UNIVERSIDADE CATÓLICA DE PELOTAS
CENTRO POLITÉCNICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

VirD-GM: Uma Contribuição Para o Modelo de Distribuição e Paralelismo do Projeto D-GM

por
Vanessa Souza da Fonseca

Dissertação apresentada como
requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Orientador: Profª. Drª. Renata Hax Sander Reiser
Co-orientador: Prof. Dr. Adenauer Corrêa Yamin

DM-2008/7-004

Pelotas, agosto de 2008

Dedico... aos meus pais Sérgio e Regina.
Pois estímulo e amor a mim dedicados foram as armas que me levaram a esta conquista.

AGRADECIMENTOS

Em primeiro lugar a Deus, por me conceder à vida, pela força interior que meu deu para que eu pudesse superar as dificuldades e alcançar mais uma conquista.

Ao meu querido irmão “Duda” pela amizade,pelo amor e acima de tudo por sempre acreditar em mim me dando força necessária para que eu pudesse alcançar os meus objetivos.

A minha cunhada Gabi, que soube compreender a cara feia e o mau humor, sempre ao meu lado me dando carinho e amizade.

Ao meu sobrinho Arthur, este doce anjo que me deu muita força para que eu vencesse mais esta etapa.

A minha orientadora e amiga Renata Reiser, pelo suporte, carinho, pela intensa dedicação, não medindo esforços para realizar mais esta conquista. Obrigado!

Ao meu amigo e Co-orientador Adenauer Yamin, pela dedicação, compreensão e carinho, sempre confiando em minha capacidade, e pela certeza da vitória não deixando desanimar.

A todos meus colegas do mestrado, que permaneceram ao meu lado me dando apóio e carinho nas horas mais difíceis, me mostrando que tudo há seu tempo. Em especial a Rosaura, Bianca Martins e André Moraes.

Ao meu colega e amigo Eduardo Moller, pelo constante estímulo e encorajamento.

Ao meu amigo Felipe Munhoz pela atenção, paciência, carinho e super dedicação, pois não mediu esforços para que esta etapa chagasse ao fim.

A minha amiga Eduarda Monteiro “Duda”, que demonstrou uma verdadeira amizade, permitindo que acreditasse no meu sonho.

A secretaria do mestrado “Katinha”, claro além de uma excelente profissional uma grande amiga, pela amizade e carinho que ela me proporcionou durante esta longa caminhada.

A minha amiga Ana Paula Sias, que mesmo longe esteve sempre ao meu lado.

Quero registrar também meu agradecimento á FAPERGS e a CAPES, que na condição de órgãos de fomento á pesquisa subsidiaram o desenvolvimento desta dissertação de mestrado. Este subsídio foi condição indispensável para a conclusão do trabalho.

Agradeço também a todas as pessoas que estiveram ao meu lado, aos meus professores, e todos aqueles que conviveram comigo nesta jornada.

Obrigado de coração a todos vocês!

Não é o desafio que nos deparamos que determina quem somos e que estamos nos tornando, mas a maneira com que respondemos ao desafio. Somos combatentes, idealistas, mas plenamente conscientes, por que ter consciência não nos obriga a ter teoria sobre as coisas: só nos obriga a sermos conscientes. Problemas para vencer, liberdade para provar. "E, enquanto acreditarmos em nossos sonhos, nada é por acaso."

— HENFIL

SUMÁRIO

LISTA DE FIGURAS 7

LISTA DE TABELAS 9

LISTA DE ABREVIATURAS E SIGLAS 10

RESUMO 12

ABSTRACT 13

1 INTRODUÇÃO 14
1.1 Tema 15
1.2 Contexto de Pesquisa 15
1.2.1 Projeto D-GM 16
1.2.2 Middleware EXEHDA 17
1.3 Motivação e Objetivos 17
1.4 Estrutura do Texto 18

2 ESCOPO DO TRABALHO 20
2.1 Modelos em Programação Paralela e Distribuída 20
2.1.1 Máquina de Turing 21
2.1.2 Máquina RAM 23
2.1.3 Modelo BSP 24
2.1.4 Modelo GCM 26
2.2 Ambientes de Programação Paralela e Distribuída 26
2.2.1 DO Builder 26
2.2.2 Java Beans 29
2.2.3 HeNCE 31
2.2.4 Paralex 32
2.2.5 GRADE 34
2.3 Considerações Finais 37

3 VIRD-GM: FUNDAMENTOS 38
3.1 A Noção Intuitiva do Modelo GM 38
3.1.1 Aplicações do Modelo GM na Computação Científica 40
3.1.2 Conceitos Básicos do Modelo GM 42
3.2 Abordagem Visual para o Modelo GM 43
3.2.1 Especificação da Linguagem Visual para o Modelo GM 44

3.3 Ferramenta VPE-GM 46
3.3.1 Descrição da Ferramenta VPE-GM 47
3.3.2 Funcionamento da Ferramenta VPE-GM 49
3.4 *Middleware* EXEHDA: Fundamentos e Tecnologias 50
3.4.1 Organização do EXEHDA 51
3.4.2 Funcionalidades dos Subsistemas do EXEHDA 55
3.5 Considerações Finais 58

4 VIRD-GM: MODELAGEM E IMPLEMENTAÇÃO 60
4.1 Visão Arquitetural da D-GM 61
4.2 Organização da VirD-GM 61
4.2.1 VirD-GM: Principais Subsistemas do EXEHDA Utilizados 62
4.2.2 Processo de Instalação, Configuração e Execução da VirD-GM 63
4.3 Modelagem dos Componentes da VirD-GM 65
4.3.1 Principais Componentes de Software da VirD-GM 65
4.3.2 Diagramas de Classes de Implementação da VirD-GM 66
4.4 Controle do Fluxo de Execução na VirD-GM 68
4.5 Considerações Finais 71

5 VIRD-GM: APLICAÇÕES DE TESTE 72
5.1 Aplicação para o Cálculo do Número π pelo Método de Monte Carlo . . 72
5.1.1 Metodologia de Implementação 72
5.1.2 Resultados Obtidos 76
5.2 Aplicação para Quebra de Senhas pelo Método da Força Bruta 77
5.2.1 Metodologia de Implementação 77
5.2.2 Resultados Obtidos 78
5.3 Considerações Finais 81

6 CONSIDERAÇÕES FINAIS 82
6.1 Principais Contribuições 83
6.2 Outras Contribuições 84
6.2.1 Suporte à Simulação da Programação Distribuída e Paralela com Ênfase no Desenvolvimento de Software Livre 84
6.2.2 Consolidação da Integração entre os Grupos de Pesquisa GFMC e G3PD . 85
6.3 Publicações Realizadas 85
6.4 Trabalhos Futuros 87

REFERÊNCIAS 88

LISTA DE FIGURAS

Figura 2.1 Máquina de Turing. 21
Figura 2.2 Modelo PRAM. 23
Figura 2.3 Modelo BSP 25
Figura 2.4 Ambiente Visual de Programação da Ferramenta DoBuilder 28
Figura 2.5 Ambiente Visual de Programação da Ferramenta MVCASE - EJB . . 30
Figura 2.6 Ambiente Visual de Programação da Ferramenta HeNCE 32
Figura 2.7 Ambiente Visual de Programação da Ferramenta Paralex 33
Figura 2.8 Comunicação de Processos da Ferramenta GRADE 35

Figura 3.1 Modelo GM com Memória e Processos Infinitos. 38
Figura 3.2 Processo elementar d^k, processo Skip e testes computacionais. 42
Figura 3.3 Construtores de Processos Aplicados a Processos Elementares. 43
Figura 3.4 Objetos Parciais Representados no Modelo GM. 43
Figura 3.5 Processo Identidade 45
Figura 3.6 Processo Elementar 45
Figura 3.7 Regra Gramatical para a Construção de um Processo Paralelo 46
Figura 3.8 Componentes da VPE-GM 47
Figura 3.9 Interface do Editor de Processos do Ambiente VPE-GM. 48
Figura 3.10 Interface do Editor de Memória do Ambiente VPE-GM. 49
Figura 3.11 Exemplo de Funcionamento do Ambiente APV-GM. 50
Figura 3.12 Arquitetura de Software do EXEHDA 52
Figura 3.13 ISAM Ambiente Pervasivo Gerenciado pelo EXEHDA 53
Figura 3.14 Organização dos Subsistemas do EXEHDA 54
Figura 3.15 Organização do Núcleo do EXEHDA 55

Figura 4.1 Visão Funcional do Modelo D-GM 60
Figura 4.2 Visão Arquitetural do Modelo D-GM 61
Figura 4.3 Componentes da Camada de Sistemas Básicos da Arquitetura D-GM 63
Figura 4.4 Processo de Verificação de Requisitos do Sistema Básico 63
Figura 4.5 Componentes do Processo de Instalação do EXEHDA 64
Figura 4.6 Seqüênciallização do Processo de Instalação do EXEHDA 64
Figura 4.7 Diagrama de Componentes de Software da VirD-GM 65
Figura 4.8 Diagrama de Classes da VirD-GM 68
Figura 4.9 Grafo Dirigido Característico de uma Aplicação 69
Figura 4.10 Matriz de Adjacência para Controle do Fluxo de Execução na VirD-GM 69
Figura 4.11 Diagrama Processo no VPE-GM 70
Figura 4.12 Código XML correspondente ao Diagrama Processo 70

Figura 5.1 Interpretação Geométrica utilizada no Cálculo do π 73
Figura 5.2 Aplicação πCalc Modelada Graficamente na Ferramenta VPE-GM . . 74
Figura 5.3 Código XML do Arquivo Descritor de Processos da Aplicação *PiCalc* 74
Figura 5.4 Código XML do Arquivo de Configuração de Memória da Aplicação *PiCalc* 74
Figura 5.5 Algoritmo que Implementa o Processo *PiCalc* 75
Figura 5.6 Aplicação *PassBreak* Modelada no VPE-GM 79
Figura 5.7 Código XML do Arquivo Descritor de Processos da Aplicação *PassBreak* 79
Figura 5.8 Código XML do Arquivo de Configuração de Memória da Aplicação *PassBreak* 80

LISTA DE TABELAS

Tabela 3.1 Correspondência entre Memória e Processos no Modelo GM 40
Tabela 3.2 Construção Indutiva da Estrutura Ordenada do Modelo D-GM 41

Tabela 5.1 Avaliação da Aplicação *PiCalc* 76
Tabela 5.2 Estado Final da Memória do Cálculo do π 77
Tabela 5.3 Avaliação da Aplicação *PassBreak* 80

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
BSP	Bulk Synchronous Parallelism
CRCW	Concurrent Read, Concurrent Write
CMOS	Complementary Metal Oxide Semiconductor
CREW	Concurrent Read, Exclusive Write
DRAM	Dynamic Random Access Memory
DSM	Distributed Shared Memory
ECL	Emmiter Coupled Logic
EREW	Exclusive Read, Exclusive Write
EXEHDA	Execution Environment for High Distributed Applications
FDDI	Fiber Distributed Data Interface
FFT	Fast Fourier Transformation
GM	Geometric Machine Model
HeNCE	Heterogeneous Network Computing Environment
HIPPI	High-Performance Parallel Interface
HPF	High Performance Fortran
Mbps	Milhões de bits por segundo
MFLOP	Milhões de Instruções de Ponto Flutuante Por Segundo
MIMD	Múltiplas seqüências de dados
MIPS	Milhões de Instruções Por Segundo
MISD	Múltiplas Seqüências de Instruções, uma Seqüência de Dados
MPI	Message Passing Interface
NFS	Network File System
OLTP	On Line Transaction Processing
PRAM	Parallel Randon Access Machine

