais (Oduvaldo Vianna Filho)

SUMÁRIO

1 INTRODUÇÃO 1

2 RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS 6
2.1 Motivação 6
2.2 Fases de uma Resolução Distribuída de Problemas 8
2.3 Métodos de Cooperação em Resolução Distribuída de Problemas 10
2.4 Vantagens e Custos de uma Resolução Distribuída de Problemas 12
2.5 Controle e Comunicação em Resolução Distribuída de Problemas 15
2.6 Um Exemplo: O Paradigma "Blackboard" 17
2.6.1 Histórico 17
2.6.2 Arquitetura Básica 17
2.6.3 O Sistema HEARSAY-II 19
2.6.4 Outras Aplicações do Paradigma 21
2.6.5 Aplicações Apropriadas à Utilização do Paradigma 22
2.6.6 Tópicos de Pesquisa Correntes 23
2.6.7 Vantagens e Desvantagens do Paradigma 31

3 GRANDES ORGANIZAÇÕES DE SOFTWARE 35
3.1 Situação Atual 35
3.2 Uma Comparação com Grandes Organizações Humanas 35
3.2.1 Modelos de Organizações Humanas 36
3.2.1.1 Agente Único 39
3.2.1.2 Grupo de Agentes 39
3.2.1.3 Hierarquia Simples 40
3.2.1.4 Hierarquia Multi-Nível 41
3.2.1.5 Organização Descentralizada 42
3.2.1.6 Mercado 43

3.2.1.7 Implementação dos Modelos Utilizando "Blackboards" 44
3.2.2 Teoria das Contingências 46
3.2.3 Tipos de Relações Laterais 48
3.2.3.1 Contato Direto 49
3.2.3.2 Elemento de Ligação 49
3.2.3.3 Força de Trabalho 50
3.2.3.4 Time 51
3.2.3.5 Elemento de Integração 51
3.2.3.6 Gerente de Integração 52
3.2.3.7 Autoridade Dual 53
3.3 Integrando Diferentes Paradigmas 54
3.3.1 Paradigmas de Linguagens de Programação 55
3.3.2 Utilização de Diversos Paradigmas 57

4 DPSK: UM NÚCLEO PARA RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS 59
4.1 Motivação 59
4.2 Arquitetura de uma Aplicação Utilizando DPSK 61
4.3 Base de Dados de Objetos 62
4.4 Gerenciador de Transações 65
4.5 Primitivas do Núcleo 67
4.5.1 Primitivas de Comunicação 67
4.5.2 Primitivas de Controle 68
4.5.2.1 Primitiva de Comando 68
4.5.2.2 Primitivas de Sincronização 68
4.5.2.3 Primitivas de Tratamento de Interrupções 69
4.6 Implementação de Relações Laterais 69
4.7 Aplicações Desenvolvidas 70
4.8 Limitações 71
4.9 Trabalhos Correlatos 71

5 PARADIGMA DE OBJETOS 74
5.1 Histórico 74

5.2 Conceitos Básicos 80
5.2.1 Objetos 80
5.2.2 Mensagens e Métodos 80
5.2.3 Classes 81
5.2.4 Hierarquias de Classes e Herança 82
5.2.5 Um Programa Segundo o Paradigma de Objetos 83
5.3 Linguagens e Ambientes 83
5.3.1 SMALLTALK-80 84
5.3.2 C++ 85
5.3.3 CLOS 87
5.4 Aplicações do Paradigma de Objetos 88

6 O SISTEMA DPSK+P 91
6.1 Objetivos 91
6.2 Mecanismos de Comunicação 94
6.2.1 Classes e Objetos Compartilhados 94
6.2.2 Métodos Compartilhados 95
6.3 Mecanismos de Controle 96
6.3.1 Consistência dos Dados Compartilhados 96
6.3.2 Controle dos Processos 98
6.3.3 Sincronização entre Processos 99
6.3.4 Interrupções 99
6.4 Contribuições 100

7 MONITORAÇÃO E DEPURAÇÃO DE SISTEMAS PARALELOS 101
7.1 Dificuldades 101
7.1.1 Existência de Vários Focos de Controle 103
7.1.2 Grande Quantidade de Informações Monitoradas 103
7.1.3 Inexistência de Estado Global 104
7.1.4 Processamento Não Determinístico 104
7.1.5 Caráter Intrusivo da Monitoração 105
7.2 Abordagens Básicas 106

7.2.1 Depuradores Paralelos Tradicionais 107
7.2.2 Depuradores Baseados em Eventos 110
7.2.2.1 Navegadores 112
7.2.2.2 Reexecutores 113
7.2.2.3 Simuladores 114
7.2.2.4 Criação de Eventos de Alto Nível 114
7.2.3 Técnicas de Exibição do Fluxo de Controle 117
7.2.3.1 Apresentação Textual 118
7.2.3.2 Diagramas Temporais 118
7.2.3.3 Animação de Processos 123
7.2.3.4 Múltiplas Janelas 125
7.2.4 Técnicas de Análise Estática 126
7.3 Situação Corrente e Tendências Futuras 128
7.3.1 Depuração de Sistemas Paralelos de Tempo Real 129
7.3.2 Coleta de Informações por Hardware 132

8 UMA FERRAMENTA DE MONITORAÇÃO PARA O SISTEMA DPSK+P 136
8.1 Informações Monitoradas 136
8.1.1 Eventos de Controle 136
8.1.2 Eventos de Comunicação 138
8.1.3 Classes e Instâncias Compartilhadas 139
8.2 Exibição das Informações Monitoradas 139
8.2.1 Exibição dos Eventos de Controle 140
8.2.2 Exibição dos Eventos de Comunicação 142
8.2.3 Exibição do Conteúdo da Memória Compartilhada 145
8.3 Estrutura da Ferramenta 148
8.4 Implementação 154
8.5 Extensões Futuras 154

9 CONCLUSÕES 156

REFERÊNCIAS BIBLIOGRÁFICAS 158

LISTA DE FIGURAS

Figura 1.1	Um Sistema de Monitoração e Controle	2
Figura 2.1	Modelo de Resolução de Problemas por Computador	7
Figura 2.2	Fases de um Processo de Resolução Distribuída de Problemas	9
Figura 2.3	O Paradigma "Blackboard"	18
Figura 2.4	Uma Hierarquia de Abstrações para Compreensão da Fala Humana	20
Figura 2.5	Técnica Mista de Controle para o Paradigma "Blackboard"	25
Figura 2.6	Estrutura de Controle Global e Local	27
Figura 3.1	Representação Hierárquica dos Principais Modelos Utilizados em Organizações Humanas	37
Figura 3.2	Um Grafo Organizacional	38
Figura 3.3	Agente Único	39
Figura 3.4	Grupo de Agentes	40
Figura 3.5	Hierarquia Simples	41
Figura 3.6	Hierarquia Multi-Nível	42
Figura 3.7	Organização Descentralizada	43
Figura 3.8	Mercado	44
Figura 3.9	Implementação dos Modelos de Organizações Humanas Utilizando "Blackboards"	45
Figura 3.10	Relações Laterais: Contato Direto e Elemento de Ligação	50
Figura 3.11	Relações Laterais: Força de Trabalho e Time	51
Figura 3.12	Relações Laterais: Elemento de Integração e Gerente de Integração	52

Figura 3.13	Relações Laterais: Autoridade Dual	53
Figura 4.1	Arquitetura de uma Aplicação Utilizando o Sistema DPSK	62
Figura 4.2	Estrutura da Base de Dados de Objetos	64
Figura 4.3	O Gerenciador de Transações	65
Figura 5.1	Mecanismos de Abstração: Classificação e Instanciação	77
Figura 5.2	Mecanismos de Abstração: Generalização e Especialização	78
Figura 5.3	Mecanismos de Abstração: Agregação e Decomposição	79
Figura 6.1	Interação entre Agentes no Sistema DPSK+P	92
Figura 6.2	Classes Compartilhadas no Sistema DPSK+P	93
Figura 7.1	O Sistema EBBA toolkit	115
Figura 7.2	Diagrama Temporal do Sistema MTDBX	119
Figura 7.3	Diagrama Temporal do Sistema IDD	121
Figura 7.4	Mapas de Concorrência	122
Figura 7.5	Animação de Processos no Sistema BELVEDERE	124
Figura 7.6	Arquitetura do Sistema MAKBILAN	134
Figura 8.1	Exibição dos Eventos de Controle entre Agentes	141
Figura 8.2	Exibição dos Eventos de Comunicação entre Agentes	143
Figura 8.3	Exibição da Estrutura de Classes Compartilhadas	146
Figura 8.4	Exibição do Conteúdo de uma Classe Compartilhada	147
Figura 8.5	Exibição das Instâncias de uma Classe Compartilhada	148

RESUMO

Este trabalho consiste no projeto e implementação de uma ferramenta de monitoração para um núcleo de resolução distribuída de problemas denominado DPSK+P. Tal núcleo permite que diversos agentes, desenvolvidos em diferentes linguagens que suportam o paradigma de objetos, compartilhem dados e métodos entre si, de modo a cooperarem no processo de resolução de determinado problema. A ferramenta de monitoração é baseada em eventos de comunicação e controle entre agentes, utilizando a técnica de animação de processos para efeito de exibição dos resultados. Além disso, propõe um método eficiente de exibição do fluxo de comunicação em sistemas baseados em memória compartilhada, aproveitando a compacidade presente nas hierarquias de classes que compõem um sistema orientado a objetos.

ABSTRACT

This work consists of the design and implementation of a monitoring facility for an object-oriented distributed problem solving kernel named DPSK+P. This kernel allows both data and method sharing between agents, that may have been developed in different object-oriented programming languages, so that they can cooperate in a problem solving activity. The monitoring facility is based on control and communication events, and its exhibition of results uses process animation techniques. It also presents an efficient framework for the exhibition of the communication flow in shared memory based systems, which profites the existing compactness in object-oriented class hierarchies.

A meus pais, sem cujo apoio, amor e dedicação durante todos estes anos, eu certamente não teria me tornado o ser humano que hoje sou

1 INTRODUÇÃO

Segundo alguns pesquisadores [1], as futuras aplicações computacionais, a serem desenvolvidas na década de 90, serão cada vez maiores e complexas. Tais aplicações deverão ser extremamente confiáveis, no sentido de serem capazes de tratar eventuais inconsistências introduzidas no sistema por usuários comuns, que não tenham cultura na área de computação. Além disso, será exigido um elevado nível de desempenho, tanto no que se refere às informações disponíveis aos usuários (como resultados de cálculos) bem como à comunicação interna entre seus diversos módulos.

Por outro lado, os progressos obtidos nos últimos anos pela Engenharia de Software introduzem novos requisitos para o projeto de tais aplicações. Facilidades de modificação, extensão e reconfiguração de sistemas serão cada vez mais requeridas pelos usuários. O modelo de execução será paralelizado, quer no que diz respeito ao ambiente computacional utilizado (multiprocessado ou distribuído) quer no que se refere à natureza da própria aplicação (multiprogramação).

Já do ponto de vista da Engenharia, o projeto de sistemas heterogêneos já é hoje cada vez mais comum. Tal característica se refere tanto à plataforma utilizada (diferentes máquinas conectadas em rede) como aos diversos módulos que compõem o software do sistema, os quais podem ter sido desenvolvidos em diferentes linguagens de programação. É importante ressaltar que tal fato não é obrigatoriamente um requisito de projeto, mas às vezes uma conseqüência de mercado, uma vez que um cliente deseja em primeiro lugar aproveitar ao máximo seus próprios recursos (máquinas e programas) antes de optar por novos. Cada vez mais, os sistemas se tornarão evolucionários, isto é, novas funções serão adicionadas a sistemas já existentes, podendo eventualmente ser

necessária a incorporação de novas máquinas ou programas ao sistema original.

Seja tomado como exemplo um sistema de monitoração e controle, apresentado na figura 1.1. Trata-se claramente de uma plataforma heterogênea, onde dados são monitorados pelos controladores lógico-programáveis (CLPs) e tratados convenientemente pelo conjunto das diferentes máquinas, resultando no envio de comandos para o campo, novamente através dos CLPs. Supõe-se ainda que o sistema deva conter um módulo estatístico, uma interface homem-máquina gráfica e um módulo inteligente de diagnose de falhas.

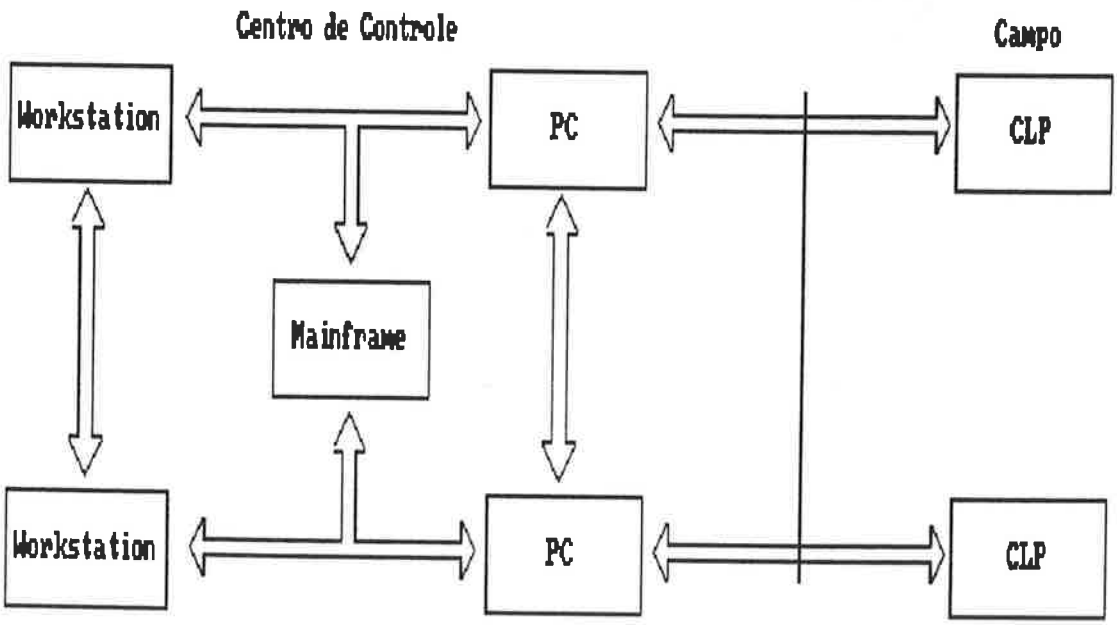

Figura 1.1 - Um Sistema de Monitoração e Controle

Seria desejável que tal sistema apresentasse as seguintes características:

- execução distribuída, em relação a ações, dados e plataformas;

- paralelismo, com unidades de execução de alta granularidade;
- suporte a atividades de coordenação e cooperação entre tarefas e grupos;
- suporte a decisões "inteligentes";
- alta interatividade com os usuários.

Além disso, supondo que já existisse um pacote estatístico desenvolvido em FORTRAN e estivesse disponível uma biblioteca em C para o projeto de aplicações gráficas, seria desejável que o módulo de diagnose, desenvolvido em uma linguagem simbólica como LISP, pudesse ser eficientemente integrado com os outros módulos, independentemente de eventuais incompatibilidades de linguagens de programação.

Finalmente, torna-se imprescindível também prover ao projetista ferramentas precisas para o desenvolvimento de tais aplicações. Somente com a utilização de ambientes do tipo CASE (Computer Aided Software Engineering) e de poderosas ferramentas de monitoração e depuração é que tais aplicações tão complexas poderão ser desenvolvidas num tempo hábil e com os graus de confiabilidade exigidos.

As características desejáveis ao sistema de controle mostrado acima são objeto de estudo de uma área da Inteligência Artificial denominada resolução distribuída de problemas (Distributed Problem Solving). Por outro lado, o paradigma de objetos [1] vem sendo estudado nos últimos anos como uma possível resposta a alguns dos requisitos colocados pela Engenharia de Software, tais como reutilização e extensão de software. Finalmente, esforços também vêm sendo dispendidos no sentido de desenvolver ferramentas de monitoração e depuração que

possam ser eficientemente utilizadas em ambientes distribuídos.

Este trabalho tem como objetivo o projeto e implementação de uma ferramenta de monitoração para um núcleo de resolução distribuída de problemas orientado a objetos. Tal núcleo denomina-se DPSK+P, e consiste numa extensão de outro núcleo, denominado DPSK (Distributed Problem Solving Kernel) [2], desenvolvido na Universidade de Carnegie Mellon, como tese de doutorado do orientador deste trabalho. Ambos os núcleos utilizam o paradigma "Blackboard" [3], aliado ao mecanismo de transações [4], utilizado correntemente na área de Bases de Dados, para possibilitar a distribuição. Além disso, aproveitam idéias provenientes da área de organizações humanas, notadamente uma teoria denominada Teoria das Contingências [5], propiciando que grandes organizações de software possam lidar de modo eficiente com a ocorrência de exceções. Mecanismos para a utilização de diferentes linguagens de programação em uma mesma aplicação, integradas pelo núcleo, também são providos.

O capítulo 2 deste trabalho apresenta os principais conceitos utilizados na área de resolução distribuída de problemas, sendo tomado como exemplo para ilustrá-los o paradigma "Blackboard". No capítulo 3, são mostradas as similaridades entre grandes organizações de software e organizações humanas, sendo também apresentadas a Teoria das Contingências e as tentativas feitas até hoje em integrar diferentes paradigmas de linguagens de programação. O sistema DPSK é detalhado no capítulo 4, bem como alguns outros sistemas com os mesmos objetivos. No capítulo 5, os principais conceitos envolvendo o paradigma de objetos são explicitados, sendo também apresentados alguns ambientes e linguagens já desenvolvidos que se baseiam em tal paradigma. O capítulo 6 apresenta o sistema DPSK+P, tendo sido dada ênfase às novas funções capazes de suportar o paradigma de objetos.

No capítulo 7, são discutidos os principais problemas envolvendo os processos de depuração e monitoração de sistemas paralelos, as principais técnicas de abordagem existentes e apresentados alguns sistemas já desenvolvidos. A solução adotada para o sistema DPSK+P é apresentada no capítulo 8. Por fim, no capítulo 9 são apresentadas as conclusões obtidas durante o desenvolvimento do trabalho.

2 RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS

2.1 Motivação

Os trabalhos realizados até hoje sobre computação distribuída, bem como em outras áreas da Ciência da Computação, são formulados em diferentes níveis de abstração. Quando a literatura se refere a computação distribuída, na maior parte dos casos, a ênfase se encontra nos aspectos ligados à conexão física de diversos processadores, ao protocolo de comunicação entre eles, e ao escalonamento das tarefas nos diversos processadores por meio de algum sistema operacional que coordene suas computações.

A resolução distribuída de problemas lida com estes aspectos em um maior nível de abstração, ou seja, seu interesse reside nas estratégias que devem ser utilizadas para que, ao decompor e coordenar as ações computacionais de um sistema distribuído, estas correspondam aos requisitos estruturais do problema que se deseja resolver. Uma vez resolvida tal questão, o aspecto de implementação pode ser convenientemente tratado, através de técnicas já conhecidas, como por exemplo sistemas multitarefa, sistemas multiprocessados, redes de computadores, etc.

Tal fato pode ser esclarecido pela figura 2.1 [6], onde se encontra representado um modelo genérico de resolução de problemas por meio de um computador. Nota-se claramente que existe um processo de mapeamento, onde algoritmos do mundo real são transformados em algoritmos abstratos que serão efetivamente computados. Enquanto a computação distribuída, segundo o enfoque acima, atua prioritariamente no domínio de soluções, a resolução distribuída de problemas, por sua vez, atua principalmente no domínio de problemas e no mapeamento ao domínio de soluções. Seu objetivo é desenvolver

mecanismos para a divisão de um problema original, geralmente muito complexo, em vários subproblemas, de complexidade baixa ou moderada, que possam ser processados de modo mais eficiente. A solução do problema original é então construída através da integração dos resultados obtidos na solução de cada um dos subproblemas.

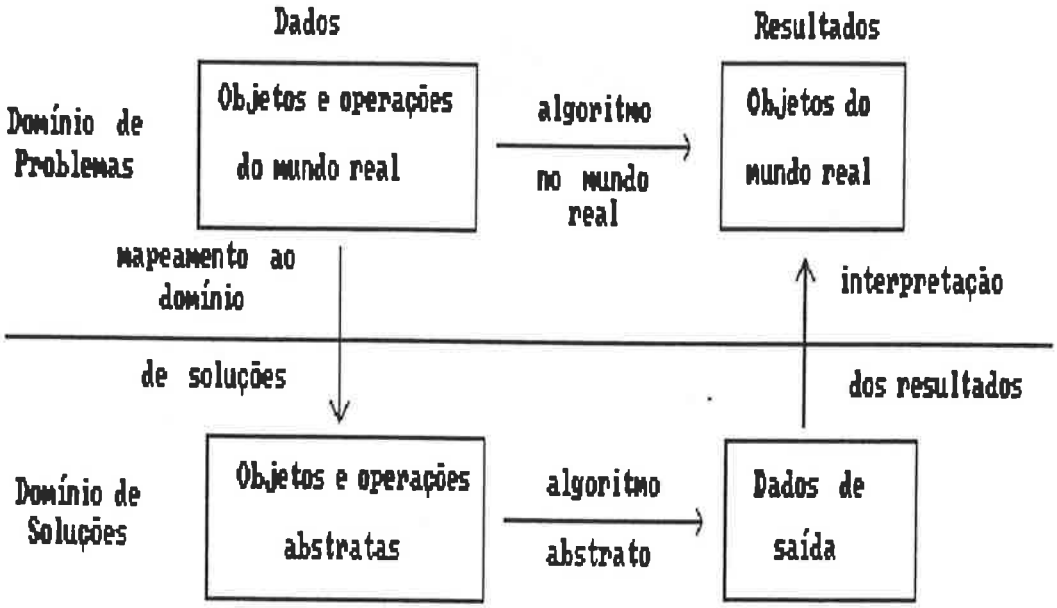

Figura 2.1 - Modelo de Resolução de Problemas por Computador

O interesse nas estratégias utilizadas em resolução distribuída de problemas provém dos inúmeros exemplos de sua utilização que podem ser observados na natureza, e nas organizações humanas em particular. Tais exemplos podem ser encontrados em diversas áreas de conhecimento humano, tais como [7]:

- funções corticais do cérebro humano;
- controles de movimentos biológicos em seres vivos;

- agentes econômicos, como por exemplo mercados;
- sistemas sócio-políticos, como por exemplo a democracia representativa;
- teorias psicanalíticas, notadamente as freudianas;
- modelos cognitivos, como por exemplo a "Society of Minds", proposta por Marvin Minsky;
- modelos de evolução das teorias da inteligência, como por exemplo o proposto por Sternberg [8].

Todos os exemplos citados acima evidenciam que a troca de informações simbólicas apropriadas entre diversos processadores que coordenam os resultados de suas computações é um meio através do qual sistemas biológicos e humanos resolvem seus problemas mais complexos. Partindo deste pressuposto, torna-se interessante incluir tal abordagem na solução de problemas por via computacional.

2.2 Fases de uma Resolução Distribuída de Problemas

A figura 2.2 mostra as diversas fases envolvidas num processo de resolução distribuída de problemas [9].

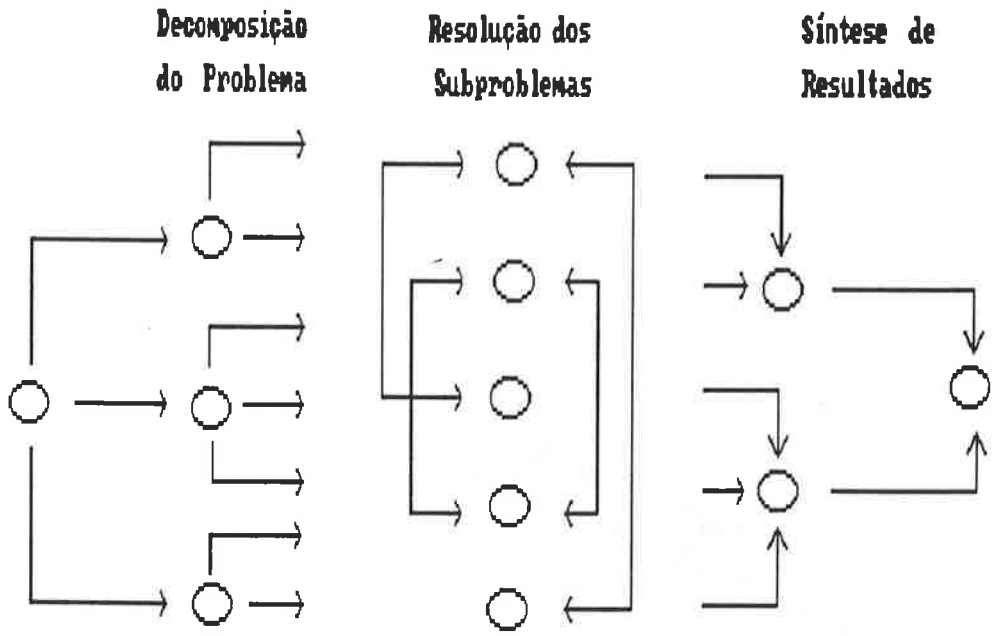

Figura 2.2 - Fases de um Processo de Resolução Distribuída de Problemas

Numa primeira fase, o problema original é decomposto em subproblemas. Tal processo ocorre até que se chegue a subproblemas primitivos, que não podem mais ser decompostos. Pela figura 2.2, pode-se observar que tal processo gera, ao seu final, uma hierarquia de subproblemas.

A segunda fase envolve a resolução dos subproblemas primitivos gerados pela fase anterior. Ainda de acordo com a figura 2.2, observa-se que, em certos casos, estes subproblemas podem não ser completamente independentes, necessitando de um mecanismo de comunicação e cooperação para serem solucionados.

Finalmente, a síntese de resultados é gerada numa terceira fase, onde integram-se os resultados gerados por cada subproblema de modo a obter a solução global do problema original. Observa-se ainda que tal processo pode também envolver uma hierarquia de subproblemas.

De acordo com o problema sendo tratado, estas fases podem variar em importância e complexidade, podendo algumas delas, em certos casos, até mesmo serem suprimidas. Por exemplo, problemas de controle se reduzem basicamente à segunda fase, ou seja, a solução global é a solução de cada um dos subproblemas, caso o problema já tenha sido formulado de modo distribuído.

2.3 Métodos de Cooperação em Resolução Distribuída de Problemas

Uma resolução distribuída de problemas pode ser caracterizada como um mecanismo de resolução cooperativo, através da ação de um conjunto de agentes descentralizados e fracamente acoplados, localizados em um certo número de processadores distintos. De qualquer modo, o aspecto mais importante a ser ressaltado é que, embora sejam independentes, tais agentes devem se balizar por um objetivo comum, que é a solução do problema originalmente proposto. Os mecanismos de cooperação têm como objetivo principal conciliar eventuais conflitos e problemas que aparecem pelo fato de estarem sendo processadas diversas tarefas em paralelo, de modo a usufruir das vantagens provenientes da utilização simultânea de diversas máquinas, como por exemplo aumentar o nível de desempenho global, partilhar bases de dados comuns e apresentar uma maior tolerância a falhas nos processadores individuais.

Quando se analisa o comportamento de um conjunto de especialistas tentando resolver um problema [9], pode-se notar que existem basicamente 2 métodos de cooperação que são correntemente utilizados: divisão de tarefas e divisão de resultados.

A divisão de tarefas ocorre quando os especialistas dividem entre si a carga de trabalho, de modo que cada um possa solucionar individualmente um subproblema do

problema original. O fato significativo a ser observado é como os especialistas decidem quem será o responsável pela resolução de qual subproblema.

Um especialista pode requerer ajuda caso o tamanho de seu subproblema seja grande demais para si ou caso ele não tenha conhecimento suficiente sobre o domínio deste subproblema. Em ambos os casos, ele tenta notificar um outro especialista particular (caso ele já conheça a priori quem possa fazê-lo) ou um grupo de especialistas, com o objetivo de que surjam voluntários que desejem e possam solucionar tal subproblema. Após um processo de negociação, quando os aspectos do subproblema a cumprir são detalhados e identificados, é realizada a escolha do melhor candidato. Um aspecto a ser ressaltado em tal processo é o de que ele ocorre dinamicamente durante o processo de resolução do problema. Tal mecanismo de negociação pode ser convenientemente utilizado de modo a evitar gargalos no processamento, possibilitando um mecanismo de controle mais apurado na alocação de recursos do que aquele que seria obtido por meio das técnicas tradicionais [10].

A divisão de resultados é adequada quando os subproblemas, apesar de serem fracamente acoplados, não podem ser resolvidos por um único especialista, dependendo de resultados obtidos da resolução de outros subproblemas. Deste modo, os especialistas conversam entre si periodicamente, reportando os resultados individuais a que chegaram durante o processo de resolução de seus subproblemas individuais. Neste modo de cooperação, os subproblemas cujos resultados serão partilhados devem utilizar uma certa quantidade de dados comuns. Com tal mecanismo, cada especialista pode aproveitar os resultados obtidos pelos outros para:

- confirmar ou descartar os resultados obtidos na resolução de seu próprio subproblema;

- agregar resultados parciais dos outros especialistas com os seus próprios, produzindo novos resultados que sejam úteis não somente a si próprio, mas também aos outros especialistas;

- considerar novas alternativas de resolução para seu próprio subproblema.

Reportando-se à figura 2.2, nota-se que o mecanismo de divisão de tarefas pode ser utilizado nas fases de decomposição e síntese de resultados. Tal mecanismo assume que a solução de cada um dos diferentes subproblemas pode ser atingida com um mínimo de comunicação entre os diversos agentes que os processam. Por outro lado, o mecanismo de divisão de resultados é útil durante a fase de resolução dos subproblemas, quando os problemas primitivos não podem ser solucionados sem uma comunicação entre os diversos agentes envolvidos no processo de resolução.

2.4 Vantagens e Custos de uma Resolução Distribuída de Problemas

Supondo que cada agente de resolução de problemas esteja sendo executado em processadores distintos, existe uma série de vantagens em adotar uma política de distribuição [7]:

- controle de complexidade: o processamento distribuído é uma estratégia básica para o controle de complexidade de uma computação. Uma computação central é custosa tanto em tempo de processamento como na quantidade de memória utilizada.

Distribuir a computação entre diversos agentes possibilita que estes realizem tarefas em paralelo, sendo capazes de tratar segmentos distintos de dados durante a maior parte do tempo. Além disso, o escopo de cada agente torna-se limitado, ou seja, o domínio de entradas para cada um é menor, o que diminui a complexidade, pois esta geralmente é função exponencial do tamanho do espaço de entradas;

- melhoria da resposta degradada: uma computação distribuída propicia respostas degradadas mais graciosas na presença de dados de entrada degradados ou de falhas em porções do sistema;

- suporte a sistemas evolucionários: na medida em que modificações e acréscimos são necessários ao sistema, a computação distribuída adapta-se mais facilmente, uma vez que as modificações são na maioria das vezes locais, desde que a decomposição original do problema tenha refletido corretamente a estrutura da tarefa a ser solucionada;

- menor capacidade de processamento local: em sistemas muito complexos, envolvendo um grande número de entradas e saídas, uma arquitetura centralizada necessita de um processador com capacidade muito grande para responder corretamente aos sinais de entrada/saída;

- replicabilidade: geralmente, uma decomposição gera um número de

subproblemas idênticos, cujo escopo esteja limitado a regiões de processamento, relacionadas com subconjuntos dos dados de entrada. Neste caso, caso seja preciso melhorar a eficiência, cada um destes subproblemas pode ser replicado em diversos agentes, tornando a decomposição um processo recursivo.

Obviamente existe um custo inerente ao processo de decomposição. Um fato que deve ser levado em conta é o de que cada agente pode ser considerado como um filtro, na medida em que este só comunica aos outros o resultado de sua computação. Assim sendo, um único agente que realiza a tarefa de outros dois pode chegar a resultados distintos com os mesmos dados de entrada , uma vez que não depende dos resultados da computação de outros agentes.

Duas estratégias podem ser citadas para minimizar tal efeito. Cada agente pode, por exemplo, assumir o menor número de premissas possíveis ao solucionar seu subproblema, se comprometendo ao mínimo com possíveis efeitos de filtragem. Desta maneira, compromissos locais são feitos de modo conservativo, ou seja, é gerada como resultado a entidade abstrata mais genérica possível dentro do escopo envolvido.

Outra estratégia utilizada é a de relaxação local, onde os resultados de agentes contíguos, que trabalham com dados em regiões vizinhas, são comparados quanto à sua consistência mútua, fazendo com que resultados muito discrepantes sejam ignorados por agentes de nível mais alto. Tal estratégia acarreta vez por outra alguns erros, no caso em que a computação de um agente com resultados discrepantes de outros é correta, mas não é propagada para níveis superiores.

2.5 Controle e Comunicação em Resolução Distribuída de Problemas

Um mecanismo de resolução distribuída de problemas deve necessariamente contemplar três aspectos, que se referem à arquitétura, controle e comunicação entre os diversos agentes que o compõem.

Arquitetura, neste contexto, se refere à forma como os agentes estão logicamente ligados entre si, formando um conjunto ou uma organização, hierárquica ou não. Esta definição tem uma importância vital no processo de resolução, uma vez que estarão definidas as linhas de autoridade entre os agentes, contemplando troca de dados e de comandos. Este aspecto será tratado no capítulo 3 deste trabalho.

Controle entre agentes, neste contexto, significa a possibilidade dos agentes influenciarem de uma maneira direta no processamento de outros agentes. De um modo geral, pode-se dividir tal controle em:

- controle de processo, onde um agente pode ativar, suspender, reativar e terminar o processamento de outros agentes. Em uma arquitetura hierárquica, correspondem a linhas de comando enviadas de níveis superiores para níveis inferiores da hierarquia;

- controle de sincronização, onde pode-se fazer com que o processamento de diversos agentes em certo instante seja definido de modo a obter um determinado estado global de processamento; existem diversos mecanismos na literatura, utilizados em sistemas operacionais, distribuídos ou não, tais como semáforos, monitores [11]

[12] e barreiras [13]. A descrição detalhada de tais mecanismos será omitida, por fugir ao escopo deste trabalho;

- controle de exceções, que correspondem ao tratamento de situações inesperadas no processamento. O mecanismo é, de certa forma, semelhante ao mecanismo de interrupções em hardware; em uma arquitetura hierárquica, correspondem aos sinais enviados de níveis inferiores para níveis superiores da hierarquia, indicando a ocorrência de alguma contingência.

Comunicação entre agentes, por sua vez, refere-se à possibilidade dos agentes trocarem informações entre si, relativas ao processo de resolução do problema em questão. Esta comunicação pode ser executada através de duas políticas:

- mecanismos síncronos, onde o envio e o recebimento de dados é sincronizado entre os agentes envolvidos, sendo que o processamento destes só prossegue após ter sido completada a comunicação. Exemplos desta política são mecanismos baseados em troca de mensagens, chamada remota de procedimentos e "rendez-vous" [11] [12]. Novamente, a descrição detalhada de tais mecanismos será omitida, por fugir ao escopo deste trabalho;

- mecanismos assíncronos, baseados na troca de dados por meio de uma memória comum. Tal memória pode ser compartilhada, podendo todos os agentes acessá-la diretamente, ou centralizada, onde existe um servidor responsável para disciplinar o

acesso. No caso da memória compartilhada, necessariamente os mecanismos de sincronização citados anteriormente serão utilizados para disciplinar o acesso.