PVM	Parallel Virtual Machine
RAM	Randon Access Machine
SE	Sistema de Execução da ferramenta *Xchart*
SIMD	Uma Seqüência de Instruções, Múltiplas Seqüências de Dados
SISD	Uma Seqüência de Instruções, uma Seqüência de Dados
SONET	Synchronous Optical NETwork
SPMD	Simple Program Multiple Data
SR	Synchronizing Resources
SRAM	Static Random Access Memory
TCP/IP	Transmition Control Protocol/Internet Protocol
UMA	Uniform Memory Access
VirD-GM	Virtual Distributed Geometric Machine
Xchart	Tecnologia baseada na Linguagem Xchart
XML	Extensible Markup Language

RESUMO

Este trabalho descreve as principais contribuições da VirD-GM (*Virtual Distributed Geometric Machine Model*) para o modelo de distribuição e paralelismo do Projeto D-GM (*Distributed Geometric Machine Project*). Para disponibilizar as abstrações do modelo GM (*Geometric Machine*) em uma plataforma com suporte à execução distribuída e/ou concorrente, considera-se o *middleware* EXEHDA (*Execution Environment for High Distributed Applications*) como ambiente de suporte à execução. O trabalho possibilitou criar e gerenciar um ambiente de programação paralela e distribuída, bem como promover a execução, sob este ambiente, das aplicações desenvolvidas no ambiente visual VPE-GM (*Visual Programming Environment for the Geometric Machine Model*). Estas aplicações são, por natureza, paralelas e direcionadas ao estudo de algoritmos paralelos para a Computação Científica. O trabalho está centrado na concepção e construção da arquitetura de *software* da VirD-GM, responsável pelo gerenciamento das computações paralelas obtidas pela aplicação de construtores de processos definidos no modelo GM. Neste contexto, esta dissertação não só viabilizou construção da visão estrutural do projeto D-GM como também consolidou sua integração com a visão funcional, caracterizada pela extensão do ambiente VPE-GM, responsável pelo ambiente de desenvolvimento e geração de código para o Projeto D-GM. Dentre as principais contribuições, destacam-se: (i) formalização das noções de concorrência e conflito intermitentes com as noções de comunicação e sincronização de processos, diretamente relacionadas com a estrutura espaço-temporal do modelo GM; (ii) definição compreendendo a modelagem e implementação dos módulos de carregamento, gerenciamento e controle da VirD-GM; (iii) estudo, aplicação e customização dos serviços disponibilizados pelo *middleware* EXEHDA; (iv) implementação das camadas de aplicação, de suporte ao ambiente de execução e de sistemas básicos; (v) controle do fluxo de dados e manipulação das dependências entre as computações concorrentes pelo uso de matrizes de adjacências, incluindo a implementação de barreiras de sincronização, garantindo a correta execução. A prototipação da VirD-GM e a avaliação obtida com o desenvolvimento de aplicações de teste demonstraram a viabilidade da abordagem teórica-prática proposta no Projeto D-GM.

Palavras-chave: Computação Distribuída, Computação Paralela, Modelo de Programação, Arquitetura de Software, Máquina Geométrica.

TITLE: "VIRD-GM: A CONTRIBUTION TO THE MODEL OF DISTRIBUTION AND PARALLELISM OF DE PROJECT D-GM"

ABSTRACT

This research describes the main contributions of the VirD-GM (Virtual Distributed Geometric Machine Model) for the model of parallelism and distribution of the Project D-GM (Distributed Geometric Machine Project). In order to provide the abstractions of the GM model (Geometric Machine) on a platform to support the implementation distributed and / or parallel computations, the middleware EXEHDA (Execution Environment for High Distributed Applications) is considered as the execution environment. The work enabled to create and manage an environment of parallel and directed programming, and promote the implementation, in this environment, of applications developed in the visual environment VPE-GM (Visual Programming Environment for the Geometric Machine Model). These applications are, by nature, parallel and restricted to the study of parallel algorithms for Scientific Computation. The work focuses on the design and construction of the software architecture of the VirD-GM, which is responsible for managing parallel computations obtained by the application of process constructors defined by the GM model. In this context, this research does not only disposes the construction of the structural vision of the project D-GM but also consolidates its integration with the functional vision. It is characterized by an extension of the visual environment VPE-GM, which is responsible for the environment development and code generation for the Project D-GM. Among the main contributions, one may consider: (i) formalization of the concepts of concurrency and conflict intermittent with the notions of communication and synchronization of processes, directly related to the space-time structure of the GM model; (ii) modeling and implementation of the loading, management and control structures of the VirD-GM; (iii) implementation and customization of services provided by the EXEHDA; (iv) construction of the levels of applications, support of execution environment and basic systems; (v) data flow control and manipulation of adjacency matrix related to concurrent computations, including the implementation of barriers of synchronization. The prototyping of VirD-GM and avaliaton achieved through the development of test applications have implemented the viability of theoretical-practical approach proposed in Project D-GM.

Keywords: Distributed Parallel Computation, Parallel Programming Model, Parallel Model Categorization, Geometric Machine.

1 INTRODUÇÃO

A perspectiva de utilizar paralelismo nos sistemas de computação é tão antiga quanto os primeiros computadores. Trabalhos desenvolvidos por Von Neumann, na década de 40, já ponderavam a possibilidade de utilizar algoritmos paralelos para a solução de equações diferenciais.

Entre os primeiros trabalhos envolvendo o paralelismo, com implementação de *hardware*, os registros apontam para o período 1944-1947, quando Stibitz e Williams, nos laboratórios da Bell Telephone, desenvolveram o sistema MODEL V. Formado por dois processadores e três canais de entrada e saída, esse multiprocessador primitivo já constituía um exemplo típico de arquitetura paralela. Nele poderiam ser executados dois programas distintos, bem como seus dois processadores poderiam ser alocados para uma única execução.

Disponível comercialmente há aproximadamente quatro décadas, o paralelismo vem sendo utilizado com sucesso em diversos ramos da atividade humana, o qual vem prestando relevantes serviços à área de pesquisa e desenvolvimento. Historicamente, tem se destacado as soluções que são oferecidas pelo paralelismo quando do processamento numericamente intensivo (B.WILKINSON; ALLEN, 2004).

Na perspectiva de um modelo teórico de computação paralela, o modelo GM (*Geometric Machine Model*) estabelece uma computação como uma transformação de estados, na qual se consideram o fluxo de dados e os recursos (memória e processadores), ambos diretamente relacionados com a idéia de sistemas dinâmicos.

A lógica linear introduzida por Girard (GIRARD, 2003, 1986, 1987; GIRARD; LAFONT; TAYLOR, 1989) tornou explícita as interpretações computacionais relacionadas com recursos, reconhecida como a lógica ciente dos recursos, possibilitando a modelagem da noção dinâmica de computação.

Seguindo como fundamentação a teoria da aproximação sugerida por Scott (SCOTT, 1971), a idéia básica no modelo GM é representar tipos de dados por certos conjuntos parcialmente ordenados denominados domínios, e o programa que executa a computação é representado por funções entre domínios.

Nesta perspectiva, no modelo GM, a noção de concorrência surge na estruturação das regras que constroem os processos, ou ainda, recursos podem ser modelados como objetos de um domínio denominado espaço coerente, e o fluxo de recursos fluindo através dos operadores lineares, tornando compatíveis as interpretações computacionais e denotacionais.

A estrutura ordenada do modelo GM é o espaço coerente de processos computacionais, indutivamente gerado, e definido a partir da escolha do conjunto de proces-

sos elementares rotuladas por posições de um espaço geométrico e do conjunto finito de construtores de processos. Pela completação, todos os processos, inclusive aqueles sem restrição quanto ao tempo de execução ou espaço de memória, são interpretados neste domínio.

Este capítulo tem por objetivo contextualizar o trabalho realizado bem como apresentar suas principais motivações e objetivos. Formado de quatro seções, ao final ele também contempla aspectos da organização do texto como um todo.

1.1 Tema

Esta dissertação está inserida nos esforços de pesquisas do projeto D-GM (*Distributed Geometric Machine Project*). O projeto D-GM fundamenta-se na versão distribuída do modelo GM, indicada por d-GM (*Distributed Geometric Machine Model*) (REISER; COSTA; DIMURO, 2003a, 2004a, 2005a), o qual considera a análise dos originais transfinitos para mostrar que a modelagem de sistemas de computação distribuído pode ser obtido por generalizações de sistemas clássicos. A d-GM considera uma memória global transfinita compartilhada por um conjunto enumerável de modelos GM. A representação no modelo d-GM é construída por uma estrutura matricial obtida pela indexação de pontos do espaço geométrico, consistindo em generalizações da modelagem de sistemas de computação convencionais no modelo GM, representado uma forte e direta relação entre estes sistemas, geralmente tratados de forma isolada na literatura. Este projeto está em desenvolvimento no Grupo de Matemática e Fundamentos da Computação (GMFC) da UCPel e vem pesquisando os diferentes aspectos pertinentes relacionados à execução distribuída e/ou paralela das computações do Modelo GM. Sua atividades iniciaram com o trabalho (REISER, 2002).

De modo mais especifico esta dissertação tem como tema central a proposição de uma arquitetura de software, denominada VirD-GM (*Virtual Distributed Geometric Machine*), para prover uma execução distribuída das aplicações do modelo GM a partir das interfaces gráficas de VPE-GM (*Virtual Programming Environment for Geometric Machine*).

No ambiente VPE-GM, as aplicações na fase de desenvolvimento são modeladas de forma gráfica em um ambiente visual. A computação ocorre a partir da exportação de especificações das aplicações para o ambiente execução. Na fase da execução a arquitetura de *software* denominada VirD-GM irá gerenciar os diferentes requisitos inerentes a uma execução efetivamente distribuída e/ou paralela.

A VirD-GM se vale de um *middleware* para suporte aos mecanismos de distribuição, comunicação e gerência das computações distribuídas. O *middleware* selecionado é denominado EXEHDA (*Execution Environment for High Distributed Applications*) e encontra-se em desenvolvimento no grupo de Pesquisa em Processamento Paralelo e Distribuído (G3PD) da UCPel.

1.2 Contexto de Pesquisa

A proposta deste trabalho está inserida no projeto D-GM e visa a modelagem das abstrações do modelo GM em ambiente de execução distribuída, implementada sob o *middleware* EXEHDA. Estes Projetos definem o contexto de pesquisa e estão resumidos

nesta seção.

1.2.1 Projeto D-GM

O modelo GM (REISER; COSTA; DIMURO, 2002; REISER, 2002) constituído por uma máquina abstrata, com memória possivelmente infinita e tempo de acesso constante, foi concebida para dar semântica aos algoritmos da Computação Científica.

Fundamentada na Teoria dos Domínios, os espaços coerentes foram introduzidos em (GIRARD; LAFONT; TAYLOR, 1989) para prover semântica à Lógica Linear, e estão sendo utilizados para prover semântica denotacional para processos e estados no modelo GM.

A estrutura ordenada que modela os estados no modelo GM é definida pelo espaço coerente de estados computacionais, caracterizando um estado de máquina como uma função do conjunto enumerável de posições de memória ao conjunto de valores de memória.

De forma análoga, a estrutura ordenada modelando os processos do modelo GM está definida pelo espaço coerente de processos computacionais, o qual é indutivamente gerado por um conjunto finito de construtores, a partir da indexação de operações por pontos de um espaço geométrico. Neste contexto, um processo elementar no modelo GM altera uma única posição de memória após a sua execução, identificando a unidade de tempo computacional. Obteve-se assim a noção de computação, concebida como uma transição de estados associada a uma localização espacial.