2.6 Um Exemplo: o Paradigma "Blackboard"

2.6.1 Histórico

Na década de 70, a Agência de Projetos Avançados de Pesquisa do Departamento de Defesa dos EUA (DARPA) patrocinou vários grupos de trabalho em institutos de pesquisa e universidades norte-americanas com um objetivo na época ousado: a construção de sistemas que pudessem compreender a fala humana. Tal objetivo claramente se distinguia dos esforços realizados até então por cientistas da fala e engenheiros acústicos, por utilizar técnicas de Inteligência Artificial para modelar certos processos cognitivos envolvidos no processo de compreensão da fala humana.

Um destes grupos que obteve resultados muito significativos foi o da Universidade de Carnegie Mellon. Seu sistema, denominado HEARSAY-II, não somente apresentou um desempenho razoável dentre os requisitos pré-fixados pela agência DARPA, como também introduziu um paradigma genérico para coordenar processos independentes que cooperem na resolução de determinado problema. Tal paradigma denominou-se "Blackboard".

2.6.2 Arquitetura Básica

A arquitetura básica do paradigma "Blackboard" que foi utilizada no sistema HEARSAY-II encontra-se apresentada na figura 2.3 [3].

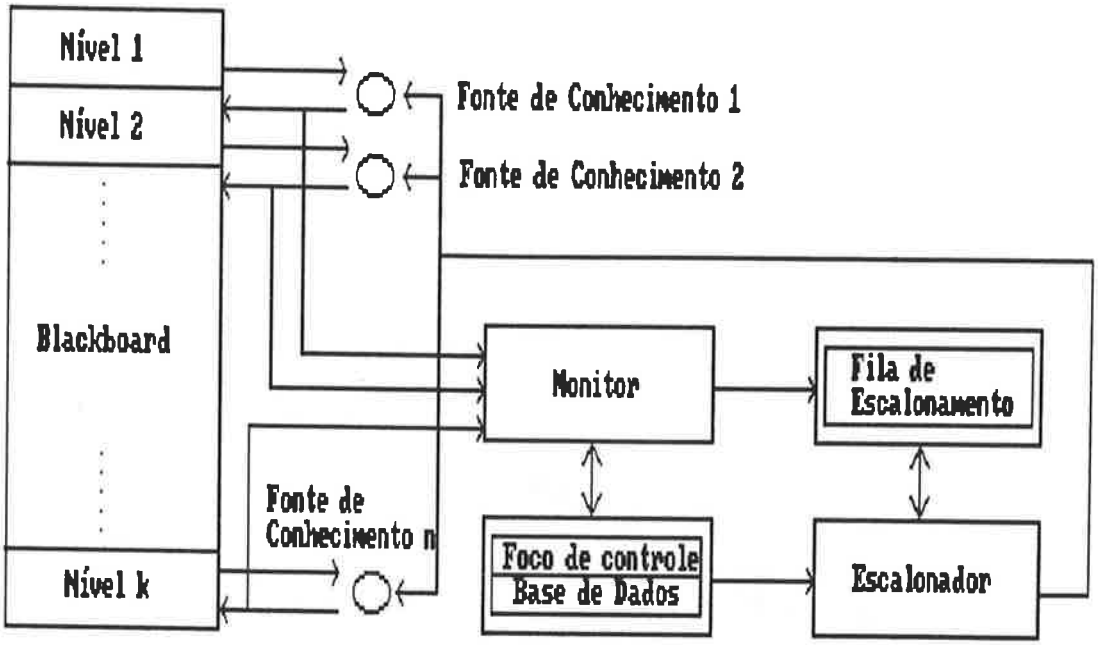

Figura 2.3 - O Paradigma "Blackboard"

Tal arquitetura é composta por uma memória global ("Blackboard") que é compartilhada por diversos módulos de programa (fontes de conhecimento), sendo que o controle de ativação dos diversos módulos é gerenciado pelo conjunto formado pelo escalonador, monitor, foco de controle e fila de escalonamento.

A memória global é dividida em níveis hierárquicos, sendo que cada fonte de conhecimento tem acesso a apenas alguns níveis.

As fontes de conhecimento são independentes entre si, e são compostas por condições de ativação e ações. As condições de ativação contém uma relação de dados cuja existência na memória compartilhada é necessária para que a fonte de conhecimento seja ativada. As ações atuam sobre a memória compartilhada, inserindo, retirando ou modificando dados já existentes, nos diversos níveis hierárquicos aos quais a fonte de conhecimento tem acesso.

A função do monitor é de verificar quais modificações foram feitas na memória compartilhada, de modo a poder eleger possíveis fontes de conhecimento que possam ser ativadas em seqüência, caso alguma dentre suas condições de ativação estejam satisfeitas. Tal mecanismo é realizado através da elaboração, pelo monitor, de uma lista de instâncias (fonte de conhecimento e dados presentes na memória compartilhada que satisfazem uma de suas condições de ativação) que são colocadas na fila de escalonamento.

A função do escalonador é a de eleger uma instância a ser executada a cada final de ativação de uma fonte de conhecimento. Para isso, o escalonador realiza um mecanismo de ordenação da fila de escalonamento, segundo algum critério de prioridade.

A função do foco de controle é a de orientar o monitor a respeito das fontes de conhecimento cuja ativação é mais prioritária para a resolução global do problema, em cada instante.

2.6.3 O Sistema HEARSAY-II

Para representar os diversos níveis de conhecimento envolvidos no domínio de compreensão da fala humana, propôs-se uma hierarquia de abstrações, conforme mostra a figura 2.4 [14].

Figura 2.4 - Uma Hierarquia de Abstrações para Compreensão da Fala Humana

Tal hierarquia apresenta em seus níveis inferiores conhecimentos mais específicos, sendo que estes vão sendo sucessivamente generalizados até que se chegue aos níveis de abstração mais altos. Deste modo, à medida que se sobe na hierarquia, vão sendo incorporados conhecimentos fonéticos, de formação de palavras, sintáticos, de significado até chegar a um conhecimento pragmático geral.

No caso específico do sistema HEARSAY-II, existe uma correspondência específica entre cada uma das fontes de conhecimento mostradas na figura 2.3 e cada um dos níveis de abstração mostrados na figura 2.4. Deste modo, existe uma fonte de conhecimento fonético, outra de conhecimento sintático e assim por diante. A fonte de conhecimento de maior nível de abstração é a responsável pela construção, segundo uma linguagem específica, de uma consulta a um banco de dados sobre produção científica, que é a tarefa para a qual o sistema foi criado. Deste modo, o sistema

consegue então compreender, e portanto responder, a perguntas do tipo "Quais os artigos que se referem à teoria da computação?" e "Quais foram os artigos escritos por McCarthy desde 1974?".

Um resultado interessante foi propiciado pela introdução de um nível adicional de abstração referente à seqüência de palavras, antes de se tentar construir uma frase inteira. Conforme o último exemplo citado acima, seqüências tais como "artigos escritos" e "McCarthy desde 1974" possibilitavam a criação de "ilhas", ou seja, trechos incompletos de frases, de modo que o passo seguinte de execução priorizava a construção de "pontes" entre tais ilhas, em detrimento de buscar outras palavras ou seqüências distintas delas. Outras características do sistema, tais como detalhes das fontes de conhecimento e políticas de escalonamento podem ser encontradas em [3] e [15].

2.6.4 Outras Aplicações do Paradigma

O paradigma utilizado no sistema HEARSAY-II tornou-se extremamente útil para resolução distribuída de problemas em geral. Vários outros sistemas foram construídos segundo tal metodologia [3] [16], como por exemplo um voltado à construção de um modelo para planejamento (OPM) e outro voltado para análise cristalográfica de proteínas (CRYSALIS). Desenvolvimentos mais recentes incluem um sistema de apoio a pilotos de caças, onde requisitos de desempenho de tempo real foram tratados [17]. Estudos também foram realizados para se criar ambientes de suporte ao desenvolvimento de aplicações em "Blackboards", como os sistemas AGE [16] ,GBB [18], ATOME [19] e BLONDIE-III [20], sendo que neste último são fornecidos meios para integrar uma rede de "Blackboards" distribuídos. Métodos para utilizar o paradigma para a construção de sistemas especialistas foram introduzidos pelo sistema HEARSAY-III [21] [22],

inclusive provendo mecanismos para implementar a persistência dos dados pertencentes à memória compartilhada, através de um gerenciador de base de dados relacional.

2.6.5 Aplicações Apropriadas à Utilização do Paradigma

Existem várias similaridades nos problemas tratados até hoje segundo o paradigma "Blackboard". Tais problemas, que seriam adequados à utilização do paradigma, teriam em comum os seguintes atributos [16]:

- vários tipos distintos e especializados de conhecimento envolvidos na resolução do problema;
- integração de informações díspares;
- domínio da aplicação naturalmente hierárquico;
- problemas envolvendo um fluxo contínuo de dados (monitoração, por exemplo), onde deve haver uma adaptação contínua aos dados de entrada, não existindo portanto uma solução final;
- conhecimento e dados esparsos e incertos.

Apesar do paradigma já existir há aproximadamente 15 anos, sistemas que estejam em campo e que efetivamente o utilizem são poucos. Existem duas prováveis causas para tal fato [23]. Em primeiro lugar, o paradigma é apropriado para aplicações complexas, e somente nos últimos anos é que sistemas baseados em conhecimento vêm tendo uma real aplicação comercial, fora dos laboratórios de pesquisa; sendo assim, ainda não houve tempo suficiente para surgirem as aplicações de campo. A segunda razão se deve à utilização do paradigma para o

que se convencionou chamar nos últimos anos de processamento exploratório. Utiliza-se o paradigma durante as tentativas iniciais de desenvolvimento de um dado sistema, devido à flexibilidade de controle e à existência de diferentes fontes de conhecimento, entre outros aspectos. Uma vez melhor conhecido o comportamento do sistema, opta-se por reimplementá-lo numa forma mais direta e eficiente.

2.6.6 Tópicos de Pesquisa Correntes

Entre os diversos tópicos de pesquisa correntes envolvendo o paradigma "Blackboard", cabe destacar três pela sua importância na resolução de problemas em Engenharia: mecanismos de controle mais complexos, suporte à concorrência e paralelismo e resolução de problemas de tempo real [23].

Em resolução distribuída de problemas em geral, e no paradigma "Blackboard" em particular, o controle de ativação de um determinado agente (instância de uma fonte de conhecimento no caso do paradigma "Blackboard") não é trivial. Surgem diversos problemas relacionados a tal fato, quer seja para compatibilizar a ativação de agentes que estejam competindo por recursos comuns, quer seja para coordenar a ativação de agentes que estejam trabalhando em aspectos e níveis de abstração diferentes do problema.

No paradigma "Blackboard" em particular, o controle é originalmente orientado aos dados ("data-driven"). A geração de determinada hipótese como conseqüência da ativação de determinada instância é que vai determinar o próximo passo de processamento a ser realizado. Tal mecanismo, apesar de ser útil em muitos casos, apresenta alguns inconvenientes, tais como

- o escalonamento é instantâneo, sendo considerados apenas os efeitos imediatos no estado do problema, não existindo portanto um mecanismo de inferência para avaliar os efeitos futuros do passo sendo executado. Isto quer dizer que um determinado agente não tem possibilidade de planejar seus próximos passos;

- no caso de não haver nenhuma instância que possa ser ativada, tal mecanismo não possibilita a detecção, e posterior busca, de dados que estejam faltando para que alguma instância possa ser ativada. Em outras palavras, apesar de desejável, não é possível orientar o controle também a metas.

Na figura 2.5, apresenta-se um modo de aplicação de uma técnica mista de controle, envolvendo simultaneamente um componente de controle direcionado aos dados e outro direcionado a metas ("goal-driven") [24]. É introduzida uma memória compartilhada auxiliar, contendo metas a serem atingidas. Tal memória tem a mesma dimensão da memória original, que contém os dados compartilhados. Além disso, é introduzido também um novo módulo de controle, denominado planejador. O monitor, ao invés de colocar diretamente na fila de escalonamento as instâncias, gera metas a serem inseridas nesta segunda memória. O planejador, por sua vez, desenvolve planos para atingir as metas colocadas nesta segunda memória. Tal esquema é certamente mais flexível para o controle em certos domínios onde existam vários módulos pertencentes a diferentes níveis hierárquicos, atuando em aspectos e níveis distintos da solução do problema.

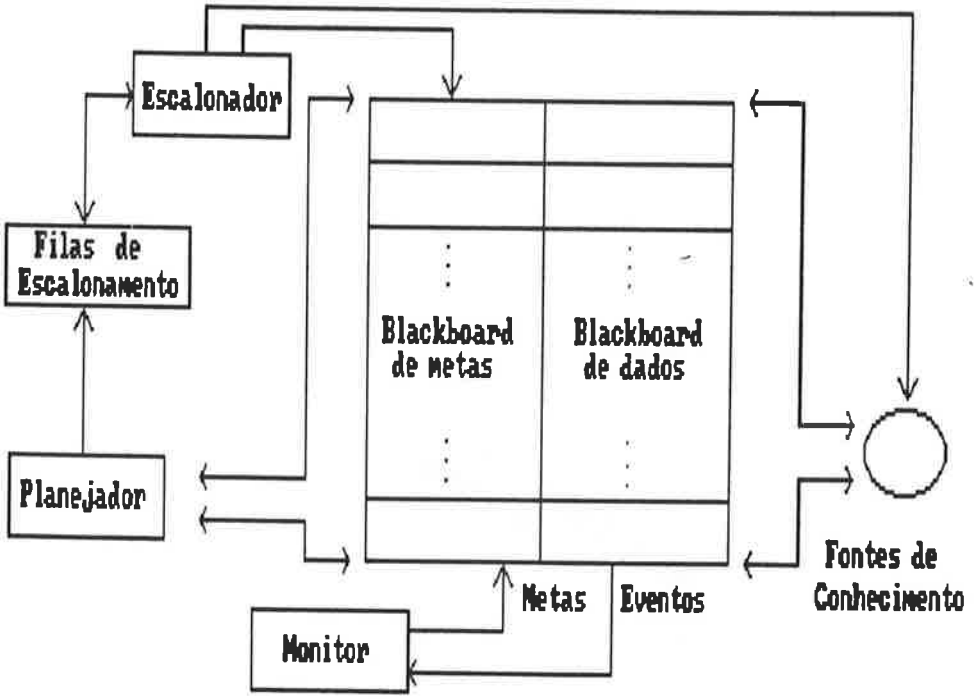

Figura 2.5 - Técnica Mista de Controle para o Paradigma "Blackboard"

Tal mecanismo de controle, entretanto, consegue melhorar apenas o controle local. Supondo um sistema mais complexo, onde existam vários agentes distribuídos, cada um contendo várias fontes de conhecimento, agrupadas em torno de um "Blackboard", torna-se necessário também aplicar uma política de controle global. Tais etapas são extremamente distintas: enquanto o controle local deve priorizar a eleição de determinada tarefa contida em sua fila de escalonamento a ser cumprida por um determinado agente, o controle global tem como tarefa principal balancear as tarefas e responsabilidades que devem ser delegadas aos vários agentes.

Por outro lado, à medida que os problemas tornam-se mais complexos, a hipótese inicial de total independência entre os subproblemas torna-se fraca, devido aos seguintes aspectos:

- se houver uma relação de precedência entre os subproblemas sendo processados, o controle local dos agentes depende do controle exercido pelos agentes que processam os subproblemas precedentes;

- para aumentar a confiabilidade, geralmente o mesmo subproblema é processado por vários agentes; caso algum termine com sucesso sua execução, os outros agentes envolvidos devem evitar o prosseguimento de execução de tarefas redundantes;

- caso agentes atuem em áreas comuns de dados (por exemplo em processamento de imagens), é necessário um mecanismo de sincronização, de modo que cada um possa avaliar como o seu controle local pode afetar os outros.

Torna-se claro, pelas razões expostas acima, que um mecanismo de controle em sistemas complexos deve levar em conta conceitos como relações de precedência, redundância e sincronização. Uma tentativa de integrar tais aspectos num mecanismo de controle é apresentada na figura 2.6, onde o controle local torna-se mais sofisticado, de modo a obter não somente informações sobre o domínio do subproblema onde esteja atuando, como também um conhecimento de nível mais alto sobre a coordenação da rede de agentes [25].

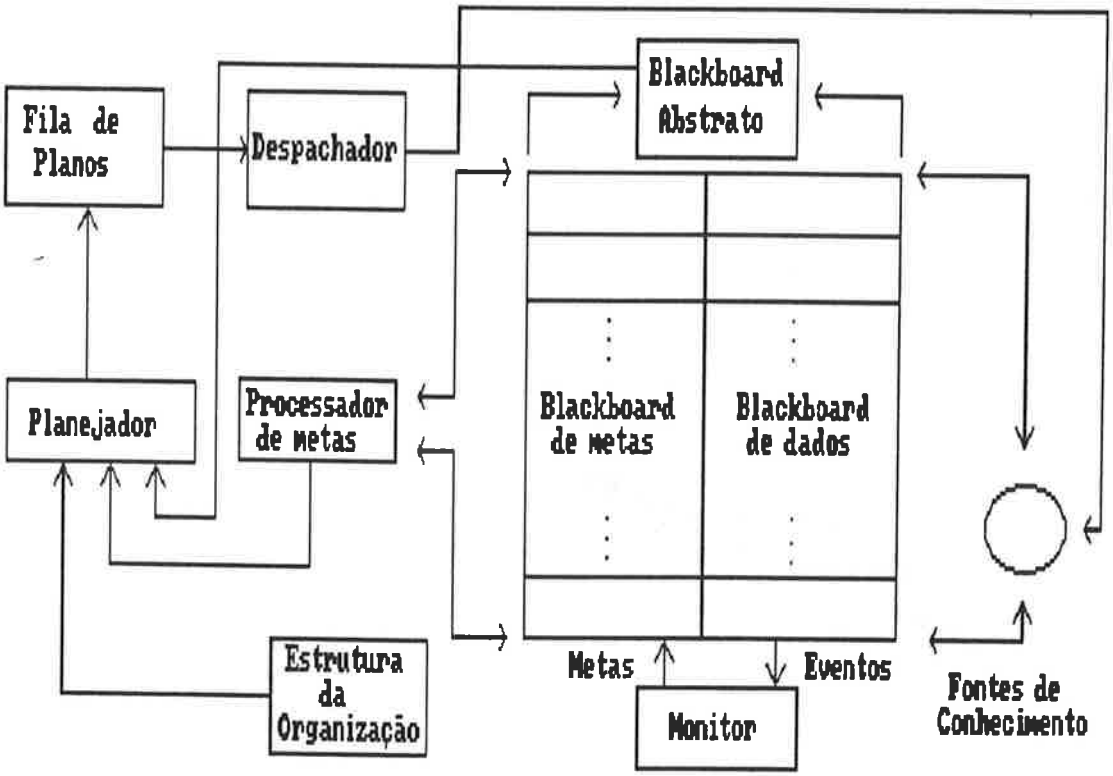

Figura 2.6 - Estrutura de Controle Global e Local

Segundo a figura 2.6, o controle local é especializado em três etapas:

- é introduzida uma estrutura organizacional, consistindo num conjunto de responsabilidades e áreas de interesse de cada agente;

- o planejador local é responsável pela geração de um plano, consistindo de uma seqüência de atividades que cumpram o papel do agente na organização. Eventualmente, como resultado desta etapa de planejamento, a organização estrutural pode ser modificada;

- as hipóteses e metas que constam das memórias compartilhadas são combinadas e

compactadas numa terceira memória compartilhada, denominada "Blackboard Abstrato". Tais informações referem-se a níveis mais altos de abstração, referentes ao papel do agente na rede, e não às suas tarefas específicas.

O planejador, a partir dos dados contidos no "Blackboard Abstrato", formula objetivos de alto nível, que vão sendo sucessivamente refinados até que seja gerada uma instância de fonte de conhecimento a ser ativada. Em outras palavras, o papel do planejador é o de integrar estratégias de longo prazo baseadas na estrutura organizacional, com táticas a médio prazo, provenientes das visões de alto nível obtidas do "Blackboard Abstrato" e com visões de mais curto prazo, indicando as ações imediatas que poderiam ser tomadas, através da fila de instâncias das fontes de conhecimento locais.

O controle global é obtido através da troca de informações contidas no "Blackboard Abstrato". O fato de serem trocadas informações de mais alto nível impede a degradação do nível de desempenho global do sistema, devido à sobrecarga dos canais de comunicação, falha de operação destes mesmos canais e atraso na deliberação de mensagens, entre outros.

Finalmente, cabe ressaltar que a técnica exposta acima ainda supõe uma relativa independência entre os subproblemas, com agentes autocontidos e fracamente acoplados. Tal fato é que determinada o reforço no mecanismo de controle local, o que seria inviável nos casos em que se lida com um número muito grande de agentes, com alta taxa de comunicação [26].

Ainda com respeito à concorrência e paralelismo, existem duas dimensões a serem analisadas [23]. Arquiteturas paralelas envolvem a execução concorrente

das fontes de conhecimento e componentes de controle, se este for mais especializado, como mostrado acima [24] [25]. Além disso, a granularidade da concorrência pode ser baixa, como no caso da ativação simultânea de várias regras dentro de uma mesma fonte de conhecimento, ou alta, como no caso de processamento simultâneo de diversas fontes de conhecimento. Cabe notar que caso o controle seja complexo como o mostrado nos exemplos citados acima, o próprio componente de controle pode ser considerado como uma fonte de conhecimento, o que serve para tornar a escolha do módulo a ser ativado mais complexa, pois em um dado momento do processamento deve-se eleger entre várias fontes de conhecimento do domínio e de controle.

Por outro lado, uma arquitetura paralela envolveria a comunicação de dados entre diferentes sistemas, cada um consistindo num "Blackboard" autônomo, como em [25]. Neste caso, mecanismos para aumentar a velocidade na transmissão de mensagens entre os diversos sistemas e para manter a consistência dos dados replicados devem ser providos.

Uma experiência interessante são os "Blackboards" transacionais [27]. Basicamente, importou-se o conceito de transação, que é correntemente usado na área de banco de dados, de modo a disciplinar o acesso à memória compartilhada pelos diversos agentes [4]. Várias fontes de conhecimento podem se comunicar, acessando dados compartilhados em transações diferentes ou em uma única transação, caso seja necessário uma vista comum desses dados. As principais características que devem estar presentes em uma transação são as seguintes:

- atomicidade: todas as ações contidas em uma transação devem ser completadas, ou todas devem ser descartadas. Tal propriedade é atingida introduzindo-se

operadores de disparo e final de transação, bem como pela manutenção temporária do estado corrente dos dados imediatamente antes do início da transação;

- consistência: duas transações de escrita não podem acessar os mesmos dados ao mesmo tempo. Tal propriedade é mantida através de semáforos;

- persistência: os resultados das transações terminadas devem sobreviver a panes do sistema. Tal propriedade é mantida através do uso de memória não volátil e de protocolos de recuperação.

Finalmente, quanto as aplicações em tempo real, discussões sobre níveis de desempenho, limitação de recursos, compromisso entre velocidade e eficácia, processamento de entrada e saída estão sendo atualmente realizadas. Algumas propostas que vêm sendo estudadas são as seguintes [23]:

- atualização dos dados de entrada e saída fora do ciclo de controle, com eventuais selos de tempo associados;

- prover mecanismos de interrupção durante a ativação das fontes de conhecimento;

- projeto de fontes de conhecimento assíncronas em relação ao ciclo de controle, que seriam ativadas sempre que algum novo dado de entrada fosse provido. Esta ativação se daria através de "demons", procedimentos que são automaticamente disparados quando um determinado parâmetro é acessado;

- prover mecanismos "inteligentes" para efetuar o controle, que pudessem mudar a política adotada em conformidade com as situações de processamento derivadas dos dados de entrada, de modo a priorizar a execução das tarefas mais relevantes, alocando a estas os recursos necessários para a execução;

- ativar mais de uma fonte de conhecimento a cada ciclo de controle, permitindo a concorrência em diferentes níveis de granularidade, conforme os requisitos dinâmicos da execução.

Um sistema para diagnóstico de comutação digital de PABX e outro para gerenciamento de carga em redes de distribuição elétrica podem ser citados como aplicações já desenvolvidas de sistemas de tempo real baseados em "Blackboards", contendo algumas das propostas relatadas acima [28].

2.6.7 Vantagens e Desvantagens do Paradigma

O uso do paradigma "Blackboard" oferece algumas vantagens [3] [29]:

- múltiplas fontes de conhecimento: a capacidade de diversas fontes de conhecimento interagirem de modo cooperativo para a resolução de um problema é especialmente útil em situações onde as informações são incompletas ou incertas. Incertezas podem surgir por fontes de ruído associadas aos dados, ambiguidades aparentes e conhecimento incompleto sobre o domínio do problema;

- múltiplos níveis de abstração : a solução de um problema de modo inteligente geralmente requer a utilização de diversas descrições em diferentes níveis de abstração;

- soluções parciais compartilhadas: cada instância colocada pelo monitor na fila de escalonamento representa uma hipótese gerada por alguma fonte de conhecimento. Deste modo, soluções parciais do problema vão sendo geradas à medida que o problema vai sendo resolvido;

- fontes de conhecimento independentes : o uso de módulos de programa independentes, cuja interação é dependente somente dos dados presentes na memória compartilhada, é extremamente útil, pois possibilita que diferentes projetistas desenvolvam, testem e modifiquem seus próprios módulos sem se importar com os outros que irão compor o sistema global;

- formação incremental de soluções: a solução de um problema é obtida de modo incremental, através da inserção e integração de sucessivas soluções parciais. Deste modo, soluções de problemas baseadas em heurísticas podem ser desenvolvidas;

- método de resolução oportunista: chama-se de oportunista ao método de resolução capaz de explorar seletivamente seus melhores dados e mais promissores métodos em um determinado instante. Uma vez que a seqüência de instâncias selecionadas na

fila de escalonamento não é fixa e sim dependente dos dados presentes na memória compartilhada, pode-se assumir que o paradigma "Blackboard" gera um comportamento oportunista;

- possibilidade de experimentos: a modularidade permite que diferentes alternativas de projeto possam ser testadas, através da substituição de uma ou mais fontes de conhecimento, incluindo controles de alto nível, como por exemplo alterando os focos de controle;

- flexibilidade na máquina de inferência: quando se constrói um sistema baseado em regras, também chamado de sistema de produção, geralmente utilizam-se políticas fixas de encadeamento de regras, para frente ou para trás [30] [31]. A figura 2.5 apresenta uma política mista de controle, onde passos de encadeamento para frente (orientados pelos eventos) e para trás (orientados pelas metas) podem ser alternados conforme o estágio de solução alcançado.

Por outro lado, podem ser apontadas também as seguintes desvantagens [3]:

- excessiva generalização: o paradigma introduzido é excessivamente genérico. Cada processo de inferência depende de um acesso à memória compartilhada. Se por um lado isto é desejável para interações entre diferentes fontes de conhecimento, torna-se indesejável para o processamento de níveis intermediários de decisão

relativos a uma única fonte de conhecimento, por acarretar uma degradação de desempenho;

- conhecimento interpretado: a cada passo de solução, o sistema avalia possíveis instâncias que possam ser ativadas, elege a mais promissora para a situação atual e a ativa. Tal processo, além de requerer tempo, envolve o uso de mecanismos sofisticados. Em outro sistema de compreensão de fala denominado HARPY [3], alguns passos do procedimento de resolução são algoritmizados, compilados e aplicados de forma repetitiva, gerando alguns resultados com melhor nível de desempenho.

3 GRANDES ORGANIZAÇÕES DE SOFTWARE

3.1 Situação Atual

Grandes organizações de software vêm se tornando usuais, tanto na comunidade acadêmica, como na indústria e no setor de serviços. Além dos problemas comuns que são tratados pela Engenharia de Software, tais como minimização de custos e aumento de produtividade, tais organizações ainda tem que se defrontar com problemas adicionais, ou seja, a falta de metodologias adequadas para prover os sistemas de características fundamentais, tais como expansibilidade, suporte à resolução distribuída de problemas e mecanismos de colaboração entre diferentes ferramentas. Tais características encontram-se presentes na quase totalidade das organizações humanas, que são muito semelhantes, em certas proporções, a tais organizações.

3.2 Uma Comparação com Grandes Organizações Humanas

O estudo de grandes organizações humanas, pela similaridade que estas apresentam com grandes organizações de software, torna-se atraente na medida em que diversas teorias já se encontram desenvolvidas e fundamentadas através de sua aplicação ao longo das últimas décadas.

Basicamente, organizações humanas conseguem realizar tarefas complexas, que envolvem o tratamento de dados incertos através de dois mecanismos [32]:

- utilizando um grande número de agentes, com mecanismos extremamente poderosos para que estes possam colaborar entre si;
- aplicando paralelismo e concorrência.

A diferença fundamental entre as duas áreas é que ainda não se conhecem métodos eficientes para implementar mecanismos de colaboração em grandes organizações de software. Tradicionalmente, a quantidade de esforço computacional para se integrar diferentes ferramentas de software é comparável ao esforço dispendido para desenvolvê-las.

Nos últimos anos, o paradigma "Blackboard" vem sendo utilizado para prover tais características a grandes organizações de software. Neste sentido, o mecanismo de colaboração se daria por meio da memória compartilhada, a resolução distribuída de problemas poderia ser implementada por meio de diferentes fontes de conhecimento, e a concorrência/paralelismo seria obtida através da extensão da arquitetura para um ambiente multiprocessado, através do uso disciplinado de transações de acesso à memória compartilhada [27]. Como exemplo de tal utilização, um "Blackboard" de controle poderia ser utilizado como integrador de ferramentas de um ambiente para desenvolvimento de software [33], que tivesse como requisitos básicos extensibilidade, modularidade e flexibilidade.

A seguir, serão apresentadas alguns modelos normalmente utilizados em organizações humanas, que podem ser aplicados em grandes organizações de software.

3.2.1 Modelos de Organizações Humanas

A figura 3.1 apresenta hierarquicamente os principais modelos utilizados em organizações humanas [34].

Figura 3.1 - Representação Hierárquica dos Principais Modelos Utilizados em Organizações Humanas

Todos os modelos apresentados na figura 3.1 podem ser representadas por meio de grafos organizacionais [32], conforme mostra a figura 3.2.

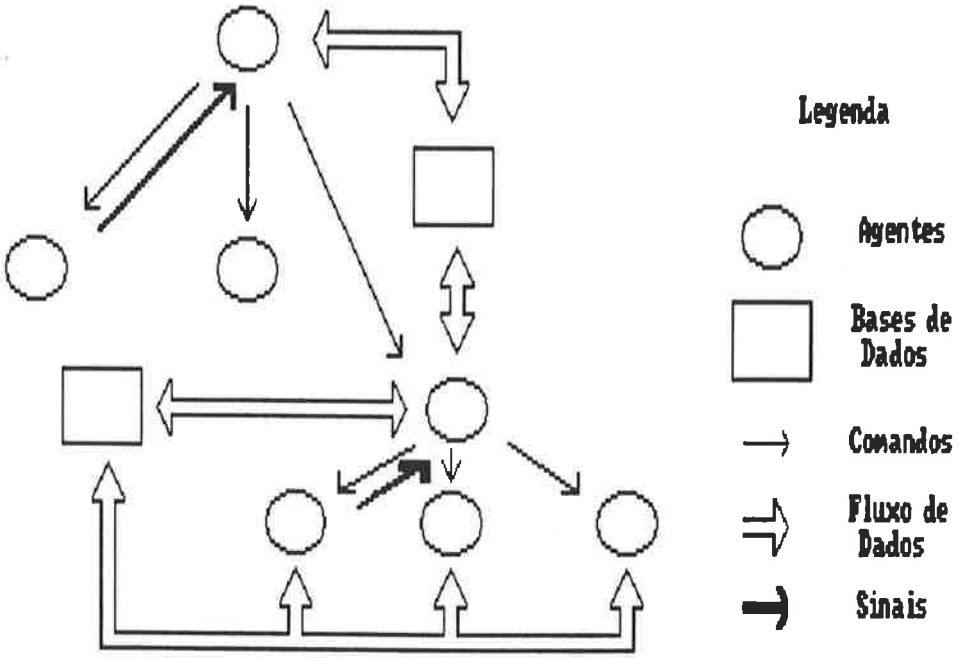

Figura 3.2 - Um Grafo Organizacional

Nestes grafos, existem dois tipos de nós: os agentes, responsáveis pela execução de determinadas tarefas, e as bases de dados, que contém os dados que devem ser compartilhados pelos diversos agentes.

Os arcos podem ser de três tipos: comandos, fluxo de dados e sinais. Os arcos de comando estabelecem linhas de autoridade entre os agentes e se direcionam de níveis de hierarquia superiores para níveis inferiores. Através dessa linhas, comandos tais como "execute esta subtarefa", "envie-me um relatório" e "pare" são enviados.

Os arcos do tipo sinal servem para prover um fluxo dos níveis inferiores para os superiores, servindo para notificar contingências, ou seja, para informar a existência de condições inesperadas, tais como "não posso executar tal tarefa".

Finalmente, os arcos do tipo fluxo de dados representam o fluxo de informações que não sejam comandos nem sinais.

Para efeito de simplificação, nas representações que se seguem de cada um dos modelos apresentados na figura 3.1 serão omitidos os arcos do tipo sinal.

3.2.1.1 Agente Único

A organização mais simples, apresentada na figura 3.3, consiste em um único agente, que realiza todas as tarefas envolvidas na resolução de um problema. Levando-se em conta a existência de uma barreira cognitiva, à medida em que problemas mais complexos vão sendo tratados, este tipo de organização torna-se insuficiente para resolvê-los. Tal princípio de limite de racionalidade foi postulado por Simon [35]:

"A capacidade da mente humana em formular e resolver problemas complexos é muito pequena, comparada com a magnitude dos problemas cuja solução é necessária para que se obtenha comportamentos racionais no mundo real - ou até mesmo para que se alcance uma razoável aproximação desta racionalidade".

Figura 3.3 - Agente Único

3.2.1.2 Grupo de Agentes

Para ultrapassar o limite de racionalidade postulado por Simon, são criados então grupos de agentes, onde

existe uma cooperação entre os diversos elementos para atingir um resultado global, sendo que cada um soluciona um subproblema. A coordenação entre os agentes do grupo é feita através de decisões conjuntas, que são tomadas através da divisão de resultados entre todos os membros do grupo. Tal organização encontra-se representada na figura 3.4.

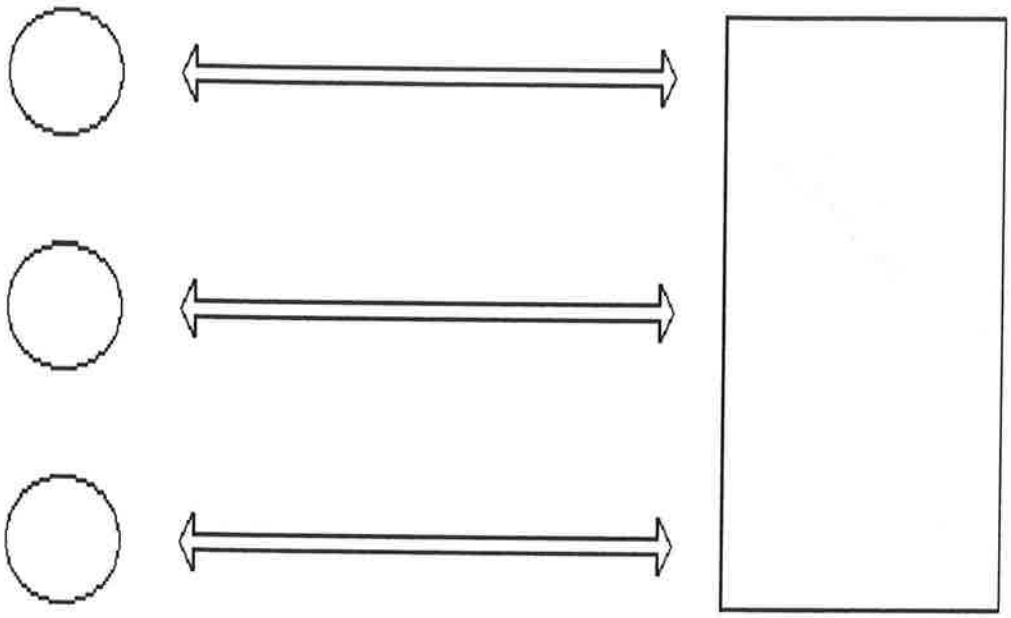

Figura 3.4 - Grupo de Agentes

3.2.1.3 Hierarquia Simples

Caso o número de integrantes do grupo cresça muito, a tomada de decisões coletiva, envolvendo todos os agentes, começa a tornar-se custosa. Criam-se então hierarquias simples, de dois níveis, onde no nível superior existe apenas um único elemento capaz de tomar decisões a respeito dos agentes que pertencem ao nível inferior, conforme mostra a figura 3.5.

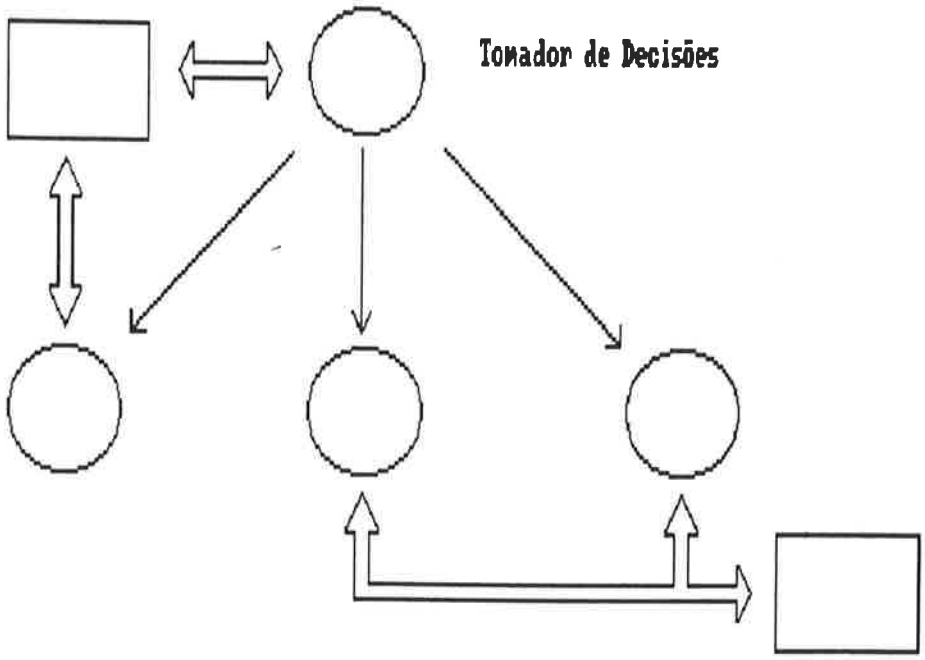

Figura 3.5 - Hierarquia Simples

3.2.1.4 Hierarquia Multi-Nível

Do mesmo modo que a organização de um agente único, a hierarquia simples torna-se insuficiente para tratar problemas mais complexos, devido ao limite de racionalidade do elemento responsável pela tomada de decisões. Como evolução, surgem as hierarquias multi-nível, com diversos níveis de gerenciamento. Deste modo, cada nível dessa hierarquia age como um filtro de decisões e informações que são enviadas aos níveis superiores. Tal organização encontra-se representada na figura 3.6.

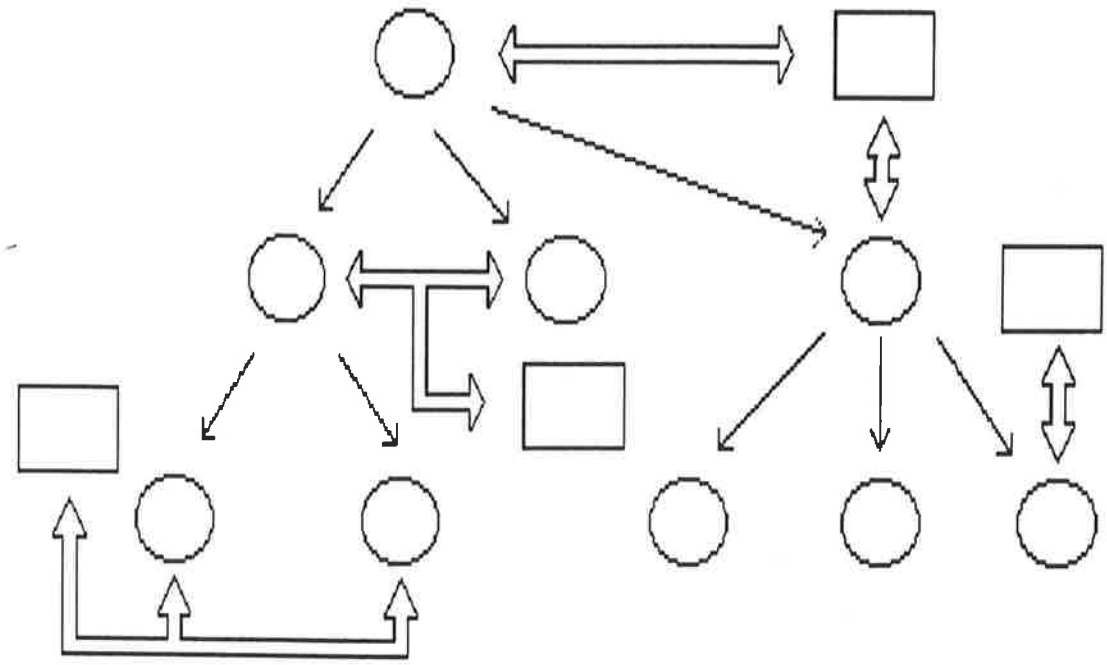

Figura 3.6 - Hierarquia Multi-Nível

3.2.1.5 Organização Descentralizada

Na medida em que o número de tarefas diferentes processadas por uma organização cresce, surge o problema de alocação de recursos. Unidades começam a competir para alocar recursos ao desenvolvimento de suas próprias tarefas. Surgem então as hierarquias multidivisão, onde a organização é dividida em linhas. O controle aqui passa a ser local, sendo criado um grupo de elite responsável pelo controle estratégico, conforme mostra a figura 3.7.

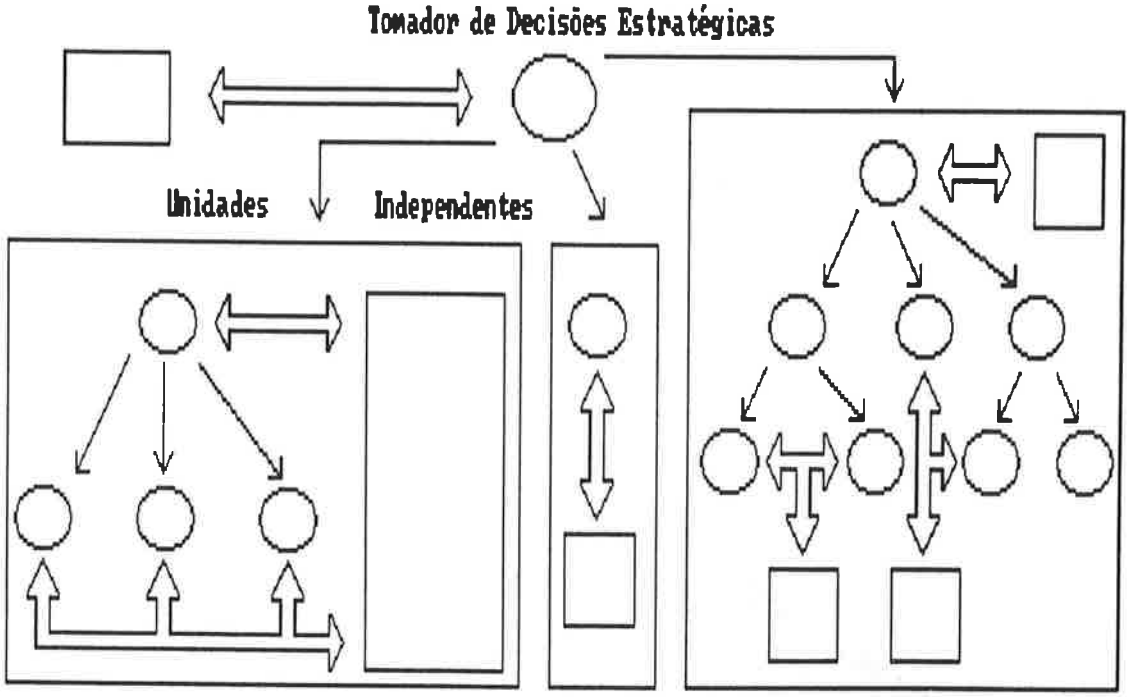

Figura 3.7 - Organização Descentralizada

3.2.1.6 Mercado

Crescendo ainda mais a organização, o esforço de coordenação e a quantidade de informação a ser tratada torna-se maior. São criados então mercados, onde são negociados contratos entre diversas organizações independentes para que sejam processadas as tarefas necessárias. Toda a comunicação encontra-se restrita a um contrato estabelecido entre as organizações. Quaisquer formas de controle entre as organizações são também suprimidas, conforme mostra a figura 3.8.

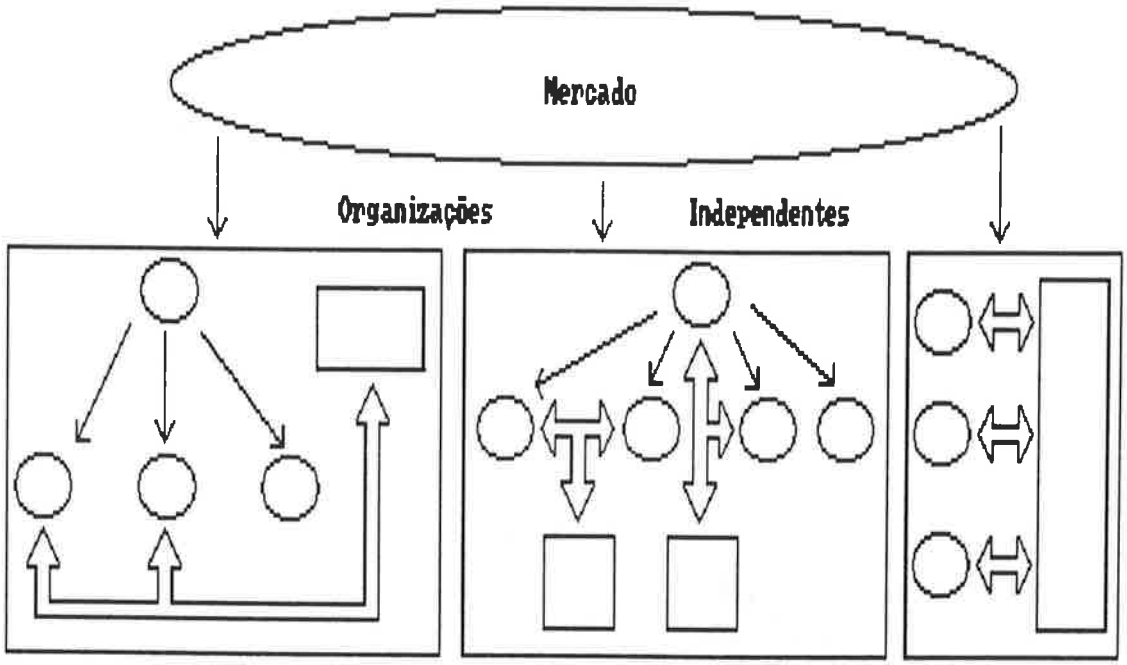

Figura 3.8 - Mercado

3.2.1.7 Implementação dos Modelos Utilizando "Blackboards"

A figura 3.9 indica como cada um destes modelos apresentados na figura 3.2 pode ser implementado através do paradigma "Blackboard".

Arquitetura	Implementação
Mercado	Criação dinâmica de BBs
Organização Descentralizada	Um conjunto de BBs trocando dados estratégicos
Hierarquia Multi-Nível	Um conjunto de BBs do tipo hierarquia simples
Hierarquia Simples	Um BB com uma FC do tipo foco de controle
Grupo de Agentes	Um BB com várias FCs
Agente Único	Um BB com uma única FC

onde BB = "Blackboard" e FC = Fonte de Conhecimento

Figura 3.9 - Implementação dos Modelos de Organizações Humanas Utilizando "Blackboards"

Referindo-se à figura 3.9, um modelo do tipo agente único poderia ser implementado através de uma única fonte de conhecimento. Já um modelo do tipo grupo de agentes seria implementado por uma coleção de agentes independentes, que compartilhariam os dados necessários para integrar suas tarefas. Uma fonte de conhecimento do tipo foco de controle, responsável pelo escalonamento de tarefas e pelas políticas de compartilhamento de dados poderia representar o agente tomador de decisões numa hierarquia simples. Uma extensão para uma hierarquia multi-nível poderia ser obtida através de uma coleção de hierarquias simples, cada uma com seu conjunto de fontes de conhecimento, incluindo um foco de controle.

Um modelo do tipo organização descentralizada poderia ser representado através de uma fonte de conhecimento do tipo tomador de decisões de alto nível, com acesso a informações estratégicas, que implantaria

políticas de partilhamento de recursos e compartilhamento de dados entre as diferentes divisões. Um exemplo deste modelo de organização é apresentado na figura 2.6, onde o planejador e o "Blackboard Abstrato" representam respectivamente o tomador de decisões e o conjunto de informações do nível estratégico.

Uma organização do tipo mercado poderia ser implementada representando os contratos por dados que ativariam fontes de conhecimento específicas. Por outro lado, se considerarmos que uma política contratual pressupõe uma coleção de processos independentes que são instanciados somente na ocorrência de um contrato [10], correspondendo a um processo de criação dinâmica de organizações, o paradigma "Blackboard" não seria o mais indicado para a implementação deste modelo de organização. Neste caso, ele deveria ser ligeiramente alterado, de modo a incorporar uma fonte de conhecimento responsável pelo mecanismo de instanciação de organizações em tempo de execução.

A partir dos dados apresentados nos parágrafos anteriores, pode-se concluir que o paradigma "Blackboard" pode ser convenientemente utilizado para incorporar conceitos de organizações humanas no modelo e implementação de grandes organizações de software.

3.2.2 Teoria das Contingências

A partir de estudos empíricos de grande escala em organizações humanas, foi criada uma teoria denominada Teoria das Contingências [5]. Tais estudos empíricos chegaram a duas conclusões básicas:

- não existe um melhor meio de organização;
- os meios de organização não são equivalentes em relação à eficácia.

Os problemas que ocorrem nas organizações humanas se devem principalmente à existência de incertezas. Estas podem ser consideradas como a diferença entre a quantidade de informação necessária para realizar uma determinada tarefa e a quantidade de informação já processada e disponível na organização. Em outras palavras, as incertezas se referem às informações que devem ser descobertas durante a execução de determinada tarefa, e que não foram consideradas em seu planejamento.

Tais descobertas durante a execução de tarefas geram exceções, que devem ser processadas durante sua realização. Usualmente, são dirigidas para níveis de decisão que encontram-se em níveis mais altos da hierarquia, acarretando alguns inconvenientes, tais como:

- geração de fluxos de informação extras de níveis inferiores para os níveis superiores da hierarquia ("bottom-up");
- geração de seqüências de interrupção nos níveis superiores;
- sobrecarga no tomador de decisões;
- necessidade eventual de replanejamento de tarefas.

A Teoria das Contingências consiste em recomendar estruturas que possam prevenir a ocorrência ou facilitar o tratamento de incertezas. Basicamente, existem duas categorias de recomendações: reduzir a necessidade de processamento de informações ou aumentar a capacidade de processamento de informações. A partir destas recomendações, pode-se observar que o tratamento de incertezas tem uma relação muito grande com a capacidade de processamento de informações existente na organização.

A redução da necessidade de processamento de informações pode ser realizada de dois modos distintos:

- relaxação do nível de desempenho exigido, através da alocação de recursos extras para realizar determinadas tarefas;
- criação de tarefas autocontidas, que minimizem a necessidade de se comunicarem com agentes localizados em níveis de decisão da hierarquia.

O aumento da capacidade de processamento de informações também pode ser realizado através de dois modos distintos:

- investimento em sistemas de informação verticais, que aumentem o fluxo entre os níveis de decisão e os níveis mais baixos da hierarquia;
- criação de relações laterais, permitindo que agentes num mesmo nível hierárquico possam ter um maior fluxo de informação para a tomada de decisões locais.

De acordo com a Teoria das Contingências, a introdução de relações laterais é a medida mais eficaz para tratar o aparecimento de incertezas.

A seguir, serão apresentadas algumas destas relações laterais.

3.2.3 Tipos de Relações Laterais

As figuras 3.10 a 3.13 apresentam exemplos de algumas relações laterais que podem ser criadas com o intuito de melhorar o desempenho de uma organização.

3.2.3.1 Contato Direto

O contato direto ocorre quando se cria um arco do tipo fluxo de dados ou sinal entre dois agentes do mesmo nível. Sem este arco, informações que deveriam ser trocadas entre estes dois agentes deveriam passar por um agente localizado num nível superior, o que não só comprometeria o desempenho, como poderia introduzir distorções na própria informação. No exemplo apresentado na figura 3.10a, os agentes B e C se comunicam através de um dado compartilhado.

3.2.3.2 Elemento de Ligação

A criação de elementos de ligação faz com um determinado agente ou divisão possa ter uma melhor visão das atividades de seu nível hierárquico. No caso do exemplo apresentado na figura 3.10b, o agente L serve como elemento de ligação do agente B junto ao agente C. Usualmente, B e C localizam-se em divisões (ou organizações) diferentes.

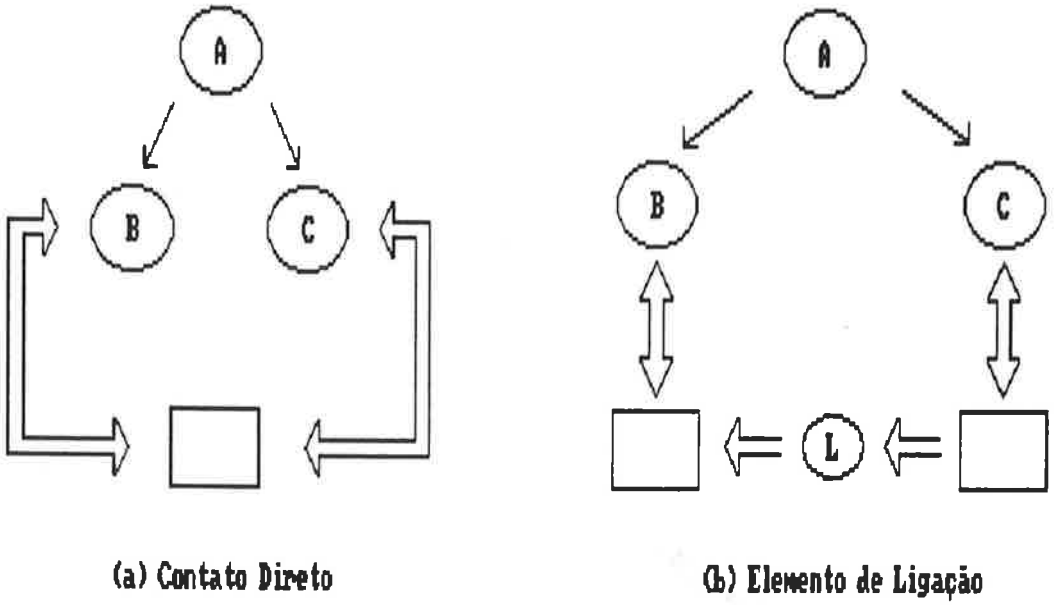

Figura 3.10 - Relações Laterais: Contato Direto e Elemento de Ligação

3.2.3.3 Força de Trabalho

A criação de forças de trabalho serve para tratar exceções que envolvam interesses de várias divisões. Cria-se então uma espécie de coleção de agentes, que compartilham dados comuns, de modo a tratar localmente a exceção, impedindo a sua propagação para níveis superiores da hierarquia. Cada elemento da coleção de agentes pertence a uma divisão afetada pelo aparecimento da exceção. No exemplo apresentado na figura 3.11a, os agentes M são responsáveis pelo tratamento de uma exceção que envolva os interesses das divisões às quais pertencem os agentes B e C. Forças de trabalho são uma espécie de síntese entre a formação de um modelo do tipo grupo de agentes e de elementos de ligação, com o intuito de tratar exceções.

3.2.3.4 Time

Times desempenham uma função muito semelhante àquela desempenhada pelas forças de trabalho. A diferença reside no fato de que servem para prevenir o aparecimento de exceções, ao invés de tratá-las após sua ocorrência. A coleção de agentes é então permanente, sendo que cada agente também pertence a uma divisão que seria afetada pelo aparecimento da exceção. Pode existir também um agente que seja o gerente do time. No exemplo apresentado na figura 3.11b, os agentes M servem para prevenir o aparecimento de exceções que envolvam as divisões às quais pertencem os agentes B e C. O agente GT gerencia a coleção de agentes.

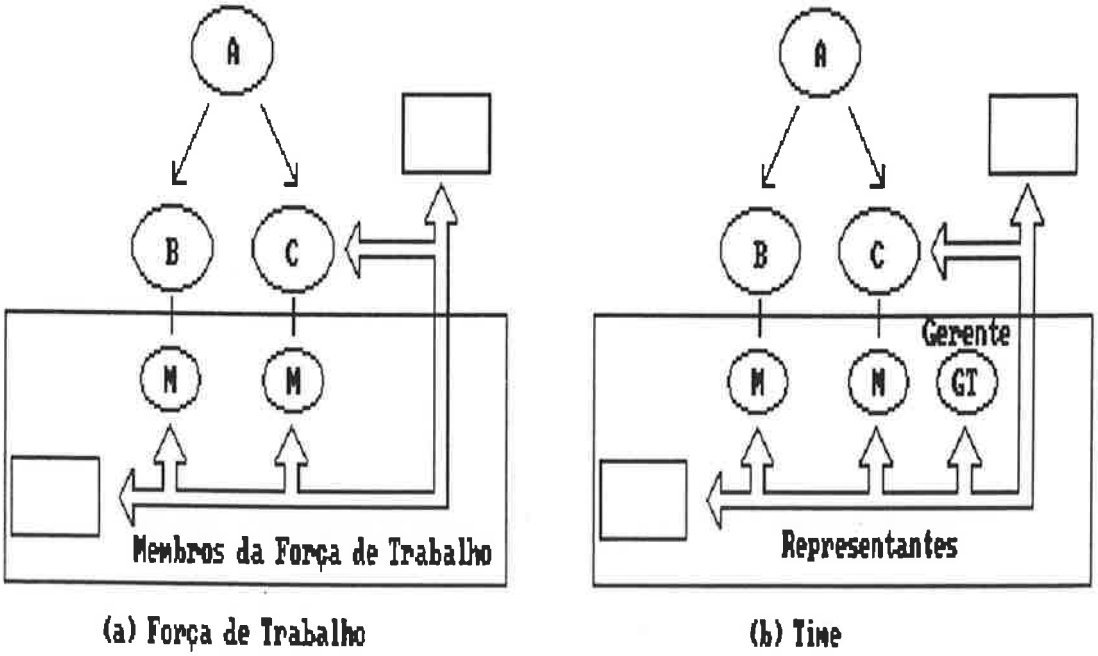

Figura 3.11 - Relações Laterais: Força de Trabalho e Time

3.2.3.5 Elemento de Integração

O elemento de integração serve como representante de níveis superiores da hierarquia em forças de trabalho ou

times, para impedir que as decisões tomadas lateralmente degradem o nível de desempenho global da organização. Pode-se observar tal elemento permite que níveis superiores da organização participem da tomada de decisões, sem no entanto sobrecarregar os canais de comunicação verticais. A figura 3.12a mostra um agente I atuando como elemento de integração.

3.2.3.6 Gerente de Integração

No caso de ocorrências periódicas de determinados tipos de exceção, o elemento de integração pode se tornar um gerente definitivo dentro da organização, cujo papel principal seja desempenhar as funções de um elemento de integração junto às diversas forças de trabalho e times. Tal situação é mostrada na figura 3.12b.

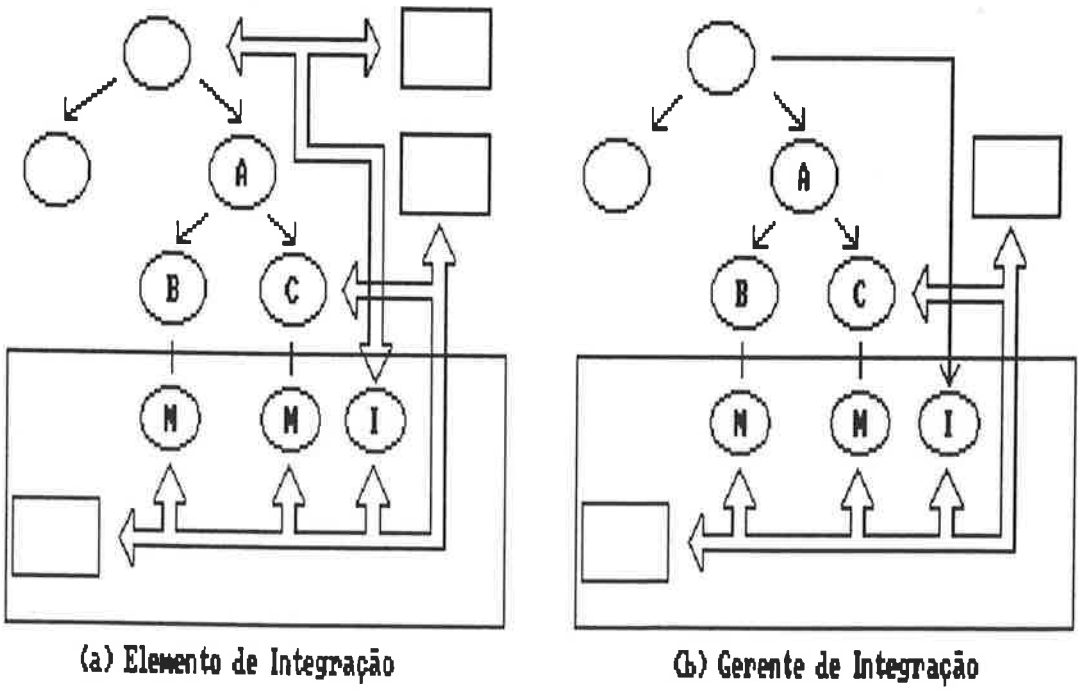

Figura 3.12 - Relações Laterais: Elemento de Integração e Gerente de Integração

3.2.3.7 Autoridade Dual

Quando existe uma autoridade dual, também chamada de gerenciamento matricial, dois ou mais arcos de comando terminam em um agente. No exemplo apresentado na figura 3.13, os serviços dos agentes B e D são compartilhados pelos agentes A e C, aumentando a confiabilidade (B pode ser acessado por C caso A apresente uma falha) e desempenho (C pode intervir mesmo durante a execução por B de uma tarefa solicitada por A). O único cuidado que se deve ter é criar um mecanismo de gerenciamento de conflitos, para evitar que os agentes B e D fiquem confusos com dois pedidos simultâneos; este mecanismo pode ser efetuado através da criação de um agente G responsável pela gerência de tais conflitos.

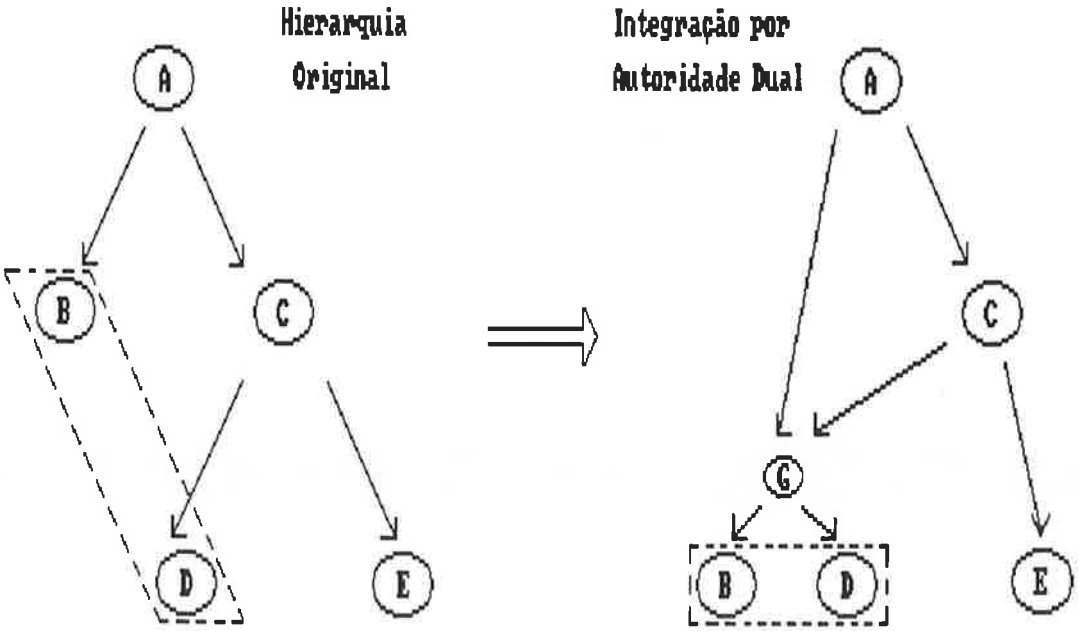

Figura 3.13 - Relações Laterais: Autoridade Dual

3.3 Integrando Diferentes Paradigmas

Tradicionalmente, grandes organizações de software, especialmente aquelas criadas para resolver problemas de Engenharia, necessitam de vários requisitos distintos para solucionar problemas reais, como por exemplo processamento numérico, simbólico, interfaces gráficas para a comunicação homem-máquina e níveis de desempenho adequados para processamento em tempo real. Tais requisitos dificilmente são obtidos quando se utiliza um único processador, tornando atraente a utilização de sistemas distribuídos. Mais ainda, dificilmente uma única linguagem de programação lida eficientemente com tais requisitos simultaneamente.

A solução mais óbvia seria a de utilizar diferentes linguagens de programação, de modo a aproveitar o que cada uma tem de mais eficiente, que posteriormente seriam integradas para se obter a solução global do problema. Infelizmente, resultados empíricos mostram que o esforço para integrar diferentes pacotes de software desenvolvidos em linguagens diferentes é similar àquele dispendido para desenvolver cada uma das ferramentas. Fazendo uma analogia com organizações humanas, seria um problema comparável ao gerenciamento e coordenação de empregados provenientes de países diferentes, cada um falando um idioma diferente, com diferentes conhecimentos educacionais e idiossincráticos. Mesmo quando são utilizados intérpretes, o resultado às vezes fica longe daquele que seria o desejado.

A seguir, serão apresentados resumidamente os principais paradigmas de linguagens de programação existentes e algumas tentativas de integração desses paradigmas.

3.3.1 Paradigmas de Linguagens de Programação

Como em outras áreas da computação, não existe um consenso sobre as características que seriam necessárias a uma determinada linguagem para que esta originasse um novo paradigma, e não que fosse simplesmente uma extensão da linguagem que a originou. Como exemplo, alguns autores consideram que uma linguagem orientada a objetos que incorpore "demons" designa um novo paradigma, denominado linguagens orientadas a acesso [36]; outros, entretanto, consideraram tal linguagem apenas uma extensão do paradigma de objetos.

De qualquer modo, existem certos paradigmas que são reconhecidos pela quase totalidade dos pesquisadores, embora a aderência das linguagens a tais paradigmas ainda não seja consenso. Por exemplo, para alguns LISP é uma linguagem funcional, enquanto para outros é uma linguagem orientada a estruturas de dados, no caso listas.

Tais paradigmas são os seguintes [37] [38]:

- paradigma procedural: utiliza um estilo de programação imperativo, onde os mecanismos de controle são a seqüência, a ramificação e a iteração de comandos. Na maior parte dos casos (pelo menos nas últimas duas décadas), a utilização de tais mecanismos é feita de forma estruturada. Exemplos de linguagens que seguem tal paradigma seriam PASCAL, FORTRAN, C;

- paradigma funcional: envolve unicamente definições e chamadas de funções, que podem ser compostas e aplicadas recursivamente. Ao serem avaliadas, é efetuado um mecanismo de substituição baseado num formalismo matemático

denominado Lambda-Cálculo. LISP seria uma linguagem que segue tal paradigma;

- paradigma lógico: utiliza um estilo de programação declarativo, no qual o controle não é explicito, ficando dependente à teoria subjacente que originou a linguagem. Tal controle é, na maior parte dos casos, uma aplicação do Teorema da Resolução, proposto por Robinson, utilizado na demonstração de teoremas matemáticos, que é implementado através de uma busca em árvores. PROLOG segue este paradigma;

- paradigma modular: corresponde a desenvolver um programa através de diversos módulos, nos quais ocorre um encapsulamento de tipos abstratos de dados. Em cada módulo, ficam armazenados não somente os dados de um mesmo tipo, bem como os métodos de acesso a tais dados. Exemplos deste paradigma são as linguagens MODULA-2 e ADA;

- paradigma de objetos: considera qualquer entidade como um objeto, ao qual estão associados parâmetros e métodos de acesso. A comunicação entre os objetos se dá através do envio e recebimento de mensagens. Os objetos são agrupados em classes, formando uma hierarquia, onde é ativado um mecanismo de herança. SMALLTALK é um exemplo deste paradigma.

3.3.2 Utilização de Diversos Paradigmas

Uma vez que para solucionar problemas reais a aplicação de um único paradigma é, na maior parte dos casos, insuficiente, vários esforços de pesquisa estão sendo feitos nos últimos anos para integrar diversos paradigmas. Infelizmente, tal tarefa não é trivial, pois além de compatibilizar dados (que em alguns casos, nem existem como tais, como em SMALLTALK), os mecanismos de controle são essencialmente distintos.

Em casos de sistemas de pequeno e médio porte, que não requeiram um alto nível de desempenho, uma solução que vem sendo adotada é a troca de dados por meio de arquivos, sendo que na maior parte dos casos tais dados encontram-se representados em um código universal, por exemplo ASCII. Tais arquivos são geralmente temporários, sendo sucessivamente criados e destruídos durante uma sessão de utilização do sistema. Um exemplo da utilização de tal técnica pode ser encontrado no sistema AIPUSP [39], destinado ao auxílio no projeto de unidades de sistemas de produção discretos. Tal sistema compreende um conjunto de ferramentas implementadas nas linguagens C e PROLOG.

Por outro lado, para sistemas maiores, cujos requisitos de desempenho sejam altos, notam-se claramente duas linhas de ação sendo investigadas: projetos de linguagens multiparadigma e de ambientes que suportem linguagens com paradigmas diferentes [38].

No primeiro caso, trata-se de prover ao programador um conjunto de construções, de modo a possibilitar a aplicação de diferentes métodos de abordagem para solucionar problemas complexos. Uma determinada linguagem de programação, que segue determinado paradigma, é então acrescida de comandos e construções adicionais para suportar outros paradigmas. Um exemplo é a linguagem C++,

que incorpora conceitos do paradigma de objetos numa linguagem procedural (C).

No segundo caso, são construídos sistemas que possibilitam que diversas aplicações, desenvolvidas em linguagens diferentes, possam se comunicar, quer trocando dados entre si, quer ativando procedimentos em outras aplicações. Usualmente, tal sistemas suportam multiprocessamento.