A estrutura ordenada dos processos é também capaz de modelar, de forma explícita, a concorrência síncrona e o conflito de acesso à memória, modelando computações concorrentes e (ou) não-determinísticas. A construção ordenada do modelo GM é desenvolvida em níveis, onde construtores de processos são representados por funções lineares, denominadas projeções e imersões, e o procedimento de completação garante solução para equações recursivas, incluindo as correspondentes construções parciais.

O modelo GM dispõe, para disciplinar a elaboração de programas, de um ambiente de programação visual denominado VPE-GM (*Visual Programming Environment for the Geometric Machine*), cuja estrutura estimula o aprendizado de técnicas de programação paralela e concorrente baseada numa semântica bidimensional inerente às linguagens visuais, provendo uma representação espaço-temporal da estrutura da memória e dos processos representados no modelo GM.

Salienta-se ainda que, o modelo GM provê interpretação para dois tipos especiais de paralelismo, o espacial, com memória global compartilhada por processos modelados por estruturas matriciais, e o temporal, na versão distribuída do modelo, considerando um conjunto enumerável de modelos GM. Esta construção torna-se interessante, sobretudo porque mostra que a modelagem de sistemas de computação paralela e/ou distribuída pode ser obtida como uma generalização da modelagem de sistemas de computação convencionais. Além disso, pela metodologia utilizada, é intrínseco que estes sistemas nascerem de forma intuitiva, conforme sugerido na literatura, devam ser as construções fundamentadas dentro da teoria das aproximações, ou mais especificamente, na Teoria dos Domínios.

1.2.2 Middleware EXEHDA

Este trabalho buscou de uma solução integrada para suporte à Computação Distribuída, considerando o *middleware* EXEHDA (*Execution Environment for Highly Distributed Applications*) como ambiente de suporte à execução das computações concebidas no modelo da D-GM, facultando assim uma execução efetivamente distribuída e paralela para os códigos gerados pelo mesmo.

O EXEHDA (YAMIN et al., 2005) pode ser entendido a partir de duas grandes perspectivas: (1) do ponto de vista das aplicações da Computação Distribuída, o EXEHDA é o provedor dos serviços que dão suporte às abstrações definidas quando da etapa de desenvolvimento. A interação das aplicações com o meio físico distribuído, através dos serviços disponibilizados pelo EXEHDA proporciona a estas aplicações uma visão integrada da infra-estrutura computacional; (2) do ponto de vista dos equipamentos que compõem o meio físico distribuído, o EXEHDA define as políticas que normatizam a organização dos recursos e os mecanismos necessários para sua gerência.

O EXEHDA faz parte dos esforços de pesquisa do Projeto ISAM (Infra-Estrutura de Suporte às Aplicações Móveis Distribuídas), em andamento na UFRGS. A elevada flutuação na disponibilidade dos recursos, inerente à infra-estrutura computacional moderna, onde não só o hardware exibe capacidades variadas de processamento e memória, mas também as bibliotecas de *software* disponíveis em cada dispositivo, motivaram a adoção de uma abordagem na qual um núcleo mínimo do *middleware* tem suas funcionalidades estendidas por serviços carregados sob demanda. Esta organização reflete um padrão de projeto referenciado na literatura como *micro-kernel*. Isto é possível porque, na modelagem do EXEHDA, os serviços estão definidos por sua interface, e não pela sua implementação propriamente dita.

Os principais serviços fornecidos estão organizados em subsistemas que gerenciam: (a) a execução distribuída; (b) a comunicação; (c) o reconhecimento do contexto; (d) a adaptação; (e) o acesso aos recursos e serviços distribuídos; (f) a descoberta e (g) o gerenciamento de recursos.

Aplicações tanto do domínio da computação em aglomerados de computadores (*clusters computing*), quanto da computação em grade e pervasiva (*grid and pervasive computing*) podem ser programadas e executadas sob gerenciamento do *middleware* proposto. Particularmente neste trabalho foi explorada a computação distribuída e/ou paralela em aglomerados de computadores.

1.3 Motivação e Objetivos

O objetivo geral deste trabalho é propor uma arquitetura de *software* para o projeto D-GM, denominada VirD-GM (*Virtual Distributed Geometric Machine*). A contribuição desta arquitetura é criar e gerenciar um ambiente físico para processamento de aplicações paralelas e/ou distribuídas, bem como promover a execução, sob este ambiente, das aplicações desenvolvidas no ambiente visual de programação do modelo D-GM. Estas aplicações são, por natureza, paralelizáveis e direcionadas ao estudo de algoritmos paralelos para a Computação Científica.

O desenvolvimento da VirD-GM pode ser creditado a três motivações principais:

- Contribuir com uma arquitetura de software para o modelo D-GM, a qual será empregada para análise, modelagem e programação de algoritmos distribuídos e/ou

paralelos da Computação Científica;

- Disponibilizar as abstrações de programações do modelo D-GM em uma plataforma com suporte à execução distribuída e/ou paralela, que faculte processamento em ambientes com elevados níveis de heterogeneidade e escalabilidade;
- Construir a visão arquitetural do projeto D-GM, que tem como foco a concepção e modelagem da VirD-GM, de forma integrada à visão funcional, caracterizada pelo do ambiente visual de desenvolvimento do projeto denominada VPE-GM.

Com base nestas motivações, destacamos como principais objetivos específicos os abaixo relacionados:

- Estudar as noções de concorrência e conflito de processos juntamente com as noções de comunicação e sincronização, tendo como referência a estrutura espaço-temporal do modelo GM;
- Utilizar o middlleware EXEHDA como ambiente de execução paralelo e distribuído para as aplicações do projeto D-GM;
- Conceber as funcionalidades dos módulos da arquitetura VirD-GM, considerando sua operação sobre o *middleware* EXEHDA;
- Modelar a passagem dos parâmetros, os quais caracterizam as aplicações desenvolvidas na ferramenta visual de desenvolvimento do modelo GM (VPE-GM), para a VirD-GM de modo que esta promova a sua execução distribuída e/ou paralela;
- Avaliar a execução de aplicações distribuídas e/ou paralelas, desenvolvidas para o projeto D-GM;
- Consolidar a integração dos grupos de pesquisa em fundamentos da computação (GMFC) e em computação paralela e distribuída (G3PD).

1.4 Estrutura do Texto

O texto desta dissertação é composto por seis capítulos, os quais são fortemente interdependentes e cujos conceitos centrais estiveram, sempre que necessário, interrelacionadas ao longo do texto.

Após breve consideração sobre alguns modelos teóricos para programação paralela e distribuída, no Capítulo 2 discorre-se sobre os trabalhos que contemplam ambientes visuais para desenvolvimento e execução de aplicações distribuídas e/ou paralelas. O estudo destas duas áreas contribuíram para a integração da abordagem teórica-prática, quando da concepção do Projeto D-GM.

No Capítulo 3, são apresentados os fundamentos da VirD-GM bem como a noção intuitiva do modelo GM. Acrescenta-se o estudo da abordagem visual para o modelo GM, centrado, principalmente, na descrição da ferramenta VPE-GM, as suas interfaces e a estruturação dos editores de processos e de memória. Segue-se o estudo do ambiente de suporte à computação distribuída, implementada sob o *middleware* EXEHDA, embasando a concepção, a modelagem e a implementação da execução das aplicações desenvolvidas no ambiente visual para o modelo D-GM.

No Capítulo 4 tem-se a apresentação da concepção da VirD-GM, de acordo com os níveis diferenciados de abstração da modelagem, incluindo os componentes funcionais e lógicos. Mostram-se os principais aspectos referentes a construção dos módulos de tradução e gerenciamento de código produzido no ambiente VPE-GM para a arquitetura de software da VirD-GM. Também é considerada a implementação do fluxo de controle e execução na VirD-GM, baseado na aplicação de grafos de dependência e a implementação da memória global e compartilhada.

A prototipação da VirD-GM como ambiente de suporte ao processamento para computações distribuídas e/ou paralelas associadas ao modelo D-GM está descrita no Capítulo 5. A avaliação das aplicações geradas na interface gráfica da VPE-GM e gerenciadas pela VirD-GM também são descritas no Capítulo 5. Mostra-se que a estratégia para validação está caracterizada pelo paralelismo de controle e de fluxo de dados, considerando-se as aplicações *PiCalc* e *PassBreak*.

Por fim, no Capítulo 6, tem-se as considerações finais, sendo apresentadas as principais conclusões, as contribuições da pesquisa, as publicações já realizadas, e as sugestões de continuidade com indicação de trabalhos futuros.

2 ESCOPO DO TRABALHO

Este capítulo caracteriza o escopo do trabalho, estando centrado em duas principais áreas de estudo: (i) os modelos em programação paralela e distribuída; e (ii) trabalhos que contemplam ambientes visuais de desenvolvimento para computações distribuídas e/ou paralelas. A revisão destas duas áreas contribuíram quando da concepção da arquitetura de software da VirD-GM.

2.1 Modelos em Programação Paralela e Distribuída

Nesta seção vamos discorrer sobre alguns dos principais modelos abstratos de máquinas e suas correspondentes versões paralelas. Também são estudados modelos mais realísticos, introduzindo, na descrição de suas caracterizações, os conceitos e fundamentos para modelagem de computações paralelas e/ou distribuídas.

Seu estudo teve por objetivo central qualificar a discussão das características do modelo GM, enquanto alternativa para o desenvolvimento de aplicações paralelas.

A existência de vários elementos de processamento (memória, número de processadores, arquiteturas da rede de interconexão) torna a definição de um modelo de computação paralela mais complexa que o modelo de computação centralizado. Esta complexidade justifica uma análise de modelos abstratos e realísticos para processamento paralelo e distribuído.

Os conceitos básicos e fundamentos para modelagem de computações paralelas e/ou distribuídas são considerados na revisão dos principais modelos de máquinas. Esta revisão esta baseada em duas abordagens:

- *Modelos abstratos para programação paralela e distribuída*, com ênfase em dois aspectos:
  - expressividade das computações interpretadas pelas diferentes formas da Máquina de Turing, permitindo análise de complexidade de algoritmos, independente dos avanços tecnológicos.
  - influência no desenvolvimento de algoritmos paralelos baseados no modelo PRAM focalizando resultados teóricos ainda presentes nas linguagens de programação atuais.
- *modelos realísticos para programação paralela e distribuída*, que neste trabalho está focado na descrição das características essenciais de dois modelos:

– o modelo BSP (*Bulk Synchronous Parallel Model*) e

– o modelo CGM (*Coarse Grained Multicomputer*);

Os algoritmos desenvolvidos nos modelos BSP e GCM, quando implementados nas máquinas existentes, apresentam resultados de *speedup* (tempo do algoritmo seqüencial dividido pelo tempo do algoritmo paralelo com p processadores) que são compatíveis com os previstos nas análises teóricas.

2.1.1 Máquina de Turing

A máquina de Turing foi concebida pelo matemático Alan Turing em 1936 (BOAS, 1990) e, por sua definição relativamente simples, mostrou-se capaz de provar que se poderia efetuar todas as operações matemáticas como computações. Conforme mostra a Figura 2.1, a máquina de Turing consiste em:

- Uma fita divida em células adjacentes. Cada célula contém um símbolo de algum alfabeto finito. Assume-se que a fita é arbitrariamente extensível para a esquerda e para a direita, e células ainda não escritas estão preenchidas com o símbolo branco.
- Um cabeçote, que pode ler e escrever símbolos na fita e mover-se para a esquerda e para a direita.
- Um registrador de estados (de número finito), que armazena o estado da máquina.
- Uma tabela de ação (ou função de transição) que diz à máquina que símbolo escrever, como mover o cabeçote (E para esquerda e D para direita) e qual será seu novo estado, dados o último símbolo lido na fita e o estado atual.