Um exemplo de tal ambiente é o sistema Mixed Language Programming (MLP) [40]. São criados processos distintos para cada aplicação, sendo que a comunicação entre as diversas aplicações é realizada através do mecanismo de chamada remota de procedimentos. Além disso, são criados agentes de processamento específicos no processo que invoca outro para traduzir os tipos de dados pertencentes à linguagem origem para uma linguagem intermediária, denominada Universal Type System (UTS), e vice-versa no processo invocado.

Um outro exemplo de ambiente é o sistema LINDA [41]. Trata-se de um núcleo de sistema operacional, que implementa a comunicação e o gerenciamento de ativação entre processos. As mensagens são trocadas através de uma memória compartilhada denominada espaço de tuplas, cujo acesso também é gerenciado pelo núcleo. Uma extensão de LINDA foi recentemente incorporada ao ambiente UNIX [42]. O sistema LINDA apresenta ainda algumas similaridades com o sistema DPSK, que serão convenientemente abordadas no capítulo 4.

Finalmente, estudos também estão sendo feitos para verificar como pode ser realizada a validação de sistemas que incorporem diversos paradigmas [43].

4 DPSK: UM NÚCLEO PARA RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS

4.1 Motivação

Os modelos que até hoje vêm sendo adotados nas diversas linhas de pesquisa desenvolvidas em resolução distribuída de problemas podem ser classificados em três principais correntes:

- generalização de modelos particulares que foram utilizados com sucesso em algum domínio particular de aplicação;
- construção de ambientes computacionais distribuídos independentes de domínio;
- acréscimo de facilidades ao processamento distribuído em linguagens de programação conhecidas.

Não se trata de uma coincidência o fato das duas últimas categorias também serem alvo de pesquisas em ambientes e linguagens para suporte a diversos paradigmas, como mostrado no capítulo 3. A origem dos esforços para estudos nas duas áreas é, de certa forma, comum, visando propiciar ferramentas e ambientes para o desenvolvimento de grandes organizações de software para resolver problemas reais complexos.

A utilização do paradigma "Blackboard" é um exemplo da primeira categoria. Infelizmente, a generalização de um método de resolução para diversos domínios nem sempre é adequada. Tal solução carece da flexibilidade necessária para adotar outros métodos de resolução que sejam mais adequados ao domínio em questão.

A segunda categoria privilegia determinado hardware, na medida em que tais ambientes foram desenvolvidos em

determinadas plataformas. Além disso, existe uma forte dependência em relação aos mecanismos de software próximos à camada do sistema operacional utilizado, principalmente no que diz respeito aos mecanismos de comunicação entre tarefas. A transportabilidade para máquinas diferentes é muito difícil e trabalhosa.

A terceira categoria é inflexível, obviamente, na medida em que privilegia determinada linguagem de programação. Mesmo havendo resultados práticos em linguagens pertencentes a diversos paradigmas, como PASCAL [44], LISP [45] e PROLOG [46], tais ferramentas são extremamente úteis para processamento distribuído, e não para resolução distribuída de problemas, conforme a diferenciação feita no capítulo 2.

O sistema DPSK (Distributed Problem Solving Kernel) [2] [47] constitui-se de um núcleo de rotinas de apoio à resolução distribuída de problemas que é flexível em relação ao método de resolução, ao hardware e ao software adotados pelo projetista. Seus requisitos básicos de projeto foram os seguintes:

- transportabilidade, não sendo customizado em relação ao hardware nem ao software;
- possibilidade de aplicação na resolução de grandes problemas de Engenharia, provendo meios para utilização de
  - quaisquer mecanismos de controle, tais como busca informada, comandos e interrupções;
  - quaisquer mecanismos de comunicação, tais como mecanismos síncronos, assíncronos e memória compartilhada;
- pequeno esforço de programação;

- núcleo pequeno e eficiente;
- possibilidade de implementar construções complexas envolvendo diversos métodos de resolução, como "Blackboards" e escalonadores.

O sistema inicialmente foi desenvolvido em ambiente UNIX 4.2, rodando em estações de trabalho da família VAX. Posteriormente, foi feita uma adaptação para máquinas IBM-PC compatíveis.

Sua principal motivação foi possibilitar a implementação das organizações descritas no capítulo anterior.

A seguir, serão apresentadas algumas características do DPSK.

4.2 Arquitetura de uma Aplicação Utilizando DPSK

A figura 4.1 mostra uma arquitetura típica de uma aplicação utilizando o sistema DPSK.

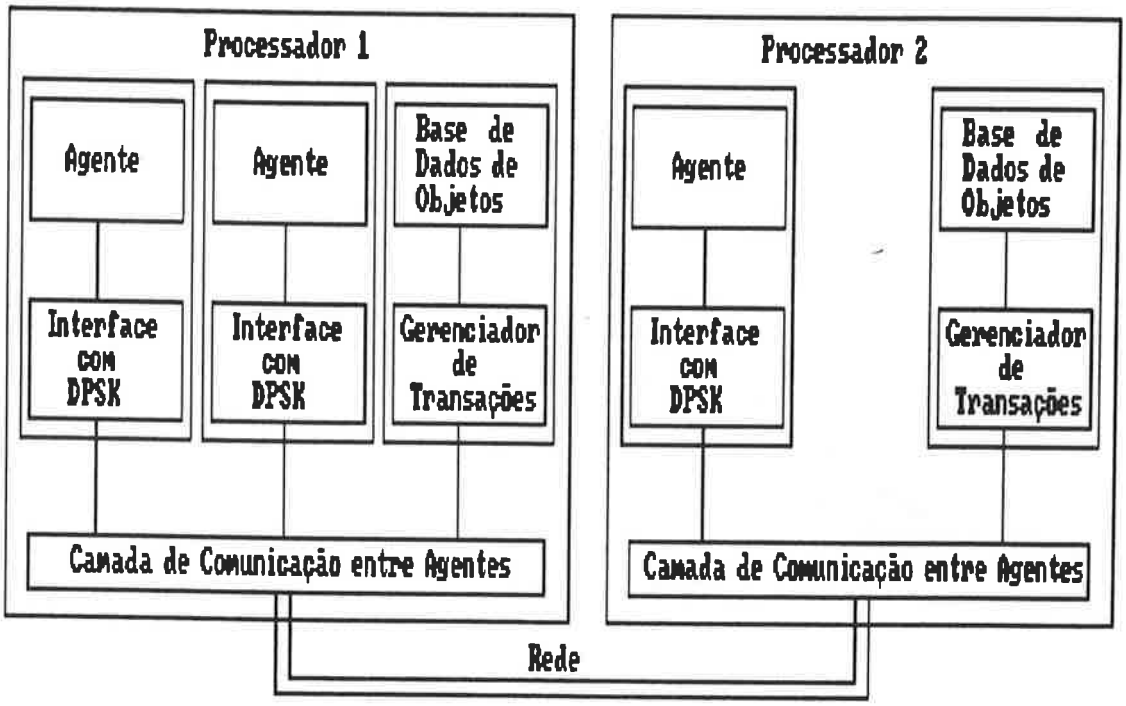

Figura 4.1 - Arquitetura de uma Aplicação Utilizando o Sistema DPSK

A aplicação compõe-se de 3 agentes de resolução de problemas, distribuídos em dois processadores. Os processadores encontram-se ligados através de uma rede local. Cada um dos agentes incorpora uma interface, onde encontram-se codificadas, na linguagem de programação utilizada pelo agente, as primitivas do núcleo. Além disso, em cada nó, existe um conjunto formado pela Base de Dados de Objetos e pelo Gerenciador de Transações, que serão detalhados a seguir.

4.3 Base de Dados de Objetos

O mecanismo de comunicação entre agentes é realizado através de uma memória compartilhada descentralizada. Outros mecanismos foram considerados, como por exemplo troca de mensagens e chamada remota de procedimentos, porém foram considerados menos eficientes para os propósitos do sistema [2]. Basicamente, o mecanismo de

comunicação por memória compartilhada permite um acoplamento mais fraco entre os agentes, pois uma vez escrito um dado em tal memória o número de agentes que podem posteriormente acessá-lo é ilimitado, tornando eventuais modificações e extensões no sistema original mais fáceis de serem implementadas. Tal requisito é fundamental para sistemas destinados a resolver problemas em Engenharia. Além disso, através da ativação de primitivas de controle adequadas, torna-se possível implementar qualquer um destes mecanismos, utilizando a memória compartilhada e mecanismos de sincronização.

São mantidos na memória compartilhada objetos do tipo

(classe (atributo valor)...(atributo valor))

sendo que o acesso comum pelos diversos agentes é controlado pelo Gerenciador de Transações.

A estrutura da Base de Dados de Objetos é mostrada na figura 4.2. Trata-se de uma rede composta por dois tipos de nós: classes e "slots". Um mecanismo de "pattern-matching" é empregado quando se deseja acessar determinado objeto, sendo empregada uma busca linear por classe, a partir da raiz ou uma busca através de uma "hashing table" pelo nome do atributo. Não se permite a duplicação de nós; caso algum par (atributo valor) pertença a mais de um objeto, existirão dois arcos apontando para o par, devidamente identificados, como por exemplo acontece na figura 4.2 com os pares (idade 2) e (idade 5). Ainda com referência à figura 4.2, estão representados os seguintes objetos:

(gato (nome Rex) (idade 2))

(cão (idade 5))

(cão (nome Piti) (idade 5) (peso 10))

(homem (nome Jaime) (idade 28) (peso 70))

(homem (idade 2) (peso 5))

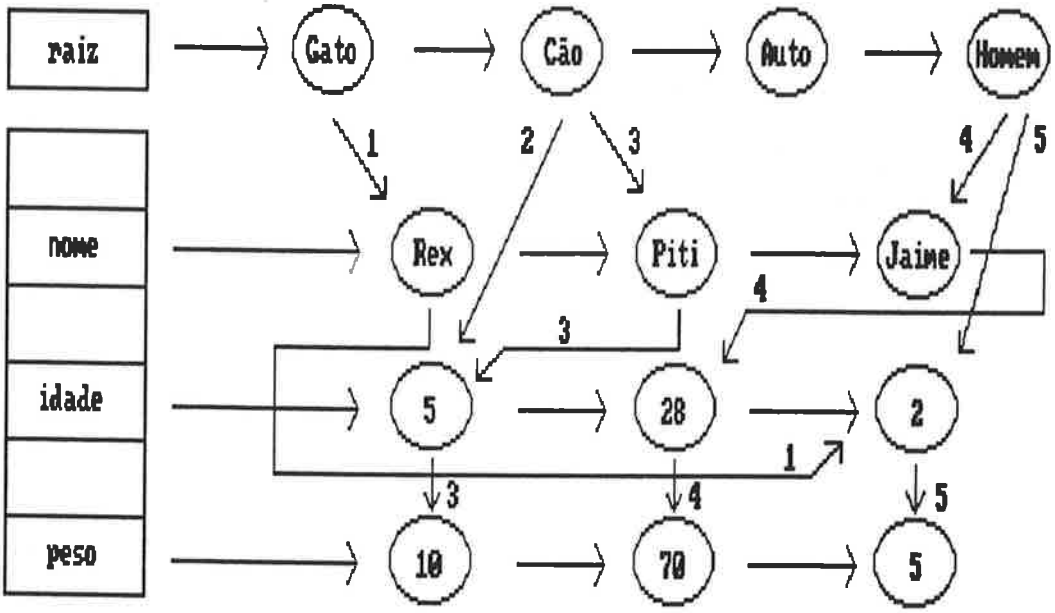

Figura 4.2 - Estrutura da Base de Dados de Objetos

Quando se deseja modificar o valor ou retirar algum atributo de determinado objeto, o arco que incidia sobre o nó correspondente ao "slot" a ser alterado toma o valor zero e é criado um novo arco incidindo no novo "slot".

Com o intuito de agrupar segmentos de memória compartilhada em um único processador, todos os elementos de uma classe são agrupados na memória do processador que criou ou acessou objetos dessa classe pela primeira vez. A localização física dos objetos é transparente aos agentes, que enxergam a memória compartilhada descentralizada como uma extensão de sua própria memória. Do ponto de vista das interfaces, tudo se passa como se

existisse apenas um Gerenciador de Transações, com uma memória compartilhada centralizada.

O mecanismo de controle de acesso aos objetos é baseado em transações [4].

4.4 Gerenciador de Transações

Os módulos que compõem o Gerenciador de Transações encontram-se representados na figura 4.3. Tais módulos são os responsáveis pelas funções de controle e comunicação entre os agentes e pelo acesso à memória compartilhada. Fornecem todo o suporte necessário aos agentes, de modo que todo o processamento distribuído fique transparente ao usuário.

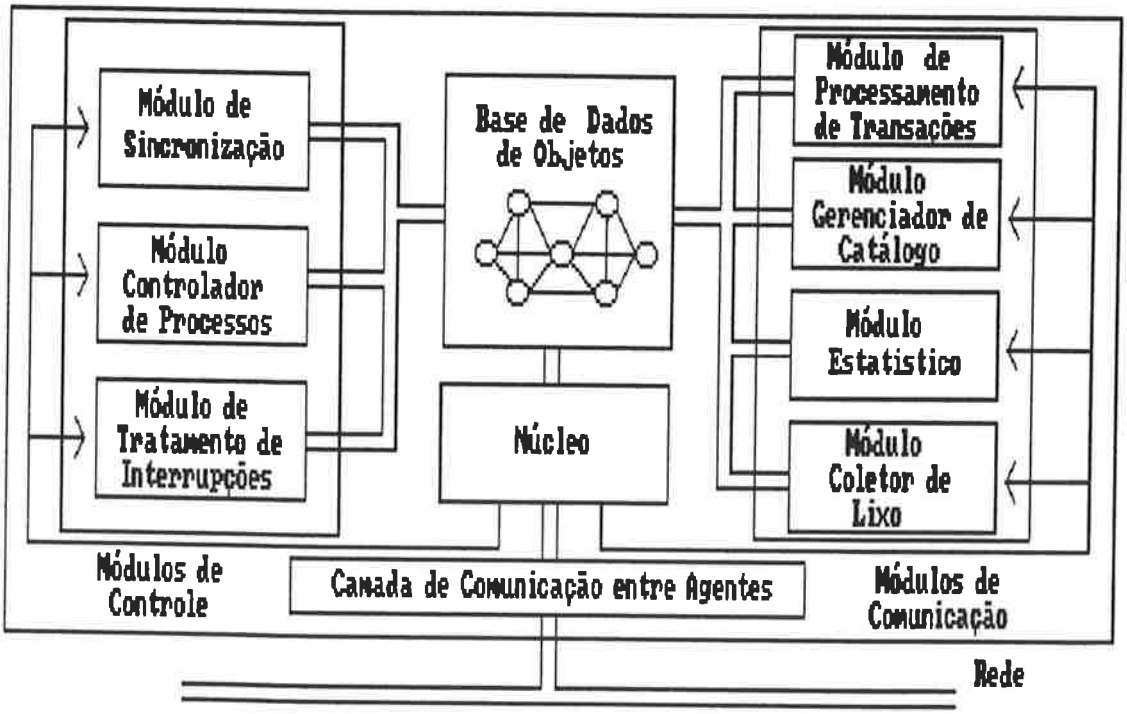

Figura 4.3 - O Gerenciador de Transações

Todas as primitivas do sistema invocadas pelos agentes se comunicam com o núcleo do Gerenciador de

Transações. Tal núcleo têm um endereço para comunicação em cada processador em que esteja instalado e suas atividades são as seguintes:

- tomar ciência do estado de todas as transações correntes no seu processador, aceitando, colocando em uma fila de espera, finalizando ou interrompendo estas transações. A execução efetiva das operações que compõem uma transação é realizada pelo Módulo Processador de Transações, sendo mantido um histórico das transações ocorridas no processador local no Módulo Estatístico;

- transferir transações para outros processadores, quando a classe dos objetos referidos pela transação estiver armazenada em outro processador. Esta operação é realizada através de uma consulta ao Módulo Gerenciador de Catálogo, que mantém uma tabela contendo o processador onde se encontram armazenados os objetos de cada classe;

- ativar periodicamente o Módulo Coletor de Lixo para remover os arcos (associados ao valor 0) e nós (aqueles cuja totalidade dos arcos incidentes tenham o valor 0) não mais utilizados;

- executar as ações de controle endereçadas aos agentes que estão localizados em seu processador, através da ativação dos Módulos de Sincronização, Controlador de Processo e Tratamento de Interrupções;

- controlar a execução de seus módulos.

4.5. Primitivas do Núcleo

O núcleo é composto por doze primitivas, que podem atualmente ser ativadas por cinco linguagens de programação: C, FORTRAN-77, FRANZ LISP, COMMON LISP e OPS5, permitindo que agentes codificados nestas linguagens possam ser integrados em um ambiente distribuído. Por razões de transportabilidade, o núcleo foi escrito na linguagem C (aproximadamente 7500 linhas de código), rodando em ambientes UNIX 4.2/4.3 em processadores genéricos, não customizados.

Existem quatro primitivas de comunicação e oito primitivas de controle entre agentes.

4.5.1 Primitivas de Comunicação

As primitivas de comunicação são as seguintes:

- Begin-Transaction (Class, Mode) : inicia o acesso aos objetos localizados na memória compartilhada designados por Class. Os modos possíveis, especificados em Mode, são leitura ou escrita. Vários agentes podem ter acesso de leitura a uma determinada classe ao mesmo tempo, porém somente um pode ter acesso de escrita em determinado instante. O valor de retorno é um identificador de transação, que é utilizado pelas outras primitivas para designar a sessão de acesso;

- Op-Transaction (Transaction_ID, Type, Pattern): realiza o acesso à memória compartilhada. Os tipos de acesso, especificados em Type, permitem criar, ler, atualizar ou retirar objetos. Os acessos são feitos a todos os objetos da classe definida pela transação, que

satisfaçam o Pattern fornecido através de pares (atributo valor);

- Abort-Transaction (Transaction_ID): interrompe uma transação corrente;

- End-Transaction (Transaction_ID): termina uma sessão de acesso à base de dados. Somente após a execução desta primitiva é que os objetos são efetivamente atualizados na memória compartilhada.

4.5.2 Primitivas de Controle

As primitivas de controle podem ser divididas em primitivas de comando, de sincronização e de tratamento de interrupções.

4.5.2.1 Primitiva de Comando

A primitiva de comando é a seguinte:

- Proc-Control (Agent, Action, Processor): permite que um determinado agente possa controlar outro, especificado em Agent e localizado em qualquer processador, especificado em Processor. As ações, fornecidas em Action, correspondem a executar, suspender, retomar ou finalizar o processamento do agente.

4.5.2.2 Primitivas de Sincronização

As primitivas de sincronização são as seguintes:

- Affirm-Event (Event): insere o evento Event na memória compartilhada;

- Check-Event (Event): checa se o evento Event já se encontra na memória compartilhada;

- Wait-Event (Event, Timeout): aguarda que o evento Event seja inserido na memória compartilhada até o limite especificado em Timeout;

- Negate-Event (Event): retira o evento Event da memória compartilhada.

4.5.2.3 Primitivas de Tratamento de Interrupções

As primitivas de tratamento de interrupções são as seguintes:

- Set-Group (Group): associa o agente ao grupo designado por Group;

- Set-Handler (Handler): associa a rotina Handler como sendo a rotina de tratamento de interrupção do agente;

- Sig-Group (Signal, Group): envia uma interrupção, identificada por Signal, ao grupo de agentes designados por Group.

4.6 Implementação de Relações Laterais

Devido à flexibilidade do mecanismo de memória compartilhada adotado no DPSK, aliado às primitivas de sincronização e tratamento de interrupções, a implementação de relações laterais, conforme discutido em 3.2.3, é extremamente facilitada. A título de exemplo, a criação de relações laterais do tipo contato direto, elementos de ligação, forças de trabalho e times seria feita da seguinte maneira:

- contato direto: definição de uma classe na memória compartilhada ao qual teriam acesso vários agentes;

- elementos de ligação: criação de um processo que pudesse acessar a base de dados de objetos, armazenando as informações referentes a uma classe que fosse acessível ao agente que representa ou ativando uma função que pudesse acessar a base de dados e retornar alguma informação sobre ela;

- forças de trabalho: criação, através de objetos pertencentes à memória compartilhada, de um mecanismo de comunicação entre as rotinas de tratamento de interrupções. Eventos poderiam ser usados para efeito de sincronização, inclusive para finalizar conjuntamente as diferentes rotinas de tratamento de interrupções;

- times: criação de um objeto comum na memória compartilhada para comunicação entre diferentes agentes, que acessam diversos objetos na base de dados.

4.7 Aplicações Desenvolvidas

Um sistema especialista distribuído para diagnose de falhas em sistemas de potência, denominado D2, foi desenvolvido utilizando o DSPK [48]. Tal sistema é composto por oito módulos, escritos nas linguagens C, COMMON LISP e OPS5, rodando em uma rede de quatro processadores do tipo VAX-STATION-II.

No Brasil, um outro sistema especialista distribuído para auxílio à operação de um processo industrial também

foi desenvolvido utilizando o DPSK [49]. Os módulos foram codificados nas linguagens C (15000 linhas) e OPS5 (650 regras), rodando em uma rede de três processadores do tipo IBM PC-XT e PC-AT compatíveis.

4.8 Limitações

O sistema DPSK apresenta as seguintes limitações:

- o uso da memória compartilhada é caro, sendo que somente dados essenciais devem ser nela armazenados;
- números em ponto flutuante perdem a precisão quando transmitidos em código ASCII de um processador a outro e quando certas conversões de linguagem pelas interfaces são executadas;
- a organização da memória compartilhada em objetos é linear, não sendo permitida a criação de hierarquias;
- cada transação só pode ser executada para manipular uma única classe de objetos.

4.9 Trabalhos Correlatos

Diversos trabalhos com objetivos semelhantes aos do sistema DPSK podem ser encontrados na literatura. Serão destacados três: os sistemas CAMELOT [50] [51], AGORA [52] e LINDA [41] [42].

O objetivo principal do sistema CAMELOT [50] [51] é o de facilitar a construção de aplicações distribuídas que sejam confiáveis, através da utilização de objetos compartilhados. Para tal, provê mecanismos de suporte a transações distribuídas e interfaces apropriadas para elas. É compatível com sistemas operacionais do tipo UNIX e com protocolos de rede do tipo ARPANET, sendo nesse

sentido independente de máquina. Oferece funções poderosas para manipular diferentes tipos de dados que podem ser compartilhados, sendo também cuidadosamente documentado. Do mesmo modo que o DPSK, adota um modelo cliente/servidor para gerenciar os objetos compartilhados. Provê também gerenciadores para áreas de disco e políticas de recuperação de dados, em caso de queda do sistema. Do ponto de vista lógico, o modelo de arquitetura do sistema CAMELOT é um modelo em camadas, contendo cinco níveis distintos: sistema operacional, camada de comunicação entre agentes (rede), camada de transações (onde as primitivas do sistema encontram-se disponíveis), camada de objetos compartilhados e camada da aplicação.

O ambiente AGORA [52] tem como principais objetivos ser capaz de executar de modo eficiente diferentes modelos computacionais, evitando restringir as possíveis implementações a arquiteturas específicas. Dentro deste escopo, pode ser então considerado como um núcleo para resolução distribuída de problemas, como o DPSK. Seu modelo foi fortemente influenciado pelo paradigma "Blackboard", existindo fontes de conhecimento que manipulam segmentos na memória compartilhada. Tal influência pode ser explicada pelo fato de que o núcleo foi originalmente desenvolvido para dar suporte a um sistema de compreensão da fala humana, denominado ANGEL, sofrendo assim influências do sistema HEARSAY-II [3]. Uma diferença em relação ao DPSK é que objetos , uma vez inseridos, não podem mais serem retirados da memória compartilhada. Apesar do custo associado a tal política, esta possibilita esquemas mais simples de manutenção de consistência dos objetos compartilhados. O modelo de arquitetura adotado contém quatro camadas: hardware, sistema operacional (MACH), máquina paralela virtual (onde as primitivas do sistema encontram-se disponíveis) e um quarto nível, onde um ambiente especializado poderia

ser criado, de modo a possibilitar ao usuário interagir com o ambiente utilizando termos familiares de seu domínio, que seriam traduzidos para as primitivas do ambiente.

O sistema LINDA [41] [42] consiste num conjunto de operadores para facilitar a construção de programas paralelos. A comunicação entre os diversos agentes se dá através de um espaço de dados compartilhados denominado espaço de tuplas. Tuplas podem ser inseridas, removidas e alteradas pelos diversos agentes, sendo que cada tupla é identificada e acessada por mecanismos de "pattern matching". O sistema DPSK também utiliza um mecanismo de "pattern matching" para acesso a seus objetos compartilhados. Um protocolo de rede, denominado S/NET, é utilizado para possibilitar a distribuição. Através dessas tuplas, modelos baseados em troca de mensagens e em memória compartilhada podem ser implementados. Uma versão denominada KERNEL LINDA foi desenvolvida para ser utilizada como mecanismo de comunicação entre processos em um sistema operacional paralelo do tipo UNIX, denominado QIX.

5 PARADIGMA DE OBJETOS

5.1 Histórico

O chamado paradigma de objetos começou a ser estudado com maior atenção a partir dos princípios da década de 80, devido à chamada "Crise do Software". Tal crise seria causada pelo grande descompasso existente entre os progressos obtidos nos últimos anos pelas áreas de hardware e software. Se por um lado, máquinas cada vez mais poderosas estariam sendo projetadas e incorporadas ao mercado, por outro não estavam sendo concomitantemente desenvolvidas metodologias que pudessem tornar a produção de software mais rápida e eficiente. Ao mesmo tempo, novas aplicações estariam surgindo, que requereriam características de processamento atualmente difíceis de serem obtidas por métodos convencionais de produção de software, como por exemplo suporte à evolução de sistemas, à execução paralelizada e reconfigurável e à utilização de plataformas heterogêneas, compostas por máquinas distintas ligadas em rede. Desta forma, a demanda pela produção de software de alta complexidade cresceria de forma gigantesca nas próximas décadas.

O paradigma de objetos surgiu então como uma tentativa de obter um processo de produção de software de grande escala, alta complexidade, alto grau de desempenho e alta confiabilidade que pudesse incorporar as características citadas acima. Para tal, o desenvolvimento de tal paradigma incorporou conceitos de outras áreas da computação, sendo que os mais básicos foram importados das áreas de Linguagens de Programação, de Bases de Dados e de Inteligência Artificial.

O conceito importado da área de Linguagens de Programação foi o de tipo abstrato de dados. Analisando-se a evolução das linguagens procedurais, nota-se claramente que a ênfase do processo de desenvolvimento de

software foi migrando de uma orientação pura a processo para uma orientação mista a processo e dados [53]. Inicialmente, o foco de atenção do projetista estava dirigido ao projeto de procedimentos que combinados gerariam o algoritmo desejado para solucionar o problema proposto. À medida em que o tamanho dos programas crescia, começa a haver uma preocupação com a organização dos dados, surgindo o conceito de módulo, que encapsula os dados acessados com seus respectivos procedimentos de acesso. Tal política levou à centralização de determinados tipos de dados, como pilhas, filas e listas ligadas, em módulos específicos. Sendo assim, um outro módulo só poderia acessar tais estruturas exclusivamente através da ativação destes procedimentos, não tendo conhecimento da forma como tais dados estariam internamente representados nos respectivos módulos. Tal mecanismo denominou-se tipo abstrato de dados.

A área de Bases de Dados contribuiu com o conceito de modelagem conceitual. A idéia básica é a de capturar, através de um mecanismo de abstração, conforme mostrado na figura 1.1, as entidades do mundo real que devem estar representadas na base de dados, independentemente do modelo lógico adotado na construção de tal base. O projeto de uma base de dados se daria então em três fases [54]. A primeira fase, denominada projeto conceitual, teria como resultado um modelo de dados conceitual, construído a partir do ponto de vista dos usuários e da aplicação, onde não haveria a preocupação com o modo pelo qual os dados seriam armazenados e processados no computador. A seguir, seria realizada uma fase de projeto lógico, onde o modelo conceitual gerado na fase anterior seria traduzido para o modelo de armazenamento correspondente ao gerenciador da base de dados (relacional ou hierárquico, entre outros) e adaptado aos aspectos de processamento (tempos de resposta, prioridades), resultando num esquema lógico. Finalmente,

a fase de projeto físico realizaria a adaptação do esquema lógico ao computador, levando em conta as características e limitações específicas da máquina.

O mecanismo de modelagem conceitual também está intimamente ligado a um desenvolvimento orientado a dados e não a processos. Quando se analisa um sistema de software, observa-se que as interfaces entre os diversos procedimentos é altamente sujeita a modificações no decurso da atividade de projeto, enquanto que as entidades de dados abstraídas do mundo real são dificilmente modificadas, desde que a modelagem tenha sido fiel ao que se desejava inicialmente representar.

Tipicamente, as operações abstratas que são utilizadas num processo de modelagem conceitual são as seguintes [1]:

- classificação/instanciação : trata-se da associação de fenômenos em categorias, baseadas na existência de propriedades comuns. No exemplo apresentado na figura 5.1, João e Maria são instâncias da categoria Estudante de Graduação;

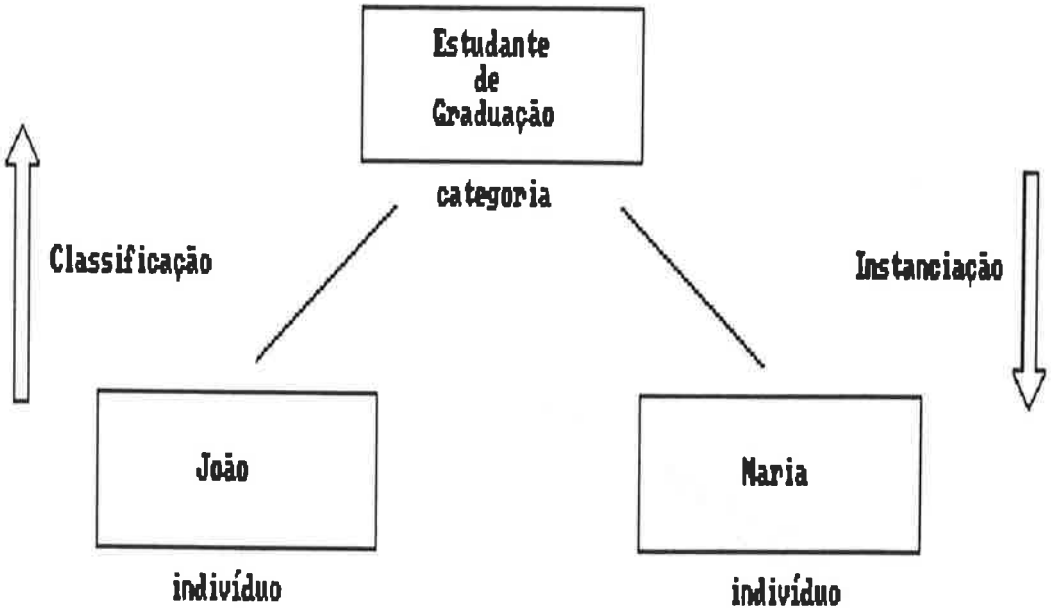

Figura 5.1 - Mecanismos de Abstração: Classificação e Instanciação

- generalização/especialização : trata-se da associação de categorias mais específicas com uma categoria mais genérica. No caso do exemplo apresentado na figura 5.2, as categorias Estudante de Graduação e Estudante de Pós-Graduação são especializações da categoria Estudante;

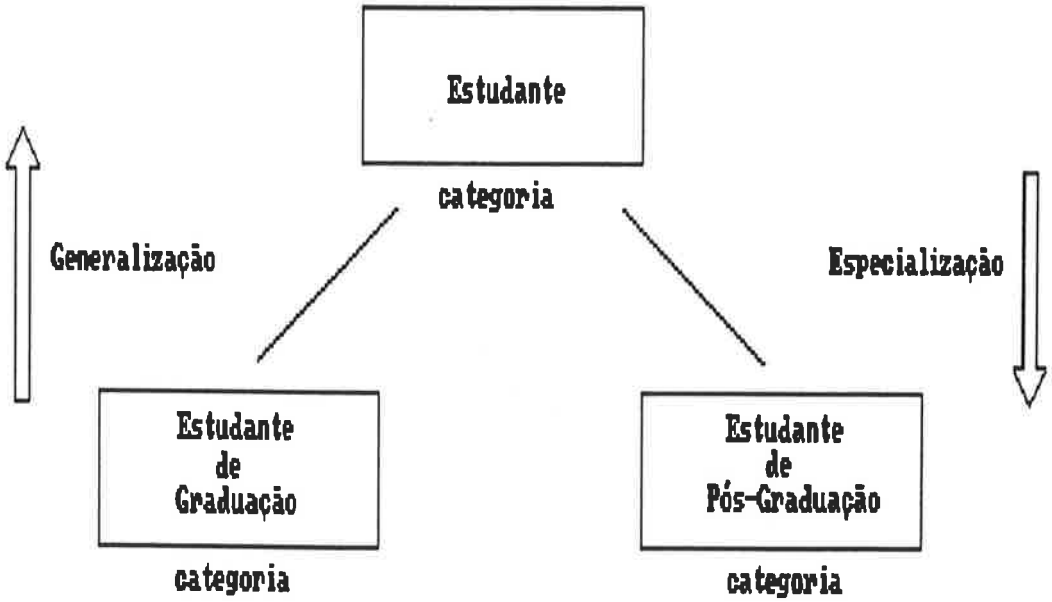

Figura 5.2 - Mecanismos de Abstração: Generalização e Especialização

- agregação/decomposição : trata-se da identificação de determinadas instâncias de uma categoria que são compostas por instâncias de outras categorias. A instância da categoria Auxiliar de Ensino, apresentada na figura 5.3, pode ser decomposta em instâncias das categorias Estudante de Pós-Graduação e Professor;

Figura 5.3 - Mecanismos de Abstração: Agregação e Decomposição

A idéia de agregação e decomposição é também muitas vezes utilizada em processos de análise e síntese de programas. Um processo de análise é composto por refinamentos sucessivos, e portanto utiliza mecanismos de decomposição, enquanto que um processo de síntese utiliza mecanismos inversos, ou seja, agregação.

A contribuição da área de Inteligência Artificial para o paradigma de objetos é uma estrutura de representação de conhecimento denominada "frame" [30]. "Frames" são, em essência, objetos estruturados, que são agrupados em hierarquias de generalização/especialização, sendo que cada "frame" é composto por um nome, um conjunto de valores, armazenados em "slots" do tipo par atributo valor e por um conjunto de ações, denominadas "demons", que podem ser disparadas quando um valor de determinado "slot" é preenchido ou modificado. Um mecanismo básico utilizado em "frames" é a herança de propriedades : um "frame" localizado em níveis inferiores

da hierarquia herda todas as propriedades (valores de "slots" e "demons") de níveis superiores da hierarquia aos quais esteja direta ou indiretamente ligado. Em aplicações mais complexas, além dos "demons", os "slots" podem conter outras informações além do valor propriamente dito, como por exemplo valores "default", domínio e cardinalidade dos valores, entre outros.

A seguir, são apresentados os conceitos básicos do paradigma de objetos [1] [55].

5.2 Conceitos Básicos

5.2.1 Objetos

Objetos são unidades conceituais, resultantes de um processo de abstração do mundo real, compostos por um estado interno, onde valores podem ser armazenados e modificados ao longo da vida do objeto, e por um comportamento, onde são definidas ações (denominadas métodos) através das quais o objeto responderá à demanda de processamento por parte de outros objetos.

No paradigma de objetos puro, a rigor somente existem objetos. Outras construções usuais das linguagens procedurais não existem, como por exemplo variáveis e comandos.