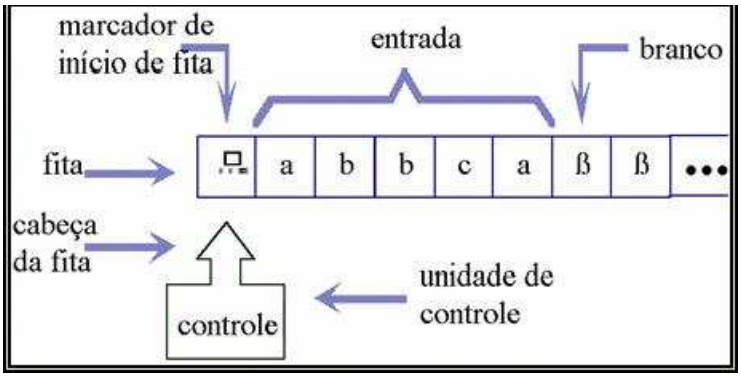

Figura 2.1: Máquina de Turing.

Quando consideram-se as computações clássicas, a máquina de Turing estabelece uma classe estável e máxima de dispositivos computacionais, ou seja, qualquer forma aceitável de expressão das idéias contidas em um algoritmo pode ser, em última instância, equivalente à que se pode obter com as construções propostas na máquina de Turing.

Os estudos envolvendo o desenvolvimento, construção e funcionamento da máquina de Turing inserem-se no paradigma dos dispositivos reconhecedores de linguagens (autômatos). Além de ser simples de descrever, de definir e de entender, a Máquina de Turing apresenta a maior generalidade possível quanto às computações clássicas.

Os modelos determinísticos apresentam a tabela de ação com no máximo uma entrada para cada combinação de símbolo e estado. Em contrapartida, tem-se a versão não-determinística da Máquina de Turing quando a tabela de ação contém múltiplas entradas para uma combinação de símbolo e estado.

Toda Máquina de Turing é capaz de executar uma certa função computável parcial a partir da cadeia formada pelos símbolos do alfabeto, modelando um computador por um programa fixo. No entanto, é também possível codificar a tabela de ação como uma cadeia de símbolos. Ou seja, este modelo de máquina é capaz de simular outras máquinas de Turing, denominado de máquina de Turing universal.

Freqüentemente, diz-se que as máquinas de Turing, ao contrário de autômatos mais simples, são tão poderosas quanto as máquinas reais, e são capazes de executar qualquer operação que um programa real executa. Ou seja, qualquer programa particular executando em uma máquina particular, dada uma entrada finita e uma quantidade de tempo limitado é, na verdade, nada além de um autômato finito determinístico, já que a máquina em que executa pode estar apenas em uma quantidade finita de configurações. Assim uma máquina de Turing poderia de fato ser equivalente a uma máquina real, com possibilidade adicional de modelar uma *quantidade ilimitada de espaço de armazenamento*, ou seja, a diferenciando-se pela habilidade em manipular uma *quantidade ilimitada de dados*. No entanto, dada uma *quantidade finita de tempo*, uma máquina de Turing (como uma máquina real) pode apenas manipular uma quantidade finita de dados.

Um dos problemas mais significativo consiste no tempo de processamento requerido pelas computações reais. Máquinas reais são muito mais complexas que uma máquina de Turing. Freqüentemente, o espaço de estados do autômato finito determinístico equivalente a Máquina de Turing pode crescer exponencialmente. Isto justifica-se pelo fato de que Máquinas de Turing descrevem algoritmos, independente de quanta memória eles utilizam. Há um limite máximo na quantidade e no acesso de memória que qualquer máquina real apresenta (conhecido como Gargalo de Von Neumann), e este limite pode crescer arbitrariamente no tempo.

Programas reais, tais como sistemas operacionais e processadores de texto, são escritos para receber entradas irrestritas através da execução, e portanto não param. Máquinas de Turing não modelam tal *computação contínua* de forma eficiente (apenas porções dela, tais como procedimentos individuais). Outra limitação da máquina de Turing é que ela não modela a organização estrita de um problema específico e existem otimizações computacionais que podem ser executadas baseadas nos índices em memória. Assim, a memória seqüencial da Máquina de Turing dificulta o acesso imediato à informação armazenada em alguma célula que compõe o estado de memória.

2.1.1.1 Versão Paralela da Máquina de Turing

As máquinas de Turing com várias fitas (possivelmente multidimensionais) e várias cabeças generalizam a Máquina de Turing, e facilitam a modelagem formal das computações para os atuais computadores digitais. Esse modelo possui uma unidade de controle, várias cabeças de leitura/escrita atuando sobre uma ou várias fitas, de entrada/saída. Se as cabeças executam a mesma função programa fica caracterizado o paralelismo, com trabalho cooperativo para que a computação ocorra mais rapidamente.

Apesar dos vários trabalhos apontando as potencialidades da Máquina de Turing Paralela (WORSCH, 1997, 1999) e suas contribuições na análise da complexidade e computabilidade de algoritmos, este modelo não é realístico.

2.1.2 Máquina RAM

As máquinas RAM (*Randon Access Machine*) (BOVET; CRESCENZI, 1994), conhecidas como máquina de registradores, apresentam-se como um modelo computacional para analisar algoritmos concretos com desempenho em máquinas reais, pois estão baseadas na diferenciação dos conceitos de programa e da máquina, viabilizando uma redução dos conjuntos de instruções essenciais.

Possuem características compatíveis com as linguagens de alto-nível, com um conjunto infinito de palavras, incluindo também uma memória de endereçamento infinita. Por máquina RAM básica, entende-se uma unidade de controle, um ou mais registradores de acumulação, um registrador contador e uma coleção infinita de registradores na memória. O acumulador e os registradores na memória não são limitados no seu tamanho, e qualquer estado computacional é uma configuração armazenada nos registradores de memória.

A máquina RAM dispõe instruções que influenciam o fluxo de controle, instruções de entrada e saída de dados, instruções de transporte de dados e instruções de desempenho aritmético. Sendo assim, os programas da máquina RAM possuem uma seqüência de instruções rotuladas através da qual cada instrução é selecionada e ajustada ao programa.

A *máquina Norma* (*Number theOretical Register MAchine*) proposta por Richard Bird em 1976, consiste em uma arquitetura de computadores atuais. Possui um número infinito de registradores naturais como memória e três instruções sobre cada registrador: adição e subtração do valor natural um e teste, verificando se o valor natural armazenado é igual a zero.

No modelo de computação seqüencial de Von Neumann, pela memória de acesso aleatório, é possível estabelecer uma relação entre os desempenhos das implementações e dos seus respectivos algoritmos através das medidas de complexidade de tempo baseadas em análises assintóticas. Neste modelo, estas medidas são capazes de refletir corretamente o desempenho dos algoritmos seqüenciais, servindo como referência para as implementações.

2.1.2.1 Versão Paralela do Modelo RAM

O modelo PRAM *(Parallel Random Access Machine)* (BOVET; CRESCENZI, 1994) é uma extensão do modelo seqüencial RAM e o mais conhecido modelo de computação paralela (vide Figura 2.2).

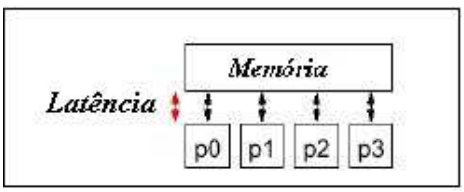

Figura 2.2: Modelo PRAM.

Descrito, essencialmente, por um conjunto de processadores que operam de modo síncrono, sob o controle de um relógio comum. Cada processador é identificado por um índice único e possui uma memória local própria, podendo comunicar-se com os demais processadores através de uma memória global compartilhada. Essa forma de comunicação possibilita o acesso (leitura ou escrita) simultâneo de vários processadores a uma mesma

posição da memória global.

Este modelo apresenta três variações pela possibilidade de acesso concorrente:

1. EREW (*Exclusive Read, Exclusive Write*) não permitindo qualquer acesso simultâneo (leitura ou escrita) a uma mesma posição da memória por mais de um processador;

2. CREW (*Concurrent Read, Exclusive Write*) possibilitando somente a leitura simultânea de uma mesma posição da memória por mais de um processador;

3. CRCW (*Concurrent Read, Concurrent Write*) viabilizando tanto a leitura, como a escrita concorrente em uma mesma posição da memória, por mais de um processador. Para evitar os possíveis conflitos gerados pelo acesso simultâneo à memória, têm-se os seguintes critérios:

   (a) CRCW *comum*, onde todos os processadores devem escrever o mesmo valor;

   (b) CRCW *prioritário*, quando todos os valores escritos, simultaneamente, podem ser diferentes, mas ficará armazenado na posição da memória o valor escrito pelo processador de maior prioridade (assumido como sendo o processador com menor índice);

   (c) CRCW *arbitrário*, como todos os valores escritos, simultaneamente, podem ser diferentes, ocorrendo neste caso uma escolha arbirátia e apenas um, entre todos os processadores, poderá escrever.

O modelo PRAM utiliza uma memória compartilhada e na análise da complexidade do algoritmo para este modelo, não são considerados custos de comunicação. Apesar de sua importância conceitual e teórica, o modelo PRAM não consegue capturar com exatidão a noção do paralelismo. As características não incorporadas ao modelo PRAM durante o desenvolvimento dos algoritmos, tais como custo adicional para referência de memória global e latência, possui grande impacto no desempenho das implementações.

Desde (MINSKY, 1967), os estudos e aplicações mostram o uso das máquinas de registradores como modelos para teoria básica da recursão. Verificou-se também o fato surpreendente de que, a Máquina de Dois Registradores pode simular todo poder computacional de uma máquina universal.

Para tentar solucionar os problemas da Modelo PRAM, consideram-se os chamados modelos realísticos, como BSP e o CGM, que serão descritos nas próximas seções.

2.1.3 Modelo BSP

O modelo BSP (*Bulk Synchronous Parallel Model*) foi proposto por Valiant em 1990 (VALIANT, 1990), sendo um dos primeiros modelos a considerar os custos de comunicação e a abstrair as características de uma máquina paralela em um pequeno número de parâmetros. O principal objetivo deste modelo é servir de modelo ponte entre as necessidades de *hardware* e *software* na computação paralela.

Uma máquina BSP consiste em um conjunto de p processadores com memória local, comunicando-se através de algum meio de interconexão, gerenciados por um roteador e com facilidade de sincronização global.

Um algoritmo BSP é uma seqüência de superpassos separados por barreiras de sincronização, como apresentado na Figura 2.3. Em um superpasso, a cada processador é atribuído além de um conjunto de operações independentes combinando passos de computação que utilizam dados disponibilizados localmente (rodada de computação), um conjunto de passos de comunicação (rodada de comunicação) através de instruções de envio e recebimento de mensagens. Uma mensagem enviada em um superpasso será recebida somente no próximo superpasso.

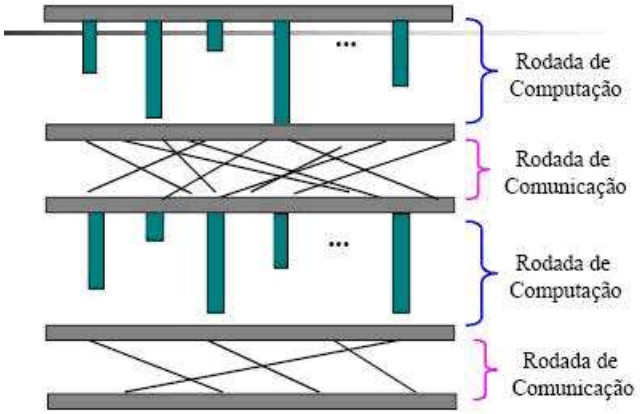

Figura 2.3: Modelo BSP

O modelo BSP possui vários parâmetros:

1. o tamanho n do problema;

2. o número p de processadores disponíveis, cada qual com sua memória local;

3. o tempo l mínimo entre dois passos de sincronização, representando o custo de sincronização, de tal forma que cada operação de sincronização contribui com l unidades de tempo para o tempo total de execução. Este parâmetro l também pode ser visto como a latência de comunicação, para que os dados recebidos possam ser acessados somente no próximo superpasso.