5.2.2 Mensagens e Métodos

Os objetos se comunicam entre si através do envio e recebimento de mensagens. Mensagens são compostas por um seletor e por parâmetros. Um objeto, ao receber uma mensagem, realiza um acoplamento entre o seletor da mensagem e algum de seus métodos. Uma vez selecionado, o método é então executado e o resultado da computação é enviado ao objeto que enviou a mensagem, assim como a passagem do controle.

Existem basicamente 2 maneiras de realizar tal acoplamento : em tempo de compilação ou em tempo de execução ("late binding"). O que deve ser ressaltado é que o resultado do envio de uma mensagem pode ser muito diferente, caso esta seja enviada para objetos distintos. Tal propriedade, no paradigma de objetos, é denominada polimorfismo.

5.2.3 Classes

Utilizando um mecanismo de abstração do tipo classificação, objetos cuja estrutura e comportamento sejam idênticos são agrupados em classes. Tal mecanismo permite que a descrição das propriedades e do comportamento de uma classe de objetos seja realizada apenas uma única vez, independentemente do número de instâncias dessa classe que possam existir na aplicação.

O mecanismo de definição de classes, por sua vez, pode ser implementado segundo duas políticas distintas. Numa primeira política, classes seriam representadas como um novo tipo de dados da linguagem hospedeira. Numa outra abordagem, mais complexa, classes também seriam objetos, contendo um estado e um conjunto de métodos que descrevem o seu comportamento. Em particular, existiriam dois métodos destinados a criar e destruir instâncias de objetos da classe. Assim, por exemplo, caso um objeto deseje criar uma instância da classe Estudante, ele deverá enviar ao objeto que representa a classe Estudante uma mensagem cujo seletor seja o nome do método que cria instâncias dessa classe. Além disso, neste segundo caso, torna-se possível criar um nível de abstração adicional denominado metaclasse, onde o comportamento do objeto classe pode ser detalhado. Tal comportamento pode descrever, por exemplo, quais métodos são automaticamente herdados e quais variáveis de instância são inicializadas quando for criada uma nova instância desta classe. Trata-se de um mecanismo extremamente poderoso, uma vez que

permite definir diferentes comportamentos para hierarquias de classe distintas, de modo a implementar diferentes modelos de orientação a objetos.

5.2.4 Hierarquia de Classes e Herança

Classes são inseridas em uma hierarquia de especialização, sendo que uma classe mais específica herda todas as propriedades da classe localizada imediatamente acima na hierarquia. No exemplo mostrado na figura 5.2, a classe Estudante é denominada superclasse das classes Estudante de Graduação e Estudante de Pós-Graduação, sendo que estas são denominadas subclasses da classe Estudante.

Caso uma classe mais específica deva ter um comportamento distinto de sua superclasse, é possível cancelar a herança através da redefinição do método correspondente. Assim sendo, ainda se referindo à figura 5.2, caso a classe Estudante tenha um método denominado Média que deva ser diferenciado na classe Estudante de Pós-Graduação, basta criar um novo método denominado Média nesta última classe. Esta operação, no paradigma de objetos, é denominada de sobrecarga de operadores.

Tal ação resulta frutífera em virtude do modo em que é realizado o acoplamento mensagem/método em uma hierarquia. Quando um objeto recebe uma mensagem, ele tenta executar o método que corresponda ao seletor enviado. Caso não exista tal método, então o método é procurado na sua superclasse e assim por diante, até que seja atingida a raiz da hierarquia de classes; se ainda assim não puder ser realizado o acoplamento, é então gerada uma mensagem de erro.

Um mecanismo denominado herança múltipla permite que uma determinada classe tenha mais de uma superclasse direta. Na figura 5.3, a classe Auxiliar de Ensino tem 2

superclasses: Estudante de Pós-Graduação e Professor. Cabe ressaltar ainda que nem todos os sistemas baseados no paradigma de objetos suportam herança múltipla; isto quer dizer que é impossível, nestes casos, utilizar os mecanismos de abstração do tipo agregação/decomposição.

5.2.5 Um Programa Segundo o Paradigma de Objetos

De acordo com os conceitos apresentados acima, construir um programa segundo o paradigma de objetos significa [1]:

- definir classes para modelar os conceitos da aplicação, utilizando mecanismos de abstração;
- agrupar estas classes em uma hierarquia do tipo generalização/especialização;
- disparar uma mensagem para uma classe que fará o papel de programa principal e iniciar a execução do programa;
- durante a execução, objetos serão instanciados a partir de classes, objetos enviarão mensagens a outros objetos, e as respostas dos objetos às mensagens recebidas comporão a ação propriamente dita do programa.

5.3 Linguagens e Ambientes

Na seqüência, serão apresentadas as características principais de algumas linguagens e ambientes que suportam o paradigma de objetos. SMALLTALK-80 segue o paradigma puro de objetos, enquanto que C++ e CLOS são extensões respectivamente dos paradigmas procedural e funcional que suportam objetos. O objetivo destas descrições não é o de fornecer descrições detalhadas de tais linguagens e

ambientes, e sim ressaltar os aspectos que as diferenciam.

5.3.1 SMALLTALK-80

SMALLTALK-80 [56] foi desenvolvido por um grupo de pesquisadores da Xerox Palo Alto Research Center, como extensão dos sistemas SMALLTALK-72, SMALLTALK-74 e SMALLTALK-76. Algumas de suas idéias fundamentais provêm da linguagem SIMULA. Trata-se não propriamente de uma linguagem, mas sim de um ambiente de programação orientado a objetos, contemplando uma poderosa interface homem-máquina, ferramentas tais como editores e compiladores e uma biblioteca de classes pré-definidas, onde constam as classes primitivas do ambiente, a partir das quais podem ser criadas novas classes.

A rigor, qualquer ação computacional em SMALLTALK-80 se resume a objetos e mensagens. Não existem comandos usuais de linguagens de programação. Assim sendo, por exemplo a operação 8 + 5 pode ser entendida como o envio de uma mensagem cujo seletor é + ao objeto 8, tendo como parâmetro o objeto 5. Ao receber tal mensagem, o objeto 8 acopla a mensagem ao seu método correspondente, realiza a computação, e retorna o objeto 13 como resposta.

O padrão SMALLTALK-80 não suporta herança múltipla, ou seja, cada classe tem somente uma única superclasse. Existe uma classe, denominada **Object**, que se encontra no topo da hierarquia de classes acessíveis ao usuário comum. Além disso, classes são objetos, existindo portanto uma hierarquia complementar de metaclasses, onde encontram-se definidos os protocolos de uso da classe definida, ainda que com poucos recursos.

Não existe um meio para destruir objetos explicitamente em SMALLTALK-80; isso é feito automaticamente por um coletor de lixo que, de tempos em

tempos, retira da memória os objetos que não foram mais usados.

Os objetos podem ter atributos internos, que representam seu estado atual. Atributos de instância não são visíveis externamente, enquanto que atributos de classe são visíveis para as instâncias diretas e indiretas da classe respectiva.

O acoplamento mensagem método é realizado em tempo de execução.

5.3.2 C++

A linguagem C++ [57] é uma extensão da linguagem C, que foi projetada para melhorar alguns aspectos desta (checagem de tipos, funções com número variável de argumentos, entre outros), suportar tipos abstratos de dados e suportar programação orientada a objetos. Um programa em C++, assim como um programa em C, é um conjunto de funções, agrupadas em diferentes arquivos, contendo uma função especial denominada main () que é a primeira a ser ativada quando o programa é executado.

Nas primeiras versões de C++, o processo de compilação era dividido em duas fases: uma pré-compilação de C++ para C, seguida de uma compilação do código em C gerado no passo anterior. Em versões mais recentes, o processo é realizado em um único passo.

Classes são definidas através de um tipo especial denominado <u>class</u>, onde se especificam os atributos e métodos que as compõem. Tanto os atributos como os métodos de uma classe podem ter três diferentes níveis de visibilidade: privativo, protegido e público. No nível privativo, somente podem ser acessados os métodos e os atributos pelos métodos da própria classe e por outras classes ou funções especiais, denominadas amigas de uma classe. Tais classes e funções amigas de uma classe são

um meio de privilegiá-las quanto aos direitos de acesso, contornando a rigidez imposta pela hierarquia de classes. No nível protegido, a permissão de acesso é estendida às subclasses da classe em questão. No nível público, encontram-se definidos os protocolos de acesso às instâncias da classe.

A definição de uma classe em C++ se dá em duas partes: uma fase de especificação e outra de implementação, geralmente dispostas em arquivos distintos. Na especificação de uma classe, são definidas as superclasses, atributos e os protótipos dos métodos, onde definem-se suas interfaces de acesso (parâmetros de entrada e saída) e seus graus de visibilidade. Na implementação de uma classe, encontram-se codificados os métodos que a compõem.

Não existe uma hierarquia de classes pré-definida em C++, podendo entretanto existirem arquivos em bibliotecas contendo classes já desenvolvidas pelo usuário.

C++ suporta herança múltipla.

Existem sempre pelo menos dois métodos em cada classe, um construtor e um destrutor de instâncias. Não existe um mecanismo automático de coleta de lixo, sendo que instâncias que não são mais utilizadas devem ser explicitamente destruídas.

O acoplamento mensagem método é realizado em tempo de compilação. Tal mecanismo é realizado indiretamente, através do uso de uma tabela de endereços. Entretanto, existe um meio de realizar acoplamento em tempo de execução, através de métodos chamados virtuais. Métodos virtuais servem para indicar que o tratamento será posteriormente especializado nas subclasses que serão definidas a partir da classe em questão. Neste caso, é

mantida uma entrada vazia na tabela de endereços que é preenchida em tempo de execução.

Uma vez que classes são definidas por um tipo especial e não são objetos, não existem metaclasses em C++.

5.3.3 CLOS

A linguagem CLOS [58] [59] é uma extensão da linguagem COMMON LISP, que suporta o paradigma de objetos. Seu desenvolvimento foi fortemente influenciado pelo paradigma funcional. Os conceitos de objetos, classes e instâncias são mantidos tais como nos exemplos anteriores, mas não existe o conceito de mensagem no modelo de objetos adotado pela linguagem CLOS.

Os métodos não são agregados a classes e sim a funções especiais denominadas funções genéricas. Uma chamada de função genérica é semelhante a uma chamada de função usual em COMMON LISP, sendo que objetos são passados como argumentos da função. Tais funções genéricas têm o efeito de dissociar os objetos das operações realizadas sobre os mesmos.

Existem primitivas para definição de classes, criação de instâncias e para definição de métodos. Os atributos são armazenadas em "slots" do tipo par atributo valor. Tais atributos não são protegidos: podem ser acessados através de métodos acoplados a funções genéricas ou através de quaisquer outras funções, se assim desejar o usuário.

Assim como em SMALLTALK-80, existe uma hierarquia de classes pré-definidas. A classe t encontra-se no topo da hierarquia de classes manipuláveis pelo usuário, e sua função é semelhante à da classe Object, em SMALLTALK-80.

Existem dois tipos de métodos em CLOS: métodos primários e métodos auxiliares. Métodos primários são métodos sem qualificadores. Métodos auxiliares são identificados por um qualificador, que pode ser do tipo before, after e around. A idéia básica é a de especializar os métodos primários definidos na superclasse através da ativação destes métodos auxiliares nas subclasses. Na ativação de uma função genérica, é avaliado e executado um método efetivo, que é composto por um método primário, podendo ser acrescido de um ou mais métodos auxiliares. Assim sendo, por exemplo, caso um método pertencente a uma subclasse necessite ser especializado, com uma ação posterior à ação realizada pelo método primário pertencente à sua superclasse, pode-se definir um método auxiliar de qualificação after que será ativado após a execução do método primário.

Classes são objetos em CLOS, existindo portanto um mecanismo de metaclasses, extremamente poderoso, que permite implementar modelos distintos de orientação a objetos.

5.4 Aplicações do Paradigma de Objetos

Uma vasta gama de aplicações tem utilizado o paradigma de objetos em seu desenvolvimento, dentre as quais cabe ressaltar aqui aquelas destinadas à resolução distribuída de problemas. Tipicamente, aplicações com atividades paralelas, interativas, mal estruturadas, atuando em domínios abertos têm sido modeladas utilizando o paradigma "Blackboard", manipulando objetos na memória compartilhada. Exemplos de tais aplicações encontram-se nas áreas de automação de escritórios e de ambientes de desenvolvimento de software, onde pode ser citado o Projeto ETHOS [1], atualmente sendo desenvolvido em várias universidades brasileiras. Tal projeto tem como objetivo o desenvolvimento de um meta-ambiente de desenvolvimento de software. A arquitetura global do

meta-ambiente segue o modelo cliente-servidor, com comunicação oportunista entre as ferramentas, ou seja, tais ferramentas são integradas por um "Blackboard" de controle. Cada ferramenta, por sua vez, pode também ser constituída por um conjunto de sub-ferramentas, integradas por um "Blackboard" local. O estudo de definição de tal meta-ambiente comporta aspectos de reutilização, replicação e migração de objetos, entre outros.

Outra aplicação do paradigma de objetos em resolução distribuída de problemas, desta vez na área de automação industrial, é o sistema DOSBART (Distributed Operating-System Based Blackboard Architecture for Real-Time Control) [60]. Trata-se de um núcleo para resolução distribuída de problemas, destinado a aplicações de controle distribuído em tempo real, desenvolvido em CLOS. Utiliza uma arquitetura baseada em "Blackboards", com mecanismos "inteligentes" de monitoração de estados, que utilizam inferência. Provê paralelismo efetivo entre as diversas tarefas, através de um núcleo para processamento distribuído de objetos.

Em outras aplicações de Inteligência Artificial, objetos também têm sido utilizados para, por exemplo, construir interfaces inteligentes e ambientes voltados a ícones [61]. Como aplicações desenvolvidas podem ser citados os sistemas STARPLAN-II, um sistema especialista para diagnose e reparo de satélites, desenvolvido pela Ford Aerospace, e o sistema PRIDE, para auxílio ao projeto de transportadoras mecânicas de papéis em máquinas copiadoras, desenvolvido pela Xerox. Na área de ferramentas, pode-se citar uma voltada ao processamento de conhecimento, GOLDWORKS-II, desenvolvida pela Golden-Hill Computers. Finalmente, ACTORS, desenvolvido no MIT, é um dos primeiros sistemas a dar suporte à concorrência de objetos.

O interesse no paradigma de objetos existe também em outras áreas que não aquelas que tradicionalmente voltam seus esforços para novos modelos computacionais. Na França, particularmente em Toulouse, um grupo de pesquisadores estuda um formalismo para o projeto de software de tempo real, no qual a estrutura da aplicação seria modelada segundo o paradigma de objetos e o controle dinâmico através de Redes de Petri [62].

Finalmente, cabe ressaltar que o paradigma de objetos tem sido também alvo de estudos na área de Engenharia de Software, onde novos modelos de desenvolvimento e de ciclo de vida estão sendo criados, com o intuito de facilitar o desenvolvimento e aumentar o nível de desempenho dos sistemas baseados neste paradigma [63].

6 O SISTEMA DPSK+P

6.1 Objetivos

O paradigma de objetos, descrito no capítulo 5, apresenta naturalmente um enfoque distribuído. A independência e forte encapsulamento de dados e métodos em um dado objeto somente vem reforçar tal enfoque. Deste modo, tornam-se desejáveis, e até mesmo naturais, esforços no sentido de estender o alcance de tal paradigma a ambientes distribuídos. Deste modo, diversos agentes desenvolvidos segundo o paradigma poderiam compartilhar objetos, mesmo que estes tivessem sido definidos em outros agentes, sendo processados na mesma máquina ou em máquinas distintas. Uma proposta neste sentido é apresentada no sistema DCK (Distributed CLOS Kernel) [64], desenvolvida utilizando as primitivas do sistema DPSK, no sentido de prover a um interpretador CLOS um mecanismo de chamada remota de funções genéricas.

O objetivo básico do sistema DPSK+P [65] é o de incorporar algumas práticas provenientes do paradigma de objetos num núcleo para resolução distribuída de problemas. Tais práticas se referem à possibilidade de definir hierarquias de classes, métodos e mecanismos de instanciação para objetos compartilhados.

Para ser eficaz, um núcleo de resolução distribuída de problemas que se proponha a permitir a existência de agentes desenvolvidos segundo o paradigma de objetos precisa contemplar mecanismos de comunicação e controle que sigam algumas práticas de tal paradigma. Os mecanismos de comunicação do sistema DPSK+P permitem que um agente possa definir objetos que possam ser manipulados por quaisquer outros agentes. A idéia básica é a de permitir que um dado agente possa tornar certos dados e métodos compartilháveis quando desejar que sejam acessíveis por outros agentes. Para atingir tal objetivo,

os dados contidos nos diferentes objetos são divididos em três níveis: um nível privado, cujo acesso é permitido apenas ao próprio objeto, um nível público, cujo acesso é permitido aos outros objetos definidos no mesmo agente, e um nível compartilhado, cujo acesso é permitido a quaisquer outros agentes, não necessariamente localizados na mesma máquina. Os dois primeiros níveis são usuais em linguagens que suportam o paradigma, como C++ [57]. A contribuição do sistema DPSK+P reside no terceiro nível, que contempla justamente o aspecto distribuído. Neste último nível ainda, torna-se também necessário prover um mecanismo pelo qual um agente possa ativar métodos definidos em objetos mantidos por outros agentes, para manipular os dados compartilhados. Tal interação entre agentes encontra-se esquematizada na figura 6.1.

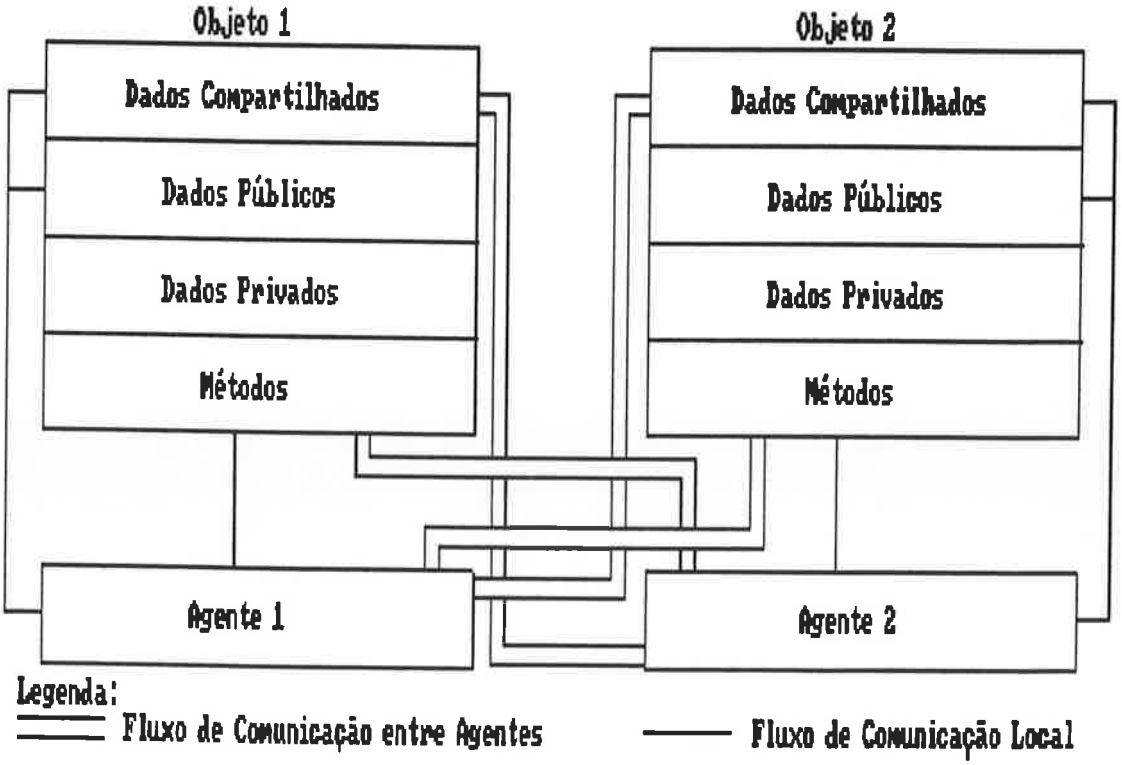

Figura 6.1 - Interação entre Agentes no Sistema DPSK+P

Para tornar mais claros os objetivos do sistema DPSK+P, seja tomada como exemplo a seguinte hierarquia de classes, mostrada na figura 6.2.

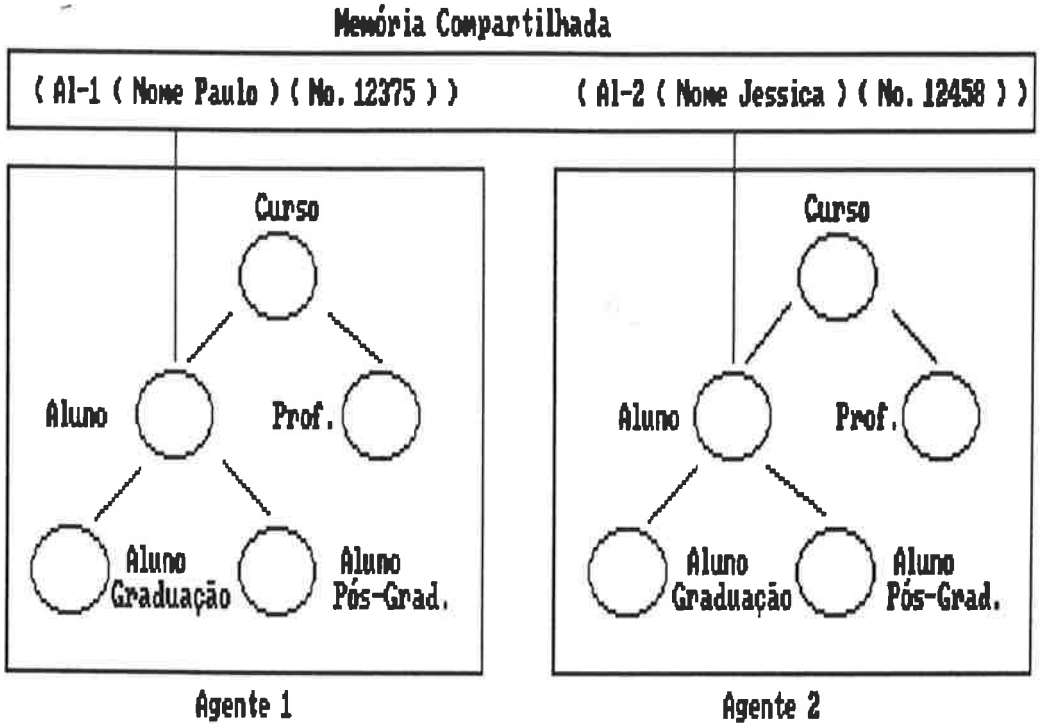

Figura 6.2 - Classes Compartilhadas no Sistema DPSK+P

Segundo a linguagem C++, os métodos e atributos do nível privado da classe Aluno só podem ser manipulados por métodos definidos na própria classe. O nível protegido estende este direito às suas subclasses, no caso Aluno de Graduação e Aluno de Pós-Graduação. Finalmente, o nível público define o protocolo de manipulação das instâncias da classe Aluno para todos os outros objetos definidos no programa. Tais direitos, entretanto, se restringem a um único agente ou programa. O que o sistema DPSK+P se propõe é estender este direito a outros agentes, ou seja, possibilitar, por exemplo, que atributos da classe Aluno definida no agente 1 possam ser acessados por objetos pertencentes à classe Aluno

definida no agente 2 e vice-versa. Tais dados compartilhados é que consistirão o chamado nível compartilhado. De um certo modo, trata-se de estender o nível público da linguagem C++ para outros agentes, localizados ou não na mesma máquina.

Para efeito de controle, serão previamente definidas certas classes responsáveis pela criação, interrupção, bloqueio e sincronização entre processos, bem como pela manutenção de consistência dos dados compartilhados.

Tais mecanismos de comunicação e controle encontram-se detalhados a seguir.

6.2 Mecanismos de Comunicação

Os mecanismos de comunicação providos pelo sistema DPSK+P permitem que um agente possa acessar dados compartilhados e ativar métodos pertencentes a objetos definidos em outros agentes. Para tal, torna-se necessária a criação de mecanismos para definir classes, instanciar e acessar objetos, bem como para definir e invocar métodos.

6.2.1 Classes e Objetos Compartilhados

Os objetos compartilhados são criados através do método MakeInstance. Uma vez fornecida a classe à qual pertence e, opcionalmente, um conjunto adicional de "slots", tal método retorna um identificador para o objeto em questão.

Classes, por sua vez, são criadas de modo análogo, através da instanciação da metaclasse **RootClass**. Os "slots" passados em MakeInstance, juntamente com aqueles herdados das superclasses, compõem a definição da classe sendo criada. Uma vez criadas, classes não podem mais ser removidas.

É interessante notar que apenas os dados compartilhados são mantidos e gerenciados pelo núcleo. Os dados privados e públicos bem como os métodos de cada objeto ficam residentes no agente que o criou.

O procedimento de leitura dos objetos compartilhados se dá através de dois passos: seleção e acesso a tais objetos. A seleção pode ser feita através de métodos especialmente criados para tal função, que modificam um ou mais "slots" desses objetos. O acesso é feito através de um mecanismo de "pattern matching", realizado pelo método GetInstance. Os parâmetros que devem ser fornecidos a tal método são a classe à qual o objeto pertence e um conjunto de "slots", especificados no passo de seleção, que devem estar presentes nas instâncias.

Operações de escrita são realizadas através da ativação dos métodos DeleteInstance, UpdateSlot e DeleteSlot.

Finalmente, cabe ressaltar que o sistema DPSK+P permite que "slots" possam ser definidos sem ser necessariamente preenchidos. Para tal, existe uma primitiva denominada MakeSlot, cujos parâmetros são o nome do atributo, o tipo de valor que pode ser armazenado (como inteiro e real, entre outros) e valor, retornando um ponteiro para o "slot".

6.2.2 Métodos Compartilhados

Os métodos utilizados para manipulação de dados contidos nos objetos são, como dito acima, localizados e gerenciados pelos próprios agentes, e não pelo núcleo. De acordo com o modelo cliente-servidor adotado no desenvolvimento do sistema, sempre que um agente desejar que outros agentes possam invocar seus métodos, ele deverá agir como um servidor. O agente que desejar invocar um método externo faz uma chamada de procedimento

normal, cabendo ao núcleo redirecioná-la a fim de que o agente no qual se encontra definido o método desejado seja notificado. Tal mecanismo é denominado na literatura de chamada remota de procedimentos.

A primitiva RegisterMethod permite que um agente anuncie um método que possa ser invocado por outros agentes. Seus parâmetros são o nome do método, a classe do parâmetro de entrada e um ponteiro para a função local que codifica o método.

Um agente que deseje invocar um método definido em outro agente deve utilizar a primitiva CallMethod, especificando o nome do método e seu parâmetro de entrada. Ao ser ativada tal primitiva, o agente fica bloqueado até que o agente no qual o método foi definido lhe devolva o resultado da execução do método em questão.

Um agente que tenha anunciado métodos, através da primitiva RegisterMethod, deve periodicamente verificar se existem pedidos de ativação externos pendentes para tais métodos. Em caso afirmativo, tais pedidos são aceitos através da primitiva AcceptMethod, que invoca o método e passa seu valor de retorno para o agente que requisitou a sua execução.

6.3 Mecanismos de Controle

Os mecanismos de controle entre agentes são implementados no sistema DPSK+P através de classes pré-definidas para manutenção de consistência dos dados compartilhados, controle, sincronização e interrupção de processos. Os métodos definidos para tais classes são ativados através da primitiva CallMethod.

6.3.1 Consistência dos Dados Compartilhados

Do mesmo modo que no sistema DPSK, o controle sobre a consistência dos dados compartilhados é realizado

através de um mecanismo de transações [4]. Uma classe denominada **Transaction** e' definida para tal, contendo os seguintes "slots":

- objects, que especifica os objetos sobre os quais se deseja realizár uma transação. Objetos podem ser classes e instâncias. No caso de classes, todas as suas subclasses podem ter seu acesso bloqueado, a critério do usuário;

- type, onde se especifica o tipo de transação, ou seja, escrita ou escrita/leitura;

- timeout, onde se especifica o tempo máximo que uma transação deve aguardar para ser completada antes de ser interrompida.

Tal classe apresenta os seguintes métodos:

- BeginTransaction, para iniciar uma transação. Tal método retorna um objeto utilizado como identificador da transação, que será utilizado pelos métodos que se seguem;

- EndTransaction, para finalizar uma transação;

- AbortTransaction, para interromper uma transação em curso. Neste caso, devido à propriedade de atomicidade, o estado dos objetos compartilhados retorna àquele em cujo instante a transação teve início.

No caso de uma tentativa de acesso a uma classe ou instância que já tiver sido bloqueada por outra transação, o método retorna um objeto nulo.

Quando o usuário desejar acessar instâncias das classes compartilhadas sem utilizar transações, basta ativar diretamente os métodos adequados. Neste caso, obviamente, o sistema não garante a consistência dos dados compartilhados.

6.3.2 Controle dos Processos

Para efeito de controle dos processos, uma classe **Process** é pré-definida, contendo os seguintes "slots":

- host, onde é especificado o computador no qual o processo deve ser executado. O nome do computador pode ser simbólico ou um endereço de rede contendo 32 bits;

- file, onde é especificado o arquivo executável a partir do qual deve ser gerado o processo;

- name, onde é especificado o nome simbólico do processo;

- pid, onde é armazenado o identificador do processo, fornecido pelo sistema operacional. Deste modo, é possível ativar vários processos simultâneos gerados a partir de um mesmo arquivo executável (clones);

- signo, onde fica armazenado o número do sinal enviado ao processo;

- group, onde fica armazenado o grupo ao qual o processo pertence;

- timeout, onde é especificado o tempo máximo que um processo pode ficar bloqueado, no caso de ativação do método SuspendProcess;

- status, onde o estado corrente do processo fica armazenado;
- creation-time, onde fica armazenado o instante de criação do processo;
- last-action, onde fica armazenada a última ação realizada sobre o processo.

Os três últimos "slots" são preenchidos pelos métodos RunProcess, SuspendProcess, ContinueProcess, KillProcess e InterruptProcess, cuja ativação resulta na execução das respectivas ações de controle. O estado corrente do processo pode ser lido através do "slot" status.

6.3.3 Sincronização entre Processos

De um modo distinto do sistema DPSK, onde existiam dados compartilhados especiais denominados eventos para fins de sincronização, o sistema DPSK+P deixa tal tarefa ao usuário. Tal política se deve ao fato da grande flexibilidade que pode ser obtida através do ativação dos métodos SuspendProcess e ContinueProcess descritos acima. Assim sendo, mecanismos como eventos, semáforos e barreiras podem ser facilmente desenvolvidos.

6.3.4 Interrupções

O sistema DPSK+P mantém a mesma estrutura do sistema DPSK no que diz respeito ao tratamento de interrupções, utilizando sinais.

Um processo é ligado a um ou mais grupos através do "slot" group da classe **Process**. Para interromper um processo, uma aplicação deverá:

1. Selecionar todas as instâncias da classe **Process** cujo "slot" group seja igual ao

grupo ao qual se deseja enviar o sinal de interrupção;

2. Para cada uma das instâncias selecionadas no passo anterior, preencher o "slot" signo com o número do sinal a ser enviado e executar o método InterruptProcess.

As funções responsáveis pelo tratamento de interrupções podem, por sua vez, obter o número do sinal ativado através de inspeção do "slot" signo na classe **Process**.

6.4 Contribuições

Comparando-se o sistema DPSK+P com o sistema DPSK, algumas evoluções importantes podem ser notadas. Em primeiro lugar, possibilitou-se uma organização hierárquica de classes, com herança de "slots", o que pode se tornar desejável em certas aplicações. Além disso, todo objeto no sistema é instância de uma classe, o que pode facilitar o desenvolvimento de mecanismos de depuração da atividade de resolução de problemas sendo desenvolvida, desde que se guarde um histórico de evolução destas instâncias. Um outro atrativo do enfoque proposto é que o núcleo é facilmente extensível, bastando definir novas classes para os novos serviços, sem alterar o número de primitivas.

Pode-se acrescentar também que aspectos tais como persistência dos dados (através do uso de bases de dados orientadas a objetos) e migração de objetos, para possibilitar reconfigurações pós-queda de sistemas, também podem ser acrescidos ao núcleo no futuro.

7 MONITORAÇÃO E DEPURAÇÃO DE SISTEMAS PARALELOS

7.1 Dificuldades

À medida que os sistemas computacionais se tornam mais complexos, aumentam as dificuldades encontradas pelos projetistas para testá-los e modificá-los de modo a atenderem as especificações para as quais foram projetados. Desta forma, monitores e depuradores complexos vêm sendo cada vez mais utilizados no intuito de aumentar a eficiência do processo de teste de tais sistemas.

De acordo com uma definição clássica, um processo de depuração consiste em localizar, analisar e corrigir falhas em programas, onde falhas podem ser entendidas como condições acidentais que causam desvios na função desejada [66]. Um depurador clássico, portanto, é uma ferramenta que possibilita interromper a execução de um programa, examinar seu estado interno e continuar a execução, podendo eventualmente reexecutá-lo com o intuito de interromper o processamento em passos anteriores ou alterar posições de memória contendo dados ou código. Tal processo é às vezes chamado também de depuração cíclica.

Um monitor, por sua vez, é uma ferramenta responsável pela extração dinâmica de informação durante uma sessão de processamento, sendo os dados coletados apresentados aos usuários em formatos adequados para análise. Um monitor faz parte, necessariamente, de uma ferramenta de depuração.

No caso de sistemas paralelos, tais processos de monitoração e depuração encontram dificuldades adicionais que não aparecem em sistemas seqüenciais. As principais dificuldades são a existência de vários focos de controle, a grande quantidade de dados a serem

monitorados e analisados, a inexistência de um estado global, a presença de características não determinísticas no processamento e o caráter intrusivo da ferramenta de monitoração ou depuração [66] [67].

Antes de analisar tais dificuldades, cabe ressaltar o contexto no qual será utilizada a expressão sistemas paralelos no decorrer deste capítulo. Existem diversas definições e diversos graus de paralelismo envolvidos nos sistemas atualmente existentes. No caso deste trabalho, a expressão **sistemas paralelos** será utilizada para designar:

- sistemas multiprogramados, onde existe apenas uma máquina contendo uma única CPU onde são executados diversos processos "ao mesmo tempo". Sistemas multitarefa são entendidos como um caso particular de sistemas multiprogramados, onde existe apenas um único usuário;

- sistemas multiprocessados, onde existe apenas uma máquina contendo diversas CPUs. O paralelismo envolvido nestes sistemas é de baixa granularidade, envolvendo por exemplo a divisão de passos em comandos de iteração, como o comando for na linguagem C, entre as diversas CPUs;

- sistemas distribuídos, onde existe um conjunto de máquinas interligadas via uma rede que executam uma única aplicação. As máquinas que compõem o sistema podem ser heterogêneas, multiprogramadas ou mesmo multiprocessadas. O paralelismo envolvido é de alta granularidade, como por exemplo resolução distribuída de problemas.

Por outro lado, freqüentemente se utiliza como sinônimos as expressões sistemas paralelos e sistemas concorrentes. **Concorrência** será entendida como competição por recursos comuns, que não envolve necessariamente paralelismo. Sistemas multiprogramados sempre são concorrentes, pelo uso comum de uma única CPU, mas sistemas distribuídos necessariamente não o são. No caso de resolução distribuída de problemas, o uso de uma memória compartilhada torna tais sistemas também concorrentes.