4. a capacidade computacional dividida pela capacidade de comunicação de todo o sistema, ou seja, a razão g entre o número de operações de computação realizadas em uma unidade de tempo e o número de operações de envio e recebimento de mensagens.

Salienta-se que os dois últimos parâmetros, l e g, são usados para computar os custos de comunicação no tempo de execução de um algoritmo BSP.

A capacidade de comunicação de uma rede de computadores está relacionada ao parâmetro g, descrevendo a taxa de eficiência de computação e comunicação do sistema. Através deste parâmetro pode-se estimar o tempo tomado pela troca de dados entre processadores. Se o número máximo de mensagens enviadas por algum processador durante uma troca simples é h, então seriam necessárias até $g * h$ unidades de tempo para a conclusão da troca. Na prática, o valor de g é determinado empiricamente, para cada máquina paralela, através da execução de *benchmarks* apropriados.

2.1.4 Modelo GCM

O modelo CGM (*Coarse Grained Multicomputer*) constitui-se em um multicomputador com granularidade grossa, proposto por Dehne et al (DEHNE; FABRI; RAU-CHAPLIN, 1993). É semelhante ao modelo BSP, no entanto, é definido por apenas dois parâmetros: tamanho n do problema; número p de processadores disponíveis, cada um com uma memória local de tamanho $O(n/p)$ (JUNIOR et al., 2000).

O termo granularidade grossa (*coarse grained*) vem do fato de que o tamanho do problema é consideravelmente maior que o número de processadores, ou seja, $\frac{n}{p} \geq p$.

Um algoritmo CGM consiste de uma seqüência de rodadas (rounds), alternando fases bem definidas de computação local e comunicação global. Normalmente, durante uma rodada de computação utiliza-se o melhor algoritmo seqüencial para o processamento dos dados disponibilizados localmente.

Em uma rodada de comunicação, uma h-relação (com $h = O(n/p)$) é roteada, isto é, cada processador envia $O(n/p)$ dados e recebe $O(n/p)$ dados. Assim, no modelo CGM, o custo de comunicação é modelado pelo número total de rodadas de comunicação e por conseqüência, o custo de um algoritmo CGM é a soma dos tempos obtidos em termos do número total de rodadas de computação local e do número de superpassos, que equivale ao número total de rodadas de comunicação.

2.2 Ambientes de Programação Paralela e Distribuída

Esta seção resume o estudo das propostas para computação distribuída e/ou paralela nas quais as aplicações são concebidas em ambientes visuais de desenvolvimento. Este estudo foi importante quando da concepção da arquitetura de software proposta neste trabalho para a VirD-GM.

2.2.1 DO Builder

O *DOBuilder* surgiu com o intuito de modelar e implementar uma ferramenta de programação visual com objetivo de atender requisitos centrais para programação distribuída e/ou paralela.

A ferramenta *DOBuilder*, como todo ambiente de programação, apresenta recursos ao usuário para ajudar no desenvolvimento de *software*, como a reutilização, extensibilidade, modularização e geração automática de código (MALACARNE; GAYER, 2001).

Outra característica do *DOBuilder* é integrar-se com a programação orientada a objetos focada em sistemas distribuídos explorando o modelo de objetos distribuídos, pois a orientação a objetos distribuídos possui soluções para maior parte dos problemas de reutilização, modularidade e encapsulamento inerentes nas aplicações, sobretudo se as aplicações possuírem grande porte.

Assim as aplicações que são organizadas através da ferramenta DO Builder seguem a filosofia orientada a objetos. Algumas das principais características necessárias para uma ferramenta de programação para ambientes paralelos e/ou distribuídos são disponibilizadas no ambiente desenvolvido. Neste sentido, a ferramenta procura contemplar requisitos considerados oportunos na engenharia de *software*, dentre vários, podemos citar:

- Programação orientada a objetos;

- Programação visual;
- Facilidade de uso;
- Modelo de programação abstrato;
- Flexibilidade;
- Rapidez de desenvolvimento;
- Portabilidade;
- Opções comuns;
- Reutilização de código; e
- Extensão do sistema.

Estes requisitos foram determinados com base na análise das qualidades de ferramentas utilizadas na programação paralela e/ou distribuída avaliadas pelo autor em (MALACARNE; GAYER, 2001).

O ambiente de programação do DO Builder foi implementado na linguagem Java. Utilizou como interface gráfica os controles do pacote Swing. A linguagem visual não define como será implementada a distribuição nem a comunicação entre os objetos. Embora seja utilizado o modelo de grafos que auxilia na programação concorrente, as tarefas de distribuição e comunicação independem da linguagem e são implementadas pelos componentes a critério do programador.

A forma de distribuição a ser adotada depende do ambiente físico de execução utilizado. Para a comunicação, os principais componentes existentes são as portas de comunicação. Há ainda outros componentes, como o par composto pela porta de chamada de serviço (cliente) e porta de serviço (servidor) que implementam uma aplicação cliente/servidor. Quando uma porta de chamada de serviço faz uma requisição à porta de serviço, um novo objeto é criado no servidor para tratar a requisição que chega. Assim como estes componentes, outros podem ser criados e adicionados à ferramenta. Por exemplo, caso haja a necessidade de uma porta de comunicação com um protocolo diferente dos que já foram implementados, basta desenvolvê-la como um componente JavaBean e adicioná-la à ferramenta.

A especificação existente no *DOBuilder* acompanha o paradigma da programação concorrente explícita, desta forma o programador fica com a responsabilidade de realizar a decomposição paralela, sincronização e comunicação. Esta necessidade de gerenciar a concorrência explicitamente constitui uma das principais motivações para uso da programação visual, pois a mesma reduz os esforços de codificação. As aplicações mais comum desenvolvidas pelo *DOBuilder* são as aplicações cliente-servidor. Estas aplicações são utilizadas na chamada remota de método e na criação remota de objetos, ou seja, ocorrem sempre com objetos localizados em diferentes endereços trabalhando com sistemas de comunicação de baixo nível, como por exemplo *Sockets*.

O *DOBuilder* também pode ser utilizado para o desenvolvimento dos componentes de comunicação para componentes de software que estejam sendo construídos em diferentes ferramentas.

Desta forma o *DOBuilder* é uma ferramenta que pode tanto ser aplicada no desenvolvimento de aplicações cliente-servidor completas como de componentes de comunicação. A ferramenta proporciona uma grande flexibilidade, possibilitando que o programador possua um bom controle sobre seus programas.

2.2.1.1 Estrutura da Ferramenta *DOBuilder*

A complexidade já existente na programação seqüencial cresce em complexidade quando se trata de programação distribuída e/ou paralela. Com intuito de minimizar o esforço despendido pelo programador o ambiente de desenvolvimento do DO Builder foi organizado conforme Figura 2.4 a relação das funcionalidades do ambiente e sua organização foram feitos em (MALACARNE; GAYER, 2001). E seus principais componentes são:

- barra de botões contendo as funções dos menus mais acessadas;
- barra de ferramentas com os componentes utilizados no programa visual;
- seção de projeto, contendo a hierarquia dos objetos da aplicação;
- seção para a edição visual da estrutura das aplicações;
- seção para a edição textual do código dos objetos da aplicação;
- janela para a edição das propriedades e métodos dos eventos dos componentes.

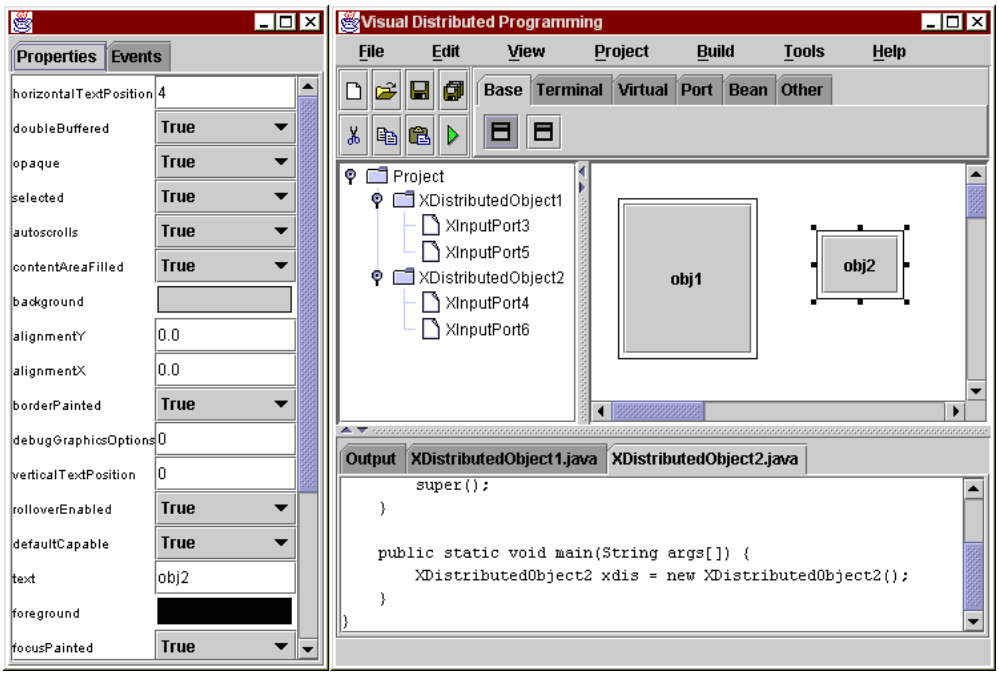

Figura 2.4: Ambiente Visual de Programação da Ferramenta DoBuilder

No DOBulder, os desenvolvedores dos componentes de software possuem facilidades para reutilização dos seus códigos. Como exemplo de aplicação que o *DOBuilder* constrói, pode-se citar qualquer tipo que faça acesso a banco de dados, transações de comércio eletrônico, bate papo, etc. O ambiente de programação foi implementado na linguagem Java.

2.2.2 Java Beans

A ferramenta *JavaBeans* apresenta uma interface de programação que possibilita criar componentes reutilizáveis sem depender da plataforma de destino da aplicação. Pode-se fazer uso de ferramentas de desenvolvimento diversas com o modelo *JavaBeans*, e os componentes podem ser combinados em *applets*, aplicações e outros componentes de software, os quais são reutilizados considerando à necessidade dos sistemas (PRADO; LUCREDIO, 2001). Os componentes do *JavaBeans* são conhecidos como *BEANS* e são definidos da seguinte forma:

"Um *JavaBean* é um componente de *software* reutilizável que pode ser manipulado visualmente em uma ferramenta de programação".

As classes disponíveis no ambiente *Javabeans* não possuem muitas diferenças de outras classes Java, portanto podem ter tratamentos iguais quando os desenvolvimento. Os componentes Java Beans devem seguir regras de padronização para que seja facilitada sua manipulação à ferramenta de programação visual. Dentre as muitas características do ambiente *JavaBeans* destacam-se, principalmente, a reutilização de componentes de software como classes, pacotes, interfaces, exceções, suporte a reutilização de *APIs*, entre outros.

A tecnologia *Enterprise Java Beans* é um modelo de servidor de componentes para JAVA. Sustenta construção de aplicações *multi-tier*, que usualmente manipulam serviços de gerenciamentos de transações, segurança, conectividade e acesso a banco de dados. A tecnologia *EJB*, dispõe de regras de negócios implementada nos componentes *beans* que estão vagos no servidor *EJB*, permanecendo independe da plataforma (PRADO; LUCREDIO, 2001). Os componentes *beans* dispõem de duas interfaces de acesso:

- *Interface Home* é utilizada como ponto inicial de contato para aplicação de clientes. Nesta interface estão alojados os métodos para localizar, gerar e transferir instância de um componente *bean*. Quando uma aplicação cliente solicita um componente *bean*, o servidor devolve uma referência para o objeto criado na Interface Home.

- *Interface Remote* é responsável pelos métodos das regras de negócios do componente *bean*, sendo assim ela aceita a chamada da aplicação Cliente, transmitindo a execução para uma instância do componente *bean* e devolve uma referencia para o objeto que cria a interface Remote.

A figura 2.5 apresenta um exemplo de uma tela da ferramenta visual *MVCASE* com criação de um componente *bean*. Nesta Figura a interface *cliente* estende a interface *EJObject* do pacote EJB, e propõe regras de negócios do componente.

2.2.2.1 Características dos Componentes *Java Beans*

Os *beans* alteram funcionalidades que sustentam as características peculiares que diferem um *bean* de outro são elas: introspecção, persistência, propriedades, métodos, personalização eventos.

As propriedades são constantes e possuem a capacidade de serem personalizadas em relação ao aspecto e o comportamento. Um *bean* como qualquer outra classe, contém um conjunto de métodos, os quais podem ser chamados por outros componentes (LINHALIS, 2000). Um *bean* também dispara eventos. Desta forma, o componente tem a capacidade de comunicar outros componentes que alguma coisa importante ocorreu.

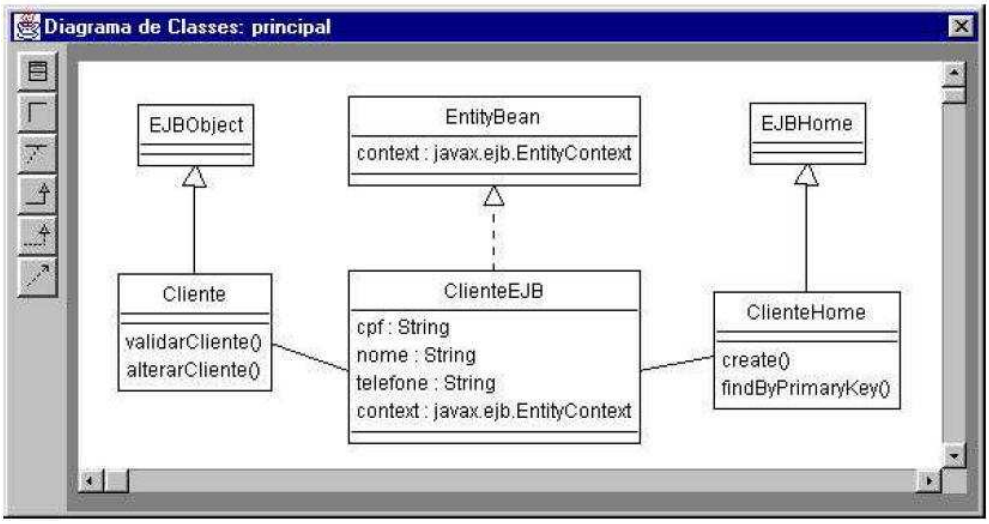

Figura 2.5: Ambiente Visual de Programação da Ferramenta MVCASE - EJB

Os Eventos são mecanismos capazes de difundir notificações de mudanças de estado entre o objeto de origem e o objeto de destino, sendo assim os eventos proporcionam um mecanismo para inserção entre componentes.

O modo de guardar e recuperar objetos do Java é denominado serialização. Sendo assim, Java faculta que os objetos sejam salvos na forma de um *stream*, ou seja, uma seqüência de *bytes* que representam os objetos. A maneira de originar a serialização é bem simples, é necessário apenas que o objeto implemente a interface serializável. Os *beans* possuem a obrigação de segurar o armazenamento. Quando um *bean* é usado, as informações referentes ao seu estado são configuradas em tempo de projeto. As informações precisam ser guardadas e após serem retomadas para fornecer a mesma aparência e comportamento a cada *bean*, quando da inicialização do programa. Geralmente, um *bean* deve acumular as suas propriedades. Desta maneira para ser considerado um *bean*, uma classe deve ser serializável.

Existem algumas vantagens na serialização de objetos Java, como:

- A portabilidade fundamental para objetos remotos: desta forma pode ser gerado um objeto em uma máquina Windows, ser serializado e permitido através de uma rede até uma máquina UNIX, que poderá ser corretamente reconstruído. Desde modo não é preciso preocupar-se com as várias formas de representações de dados, com ordem dos *bytes* e outros diversos detalhes. Esta portabilidade possui uma grande importância para os agentes móveis, pois permitem migrar sem possuir o conhecimento das características dos computadores onde entram em execução.

- Suporte a persistência: a classe somente será armazenada de forma constante se ela for serializável. Assim, uma classe serializável pode realizar o papel de banco de dados para uma aplicação.

Na programação baseada em componentes é bom salientar que é normal usar uma ferramenta visual para conceber a aplicação. Na maneira visual de gerar programas é necessário saber configuração das propriedades dos objetos em tempo de programação, assim esta forma de programar necessita que um componente exponha suas propriedades

para que possam ser lidas e manipuladas. Algum tempo atrás, um componente de *software* teria que publicar a definição de sua API em arquivos diferentes, com o surgimento da introspecção isso não é mais necessário. O modo de introspecção, permite que uma classe mostre as variáveis e métodos de outras classes, desta forma torna mais facilitado o desenvolvimento da capacidade social nos agentes.

A máquina virtual Java carrega classes em tempo de execução com emprego do recurso *Class Loading*. Para tornar possível o carregamento dinâmico o código das aplicações e as classe referenciadas por elas devem atender os seguintes aspectos:

- Uma aplicação serializada deve conter suas classes, assim como qualquer classe referenciada;
- As classes de uma aplicação são carregadas através do *CLASSPATH*. O *CLASSPATH* é uma variável de ambiente que contém o diretórios e repositórios JARs;
- As classes de uma aplicação são carregadas a partir de um servidor WEB ou repositório com funções análogas.

Um exemplo de uso dos beans é o emprego dos mesmos na construção de agentes para consulta de bancos de dados distribuídos.

A característica da serialização facilita persistência e permite a implementação de agentes móveis, com a facilidade de verifique de uma máquina para outra. Java possui uma segurança que determina restrições para classes carregadas, não permitindo que as aplicações possuam uma liberdade irrestrita (LINHALIS, 2000). Este fato aumenta o nível de segurança do sistema, pois os acessos aos recursos dos equipamentos precisam ser autorizados.

2.2.3 HeNCE

A ferramenta HeNCE (*Heterogeneous Network Computing Environment*) (BEGUELIN et al., 2006) foi desenvolvida para gerar código *PVM* e possui facilidades para balanceamento carga e visualização da execução das aplicações. A Figura 2.6 apresenta a interface da ferramenta HeNCE. À esquerda, tem-se o editor de aplicações. À direita estão a estrutura da máquina virtual, uma janela com mensagens do ambiente e as saídas das taks e, abaixo, o histórico da utilização dos processadores.

2.2.3.1 Modelo de Programação

Representa-se uma aplicação em HeNCE como um grafo dirigido. O grafo na ferramenta HeNCE não é orientado ao fluxo de dados, diferentemente de outras ferramentas existentes, de modo que a execução dos nodos que compõem a aplicação segue o fluxo de controle. O controle vai passando de um nodo para outro à medida que os processos iniciam e encerram a sua execução. Quando um nodo chega ao fim da execução, ele deixa de existir e o próximo nodo no grafo inicia a execução. Se um nodo deve receber dados de mais de um nodo, ele irá começar a executar somente quando todos seus precedentes já tiverem terminado as suas computações.

A linguagem visual da ferramenta HeNCE contempla representação para diveros tipos de nodos:

- Nodo de computação: elemento básico da computação, contém o código do usuário;

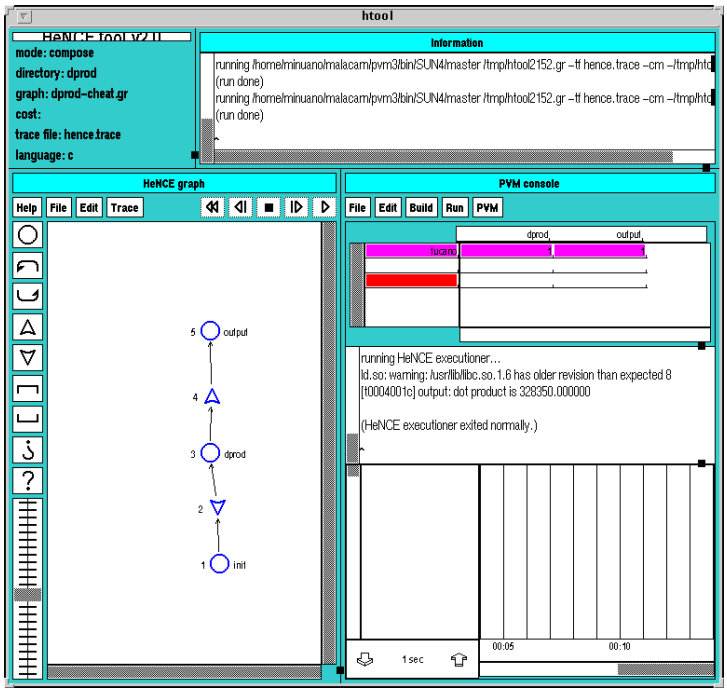

Figura 2.6: Ambiente Visual de Programação da Ferramenta HeNCE

- Nodos *loop end* e *loop begin*: repetem a execução de uma determinada parte do grafo;
- Nodos *fan end* e *fan begin*: criam estruturas paralelas. Replicam o subgrafo presente entre eles e avaliam as replicações em paralelo;
- Nodos *conditional end* e *conditional begin*: conforme o valor da expressão, executam ou não o subgrafo contido entre os nodos condicionais;
- Nodos *pipeline end* e *pipeline begin*: criam uma estrutura pipeline.

A ferramenta HeNCE realiza gerenciamento de mensagens. Sendo assim, o usuário não precisa implementar a comunicação através de chamadas de envio e recebimento de mensagens. Na anotação dos nodos de computação são declarados os dados de entrada, de saída e a chamada à função computada pelo nodo.

Os processos são criados automaticamente pela ferramenta, liberando o programador do uso explícito de rotinas do tipo *spawn* e *fork*. Os nodos de computação básica são criados conforme as construções existentes no grafo (BEGUELIN et al., 2006). A escolha das máquinas onde os processos irão rodar é determinado em parte por uma matriz de custos. A matriz de custos relaciona os hosts componentes da máquina virtual *PVM* com os nodos do grafo. Para cada par (máquina, nodo) é associado um valor numérico. Quanto maior esse valor, maior será a chance do processo correspondente ao nodo naquela máquina. Com isto fica caracterizado realizar um escalonamento, que leva em conta as características do código dos processos e das arquiteturas das máquinas.

2.2.4 Paralex

Paralex é um ambiente de programação que utiliza gráficos para definir, editar, executar e depurar aplicações paralelas. As características principais que diferenciam o

ambiente Paralex dos demais é o suporte a tolerância a falhas e balanceamento dinâmico de carga (DAVOLI et al., 1996). O Paralex foi criado por pesquisadores da Universidade do Texas e da Universidade de Bolonha na Itália Estados Unidos.