Para efeito de facilitar a linguagem, a expressão **sistemas paralelos** será utilizada daqui em diante para designar qualquer um destes sistemas. No caso de ser necessária uma designação mais precisa, esta será então utilizada.

7.1.1 Existência de Vários Focos de Controle

O fato de existirem vários focos de controle impede que técnicas clássicas de depuração, como "traces" e "breakpoints", sejam utilizadas de forma eficiente em sistemas paralelos. Meios devem ser providos para que as comunicações entre processos sejam monitoradas e exibidas, quaisquer que sejam as políticas adotadas para tal mecanismo, como troca de mensagens ou memória compartilhada, entre outras. Além disso, no caso de sistemas distribuídos, é desejável que a ferramenta de monitoração esteja localizada em apenas uma máquina, de modo a facilitar o processo de análise pelo usuário, devendo então a ferramenta concentrar informações de controle de máquinas distintas.

7.1.2 Grande Quantidade de Informações Monitoradas

O processo de análise poderá também se tornar muito complicado, caso não se consiga exibir de modo claro as ligações entre as informações exibidas e os processos a

partir dos quais tais informações foram colhidas. Dada a natureza extremamente complexa das interações que ocorrem em um sistema paralelo, a quantidade de dados a serem exibidos é muito grande e heterogênea (especialmente em resolução distribuída de problemas, onde os agentes podem ter sido desenvolvidos em várias linguagens e tratando níveis conceituais diferentes do problema), devendo ser providos mecanismos para filtrar tal quantidade de dados, de modo a apresentar ao usuário somente aqueles que efetivamente o interessam no momento.

7.1.3 Inexistência de Estado Global

Sistemas multiprocessados têm geralmente associado um relógio a cada uma das CPUs que os compõem. Em sistemas distribuídos, também não existe um relógio global que sincronize o processamento nas diversas máquinas. Nestes casos, o conceito de estado global de processamento inexiste, uma vez que não se pode determinar com razoável precisão a ordem dos processos que estão sendo executados concorrentemente, em processadores distintos. Mais ainda, supondo um sistema distribuído onde a ferramenta de monitoração se localize em apenas uma máquina, o processo de coleta de dados apresenta atrasos inerentes à utilização de uma rede, fazendo com que dados coletados em diferentes máquinas possam refletir o estado de cada uma delas em instantes diferentes, e não no mesmo instante como se desejava a princípio.

7.1.4 Processamento Não Determinístico

Em sistemas paralelos, nem sempre um mesmo conjunto de entradas produz um mesmo conjunto de saídas, tornando o seu comportamento não determinístico. Tal comportamento se deve ao aparecimento de corridas entre processos, sempre que duas atividades possam ser processadas em paralelo. Por exemplo, caso num sistema paralelo um

processo deva escrever e outro deva ler o conteúdo de uma dada posição de memória, o comportamento do sistema provavelmente será drasticamente alterado caso o segundo processo leia o valor antigo ao invés do novo. Deve-se ressaltar, entretanto, o fato de que tal comportamento é absolutamente válido, dado que nem todas as atividades realizadas em paralelo necessitam obedecer uma relação de ordem total, mas sim várias relações de ordem parcial; caso contrário, não haveria a necessidade de paralelismo. Tal comportamento introduz uma característica de não repetibilidade de processamento, o que pode tornar o processo de depuração muito complicado, pois nem sempre é possível repetir a ocorrência de uma falha, se a causa desta se deve a condições críticas de carga das máquinas ou processadores envolvidos.

7.1.5 Caráter Intrusivo da Monitoração

O caráter não determinístico descrito acima é ressaltado pelo chamado "efeito de ponta de prova". Quaisquer tentativas no sentido de colher mais informações para análise podem aumentar a dificuldade em repetir um comportamento errôneo que tenha sido observado. Isto se deve ao caráter intrusivo das ferramentas de depuração ou monitoração, no sentido de que novos processos serão executados no sistema, alterando a carga dos processadores ou máquinas e introduzindo, eventualmente, novas corridas. Este comportamento tem sido descrito por alguns autores como um efeito em software do Princípio da Incerteza de Heisenberg. Devido a este efeito, corridas críticas podem ser modificadas pela inserção de pontas de prova, tornando o comportamento do sistema diferente daquele no qual as ferramentas de depuração ou monitoração não estão ativas.

7.2 Abordagens Básicas

Para efeito de classificação das pesquisas correntes na área de depuradores e monitores de sistemas paralelos, as técnicas que vêm sendo estudadas podem ser divididas em quatro grupos distintos [66]:

- aplicação de técnicas tradicionais de depuração, através da combinação de vários depuradores, um para cada processo sendo executado. Tais depuradores são denominados depuradores paralelos tradicionais;

- criação de entidades denominadas eventos, cuja seqüência caracteriza a execução de um programa paralelo. São desenvolvidos então métodos para geração e análise de históricos destes eventos;

- criação de técnicas para exibição do fluxo de controle e de dados de um sistema paralelo;

- utilização de mecanismos de análise estática, baseados em análise do fluxo de dados de programas paralelos, que possibilitam que eventuais erros sejam detectados sem que seja necessária a execução do programa.

Cada um desses grupos se encontra detalhado a seguir. Para efeito de simplificação de linguagem, todas as técnicas serão referidas no contexto de projeto de depuradores, podendo, sem perda de generalidade, também serem aplicadas no caso de monitores.

7.2.1 Depuradores Paralelos Tradicionais

A abordagem mais natural que pode ser seguida no intuito de construir uma ferramenta de depuração para um sistema paralelo é a de montar uma coleção de depuradores seqüenciais convencionais, associando um depurador a cada processo sendo executado. O acréscimo necessário devido à existência de paralelismo é que devem ser providos meios para diferenciar a saída e controlar a execução de cada um dos depuradores que o compõem.

O efeito "ponta de prova", entretanto, torna tais depuradores inadequados para detectar falhas decorrentes de restrições temporais, como "time-outs" por exemplo. Por outro lado, para outros tipos de falhas mais comuns, tais depuradores se mostram bastante eficientes. Uma outra crítica que pode ser feita é que o nível de abstração utilizado pelos depuradores é adequado quando se deseja analisar instruções ou procedimentos de cada um dos processos, porém é por vezes inadequado para o caso de observação do nível interprocessos.

Além das capacidades existentes nos depuradores seqüenciais que os compõem, os depuradores paralelos tradicionais devem apresentar os seguintes requisitos:

- capacidade de direcionar comandos de qualquer depurador seqüencial para um determinado processo;
- capacidade de direcionar comandos simultâneos de vários depuradores seqüenciais para um conjunto arbitrário de processos;
- diferenciar as informações de saída provenientes de cada um destes processos.

Uma primeira idéia é a de executar concorrentemente várias cópias de um depurador seqüencial em vários terminais reais ou virtuais, como por exemplo numa estação de trabalho. Se o sistema possuir um gerenciador de janelas, o primeiro e o terceiro requisitos acima estão solucionados. O segundo requisito poderia ser cumprido caso o usuário ativasse manualmente o comando desejado em cada uma das janelas desejadas. Tal política, entretanto, apresenta dois inconvenientes. Primeiro, ela se torna pouco prática para aplicações com um grande número de processos concorrentes. Além disso, o efeito "ponta de prova" é realçado no caso do segundo requisito.

Uma idéia alternativa é controlar o conjunto de depuradores seqüenciais a partir de um único dispositivo de saída ou de uma única janela. Neste caso, cria-se um nível hierárquico superior, responsável pelo gerenciamento dos diversos depuradores seqüenciais, que pode cumprir eficientemente todos os requisitos acima. Atrasos no caso de sistemas distribuídos ainda podem ocorrer, mas são minimizados em relação ao caso anterior.

A possibilidade de inserir "breakpoints" nos diversos processos é uma facilidade imediata, decorrente do uso de depuradores convencionais. No caso de depuradores paralelos tradicionais, existem dois métodos para lidar coma existência de "breakpoints": todos os processos são interrompidos ou apenas o processo que encontrou o "breakpoint" é interrompido. O primeiro método apresenta a desvantagem de eventualmente interromper os diversos processos em tempos diferentes, devido ao atraso inerente à rede, em sistemas distribuídos. No segundo caso, há um grande impacto em sistemas com restrições temporais. A utilização de "breakpoints" para depurar sistemas com tais restrições é extremamente difícil, uma vez que a inexistência de um relógio global limita a possibilidade de expressar

condições ou predicados de parada dos diversos processos. Nestes casos, a tendência é a de expressar tais condições não apenas utilizando os estados internos dos processos, mas também entidades denominadas eventos, que representam ações interprocessos, como descrito em 7.2.2.

Finalmente, cabe ressaltar que depuradores paralelos convencionais necessitam de um grande suporte do sistema operacional para serem eficientes. No caso de depuradores seqüenciais complexos, o sistema operacional ou o hardware devem prover as seguintes capacidades:

- leitura/escrita em registradores/posições de memória;
- inserção de "breakpoints", com interrupção do processamento quando estes são encontrados;
- interrupção do processamento durante a ocorrência de exceções;
- executar um programa passo a passo;
- interrupção do processamento em acessos à memória.

Os depuradores paralelos tradicionais requerem, por sua vez, algumas das seguintes capacidades adicionais:

- interrupção do processamento durante a ocorrência de comunicação entre processos;
- modificação/inserção/remoção do fluxo de comunicação entre processos;
- controle do relógio usado para "timeouts".

7.2.2 Depuradores Baseados em Eventos

Eventos podem ser definidos como ações atômicas visíveis acima do escopo de um determinado processo, que representam mecanismos de controle ou comunicação entre processos. De um modo geral, a diferença entre eventos e estado global é muito tênue para sistemas genéricos, mas muito bem definida para sistemas particulares. Por exemplo, num sistema baseado em troca de mensagens um evento seria "o processo A enviou uma mensagem para o processo B", enquanto que um estado global seria "a fila de mensagens do processo B contém uma mensagem enviada pelo processo A". Pode-se notar que a primeira definição encontra-se acima do escopo dos processos A e B, enquanto que a segunda pode ser encarada como exclusivamente referente ao processo B, conforme a definição adotada.

Como a definição é muito tênue, cada sistema possui sua própria definição de eventos. Em sistemas baseados em troca de mensagens, um envio ou recebimento de mensagens constitui um evento, enquanto que um acesso a uma posição de memória compartilhada constitui um evento em sistemas baseados neste modelo de comunicação entre processos. Sistemas mais flexíveis permitem que o próprio usuário defina os eventos que o interessam, como mostrado em 7.2.2.4. De qualquer modo, em sistemas com mecanismos explícitos de comunicação entre processos os eventos podem ser divididos em duas classes: os que se referem a atividades de comunicação entre processos e os que sinalizam atividades internas dos processos. Tal diferenciação, no entanto, parece não se aplicar em sistemas baseados em memória compartilhada, uma vez que cada acesso à memória é potencialmente uma comunicação entre processos.

Do ponto de vista de manipulação dos eventos, pode-se considerar a existência de duas classes de

depuradores. Na primeira classe, apresentam-se ao usuário os eventos somente quando estes ocorrem. Já os depuradores que pertencem à segunda classe são mais complexos e poderosos. Tais depuradores, objeto de discussão no decorrer deste capítulo, gravam históricos que contêm uma relação de todos eventos gerados durante o processamento. Esta segunda abordagem é extremamente interessante, pois possibilita atenuar o efeito não determinístico dos sistemas paralelos. Ao invés de reexecutar o processamento de forma normal, pode-se reexecutá-lo utilizando como controle as informações guardadas neste histórico de eventos. Tal procedimento é denominado "replay", sendo que nos parágrafos seguintes serão denominados reexecutores as ferramentas que realizam tal procedimento. Um exemplo dessa classe de depuradores é o sistema Instant Replay [68].

Outra característica que difere os depuradores baseados em eventos é o modo pelo qual os eventos são colhidos. Alguns depuradores monitoram diretamente as vias do sistema, via hardware, enquanto que outros necessitam que sejam inseridas explicitamente instruções especiais para este fim nos processos sendo depurados. O sistema Instant Replay adota uma solução intermediária, gerando rotinas de sistema modificadas, que além de suas funções normais gravam também automaticamente o histórico de eventos, cabendo ao usuário a escolha de qual rotina utilizar em cada caso.

Um outro ponto a ser considerado no projeto de tais depuradores é a ordem em que os eventos são registrados no histórico. Uma relação de ordem total é mais simples de ser utilizada, porém introduz relações adicionais desnecessárias, o que leva algumas vezes a interpretações errôneas por parte do usuário, que vislumbra relações entre eventos que efetivamente não existem. Um modo eficiente de registrar os eventos segundo uma relação de

ordem parcial é gerar um arquivo de eventos para cada processo, como é feito no sistema Instant Replay.

Uma vez que o fluxo de informações num sistema paralelo é muito grande, torna-se necessário decidir quais informações sobre os eventos serão efetivamente guardadas no histórico. Uma primeira abordagem é gravar todas as informações possíveis, de modo a minimizar a atuação do depurador e, portanto, o efeito "ponta de prova". Por outro lado, é adequado limitar essa enorme quantidade de informação. Este procedimento será mais efetivo caso se conheça a priori a futura utilização de tal histórico, dependendo das características funcionais do depurador.

Existem basicamente três níveis de utilização desse histórico de eventos, que requerem uma quantidade crescente de detalhes:

- navegadores, cujo objetivo é possibilitar ao usuário simplesmente consultar o histórico de eventos;

- reexecutores, que possibilitam ao depurador utilizar o histórico de eventos para controle da reexecução de todos os processos que compõem o sistema;

- simuladores, que possibilitam reexecutar o sistema com um ou alguns processos (não todos) utilizando o histórico como controle.

7.2.2.1 Navegadores

Navegadores requerem informações mínimas sobre cada evento, pois em certos casos apenas a ocorrência ou não de certo evento podem auxiliar o usuário em detetar uma falha. Alguns sistemas provêm mecanismos de filtragem

para excluir do histórico certos eventos que não interessam ao usuário. Tais mecanismos são automaticamente realizados, descartando certos tipos de eventos ou eventos relacionados a processos que não estão sendo depurados, sempre com a orientação do usuário. Outros sistemas armazenam o histórico em bases de dados, para facilitar uma posterior consulta por parte do usuário.

7.2.2.2 Reexecutores

Reexecutores necessitam de um maior nível de detalhe nas informações colhidas, de modo a ser possível determinar qual o próximo evento no qual cada processo estará engajado. De modo geral, conteúdo de mensagens ou de posições de memória compartilhada não são necessários, uma vez que todos os processos do sistema serão reexecutados a partir das condições iniciais. Tais ferramentas são úteis somente se informações adicionais puderem ser colhidas durante o processo de reexecução. Estas informações podem ser saídas de depuradores seqüenciais ligados a processos sendo depurados ou mesmo provenientes de modificações realizadas nos processos originais, eventualmente com mais instruções de depuração. O comportamento dinâmico do sistema sendo reexecutado pode, no entanto, ser alterado devido a mudanças nas sincronizações entre os processos, uma vez que os reexecutores garantem apenas a manutenção da seqüência de ativação dos diferentes processos. Para sistemas de tempo real, tais condições se tornam mais críticas, já que mecanismos de entrada/saída externa, interrupções e restrições temporais introduzem dificuldades adicionais.

Certos sistemas mais complexos comparam automaticamente, em tempo de execução, o histórico de eventos gerado com predicados lógicos, interrompendo a

execução e tomando ações de depuração no caso de violação destes predicados.

7.2.2.3 Simuladores

Os simuladores necessitam conhecer detalhadamente todos os eventos que sejam visíveis ao processo sendo simulado, incluindo o conteúdo de troca de mensagens ou valores escritos na memória compartilhada. Isso acontece devido ao fato de que somente parte dos processos (aqueles que serão simulados) terão sua execução controlada pelo histórico de eventos, sendo os restantes executados normalmente. Um fato curioso que surge da comparação entre navegadores, reexecutores e simuladores é que para simular a execução de apenas alguns processos torna-se necessário guardar mais informações do que no caso de simular a execução de todos os processos.

7.2.2.4 Criação de Eventos de Alto Nível

Uma abordagem interessante surge quando se considera uma seqüência de eventos não como uma descrição de estados mas como uma descrição de comportamento do sistema sendo depurado [69]. Tal abordagem, denominada Behavioral Abstraction, permite ainda limitar a quantidade de eventos a ser examinada pelo usuário através de mecanismos de filtragem (ignorando eventos que não interessam) e condensação (criando hierarquias de eventos, através da combinação de eventos primitivos).

Um conjunto de ferramentas denominado EBBA (Event Based Behavioral Abstraction) toolkit foi implementado segundo tal metodologia, para depuração de sistemas distribuídos, como representado na figura 7.1 [70].

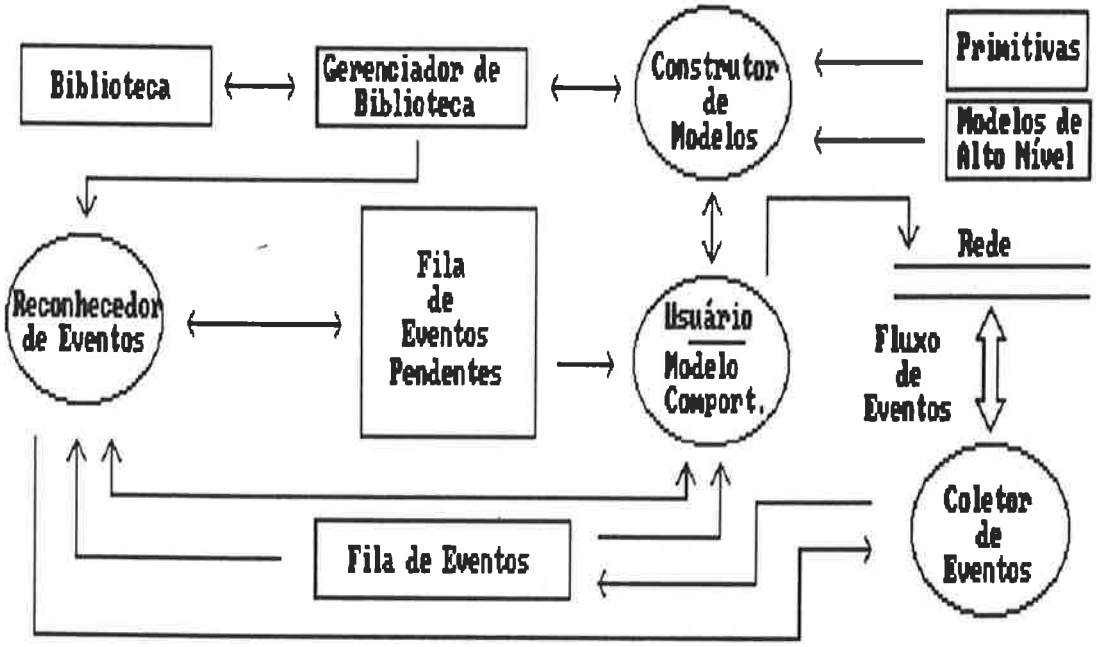

Figura 7.1 - O Sistema EBBA toolkit

O sistema consiste dos seguintes componentes:

- uma biblioteca básica de eventos;
- um construtor de modelos, que permite ao usuário criar eventos novos ou combinar alguns já existentes para determinar seu modelo de eventos;
- uma gerenciador de biblioteca, cuja função é a de incorporar os modelos de eventos construídos pelo usuário e interagir com os outros componentes do sistema;
- um coletor de eventos, cuja função é a de inserir todos os eventos gerados pelos diversos processos numa fila de eventos;
- um reconhecedor de eventos, que a partir dos eventos presentes na fila de eventos realiza uma filtragem baseada nos eventos

definidos no modelo do usuário, atualizando uma fila de eventos pendentes.

Todos os componentes do sistema são autônomos, cabendo ao usuário manipular a fila de eventos pendentes na tarefa de análise. Para interagir com o depurador, os processos que devem ser depurados devem ser ligados com uma biblioteca do sistema. Os componentes do depurador se localizam em uma única máquina.

Do ponto de vista de funcionalidade, o EBBA toolkit permite que sejam criados três níveis diferentes de depuração, com níveis de atuação remota crescentes:

- depuração remota simples, onde todo o fluxo de eventos é enviado para a máquina onde se encontra fisicamente localizado o conjunto de ferramentas, sem atuação remota alguma;

- depuração remota filtrada, com ações pré-definidas, quando são definidas listas de atenção com o intuito de realizar uma filtragem preliminar, diminuindo o fluxo de eventos enviados para a máquina onde se localiza o depurador. Além disso, possibilita-se pré-definir ações que devem ser realizadas quando certos eventos ocorrem, de modo a diminuir o tempo de resposta nestes casos;

- depuração remota complexa, quando se permite que as máquinas remotas examinem o fluxo de eventos, permitindo ações de depuração locais, baseadas na ocorrência de eventos locais e remotos.

Analisando-se as características do sistema EBBA toolkit descrito acima, pode-se concluir que são grandes

as possibilidades de utilização de eventos como descritores de sistemas paralelos.

7.2.3 Técnicas de Exibição do Fluxo de Controle

Quando foram descritos os depuradores paralelos tradicionais e os depuradores baseados em eventos, propositadamente não foram detalhados os meios de exibição para o usuário das informações colhidas durante o processo de monitoração. Tal aspecto será discutido agora.

A existência de um único dispositivo de saída é suficiente para exibir as informações colhidas por um depurador seqüencial. No caso de sistemas paralelos, tal característica se torna limitante, tanto devido à existência de diversos focos de controle como pela distribuição, lógica ou física, dos dados a serem apresentados.

Existem basicamente quatro técnicas distintas para apresentação do fluxo de controle num sistema paralelo [66]:

- apresentação textual dos dados colhidos, aproveitando características dos dispositivos de saída tais como cores e realces. Neste caso, não existe uma representação explícita da passagem do tempo na tela;

- diagramas temporais, que apresentam o resultado da execução de um programa utilizando um eixo para representar os processos e outro eixo para representar o tempo. Os pontos marcados na tela terão associados a si rótulos que indicam a atividade desenvolvida pelo processo no instante em questão;

- animação de processos, onde os dois eixos representam dimensões espaciais, sendo que a tela mostra o estado global dos diversos processos em um determinado instante de tempo, como se fosse uma foto. Na medida em que o tempo passa, novas fotos passam a ser exibidas, e efeito final é o de um filme que conta a evolução dos processos;

- múltiplas janelas, que permitem visões simultâneas do sistema, sendo que usualmente cada processo tem associado a si uma janela.

7.2.3.1 Apresentação Textual

A apresentação textual dos dados é a forma mais simples e mais comum, sendo às vezes o único modo possível de exibição no caso de depuradores paralelos convencionais. Para depuradores baseados em eventos, tal técnica às vezes é útil, principalmente no caso de navegadores, quando se deseja analisar a seqüência de eventos tal como vista por um único processo. A ordem de exibição das informações pode ser causal e não necessariamente temporal, de modo a facilitar ao usuário o reconhecimento de ocorrências relacionadas entre si nos diversos processos. Um problema que pode ocorrer é que para sistemas razoavelmente complexos, com muitos processos, as informações não podem ser exibidas em uma única tela simultaneamente. Uma solução que pode ser adotada para este caso, por exemplo, é realizar uma filtragem dos dados, exibindo apenas aqueles que efetivamente interessam no momento.

7.2.3.2 Diagramas Temporais

Diagramas temporais são uma representação bidimensional do estado de um sistema paralelo no tempo.

De acordo com a política de comunicação entre processos adotada, tais diagramas podem ser ligeiramente diferentes.

Em sistemas baseados em memória compartilhada dispositivos de saída sem capacidade gráfica podem ser eficientemente utilizados, como mostra a figura 7.2.

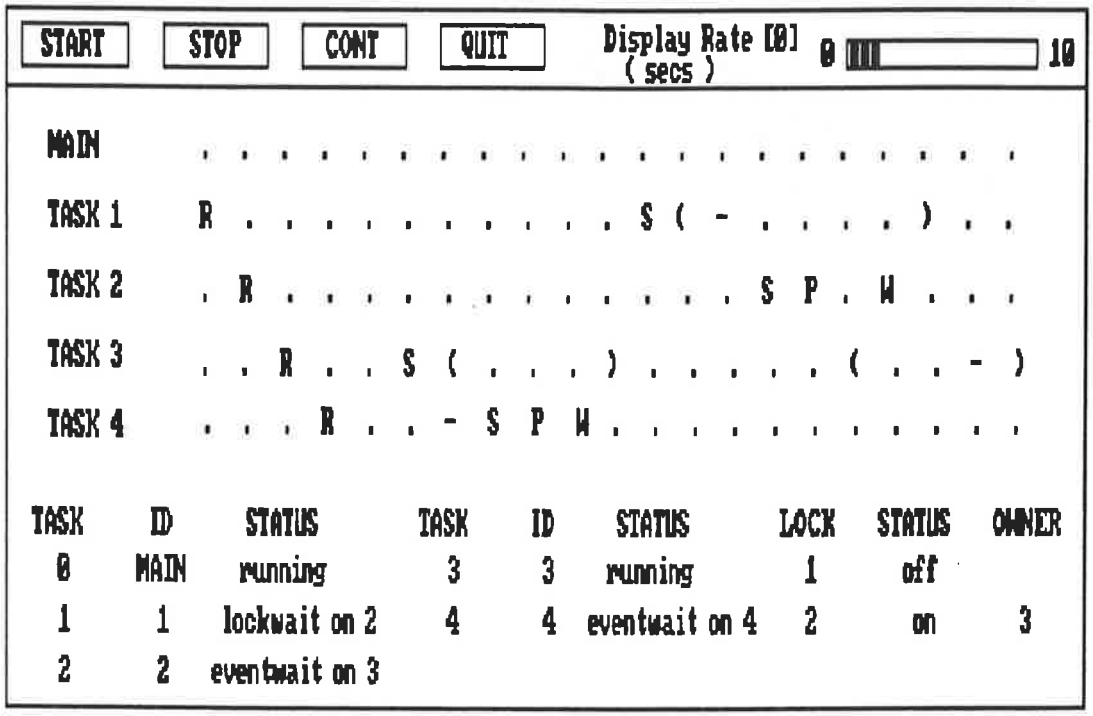

Figura 7.2 - Diagrama Temporal do Sistema MTDBX

Esta figura, gerada pelo sistema MTDBX [66], representa o estado de um sistema multitarefa. Caracteres são associadas ao estado dos processos, sendo a unidade de tempo a ocorrência de um evento. Em se tratando de um sistema multitarefa, apenas um processo terá associado a si um caractere em cada instante, sendo os outros representados por um ponto. A grande vantagem desta abordagem é que informações adicionais podem ser providas, aumentando o nível de detalhe dos estados dos

diversos processos, como pode ser observado na parte inferior da figura. Através de comandos presentes na parte superior da figura, o usuário pode modificar o foco de atenção para intervalos de tempo distintos.

Sistemas baseados em troca de mensagens também podem ser eficientemente representados de modo análogo. Entretanto, caso seja disponível um dispositivo de saída com capacidades gráficas, pontos na tela podem ser diretamente ligados, representando o envio e recebimento de mensagens entre determinados processos. Tal abordagem certamente é mais expressiva que a anterior, ressaltando-se o fato de que a unidade de tempo passa a ser o relógio, o que limita a utilização de tal abordagem a sistemas multitarefa, pois nos outros casos geralmente não existe um relógio global, como já visto. Tal abordagem é adotada pelo sistema IDD [66] e encontra-se representada na figura 7.3. Da mesma forma que no caso anterior, existem comandos disponíveis para alterar o intervalo de tempo a ser analisado.

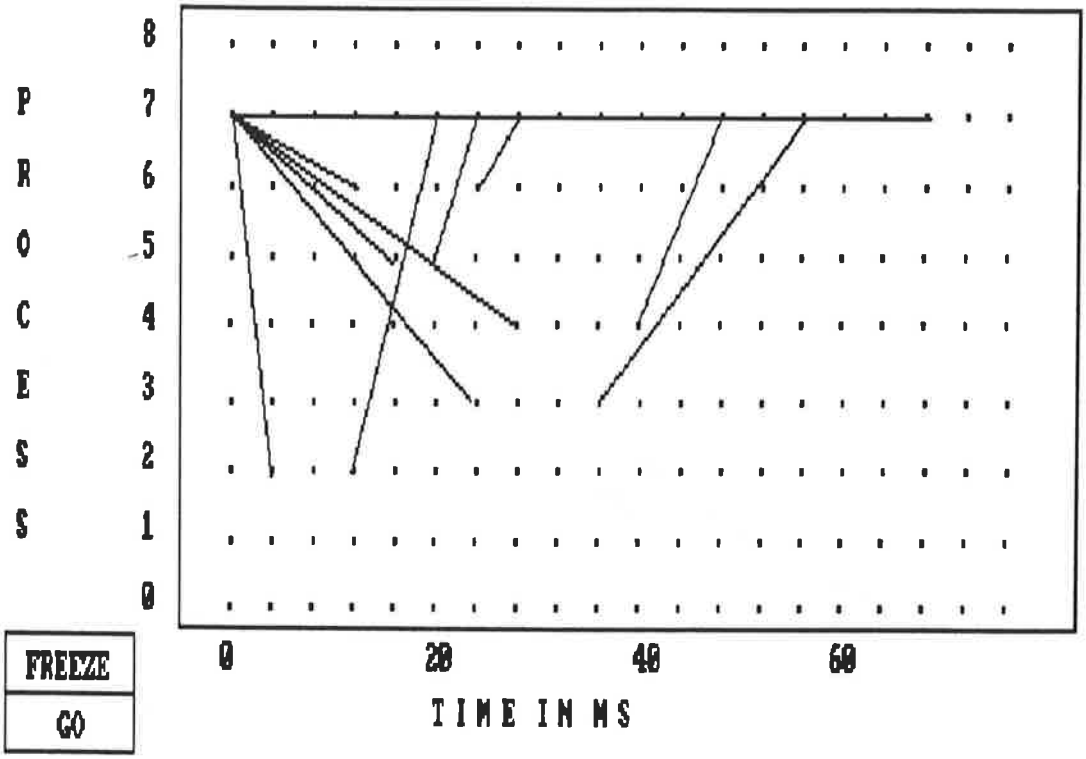

Figura 7.3 - Diagrama Temporal do Sistema IDD

Uma tentativa de solucionar este problema da inexistência de um relógio global é a utilização de uma técnica denominada mapas de concorrência [71]. Segundo esta técnica, os eventos são ordenados de modo a descrever sua ordem relativa, e não o instante exato de tempo em que ocorreram. Tal ordem é inferida através de restrições temporais que ocorrem num programa, como por exemplo o fato de que um recebimento de mensagem só pode ocorrer após o seu envio. Um exemplo de utilização de mapas de concorrência para um caso de paralelismo de baixa granularidade é apresentado na figura 7.4, representando a execução do seguinte pseudocódigo de dois "loops" numa máquina multiprocessada com cinco processadores:

DOALL J = 1,4
LOOP1
BARREIRA DE SINCRONIZAÇÃO
DOALL K = 1,5
LOOP2
BARREIRA DE SINCRONIZAÇÃO

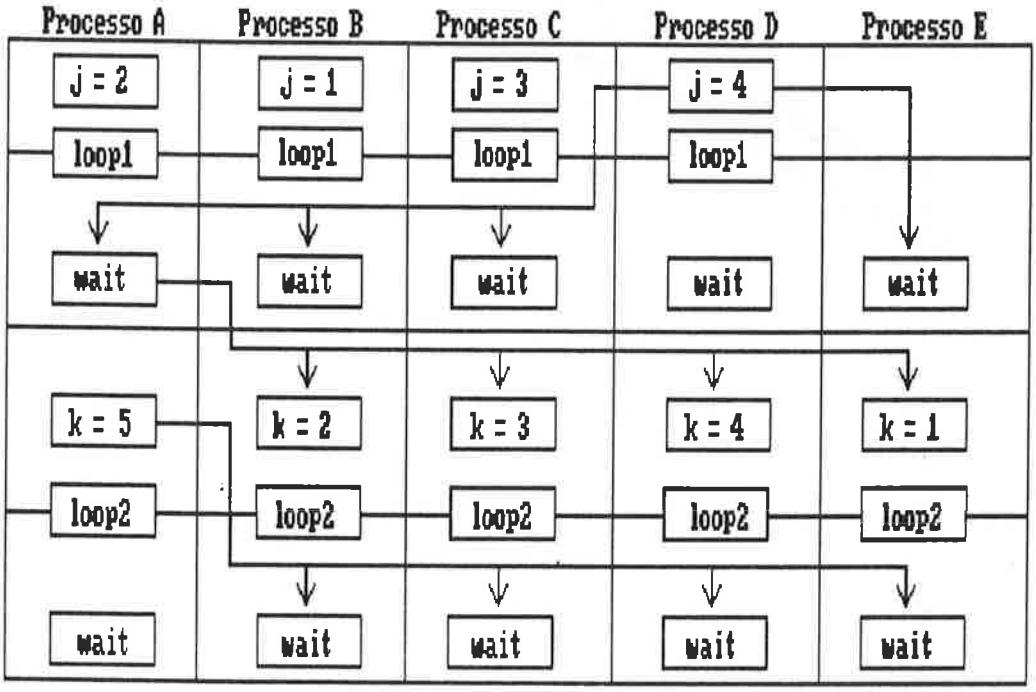

Figura 7.4 - Mapas de Concorrência

Nesta figura, cada coluna apresenta seqüências de eventos que ocorrem em um dado processo, enquanto que cada linha representa um intervalo de tempo. Eventos que aparecem em colunas diferentes numa mesma linha podem ocorrer simultaneamente, sendo que em cada coluna os eventos são divididos em blocos dependentes, nos quais uma relação de ordem parcial é obedecida, devido às restrições temporais entre os processos. A definição destes blocos se faz a partir de eventos predecessores ou sucessores, sendo as restrições temporais associadas a

estes eventos representadas por arestas entre blocos. No caso do exemplo da figura 7.4, uma dependência é representada pelas arestas que partem do evento j = 4 no processo D para os eventos **wait** nos outros processos, significando que até que seja atribuído o último valor da variável j os outros processos devem continuar em loop1. As barreiras de sincronização são representadas por arestas que partem dos eventos **wait** para os eventos que representam a atribuição de valores à variável k, significando que nenhum processo pode iniciar tal atribuição antes que todos tenham ultrapassado tal barreira.

7.2.3.3 Animação de Processos

Em animação de processos, cada processo tem associado a si uma porção da tela, sendo representado através de figuras geométricas ou ícones. A ligação entre os processos pode ser feita por meio de segmentos de reta, que se utilizam de mudança de cores ou de largura para indicar a ocorrência de algum evento referente ao nível interprocessos. Assim, em cada instante, tem-se um retrato do estado global do sistema. Tal abordagem é extremamente útil no caso de troca de mensagens, porém mais complicada no caso de comunicação por memória compartilhada.

O sistema BELVEDERE [66] utiliza esta técnica para um sistema baseado em troca de mensagens, como mostrado na figura 7.5.