O modelo de programação da ferramenta é baseado no fluxo de dados estático. O programa é representado em um grafo dirigido, onde os nodos correspondem à computação (funções, procedimentos, programas) e os arcos indicam o fluxo de dados (com tipos definidos, vide a Figura 2.7).

O modelo de programação da ferramenta Paralex contém propriedades importantes: representação explícita do paralelismo, um número pequeno de abstrações que facilitam o apredizazado por parte de programadores já familiarizados com o modelo seqüencial de programação, tolerância a falhas e reutilização de código. Há quatro tipos de nodos na linguagem visual de Paralex: nodos de computação, filtros (particionamento para paralelismo de dados), subgrafos e nodos cíclicos (um nodo executa enquanto determinada condição for verdadeira).

A especificação lógica da aplicação, é gerado um arquivo contendo a composição da máquina virtual. Para implementar a tolerância a falhas e a conversão de tipos entre plataformas diferentes é utilizado o sistema ISIS (DAVOLI et al., 1996). A heterogeneidade, com respeito aos códigos executáveis, é conseguida através da compilação remota dos arquivos nas diversas plataformas e armazenando os arquivos num sistema de arquivos acessível por todas as máquinas.

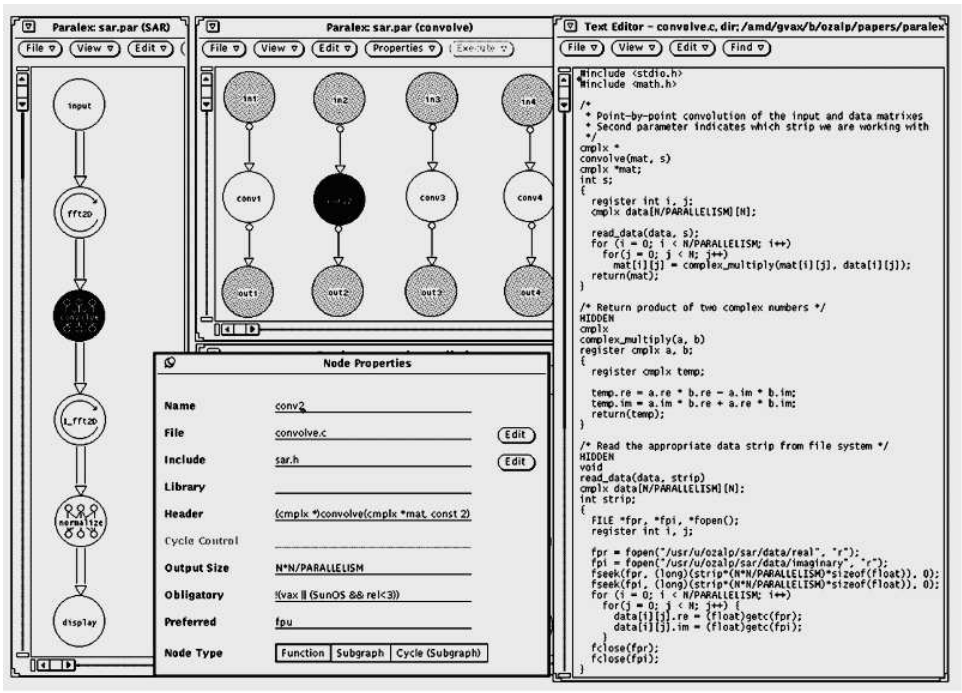

Figura 2.7: Ambiente Visual de Programação da Ferramenta Paralex

A tolerância a falhas é implementada com base no nível k, isto é, se uma aplicação irá rodar com nível k de tolerância a falhas, isto significa que cada processo será replicado em k+1 máquinas distintas, garantindo o sucesso da computação mesmo que ocorram k falhas. A técnica básica utilizada é a replicação passiva, implementada pela ferramenta ISIS. Cada nodo que requer tolerância a falhas é instanciado como um membro de um grupo que contém as réplicas do nodo. Um dos membros do grupo é o coordenador que

em caso de falha é automaticamente substituído por uma das cópias. Outro diferencial da ferramenta Paralex é o balanceamento dinâmico de carga. As decisões são baseadas em heurísticas simples, tentando colocar na mesma máquina os processos que possuem dependência de dados entre si.

O sistema de administração do balanceamento de carga dinâmico é composto por processos *daemons* (um por máquina) e um processo de controle localizado na máquina onde a aplicação foi disparada. Os processos *daemons* monitoram a carga das máquinas analisando o tamanho da fila de processos do sistema operacional local (DAVOLI et al., 1996). As características de depuração do Paralex consistem de dois elementos: monitoração e controle. O elemento de monitoração é usado para informar visualmente o programador sobre o estado da execução do programa no próprio grafo onde o programa foi desenvolvido. O controle é o elemento que faz a parte de baixo nível da visualização, coletando dados e agindo sobre os processos monitorados. A visualização não apresenta facilidades para a avaliação do desempenho.

2.2.5 GRADE

O ambiente GRADE (*GRaphical Application Development Environment*) é formado por conjunto de ferramentas desenvolvidas em diferentes instituições e procura se constituir num ambiente completo de desenvolvimento (P.KACSUK et al., 1997). Suas principais caracteristicas do ambiente são:

- interface visual para definir todas as atividades paralelas da aplicação;
- os programadores não precisam conhecer a sintaxe de baixo nível do sistema de trocas de mensagens. O ambiente GRADE gera todas as chamadas das bibliotecas de trocas de mensagens automaticamente, conforme o gráfico da aplicação vide Figura 2.8;
- a compilação e distribuição dos códigos executáveis é realizada automaticamente no ambiente heterogêneo;
- as informações de depuração e monitoração são apresentadas ao usuário diretamente no gráfico, durante a depuração on-line ou na visualização *post-mortem* do arquivo de*trace*.

As ferramentas que constituem o ambiente GRADE são as seguintes:

- **GRAPNEL** (*GRAphical Process's NEt Language*): é a linguagem de programação visual da ferramenta. Desenvolvida no LPDS (*Laboratory of Parallel and Distributed Systems*), no Instituto de Pesquisa em Computação e Automação da Academia Húngara de Ciências em Budapeste, na Hungria;
- **GRED** (*GRaphical EDitor*): editor gráfico que suporta a linguagem GRAPNEL. Foi desenvolvido no LPDS em cima do sistema de janelas X Window. A linguagem de programação usada foi C++;
- **GRP2C**: pré-compilador que traduz o programa especificado visualmente em código C pronto para ser compilado;

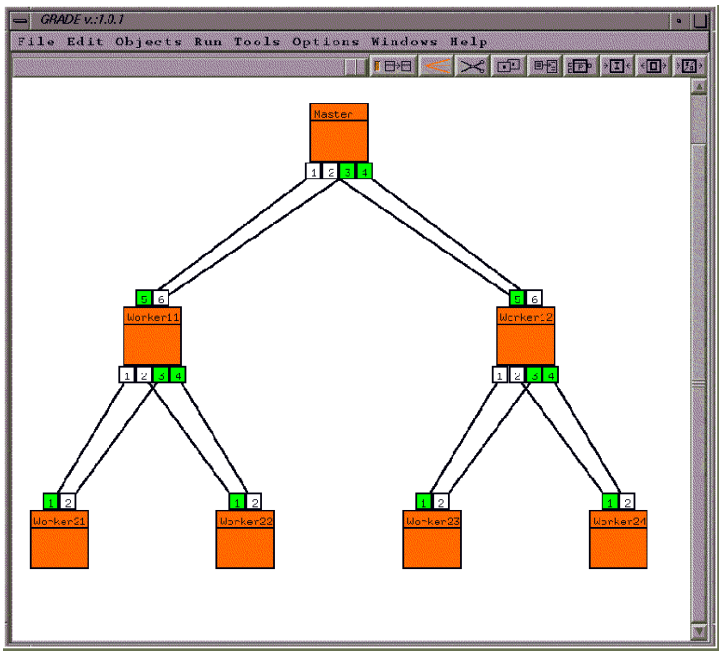

Figura 2.8: Comunicação de Processos da Ferramenta GRADE

- **Tape/PVM**: ferramenta de monitoração para gerar um arquivo de trace durante a execução de uma aplicação PVM. Construído no IMAG, França;
- **DDBG** (*Distributed DeBuGger*): depurador distribuído, criado na Universidade Nova de Lisboa, Portugal;
- **PROVE**: ferramenta de visualização usada para analisar e interpretar a informação contida no arquivo de trace e apresentá-la ao programador, graficamente. Desenvolvida no LPDS.

O ciclo de desenvolvimento de um programa no ambiente GRADE é resumido da seguinte forma: o programador usa a ferramenta GRED para editar um programa com a linguagem gráfica GRAPNEL. Ao final desta fase, o editor GRED gera um arquivo GRP com a descrição do programa. Este arquivo possui as informações necessárias para que a aplicação seja editada posteriormente, bem como processada para gerar um programa em C com chamadas a rotinas da biblioteca PVM. A geração dos programas em C é tarefa do pré-compilador GRP2C que além desses arquivos gera um arquivo Makefile e um arquivo de Referências Cruzadas (relaciona as linhas dos programas textuais com os elementos gráficos dos programas na linguagem GRAPNEL). A geração do código C é automatizada e feita em todas as estações do *cluster* heterogêneo. Depois desta fase, o código está pronto para a geração dos arquivos executáveis através de um compilador C tradicional.

Vários arquivos C são gerados: para cada processo da janela de aplicação e para um processo extra chamado servidor. Esses arquivos fontes contêm chamadas à biblioteca GRAPNEL, que encapsula as rotinas do ambiente PVM. O processo servidor começa a executar quando a aplicação inicia e é responsável pela criação e administração dos processos definidos pelo usuário. Depois dos executáveis já estarem disponíveis, o programa paralelo pode ser executado ou em modo de depuração ou de rastreamento.

Em modo de depuração, o DDBG controla a execução do programa dando opções para estabelecer pontos de parada e/ou executa passos a passo. No modo de rastreamento, um arquivo de *trace* é gerado contendo todos eventos escolhidos pelo usuário para serem investigados. Estes eventos são visualizados pela ferramenta de visualização gráfica PROVE que assiste o programador na busca de gargalos no desempenho dos programas.

2.2.5.1 Modelo de Programação

A linguagem visual do ambiente GRADE é a linguagem GRAPNEL, sendo GRED o editor utilizado na edição dos programas nesta linguagem. As principais características da linguagem GRAPNEL podem ser resumidas no que se segue:

- GRAPNEL é uma linguagem híbrida, isto é, suporta gráficos no processamento das atividades paralelas e texto no código seqüencial dos processos. A linguagem textual suportada é C;
- os programas são baseados no paradigma de passagem de mensagens.

A linguagem suporta desenvolvimento *top-down* e divide o programa em três níveis de hierarquia:

- no nível mais alto, visualizado na janela da aplicação, é descrito graficamente o plano geral da aplicação com respeito à comunicação entre os processos. Devem ser definidos graficamente os processos, os grupos de processos, as portas de comunicação e as conexões entre os processos. A descrição da funcionalidade desses elementos, entretanto, não aparece neste nível;
- no nível intermediário são definidas graficamente as operações de envio e recebimento de mensagens, bem como outras estruturas do programa, dentro do código do processo. Existe uma janela para cada processo que descreve a sua estrutura interna;
- no nível mais baixo os códigos textuais das estruturas dos processos são editados.

Os processos são unidades simples ou membros de um grupo. Um grupo é uma coleção ordenada de processos, onde cada processo possui um número dentro do grupo. Tanto processos como grupos são definidos graficamente como retângulos.