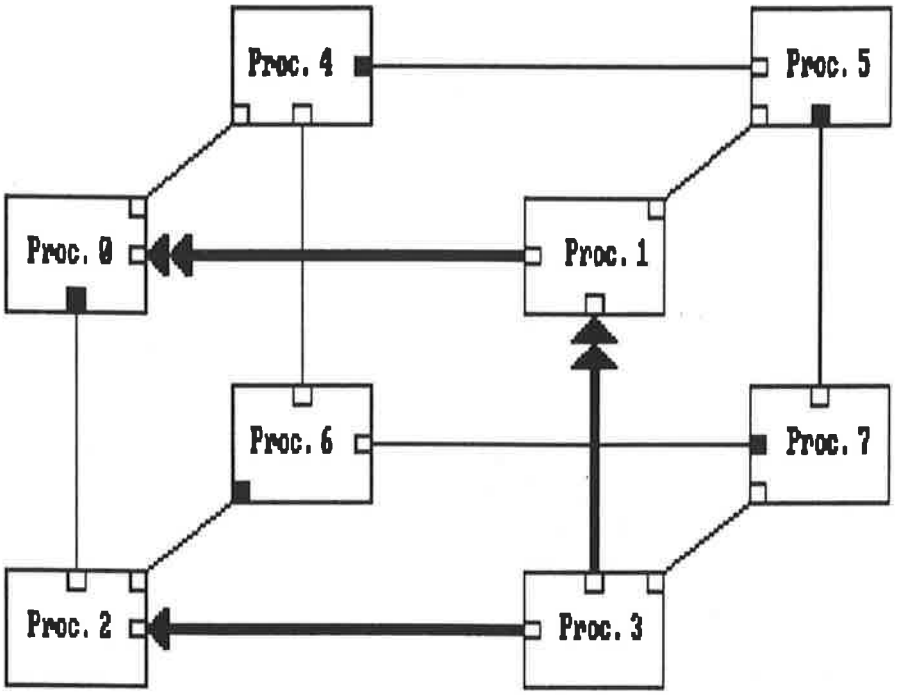

Figura 7.5 - Animação de Processos no Sistema BELVEDERE

Tal figura representa uma foto do estado global de processamento num determinado instante de tempo. A atividade dos processos é indicada através da exibição intensificada de portas (pequenas caixas) e canais (linhas), sendo que portas intensificadas representam recebimento e canais intensificados envio de mensagens. O sentido das setas nas linhas indica a direção do fluxo de mensagens, enquanto que sua quantidade o número de mensagens enviadas. Neste sistema ainda, a localização dos processos na tela é muito importante, sendo especificada pelo usuário. Permite também que os eventos sejam mostrados segundo diferentes perspectivas: de processadores, canais ou dados. No caso de perspectiva de um único processador, os eventos serão exibidos de acordo com a ordem em que foram detectados por este processador. Lembrando que no caso de aplicações distribuídas existe um atraso de comunicação devido à rede, possivelmente dois processadores distintos detectarão eventos em ordens

distintas e, portanto, apresentariam perspectivas diferentes.

Já o objetivo do sistema VOYEUR [72] é o de prover ao usuário a possibilidade de criar vistas específicas para visualizar processos paralelos. Encontram-se atualmente disponíveis quatro tipos de vistas: ícones, vetor, "trace" e lista ligada. Na vista por ícones, os eventos permitem ao sistema determinar em que posição da tela determinado ícone deve ser colocado e quando iniciar um novo quadro de animação. A visão por vetores é uma variação da visão por ícones, permitindo representar a variação temporal de funções matemáticas bidimensionais. A visão por "trace" associa uma caixa a cada processo, onde estão representados seus valores locais. A ligação entre as caixas representam as conexões entre os processos. Finalmente, a visão por lista ligadas é uma variação da anterior, que apresenta também algumas estruturas de dados especiais do programa. Um fato interessante é que no sistema VOYEUR uma interface separa logicamente os processos de exibição e de coleta do fluxo de eventos dos processos sendo monitorados. Deste modo, o usuário pode convenientemente mudar a visão, utilizando o mesmo fluxo de eventos.

7.2.3.4 Múltiplas Janelas

Sistemas de múltiplas janelas são usuais nos dias de hoje, especialmente em estações de trabalho. Por outro lado, ao se comparar técnicas de animação de processos com diagramas temporais, pode-se observar que ambos são complementares. Enquanto os diagramas temporais refletem de um modo bastante eficiente a ocorrência de padrões de comportamento do sistema no tempo, certamente pecam pela ausência de informações mais precisas sobre o estado global dos processos num dado instante. Técnicas de animação são eficientes para obter um retrato instantâneo do sistema, mas não provêm meios de guardar a ocorrência

de eventos passados. Um sistema que certamente seria muito eficiente poderia constar de duas janelas, cada uma exibindo o fluxo de controle segundo uma destas técnicas, de modo a fornecer ao usuário informações precisas sobre o estado global instantâneo e sobre a evolução de comportamento do processos.

7.2.4 Técnicas de Análise Estática

Em certas condições críticas, o efeito "ponta de prova" que se manifesta durante a utilização de depuradores paralelos tradicionais ou de depuradores baseados em eventos não permitem que o usuário possa corrigir determinadas falhas. Para estes casos, técnicas de análise estática vêm sendo estudadas, cujo objetivo é a detecção de certas classes de anomalias em programas paralelos sem que estes necessitem ser executados. Não se trata, entretanto, de provas formais de corretude, tais como pesquisadas em Engenharia de Software, mas apenas de procurar garantir que o programa paralelo não irá atingir estados pré-identificados como incorretos.

Essas técnicas vêm sendo utilizadas para detecção de duas classes de erros: aqueles devido à sincronização entre processos, como bloqueios mortais, e aqueles decorrentes de má utilização dos dados, como por exemplo escrita paralela em posições de memória sobrepostas.

Duas linhas de pesquisa bem definidas podem ser distinguidas [66]. A primeira delas consiste em aplicar técnicas de análise de fluxo de dados em programas paralelos, semelhantes àquelas utilizadas por otimizadores e vetorizadores de compiladores, cujo objetivo é determinar propriedades na utilização dos dados. Tais técnicas possibilitam detectar erros pertencentes à segunda classe descrita acima, como por exemplo referências a variáveis não inicializadas, referências a variáveis que estão sendo definidas em

paralelo e definições de variáveis que nunca serão utilizadas. Erros pertencentes à primeira classe, tais como processos aguardando o término de outros processos que não estão em execução ou que já garantidamente encerraram a sua execução, também podem ser detectados.

A segunda linha de pesquisa tenta responder à seguinte pergunta:

Quando é possível executar duas instruções S1 e S2 num programa paralelo?

Para atingir tal objetivo, uma função do tipo parallel (i,j) é construída, que retorna o valor verdade se ambas as instruções podem ser executadas em paralelo. Caso o programa paralelo possa ser representado por uma Rede de Petri [62], esta função pode ser implementada a partir do grafo de atingibilidade de estados, sendo o único inconveniente a explosão combinatória desse grafo, quando o número de processos paralelos é muito grande. Outras técnicas vêm sendo também pesquisadas para tentar solucionar tal explosão.

Finalmente, cabe ressaltar alguns esforços no sentido de combinar técnicas de análise estática com a coleta dinâmica dos dados. Um enfoque é utilizar os resultados obtidos por análise estática para a geração de casos de teste para a posterior utilização de depuradores dinâmicos. Por outro lado, resultados obtidos a partir da utilização de depuradores dinâmicos podem servir para guiar o processo de análise estática, no sentido de diminuir a ocorrência de situações que, embora teoricamente possíveis de serem atingidas, nunca irão efetivamente ocorrer na prática.

Evidentemente, todas as técnicas, estáticas ou dinâmicas, detalhadas até agora pressupõem que já exista um sistema paralelo a ser testado. Esforços também vêm

sendo feitos em Engenharia de Software para minimizar a ocorrência de erros em tais sistemas durante a fase de desenvolvimento dos mesmos. Um exemplo que pode ser citado são os compiladores que automaticamente vetorizam passos de programas paralelos para sistemas multiprocessados, distribuindo as instruções entre diversos processadores e garantindo que não existem falhas devido à essa distribuição.

7.3 Situação Corrente e Tendências Futuras

Sem dúvida alguma, grandes progressos já foram obtidos na área de depuradores e monitores de sistemas paralelos. Diversas ferramentas já foram projetadas e implementadas, ou encontram-se em processo de implementação. Além das já citadas no decorrer deste capítulo, pode-se acrescentar o sistema JADE [67], um monitor para sistemas baseados em troca de mensagens que permite apresentar as informações de forma gráfica ou textual, e o sistema TÊMIS [73], desenvolvido no Brasil, que consiste num núcleo para desenvolvimento de programas paralelos em MODULA-2, em ambiente PC-DOS. Tal sistema, inclusive, permite contempla um reexecutor baseado em um histórico de eventos.

Pode-se observar também que o escopo de atuação destes sistemas contemplam todos os tipos de paralelismo definidos no início deste capítulo, de alta ou baixa granularidade, desde aplicações para sistemas multitarefa, como o próprio TÊMIS, passando por máquinas multiprocessadas, como o sistema PPUTT [74] e contemplando inclusive núcleos para resolução distribuída de problemas, como por exemplo o depurador construído para o sistema AGORA [75], referenciado no capítulo 4 deste trabalho.

Cabe citar também a existência de trabalhos no sentido de adaptar depuradores e monitores de sistemas

paralelos para tratar sistemas de tempo real e para realizar a coleta de informações através de hardware específico.

7.3.1 Depuração de Sistemas Paralelos de Tempo Real

O processo de depuração de sistemas de tempo real, paralelos ou não, apresenta dificuldades adicionais em relação aos sistemas convencionais, pelo fato de existirem explicitamente, por requisitos de projeto, restrições temporais. O usuário precisa então detectar não somente falhas devidas à erros de lógica, como também aquelas provenientes da violação de tais restrições.

Tradicionalmente, a maioria dos sistemas de tempo real contemplam um escalonador que divide o processamento em ciclos e subciclos, sendo estes últimos associados a atividades periódicas, como a execução de um conjunto de rotinas. Caso ocorra uma falha devido à violação de restrições temporais em um subciclo, não é possível, em geral, detectar a rotina que a originou. Mais do que isto, se puderem ser escalonadas rotinas provenientes de processos diferentes num mesmo subciclo, a ocorrência destas violações tem efeito sobre vários processos.

O monitor de tempo real ART [76] apresenta uma proposta interessante para solucionar tal problema em sistemas de tempo real distribuídos, através da criação de constructos de linguagem para expressar explicitamente as restrições temporais em um processo, tornando-as autocontidas e impedindo que sua ocorrência num determinado processo influa em outro. A idéia básica é estender o conceito de encapsulamento de dados utilizado em Engenharia de Software para tratar tais restrições, criando uma espécie de encapsulamento temporal. Além disso, é criado um escalonador que possa lidar eficientemente com tais constructos.

O sistema se constitui de três módulos: um coletor de eventos, um despachador de eventos que realiza a filtragem dos mesmos e os envia para um visualisador, cuja tarefa é exibí-los de modo conveniente para o usuário. O visualisador se encontra localizado fisicamente em outra máquina, de modo a minimizar o efeito "ponta de prova", que certamente é mais crítico para aplicações de tempo real. O sistema operacional desenvolvido é baseado no paradigma de objetos, implementando a ativação de métodos remotos através de chamada remota de procedimentos. Ao definir as operações a serem aplicadas aos objetos nos métodos, o usuário define explicitamente um intervalo de tempo máximo para que a operação seja completada e um procedimento a ser realizado em caso de violação desta restrição. Através deste último, o usuário pode então garantir que o estado de processamento fique consistente, mesmo no caso de uma violação deste tipo.

O encapsulamento temporal é realizado através de um escalonador que aplica duas políticas distintas. Os processos que tenham fortes restrições temporais são escalonados segundo uma política fixa, enquanto que os outros são escalonados através de uma função de avaliação, cujo objetivo é o de garantir que os processos da primeira classe, mais prioritários, não tenham suas restrições temporais violadas. Do ponto de vista do sistema operacional, o controle sobre essas violações se dá de modo semelhante ao controle sobre violações de limites em vetores. O sistema operacional acompanha as operações realizadas sobre os objetos e verifica se o intervalo de tempo definido pelo usuário é factível, através de estimativas de atrasos de comunicação na rede e de desvios entre os relógios das diversas máquinas. Caso chegue à conclusão que tais requisitos não podem ser cumpridos, o controle é passado então para o procedimento que trata a violação da respectiva restrição.

O tratamento de exceções também é uma característica marcante em sistemas de tempo de real, que pode ser convenientemente aproveitada no projeto de monitores e depuradores. Uma proposta neste sentido é apresentada no sistema operacional MACH [77], do tipo UNIX, destinado a máquinas multiprocessadas. O núcleo do sistema é baseado em cinco abstrações: tarefas, fluxos de controle, portas mensagens e objetos de memória. Tarefas são unidades básicas de alocação de recursos, enquanto que fluxos de controle são unidades básica de processamento. Deste modo, clones de processos constituem dois fluxos de controle sobre uma única tarefa. A comunicação se dá através do envio de mensagens a portas previamente alocadas, que constituem canais de comunicação.

A ocorrência de uma exceção suspende o fluxo de controle relacionado e ativa um procedimento de tratamento, que pode ter sido desenvolvido pelo usuário ou pode fazer parte de uma ferramenta de depuração. Tal processo se dá através do envio de mensagens a portas especiais, criadas para este fim. Tais portas podem ser ligadas tanto a fluxos de controle como a tarefas. Exceções relacionadas a fluxos de controle são preferencialmente tratadas pelo usuário, enquanto que aquelas relacionadas a tarefas são tratadas pelos depuradores.

Devido a sua grande flexibilidade, o sistema operacional MACH permite que a depuração de sistemas paralelos seja realizada de acordo com as características da aplicação. Para diversos fluxos de controle em uma única tarefa, o depurador deve apenas se ligar à porta relacionada com a ocorrência de exceções nesta tarefa. Para o caso de várias tarefas, a ligação deve ser feita com as portas de todas as tarefas, sendo que a diferenciação é feita internamente. Finalmente, no caso de sistemas distribuídos, a aplicação e o depurador podem

estar em máquinas diferentes ou o próprio depurador ser distribuído, uma vez que existe uma transparência em relação à rede.

7.3.2 Coleta de Informações por Hardware

Dentre todas as dificuldades apontadas em 7.1, a única que efetivamente não pode ser eliminada através de software é o caráter intrusivo do monitor. Uma idéia que vem sendo ultimamente pesquisada é a introdução de um hardware específico para interagir com os sinais de controle e conteúdos de memória do sistema a ser depurado. Neste caso, o efeito "ponta de prova" desaparece, facilitando inclusive a detecção de falhas por violações de restrições temporais.

A interceptação dos sinais a serem monitorados através de um hardware deste tipo apresenta as seguintes vantagens adicionais [78]:

- possibilidade de coleta de informações impossíveis de serem obtidas por software, como por exemplo taxas de utilização de vias;
- o comportamento do sistema sendo monitorado apresenta sempre as mesmas características dinâmicas, independentemente da ativação ou não do monitor;
- o hardware pode ser incorporado definitivamente, uma vez que não influi no comportamento do sistema sendo monitorado.

Deve-se, no entanto, observar que tal hardware não pode ser comparado aos emuladores que hoje se encontram disponíveis comercialmente. Apesar de colher os mesmos dados que estes, o objetivo é que sejam fornecidas visões

de alto nível como aquelas fornecidas pelos depuradores citados no decorrer deste capítulo. Em outras palavras, caso os processos ativos num sistema paralelo tenham associados a si descritores localizados fisicamente em posições pré-determinadas de memória, tal hardware pode acessá-los e construir modelos de mais alto nível para o usuário, como por exemplo eventos.

Uma tentativa neste sentido para o caso de um sistema multiprocessado é apresentada no sistema MAKBILAN [79]. Tal sistema cria e mantém, em tempo real, uma descrição do sistema sendo monitorado, colhendo informações de diferentes níveis, relativos ao hardware (como acesso a vias) e ao sistema operacional (como alocação de recursos). O modo de exibição destas informações é definido pelo usuário, estando disponíveis diferentes vistas, inclusive um mecanismo de animação de processos, sendo a exibição realizada em um "display" rápido. Também possibilita a definição de eventos de alto nível que tenham significado para o usuário. A arquitetura do sistema MAKBILAN encontra-se representada na figura 7.6.

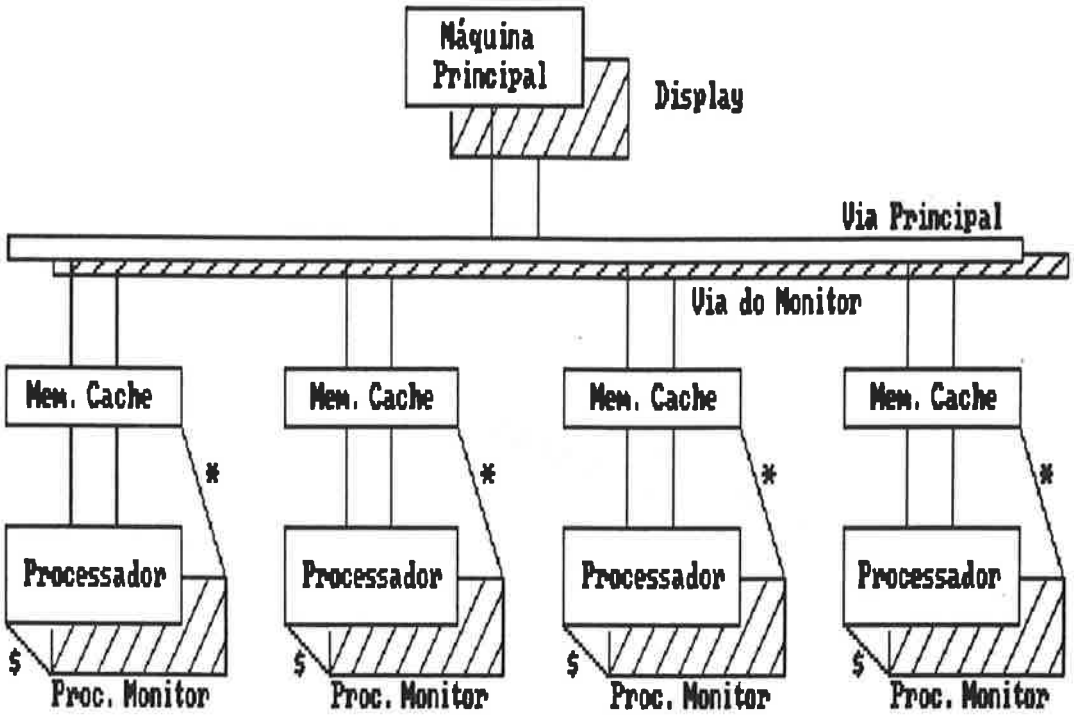

Figura 7.6 - Arquitetura do Sistema MAKBILAN

O sistema multiprocessado a ser monitorado, mostrado em primeiro plano, se constitui de um computador principal, que se liga a diversos outros processadores através de uma via de dados e controle e de memórias "cache". O hardware responsável pelo processo de depuração, mostrado em segundo plano com hachuras, é praticamente uma réplica da máquina original. A ligação entre as duas máquinas é feita através de acesso às memórias "cache" pelos processadores da máquina depuradora (representados na figura 7.6 por *) e também por contato direto com os processadores da máquina principal (representados por $) para coleta do estado de processamento desta última.

Finalizando, seria interessante resumir as principais tendências, atuais e futuras, no projeto de ferramentas de depuração e monitoração de sistemas

paralelos, tomando como base os exemplos e técnicas apresentadas.

Em primeiro lugar, é clara a tendência de dividir uma ferramenta deste tipo em pelo menos três módulos, responsáveis pela coleta, despacho e exibição das informações monitoradas, sendo que em aplicações com restrições temporais críticas o módulo de exibição geralmente se localiza fisicamente em outra máquina. Além disso, a ligação entre a informação colhida e o modo de exibí-la ao usuário é definida por este último, podendo inclusive ter várias vistas da mesma informação, privilegiando padrões de comportamento durante o tempo, no caso de diagramas temporais, ou retratos instantâneos do estado global de processamento, no caso de animação de processos.

Em segundo lugar, é cada vez mais comum a criação de eventos de alto nível de abstração para representar a atividade de processamento, principalmente no que se refere à comunicação entre processos. Isto é feito tanto para facilitar o processo de análise pelo usuário, bem como para contornar o problema do processamento não determinístico dos sistemas paralelos, através da criação de históricos de eventos que serão utilizados para controle da reexecução.

Uma última tendência que pode ser observada é a utilização de hardware específico para a coleta de informações, com a finalidade de eliminar o efeito "ponta de prova" das ferramentas de monitoração.

8 UMA FERRAMENTA DE MONITORAÇÃO PARA O SISTEMA DPSK+P

8.1 Informações Monitoradas

O sistema DPSK+P, descrito no capítulo 6 deste trabalho, contempla primitivas para comunicação e controle entre diversos agentes componentes de um processo de resolução de problemas. Além disso, permite ao usuário criar uma hierarquia de classes compartilhadas, de modo a possibilitar a troca de informações entre tais agentes.

Para efeito de monitoração, foram definidos vários eventos elementares correspondentes a ações de comunicação e controle entre agentes, segundo a filosofia apresentada no capítulo anterior. Além destes eventos, informações sobre classes e instâncias contidas na memória compartilhada encontram-se também disponíveis para fins de consulta pelo usuário. Tais eventos e informações encontram-se descritos a seguir.

8.1.1 Eventos de Controle

Os eventos de controle definidos e tratados pela ferramenta de monitoração são os seguintes:

- E1: ativação de um agente;
- E2: suspensão de processamento de um agente;
- E3: retomada de processamento de um agente;
- E4: ativação de uma exceção em um agente;
- E5: término forçado de processamento de um agente;

- E6: término normal de processamento de um agente;
- E7: ativação de uma chamada remota de procedimentos por um agente;
- E8: processamento de uma chamada remota de procedimentos por um agente;
- E9: término de processamento de uma chamada remota de procedimentos por um agente.

Reportando-se às classes e métodos primitivos descritos no capítulo 6 deste trabalho, tais eventos ocorrem nas seguintes situações:

- E1: ativação do método RunProcess, definido na classe **Process**;
- E2: ativação do método SuspendProcess, definido na classe **Process**;
- E3: ativação do método ContinueProcess, definido na classe **Process**;
- E4: ativação do método InterruptProcess, definido na classe **Process**;
- E5: ativação do método KillProcess, definido na classe **Process**;
- E6: término de processamento do agente, notificado ao núcleo pelo sistema operacional;
- E7: ativação do método CallMethod por um agente;
- E8: ativação do método AcceptMethod por um agente;

. E9: final de processamento do método AcceptMethod por um agente.

8.1.2 Eventos de Comunicação

Toda a comunicação entre agentes no sistema DPSK+P se dá através de operações de leitura e escrita em objetos compartilhados. A manutenção da consistência desses objetos pode ser realizada pelo núcleo, através de transações, se o usuário assim o desejar.

Os eventos de comunicação definidos e tratados pela ferramenta de monitoração são os seguintes:

. E10: início de uma transação;

. E11: término normal de uma transação;

. E12: interrupção de uma transação;

. E13: leitura de objetos compartilhados;

. E14: escrita na memória compartilhada.

Novamente se reportando às classes e métodos primitivos descritos no capítulo 6, tais eventos ocorrem nas seguintes situações:

. E10: ativação do método BeginTransaction, definido na classe **Transaction**;

. E11: ativação do método EndTransaction, definido na classe **Transaction**;

. E12: ativação do método AbortTransaction, definido na classe **Transaction** ou término forçado pelo próprio núcleo, devido a condições anômalas, como por exemplo "time-outs";

- E13: ativação dos métodos GetInstance ou GetSlot;
- E14: ativação dos métodos MakeInstance, DeleteInstance, UpdateSlot ou DeleteSlot.

8.1.3 Classes e Instâncias Compartilhadas

O conteúdo da memória compartilhada, no que se refere às classes e instâncias criadas pelos diversos agentes, também é convenientemente tratado pela ferramenta de monitoração. Foi criado um navegador, que permite ao usuário visualizar a hierarquia de classes, seu conteúdo (definição dos "slots", agente criador) e suas instâncias (máquina onde se encontram, identificador).

Estas informações tornam-se disponíveis à ferramenta de monitoração através de operações de leitura dos objetos compartilhados, que são ativadas pelos métodos GetInstance e GetSlot. Uma vez que o núcleo é capaz de identificar a ferramenta de monitoração, esta pode inclusive acessar instâncias que estejam sendo modificadas numa transação, e cujo estado, portanto, seja momentaneamente inconsistente. Tal acesso é vedado a outros agentes, porém é essencial para efeito de monitoração.

8.2 Exibição das Informações Monitoradas

Elegeu-se a técnica de animação de processos, descrita no capítulo anterior, para a exibição dos eventos elementares de comunicação e controle mostrados acima. Tal escolha deveu-se à priorização de informações referentes ao estado global de processamento, em detrimento de uma evolução temporal mais apurada. Cabe ressaltar, entretanto, que a partir dos eventos definidos, torna-se extremamente simples adicionar à

ferramenta de monitoração uma facilidade deste tipo, devido ao caráter modular de seu desenvolvimento, que será convenientemente detalhado em 8.3.

A fim de minimizar os problemas decorrentes desta ausência de informações temporais mais precisas, o método de animação de processos adotado foi ligeiramente modificado em relação àquele originalmente descrito no capítulo 7. Optou-se, no projeto da ferramenta, pela manutenção permanente da última ação de controle ou comunicação realizada. Tal característica será descrita com maiores detalhes a seguir, assim como a exibição de informações referentes ao conteúdo da memória compartilhada.

8.2.1 Exibição dos Eventos de Controle

Para efeito de monitoração das ações de controle, cada agente encontra-se representado na ferramenta de monitoração por um círculo, localizado em uma determinada posição na tela. Os diferentes círculos serão preenchidos com cores diferentes, de modo a representar a ocorrência dos diferentes eventos de controle. Além disso, setas ligarão os círculos entre si, com o objetivo de representar logicamente a ligação entre o agente ativo (aquele que ativou a ação de controle) e o agente passivo (aquele ao qual a ação de controle foi endereçada). No caso de chamada remota de procedimentos ou da ativação de exceções, o nome do método sendo ativado ou o grupo e sinal referentes à exceção também serão respectivamente exibidos. Um exemplo de exibição destes eventos de controle encontra-se representado na figura 8.1.

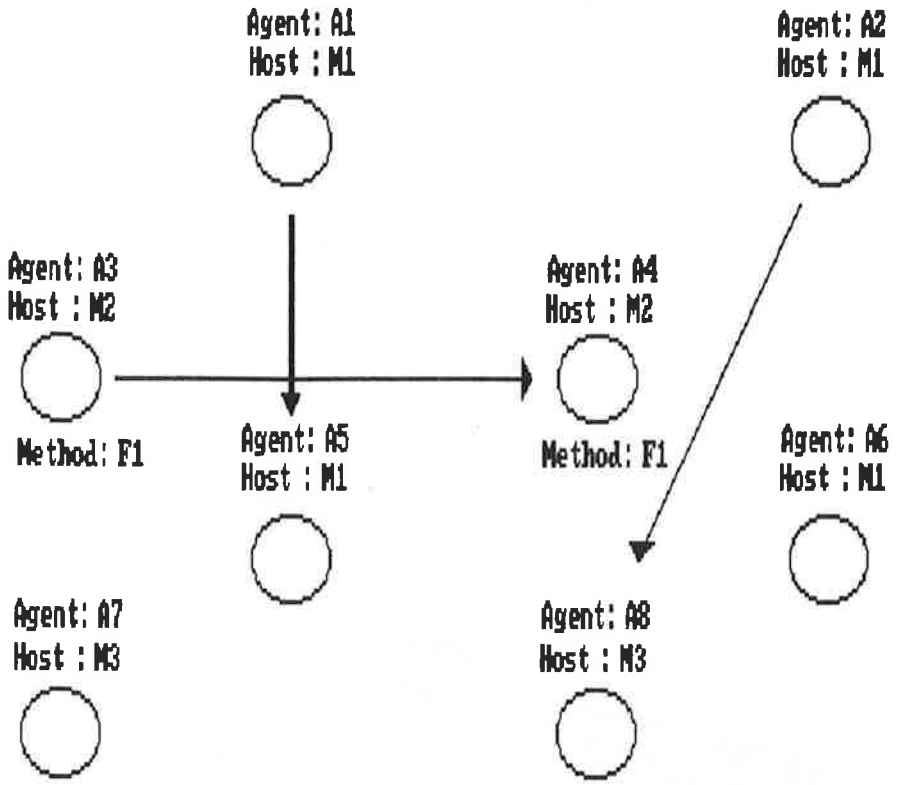

Figura 8.1 - Exibição dos Eventos de Controle entre Agentes

Esta figura representa um conjunto de oito agentes, fisicamente localizados em três máquinas distintas. Os agentes A1, A2, A5 e A6 localizam-se na máquina M1, os agentes A3 e A4 na máquina M2 e os agentes A7 e A8 na máquina M3. Os agentes A1 e A2 ativaram ações de controle nos agentes A5 e A8, respectivamente, sendo que a especificação destas ações se dá em função da cor que tomam os agentes que as sofreram. Já os agentes A3 e A4 estão envolvidos num mecanismo de chamada remota de procedimentos, através do método F1.

As setas ligando os agentes somente serão apagadas quando o agente que sofreu a ação de controle sofrer uma outra ação de controle posterior. Tal persistência permite que, a cada instante, estejam representados na tela os últimos eventos nos quais os agentes estiveram

envolvidos, minimizando em parte a inexistência de um diagrama temporal de eventos, que contivesse informações mais precisas sobre a evolução de comportamento dos mesmos.

A princípio, sempre que um novo agente for ativado, ele será exibido pela ferramenta, desde que o limite de agentes simultaneamente visíveis não seja excedido. O usuário pode, entretanto, controlar a visibilidade destes agentes, especificando aqueles que deseja observar num determinado momento. Este efeito, que corresponde a uma filtragem de eventos de controle, é extremamente útil em resolução distribuída de problemas, quando se deseja depurar o processo de resolução de um determinado subproblema, no qual estão envolvidos apenas um determinado número de agentes.

Para efeito de simplificação do protótipo, o limite de agentes simultaneamente visíveis é oito. Tal limite foi fixado levando em conta o número médio de agentes presentes em aplicações típicas de resolução distribuída de problemas. Por outro lado, como a ferramenta de monitoração foi implementada de modo distribuído, a ser detalhado em 8.3, pode-se ativar mais de um navegador de eventos de controle ao mesmo tempo, de modo a possibilitar a visibilidade simultânea de mais de oito agentes.

8.2.2 Exibição dos Eventos de Comunicação

Para efeito de monitoração do mecanismo de comunicação entre agentes, estes serão representados por linhas horizontais, que acessarão classes compartilhadas, representadas por retângulos. As operações de escrita e leitura de instâncias dessas classes serão representadas por setas ligando agentes a classes, respectivamente saindo e chegando nos agentes, como mostra a figura 8.2.

Os valores associados às setas referem-se a identificadores das instâncias acessadas.

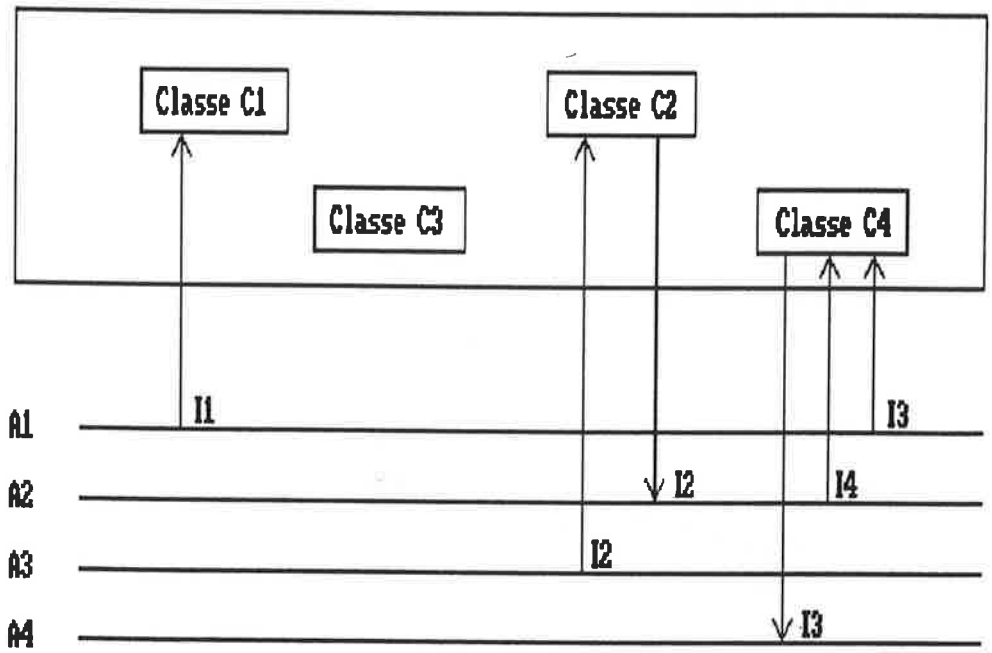

Figura 8.2 - Exibição dos Eventos de Comunicação entre Agentes

Segundo esta figura, o agente A1 realizou uma operação de escrita na instância I1 da classe C1. Já os agentes A3 e A1 estabeleceram uma comunicação efetiva com os agentes A2 e A4, respectivamente, através das instâncias I2 e I3 das classes C2 e C4.

A ocorrência de transações será visualizada através da atribuição de cores aos agentes e classes envolvidos. Assim, referindo-se ainda à figura 8.2, caso o acesso do agente A1 à classe C1 tenha sido feito através de uma transação, a linha que o representa bem como o conteúdo do retângulo que representa C1 tomaram a mesma cor quando a transação se iniciou. Ao final da mesma, ambos voltaram a exibir as cores padrões.

Originalmente, a exibição dos identificadores de instância não seria contemplada pela ferramenta de monitoração. Tal exibição, entretanto, tornou-se necessária para tratar casos de comunicação não atômica, isto é, casos em que uma operação de leitura de determinada instância ocorra após uma operação de escrita de outra instância da mesma classe, por outro agente. Tal situação é ilustrada na figura 8.2, indicando que o agente A4 lê a instância I3 da classe C4 após uma outra instância, I4, da mesma classe ter sido criada ou modificada pelo agente A2. Para efeito de simplificação do protótipo, o número de instâncias simultaneamente representadas foi limitado a três.

A última operação realizada na memória compartilhada será sinalizada através de uma cor diferente nas setas ligando agentes a classes. Novamente, os últimos acessos realizados em cada classe terão persistência até o acesso seguinte, minimizando em parte a ausência de um diagrama temporal de eventos, como no caso da exibição dos eventos de controle.

Tal procedimento, entretanto, apresenta um outro efeito, muito mais importante, e no qual consiste a contribuição básica deste trabalho. Trata-se de um mecanismo poderoso que permite representar ações de comunicação efetiva entre agentes, num esquema de memória compartilhada. Tentativas de representar tais ações em esquemas de troca de mensagens são usuais na literatura, o que não ocorre no caso de sistemas baseados em memória compartilhada, como visto no capítulo anterior. Tal fato decorre de que cada elemento em tal memória é, potencialmente, um mecanismo de comunicação, inviabilizando uma representação concisa e eficiente, notadamente para sistemas com aplicação prática. No caso do sistema DPSK+P, isso somente é possível pela utilização do paradigma de objetos na memória

compartilhada. Cabe notar que a especificação das instâncias acessadas pelos diversos agentes é extremamente compacta, sendo representadas somente aquelas diretamente envolvidas com a ocorrência dos últimos eventos de comunicação para cada classe. Embora tal esquema impeça a inspeção direta de valores contidos nestas instâncias, ele deve ser considerado como um mecanismo de abstração, ou seja, suficiente para efeitos de monitoração de alto nível, cujo comportamento é semelhante àquele apresentado no capítulo anterior, quando eventos de alto nível foram criados a partir de eventos primitivos. Somente através deste mecanismo de abstração torna-se viável exibir de modo compacto as ações de comunicação, através da limitação da quantidade de "canais de comunicação" lógicos, representando o conteúdo da memória compartilhada. Além disso, os meios de exibição da estrutura de classes compartilhadas descritos a seguir permitem que o usuário, ao detectar uma falha de processamento, inspecione estas instâncias com um maior grau de profundidade.

Novamente, para efeito de simplificação do protótipo, o número de classes compartilhadas exibidas simultaneamente limita-se a cinco. Como no caso do navegador de eventos de controle, o usuário pode modificar, em tempo de execução, quais as classes e agentes que deseja tornar visíveis, de acordo com a fase de resolução do problema sendo monitorada. Cabe ressaltar ainda que caso o usuário deseje observar um número de classes que exceda ao limite fixado, existe a opção de ativar mais de um navegador de eventos de comunicação ao mesmo tempo, como no caso dos eventos de controle.

8.2.3 Exibição do Conteúdo da Memória Compartilhada

A ferramenta de monitoração permite ao usuário visualizar as classes e instâncias contidas na memória compartilhada em três níveis de abstração distintos.

Um primeiro nível apresenta a hierarquia de classes, conforme mostra a figura 8.3.

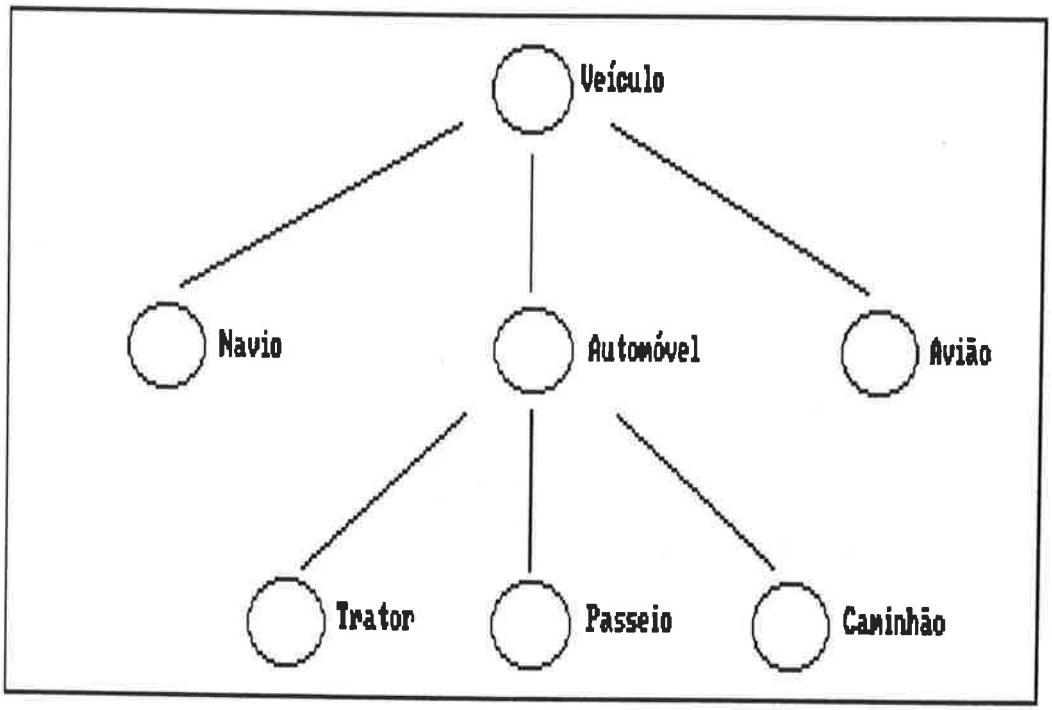

Figura 8.3 - Exibição da Estrutura de Classes Compartilhadas

Caso o conteúdo de uma determinada classe necessite de um maior nível detalhe, este é fornecido por um segundo nível, mostrado na figura 8.4.

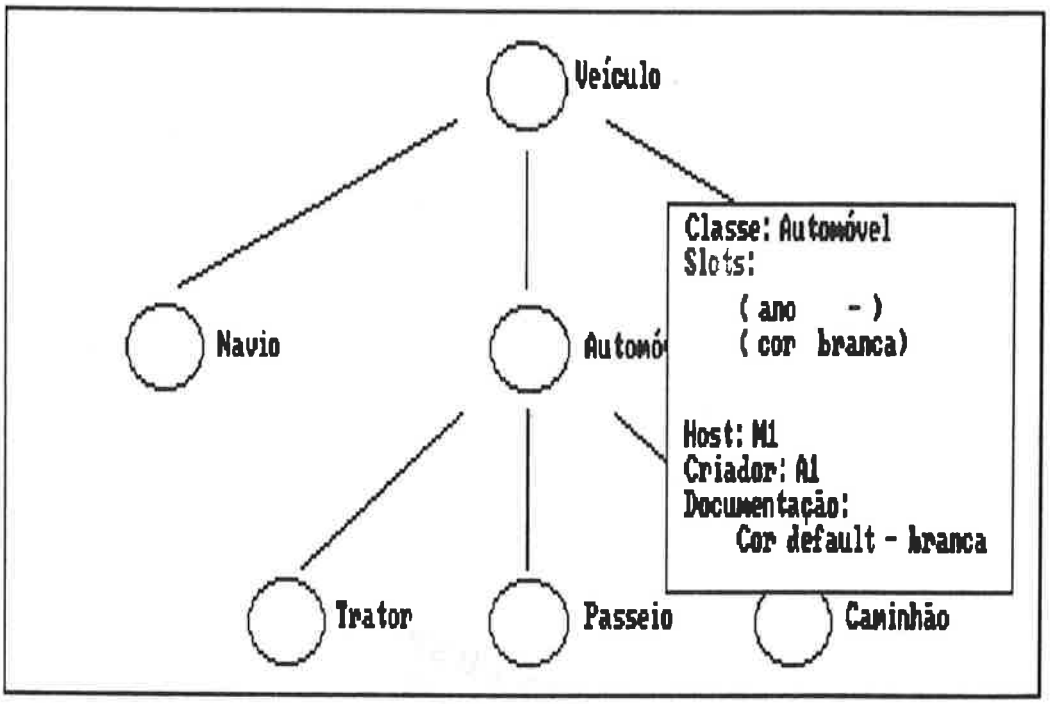

Figura 8.4 - Exibição do Conteúdo de uma Classe Compartilhada

Finalmente, um terceiro nível provê ao usuário informações sobre as instâncias criadas a partir das classes, como apresentado na figura 8.5

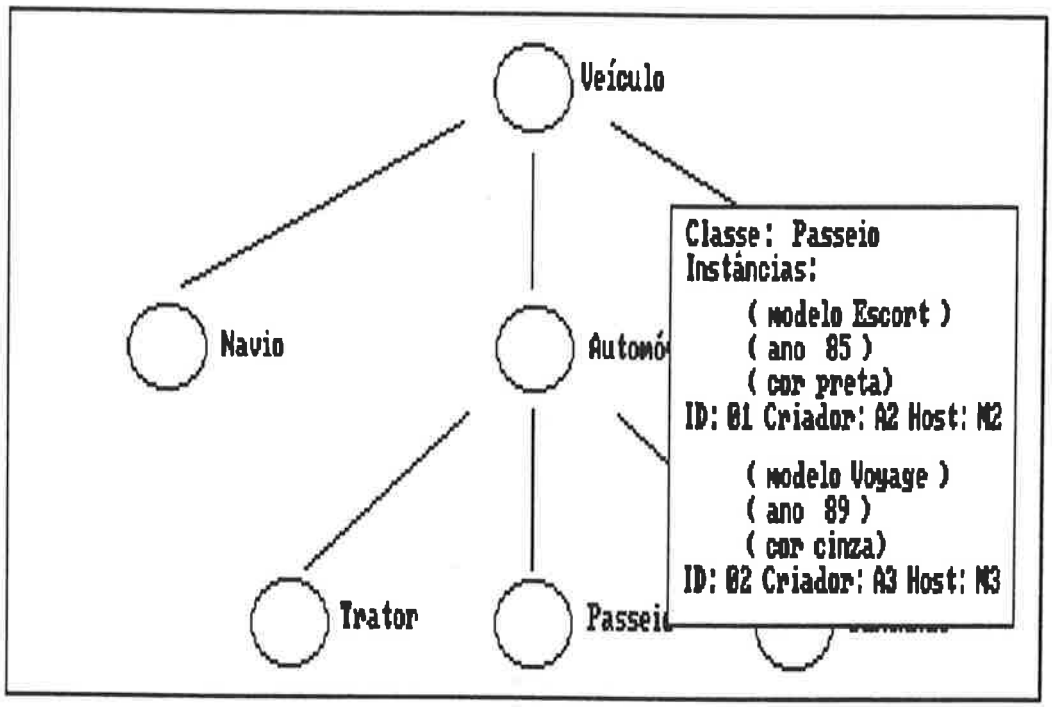

Figura 8.5 - Exibição das Instâncias de uma Classe Compartilhada

Para efeito de simplificação da ferramenta de monitoração, o navegador da estrutura de classes permite visualizar simultaneamente um número máximo de cinco níveis hierárquicos. Através do manuseio de um "mouse", a raiz da estrutura sendo exibida pode ser deslocada, permitindo a navegação na hierarquia, tanto vertical como horizontalmente. Além disso, a utilização do "mouse" permite também a mudança entre os diferentes níveis de abstração descritos acima.

8.3 Estrutura da Ferramenta

A ferramenta de monitoração se compõe de três módulos distintos: um navegador de eventos de controle, um navegador de eventos de comunicação e um navegador da hierarquia de classes compartilhadas. Seu desenvolvimento foi baseado na utilização de um sistema com múltiplas janelas, de modo a associar a cada navegador uma janela

distinta. Com isso, torna-se possível a exibição dos dados referentes aos diferentes módulos ao mesmo tempo, caso isso seja desejado pelo usuário. Além disso, um desenvolvimento modular permite a incorporação de facilidades futuras, como por exemplo um módulo responsável pela criação e exibição de um diagrama temporal de eventos.

Do ponto de vista de comportamento dinâmico, a ferramenta de monitoração compreende três processos distintos, executados simultaneamente, que correspondem a cada um dos navegadores citados. Tal enfoque permite a atualização das três janelas ao mesmo tempo, mesmo que alguma delas se encontre total ou parcialmente obscurecida por outra.

Como visto no capítulo 7, é política usual separar as operações de coleta e de exibição das informações monitoradas, de modo a aumentar a eficiência deste processo. Tal separação permite, inclusive, que a exibição das informações monitoradas se dê em uma máquina distinta, quando a aplicação apresentar fortes restrições temporais. No caso da ferramenta de monitoração do DPSK+P, o próprio núcleo funciona como um módulo de coleta de eventos, uma vez que são de seu conhecimento todas as ações de controle e de interação com a memória compartilhada que possam vir a ser de interesse da ferramenta de monitoração.

Foram analisados dois modelos distintos de comunicação entre a ferramenta de monitoração e o núcleo. Um primeiro modelo considera a ferramenta como um grupo de agentes de resolução de problemas adicional, que estaria interessado em acessar alguns objetos especiais contidos na memória compartilhada, ou seja, instâncias das classes utilizadas pelo núcleo para comunicação e controle entre agentes.

Utilizando tal modelo de comunicação, quando o usuário criasse instâncias das classes de controle sendo monitoradas (**Process** e **Transaction**), a ferramenta poderia ser sinalizada pelo núcleo (por exemplo, através do preenchimento de um valor previamente estabelecido no "slot" signo pertencente à instância da classe **Process** referente à ferramenta de monitoração, seguido da ativação do método InterruptProcess), ativando então uma transação com o objetivo de ler tais instâncias. Tal opção, no entanto, apresenta o inconveniente de bloquear parte do conteúdo da memória compartilhada, podendo então ser perceptível o efeito "ponta de prova", no caso de aplicações críticas.

Um segundo modelo, que evita tal inconveniente, considera a ferramenta de monitoração como um grupo de agentes particular, que ativa algumas primitivas especialmente criadas pelo próprio núcleo para efeito de monitoração. Esta solução é, ao mesmo tempo, simples e elegante , uma vez que se trata apenas de aproveitar informações já disponíveis no núcleo. Tais primitivas seriam especialmente construídas para fins de monitoração, com o único objetivo de fornecer informações à ferramenta de monitoração. Não existem, portanto, acessos adicionais à memória compartilhada, uma vez que o núcleo tem disponíveis para si todas as informações referentes a ações de comunicação e controle entre agentes. Do ponto de vista da ferramenta de monitoração, basta ativar estas primitivas para obter uma cópia das informações desejadas, sem o ônus de um acesso à memória compartilhada.

Este segundo modelo foi o adotado no desenvolvimento da ferramenta. Para facilitar a troca de informações com o núcleo, foi aproveitada a mesma estrutura de mensagens utilizada para a comunicação interna entre os diversos

módulos que o compõem, que é convenientemente preenchida de acordo com o evento sendo enviado.

Além da identificação, o núcleo envia as seguintes informações a respeito de cada um dos eventos de controle e comunicação:

- E1: nome simbólico do agente que foi criado e identificação dos agentes envolvidos;
- E2: identificação dos agentes envolvidos;
- E3: identificação dos agentes envolvidos;
- E4: identificação dos agentes envolvidos, grupo e número do sinal enviado;
- E5: identificação dos agentes envolvidos;
- E6: identificação do agente que terminou seu processamento;
- E7: identificação do agente envolvido e nome simbólico do procedimento remoto ativado;
- E8: identificação dos agentes envolvidos e nome simbólico do procedimento remoto sendo processado;
- E9: identificação do agente envolvido e nome simbólico do procedimento remoto que terminou seu processamento;
- E10: identificação do agente e classe envolvidos, tipo e identificação da transação sendo iniciada;
- E11: identificação do agente, classe e transação sendo finalizada;

. E12: identificação do agente, classe e transação sendo interrompida;

. E13: identificação do agente, classe e instância acessada para leitura;

. E14: identificação do agente, classe e instância acessada para escrita.

O termo identificação empregado acima se refere a um par de inteiros do tipo (máquina, identificador). A especificação da máquina é necessária, pois em ambientes distribuídos pode haver dois agentes sendo processados em máquinas distintas com o mesmo identificador.

A título de ilustração, um esboço de código referente ao navegador de eventos de controle num dialeto C-like seria o seguinte:

```
.............
InitMonitor ("moncontrol");
while (true)
{
     flag = ReceiveEvents (timeout, &my_event);
     if (flag)
     {
         switch (my_event.eventno)
         {
             case 1:
             /* tratamento de ativação de agente */
             break;
             case 2:
             /* tratamento de suspensão de agente */
             break;
             ...
         }
     }
```

```
        flag = CheckManagerEvents (&manager_event);
        if (flag)
        {
            switch (manager_event.type)
            {
                case 1:
                /* tratamento de ativação de "mouse" */
                break;
                case 2:
                /* tratamento de ativação de teclado */
                break;
                ...
            }
        }
    }
```

A primitiva InitMonitor tem a função de identificar o navegador junto ao núcleo. Esta identificação permite que o núcleo passe a lhe enviar o fluxo de eventos de controle correspondente, possibilitando ao usuário ativar simultaneamente mais de um navegador, de tipos diferentes ou do mesmo tipo, numa mesma máquina ou em máquinas distintas.

A primitiva ReceiveEvents retorna como parâmetro um valor não nulo caso exista um evento a ser encaminhado ao navegador. O "timeout" serve para evitar que o navegador fique indefinidamente bloqueado até a chegada de um evento de controle. Com isso, possibilita-se que o navegador possa também receber eventos do gerenciador de janelas, por exemplo através de uma primitiva CheckManagerEvents. Esta primitiva depende, obviamente, do gerenciador de janelas utilizado, não fazendo parte das primitivas do sistema.

Os outros navegadores teriam um pseudo-código análogo, diferindo apenas na sua identificação e nos eventos que tratam.

8.4 Implementação

A ferramenta de monitoração está sendo desenvolvida numa estação de trabalho do tipo SUN SPARC-II, com monitor colorido, rodando em ambiente UNIX [80]. A linguagem C [81] está sendo utilizada para seu desenvolvimento, para não penalizar sua portabilidade. Além disso, o sistema XWINDOWS v. 11 [82] está sendo também utilizado tanto como gerenciador de janelas como para gerador de figuras gráficas.

Até o presente momento, os navegadores de eventos de controle e comunicação já foram desenvolvidos e encontram-se integrados ao núcleo. Tais módulos contém aproximadamente 2500 linhas de código.

8.5 Extensões Futuras

Como já citado no decorrer deste capítulo, o desenvolvimento modular da ferramenta de monitoração permite que extensões lhe sejam incorporadas no futuro, de modo a provê-la de capacidades adicionais que possam auxiliar o projetista durante o ciclo de desenvolvimento das aplicações. Podem ser citadas de imediato duas possíveis extensões:

- a incorporação de diagramas temporais, que possam fornecer dados mais precisos sobre a evolução temporal dos eventos de controle e comunicação entre agentes;

- a possibilidade de definição de eventos de controle e comunicação de um maior nível de abstração, juntamente com um mecanismo de filtragem mais apurado, de modo que o usuário possa escolher quais eventos deseja analisar em cada instante. Neste caso, seria necessário também desenvolver-se um editor de eventos, que pudesse

construir novos eventos a partir dos eventos pré-definidos no sistema, bem como um reconhecedor de eventos, que pudesse realizar o mecanismo de filtragem, semelhante ao apresentado no sistema EBBA toolkit [70].

Por outro lado, ferramentas mais poderosas do que navegadores poderiam ser construídas caso o fluxo de eventos fosse armazenado em memória não volátil. Como a comunicação entre a ferramenta de monitoração e o núcleo aproveita estruturas internas deste último, existem informações tais como selos de tempo que não estão sendo atualmente utilizadas. De posse de tais informações, reexecutores e simuladores, de acordo com a nomenclatura utilizada no capítulo 7, poderiam ser construídos, uma vez que tais selos de tempo estabelecem univocamente uma relação de ordem parcial dos eventos no tempo.

9 CONCLUSÕES

Nas próximas décadas, requisitos emergentes tais como suporte à evolução de sistemas e à execução paralelizada em plataformas heterogêneas terão de ser contemplados no projeto de grandes organizações de software, de modo a dotá-las de níveis de desempenho aceitáveis. Para obtê-los, técnicas de resolução distribuída de problemas podem ser aplicadas, em conjunto com resultados consolidados oriundos do estudo das organizações humanas. O paradigma de objetos, por outro lado, fornece um arcabouço teórico extremamente útil que também lida eficientemente com tais requisitos.

Do ponto de vista do projetista, serão necessárias ferramentas cada vez mais poderosas para auxiliá-lo durante o ciclo de desenvolvimento, principalmente monitores e depuradores que tratem convenientemente os aspectos inerentes à distribuição de tarefas. Tais ferramentas apresentam dificuldades de projeto adicionais àquelas já existentes em sistemas convencionais, que já estão sendo investigadas pela comunidade acadêmica. Dentre as principais técnicas sendo investigadas, cabe ressaltar poderosos mecanismos de abstração no sentido de definir eventos de alto nível referentes aos processos de controle e comunicação entre agentes, bem como mecanismos eficientes para a exibição dos resultados coletados, quer priorizando aspectos dinâmicos, como diagramas temporais, quer priorizando o estado global instantâneo, como no caso de animação de processos.

A ferramenta de monitoração construída para o sistema DPSK+P consiste num conjunto de três navegadores, que exibem eventos de comunicação e controle entre agentes, bem como o conteúdo da hierarquia de classes compartilhadas. Além disso, a técnica tradicional de animação de processos foi ligeiramente modificada de modo a minimizar em parte a ausência de informações mais

precisas sobre a evolução temporal dos eventos sendo analisados.

Do ponto de vista de implementação, a ferramenta foi desenvolvida de maneira modular, possibilitando a incorporação futura de novos módulos, como por exemplo diagramas temporais de eventos. Além disso, ferramentas mais poderosas do que navegadores, tais como reexecutores ou simuladores, também podem ser construídas, desde que se armazene em memória não volátil o fluxo de eventos enviado pelo núcleo.

Finalmente, introduziu-se ainda um mecanismo poderoso para a representação de eventos de comunicação em esquemas baseados em memória compartilhada, através da utilização da abstração fornecida pelas classes no paradigma de objetos. Tal mecanismo permite que o tratamento de eventos deste tipo seja realizado de modo extremamente conciso e eficiente.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] TAKAHASHI, T.; LIESENBERG, H. K. E. **Programação orientada a objetos.** São Paulo, IME-USP, 1990. /Apresentado à 7. Escola de Computação, São Paulo, 1990/

[2] CARDOZO, E. **DPSK:** a kernel for distributed problem solving. Pittsburgh, 1987. 144 p. PhD (Thesis) - CAED, Carnegie-Mellon University.

[3] ERMAN, L. D. et al. The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty. **Computing Surveys**, v.12, n.2, p.213-53, June 1980.

[4] DATE, C. J. **An introduction to database systems.** 4.ed. Reading, Addison-Wesley, 1986. v.1, cap.18, p.413-28.

[5] GALBRAITH, J. **Designing complex organizations.** Reading, Addison-Wesley, 1973.

[6] TAKAHASHI, T. **Introdução à programação orientada a objetos.** Curitiba, EBAI, 1988. Cap.1, p.2-4. /Apresentado à 4. Escola Brasileiro Argentina de Informática, Curitiba, 1988/

[7] CHANDRASEKAN, B. Natural and social system metaphors for distributed problem solving: introduction to the issue. **IEEE Transactions on Systems, Man, and Cybernetics**, v.11, n.1, p.1-5, Jan. 1981.

[8] CASTILHO, A. V. **Análise de novas perspectivas para o estudo da inteligência humana:** as medidas de raciocínio analógico e sua inserção no panorama atual de pesquisa. São Paulo, 1985. 180 p. Tese (Doutorado) - Instituto de Psicologia, Universidade de São Paulo. Cap.5, p.67-71.

[9] SMITH, R. G.; DAVIS, R. Frameworks for cooperation in distributed problem solving. **IEEE Transactions on Systems, Man, and Cybernetics**, v.11, n.1, p.61-70, Jan. 1981.

[10] SMITH, R. G. The contract net protocol: high-level communication and control in a distributed problem solver. **IEEE Transactions on Computers**, v.29, n.12, p.1104-13, Dec. 1980.

[11] SANTOS, S. M. **Programação concorrente:** mecanismos de sincronização e comunicação de processos. São Paulo, IME-USP, 1984. p.23-83. /Apresentado à 4. Escola de Computação, São Paulo, 1984/

[12] TANENBAUM, A. S. **Operating systems:** design and implementation. Englewood Cliffs, Prentice-Hall, 1987. Cap.2, p.51-80.

[13] SOUZA, P. S. **Primitivas de controle e comunicação para processamento paralelo em sistemas computacionais multiprocessados.** São José dos Campos, 1989. 116 p. Dissertação (Mestrado) - Instituto Tecnológico da Aeronáutica. Cap.2, p.13-5.

[14] JOHNSON, G. **Machinery of the mind:** inside the new science of artificial intelligence New York, Times Books, 1986. Cap.6, p.123-40.

[15] LESSER, V. R.; ERMAN, L. E. A retrospective view of the Hearsay-II architecture. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 5., Cambridge, Massachusetts, 1977. **Proceedings.** Palo Alto, Morgan Kaufmann, 1977. p.790-800.

[16] ENGELMORE, R.; MORGAN, T., ed. **Blackboard systems.** Reading, Addison-Wesley, 1988.

[17] POMEROY, B.; IRVING, R. A Blackboard approach for diagnosis in Pilot's Associate. **IEEE Expert**, v.5, n.4, p.39-46, Aug. 1990.

[18] CORKILL, D. D. et al. GBB: a generic Blackboard development system. In: NATIONAL CONFERENCE OF THE AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE, 5., Philadelphia, 1986. **Proceedings.** Menlo Park, AAAI, 1986. p.1008-14.

[19] LÂASRI, H.; MAÎTRE, B. Flexibility and efficiency in Blackboard systems: studies and achievements in ATOME. In: JAGANNATHAN, V. et al., ed. **Blackboard architectures and applications**. San Diego, Academic Press, 1989. p.309-22.

[20] VAN LIEMPD, G. et al. BLONDIE-III. **IEEE Expert**, v.5, n.4, p.48-55, Aug. 1990.

[21] BALZER, R. et al. Hearsay-III: a domain-independent framework for expert systems. In: NATIONAL CONFERENCE OF THE AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE, 1., Stanford, 1980. **Proceedings**. Menlo Park, AAAI, 1980. p.108-10.

[22] ERMAN, L. D. et al. The design and an example use of Hearsay-III. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 7., Vancouver, 1981. **Proceedings**. Palo Alto, Morgan Kaufmann, 1981. p.409-15.

[23] JAGANNATHAN, V. et al., ed. **Blackboard architectures and applications**. San Diego, Academic Press, 1989.

[24] CORKILL, D. D. et al. Unifying data-directed and goal-directed control: an example and experiments. In: NATIONAL CONFERENCE OF THE AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE, 2., Pittsburgh, 1982. **Proceedings**. Menlo Park, AAAI, 1982. p.143-7.

[25] DURFEE, E. H. et al. Coherent cooperation among communicating problem solvers. **IEEE Transactions on Computers**, v.36, n.11, p.1275-91, Nov. 1987.

[26] KORNFELD, W. A.; HEWITT, C. E. The scientific community metaphor. **IEEE Transactions on Systems, Man, and Cybernetics**, v.11, n.1, p.24-33, Jan. 1981.

[27] ENSOR, J. R.; GABBE, J. D. Transactional Blackboards. In: ENGELMORE, R.; MORGAN, T., ed. **Blackboard systems**. Reading, Addison-Wesley, 1988. p.465-73.

[28] ARLABOSSE, F. et al. Blackboard and alternative architectural design for two real-scale industrial applications. In: JAGANNATHAN, V. et al., ed. **Blackboard architectures and applications**. San Diego, Academic Press, 1989. p.433-54.

[29] FEIGENBAUM, E. A. Foreword. In: ENGELMORE, R.; MORGAN, T., ed. **Blackboard systems**. Reading, Addison-Wesley, 1988. p.v-viii.

[30] NILSSON, N. J. **Principles of artificial intelligence**. Palo Alto, Tioga, 1980.

[31] RICH, E. **Artificial intelligence**. 4.ed. Auckland, McGraw Hill, 1986. Cap.3, p.56-60.

[32] TALUKDAR, S. et al. Building large-scale software organizations. In: KOWALIK, J. S.; KITZMILLER, C.T., ed. **Coupling symbolic and numerical computing in expert systems, II**. Amsterdam, North-Holland, 1988. p.129-43.

[33] LUCENA, C. J. P.; TAKAHASHI, E.T. A anatomia de um ambiente de desenvolvimento de software. In: REUNIÃO DE TRABALHO DO PROJETO ESTRA, 5., São Paulo, 1989. **Coletânea de resultados de pesquisa:** outubro/88 - abril/89. São Paulo, SID Informática, 1989. v.1, p.1-46.

[34] FOX, M. S. An organizational view of distributed systems. **IEEE Transactions on Systems, Man, and Cybernetics**, v.11, n.1, p.70-80, Jan. 1981.

[35] SIMON, H. A. **Models of man**. New York, Wiley, 1957.

[36] STEFIK, M. J. et al. Integrating acess-oriented programming into a multiparadigm environment. **IEEE Software**, v.3, n.1, p.10-8, Jan. 1986.

[37] FURTADO, A. L. **Paradigmas de linguagens de programação**. Campinas, EBAI, 1986. /Apresentado à 2. Escola Brasileiro Argentina de Informática, Campinas, 1986/

[38] MULTIPARADIGM languages and environments. **IEEE Software**, v.3, n.1, Jan. 1986. Special issue.

[39] LUCENA, C. J. P. et al. **AIPUSP: ambiente inteligente para o projeto de unidades de sistemas de produção.** In: REUNIÃO DE TRABALHO DO PROJETO ESTRA, 5., São Paulo, 1989. **Coletânea de resultados de pesquisa:** outubro/88 - abril/89. São Paulo, SID Informática, 1989. v.1, p.103-41.

[40] HAYES, R.; SCHLICHTING, R. D. Facilitating mixed language programming in distributed systems. **IEEE Transactions on Software Engineering**, v.13, n.12, p.1254-64, Dec. 1987.

[41] AHUJA, S. et al. Linda and friends. **Computer**, v.19, n.8, p.26-34, Aug. 1986.

[42] LELER, W. Linda meets Unix. **Computer**, v.23, n.2, p.43-54, Feb. 1990.

[43] ZAVE, P. A compositional approach to multiparadigm programming. **IEEE Software**, v.6, n.5, p.15-25, Sept. 1989.

[44] HANSEN, P. B. The programming language Concurrent Pascal. **IEEE Transactions on Software Engineering**, v.1, n.2, p.199-207, June 1975.

[45] HALSTEAD JR., R. H. Parallel symbolic computing. **Computer**, v.19, n.8, p.35-43, Aug. 1986.

[46] SHAPIRO, E. Concurrent Prolog: a progress report. **Computer**, v.19, n.8, p.44-58, Aug. 1986.

[47] CARDOZO, E.; TALUKDAR, S. N. A kernel for distributed problem solving. In: SIMPÓSIO BRASILEIRO EM INTELIGÊNCIA ARTIFICIAL, 4., Uberlândia, 1987. **Anais**. Uberlândia, UFU, 1987. p.267-77.

[48] CARDOZO, E.; TALUKDAR, S. N. D2: a distributed expert system for fault diagnosis. In: IEEE POWER INDUSTRY COMPUTER APPLICATION CONFERENCE, Montreal, 1987. **Proceedings**. New York, IEEE, 1987. p.101-6.

[49] FERNANDES, J. F. M.; CARDOZO, E. **Sistemas especialistas utilizando a arquitetura do quadro negro (Blackboard)**. /Não publicado/

[50] SPECTOR, A. Z. Distributed transaction processing and the Camelot system. In: PARKER, Y., ed. **Distributed operating systems:** theory and practice. New York, Springer, 1987.

[51] SPECTOR, A. Z. et al. **Camelot:** a distributed transaction facility for Mach and the Internet: an interim report. Pittsburgh, CMU, 1987. (Technical Report CMU-CS-87-129)

[52] BISIANI, R. et al. The architecture of the Agora environment. In: HUHNS, M., ed. **Distributed artificial intelligence.** Palo Alto, Morgan Kaufmann, 1987. p.99-107.

[53] STROUSTRUP, B. What is object-oriented programming. **IEEE Software**, v.5, n.3, p.10-20, May 1988.

[54] DOLDER, H. E.; LUBOMIRSKY, E. **Sistema experto para el diseño de bases de datos.** Termas del Rio Hondo, EBAI, 1989. /Apresentado à 5. Escola Brasileiro Argentina de Informática, Termas del Rio Hondo, 1989/

[55] STEFIK, M.; BOBROW, D. G. Object-oriented programming: themes and variations. **The AI Magazine**, v.6, n.4, p.40-62, Jan. 1986.

[56] THE XEROX LEARNING RESEARCH GROUP. The Smalltalk-80 system. **Byte**, v.6, n.8, p.36-48, Aug. 1981.

[57] STROUSTRUP, B. An overview of C++. **Sigplan Notices**, v.21, n.10, p.7-18, Oct. 1986.

[58] WINSTON, P. H.; HORN, B. K. P. **Lisp.** 3.ed. Reading, Addison-Wesley, 1988. Cap.14, p.183-205.

[59] KEENE, S. E. **Object-oriented programming in Common Lisp:** a programmer's guide to CLOS. Reading, Addison-Wesley, 1989.

[60] LARNER, D. L. Factories, objects & Blackboards. **AI Expert**, v.5, n.4, p.38-45, Apr. 1990.

[61] TELLO, E. R. **Object-oriented programming for artificial intelligence:** a guide to tools and system design. Reading, Addison-Wesley, 1989.

[62] VALETTE, R. et al. Approche orientée objet Hood & reseaux de Petri pour la conception de logiciel temps reel. In: JOURNÉES INTERNATIONALES SUR LE GÉNIE LOGICIEL & SES APPLICATIONS, Toulouse, 1988. **Actes**. Toulouse, LAAS, 1988. v.1, p.252-65.

[63] BOOCH, G. Object-oriented development. **IEEE Transactions on Software Engineering**, v.12, n.2, p.211-21, Feb. 1986.

[64] VALENTE, A. S. M. **Extensões orientadas a objetos da linguagem Lisp:** uma implementação do CLOS. São José dos Campos, 1990. 117 p. Dissertação (Mestrado) - Instituto Tecnológico da Aeronáutica.

[65] CARDOZO, E. et al. **DPSK+P:** a distributed problem solving kernel for object oriented applications. /A ser publicado/

[66] MCDOWELL, C. E.; HELMBOLD, D. P. Debugging concurrent programs. **Computing Surveys**, v.21, n.4, p.593-622, Dec. 1989.

[67] JOYCE, J. et al. Monitoring distributed systems. **ACM Transactions on Computer Systems**, v.5, n.2, p.121-50, May 1987.

[68] LEBLANC, T. J.; MELLOR-CRUMMEY, J. M. Debugging parallel programs with Instant Replay. **IEEE Transactions on Computers**, v.36, n.4, p.471-82, Apr. 1987.

[69] BATES, P.; WILEDEN, J. C. An approach to high-level debugging systems (preliminary draft). **SIGPLAN Notices**, v.18, n.8, p.107-11, Aug. 1983. /Apresentado ao ACM SIGSOFT/SIGPLAN Software Engineering Symposium on High-Level Debugging, Pacific Grove, 1983/

[70] BATES, P. Debugging heterogeneous distributed systems using event-based models of behavior. **SIGPLAN Notices**, v.24, n.1, p.11-22, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[71] STONE, J. M. A graphical representation of concurrent processes. **SIGPLAN Notices**, v.24, n.1, p.226-35, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[72] SOCHA, D. et al. Voyeur: graphical views of parallel programs. **SIGPLAN Notices**, v.24, n.1, p.206-15, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[73] CAMPOS, S. V. A.; CARVALHO, O. S. F. Têmis: um núcleo para desenvolvimento de programas paralelos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 10., Vitória, 1990. **Anais**. Vitória, UFES, 1990. p.121-37.

[74] FOWLER, R. J. et al. An integrated approach to parallel program debugging and performance analysis on large-scale multiprocessors. **SIGPLAN Notices**, v.24, n.1, p.163-73, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[75] FORIN, A. Debugging of heterogeneous parallel systems. **SIGPLAN Notices**, v.24, n.1, p.130-40, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[76] TOKUDA, H. et al. A real-time monitor for a distributed real-time operating system. **SIGPLAN Notices**, v.24, n.1, p.68-77, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[77] BLACK, D. L. et al. The Mach exception handling facility. **SIGPLAN Notices**, v.24, n.1, p.45-56, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[78] GENTLEMAN, W. M.; HOEKSMA, H. Hardware assisted high level debugging (preliminary draft). **SIGPLAN Notices**, v.18, n.8, p.140-4, Aug. 1983. /Apresentado ao ACM SIGSOFT/SIGPLAN Software Engineering Symposium on High-Level Debugging, Pacific Grove, 1983/

[79] RUBIN, R. V. et al. Debugging parallel programs in parallel. **SIGPLAN Notices**, v.24, n.1, p.216-25, Jan. 1989. /Apresentado ao ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Madison, 1988/

[80] CHRISTIAN, K. **Sistema operacional UNIX**. Rio de Janeiro, Campus, 1985.

[81] KERNIGHAN, B. W.; RITCHIE, D. M. **The C programming language**. Englewood Cliffs, Prentice Hall, 1978.

[82] JONES, O. **Introduction to the X Window system**. 4.ed. Englewood Cliffs, Prentice Hall, 1989.