Os grupos podem ser usados para especificar uma operação de comunicação coletiva (*multicast*, por exemplo) ou como um mecanismo de abstração para suportar o projeto estruturado em nível de processo, isto é, os processos podem ser inseridos em um grupo e serem gerenciados como um processo único. Como os grupos de processos podem ser aninhados, eles suportam projeto hierárquico. Os processos podem ainda ser estáticos ou dinâmicos (P.KACSUK et al., 1997).

Os processos estáticos são criados quando a aplicação se inicia e encerram somente quando a aplicação também termina. Para permitir a criação dinâmica de processos, a linguagem visual oferece um símbolo especial chamado DPS (*Dynamic Process Structure)* para ser usado no nível interno do processo.

Os processos dinâmicos são disparados por outros processos em tempo de execução em qualquer posição do fluxo de controle, mas somente de uma maneira estruturada. Isto significa que os processos criados existem somente em seu espaço local,

podendo se comunicar apenas com o pai e entre si. O pai não pode terminar enquanto todos os filhos também não tiverem terminado a execução.

A comunicação entre os processos pode ser ponto-a-ponto ou em grupo, e bloqueante ou não-bloqueante. A comunicação sempre é feita através das portas que pertencem aos processos ou aos grupos conectados por canais. Para assegurar que a forma dos dados transmitidos são compatíveis tanto no transmissor como no receptor da mensagem, cada porta tem o seu próprio protocolo.

2.3 Considerações Finais

Foram caracterizados neste capítulo, as áreas de estudo que constituem o escopo desta dissertação, e apresentados os principais modelos para programação paralela e distribuída, bem como projetos que contemplam ambientes visuais de desenvolvimento direcionados á construção de aplicações distribuídas e/ou paralelas.

O estudo resumido neste capítulo constituiu base conceitual e tecnológica para os capítulos que se seguem, particularmente o que diz respeito à modelagem da VirD-GM.

3 VIRD-GM: FUNDAMENTOS

O estudo resumido neste capítulo teve como finalidade central explorar os aspectos de fundamentação necessários a concepção da VirD-GM.

O mesmo está organizado em 3 seções; (*i*) a noção intuitiva do modelo GM (REISER, 2002), a qual fundamenta-se na idéia de simples compreensão por se tratar de um modelo para interpretar o comportamento de programas concorrentes e não-determinístico baseado na idéia de processo; (*ii*) a abordagem visual para o modelo GM, caracterizada pela ferramenta VPE-GM que apresenta como principais componentes os Editores de Processos e de Memória, os quais compõem o ambiente de desenvolvimento da VirD-GM; (*iii*) o *middleware* EXEHDA (YAMIN et al., 2005) considerando os fundamentos científicos e tecnológicos para suporte á execução das computações geradas na VPE-GM.

3.1 A Noção Intuitiva do Modelo GM

As definições básicas no modelo GM são obtidas por uma construção ordenada, definida por uma estrutura matemática fundamentada na Teoria dos Domínios. Neste contexto, os conceitos básicos como memória e processos são construídos a partir de um espaço geométrico, caracterizando pela construção geométrica a *indexação da memória e dos processos*. Esta construção geométrica considera apenas a disposição dos pontos no espaço geométrico, sem analisar suas propriedades métricas. Ou seja, o conjunto S de estados e o conjunto $\mathcal{P}$ de processos são, respectivamente, definidos pelos conjuntos de valores e de ações computacionais, ambos rotulados por elementos dos conjunto I representando posições do espaço geométrico.

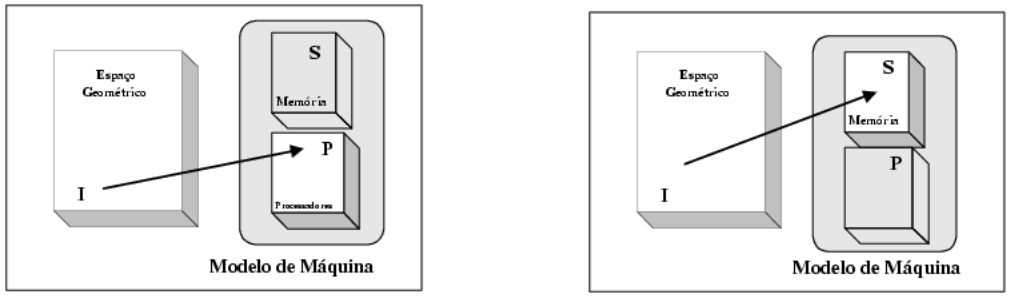

Figura 3.1: Modelo GM com Memória e Processos Infinitos.

Conforme mostra o esquema da Figura 3.1(a), os pontos do espaço geométrico podem ser entendidos como posições de memória e, portanto, ao se considerar um conjunto I possivelmente infinito enumerável , representando posições do espaço geométrico, a modelagem GM contempla interpretação para estados sem restrição quanto ao seu tamanho - tem-se um modelo para *memória infinita*.

Por outro lado, a noção dinâmica de processo está relacionada com transformações entre estados de computação. Compatível com a idéia de que a memória não é necessariamente limitada, a máquina GM modela processos sem restrição quanto ao seu tamanho - um modelo para *processos infinitos*, Figura 3.1(b).

Quando, a partir de um estado inicial cada transformação alcança um estado final, o tempo de execução associado a tal transformação é indicado por um número infinito enumerável de unidades de tempo computacional (utc), ou ainda, tem-se uma *discretização da computação*. Assim, as sucessivas computações mostram a evolução do sistema, e são interpretadas na construção temporal do modelo GM. O modelo de máquina que se está estudando contempla interpretação tanto para estas transformações como para àquelas que não possuem restrição quanto ao seu tempo de execução - um modelo para *computação infinita*.

Um processo computacional elementar é visto como um conjunto (enumerável) de funções que, após execução, altera apenas o valor de uma posição em cada estado de computação. Todo processo que, após ou durante execução, altera mais de uma posição em cada estado computacional foi construído no modelo GM por sucessivas aplicações de construtores (produtos paralelo e seqüencial e pelas somas determinística e não-determinística) sobre processos elementares - caracterizando um modelo de *construção indutiva*, conforme resumo na Tabela 3.1.

As construções obtidas pela aplicação do produto paralelo determinam transformações aplicadas a diferentes posições de memória mas executadas simultaneamente, caracterizando as *computações síncronas*. Na estrutura dual, tem-se as transformações que geram conflito de acesso à memória, indicando *computações não-determinísticas*. Da composição de construtores resultam *computações seqüenciais*, e , quando da aplicação de somas determinísticas, o controle do fluxo de dados é obtido pelo uso de testes, nas versões universal ou existencial.

A estrutura indutiva e ordenada do modelo GM é obtida em níveis, denotados por $\mathbb{D}_n$ na Tabela 3.1, e denominados subespaços coerentes. Cada nível $\mathbb{D}_n$ corresponde ao domínio de processos finitamente gerados, seja pela reconstrução dos objetos já definidos nos níveis anteriores, seja pela construção de novos objetos, executados em $2^n\ utc$.

Baseada nas idéias apresentadas nos parágrafos anteriores é construída a estrutura do modelo. No contexto deste trabalho, restringe-se a noção fundamental de processo computacional elementar, para processos uni-dimensionais. Entretanto, esta noção pode ser estendida ao se considerar processos computacionais elementares bi-dimensionais, como por exemplo modelando uma estrutura matricial. Pode-se considerar, numa abordagem mais abrangente, um modelo GM para manipulação de *estruturas multidimensionais*.

No procedimento de completação, define-se o espaço coerente $\mathbb{D}_\infty$ como ponto fixo para a equação recursiva $D_{n+1} = \mathbb{P}_n \coprod (\mathbb{P}_n \prod \mathbb{P}_n) \coprod (\mathbb{P}_n \prod_B \mathbb{P}_n)$. Neste caso, $\mathbb{P}_n \prod \mathbb{P}_n$ e $\mathbb{P}_n \prod_B \mathbb{P}_n$ denotam o produto direto de espaços vetoriais modelando os processos seqüenciais e condicionais, respectivamente. Tem-se ainda que, cada subnível $\mathbb{P}_n = \mathbb{D}_n \coprod \overline{\mathbb{D}}_n \coprod \overline{\mathbb{D}}_n^\top$ resulta da soma direta entre os domínios de processos elementares,

GM	*Processos*		*Memória*	
Execução	*Tipos*	*Construtores*	*Computações*	*Características*
Simultânea	Concorrentes	Produto Paralelo	Paralelas	Sem conflito
	Conflitantes	Soma não-determinística	Não-Determinísticas	Com Conflito
Não-Simultânea	Seqüenciais	Produto Seqüencial	Seqüenciais	Evolução temporal
	Condicionais	Soma Determinística	Controladas	Testes

Tabela 3.1: Correspondência entre Memória e Processos no Modelo GM

ou síncronos ou não-determinísticos, e de dimensão n, executado no máximo en 2 *utc*, ou ainda, a computação final é obtida após 2 operações elementares.

3.1.1 Aplicações do Modelo GM na Computação Científica

Pela estrutura ordenada do modelo GM, obtém-se semântica para algoritmos da Computação Científica, aplicados principalmente às áreas da Matemática Intervalar, de Computação Estocástica e Computação Quântica, familiares à pesquisa desenvolvida junto ao Grupo de Pesquisa em Fundamentos e Matemática da Computação.

Na abordagem tridimensional da memória e dos processos, tem-se a versão estocástica do modelo GM (REISER; COSTA; DIMURO, 2004b), considerando-se a aritmética estocástica envolvendo as operações de adição e multiplicação por escalares, com ênfase na estrutura abstrata do conjunto de números estocásticos, identificados pela média e o desvio padrão. Processos estocásticos que utilizam processamento matricial interpretam a manipulação operações e são capazes de analisar propriedades familiares aos espaços vetoriais, seguindo a fundamentação apresentada em (MARKOV; ALT, 2004; MARKOV, 2004).

Nos trabalhos (REISER; COSTA; DIMURO, 2003b; REISER et al., 2003a; REISER; COSTA; DIMURO, 2003c) considera-se uma extensão intervalar do modelo GM, capaz de prover semântica para os algoritmos da Matemática Intervalar. Neste caso, os estados de memória são valorados por representações no domínio dos intervalos de reais, enquanto programas e testes computacionais são representados na estrutura ordenada do modelo GM por operações elementares, rotulados pelas posições do espaço euclidiano tridimensional. O conjunto dos intervalos de reais, na abordagem centro-raio, intuitiva para análise do erro computacional, define os conjuntos de dados de entrada e de saída.

De caráter inovador, a definição a extensão quântica do modelo GM (qGM) (REISER; COSTA; DIMURO, 2004c, 2005b; REISER; COSTA; AMARAL, 2007a; CARDOSO et al., 2006; CARDOSO; REISER; COSTA, 2005) inclui a construção do espaço coerente dos estados de memória do modelo qGM, com memória global representada por objetos onde a relação de aproximação, interpretando computações (parciais), está definida pela inclusão.

Na extensão quântica do modelo qGM consideram-se os domínios qualitativos, introduzidos por Girard (GIRARD, 1986) como uma generalização dos espaços coerentes. A construção da estrutura ordenada que modela os processos no Modelo qGM é definida como o Domínio dos Processos Quânticos. O modelo garante representação para as portas lógicas reversíveis, referenciadas na literatura por operações de transformação unitárias, a partir da sincronização de processos elementares clássicos. Também está em

D-GM				
Níveis		*Estrutura Ordenada*		*Computações*
Básico	$\mathbb{D}_0 - \mathbb{P}_0$	$\mathbb{P}_0$	$\mathbb{D}_0 \coprod \overline{\mathbb{D}}_0 \coprod \overline{\mathbb{D}}_0^{\top}$	$2^0 = 1$ *utc*
		$\mathbb{D}_0$	Processos Elementares	
		$\overline{\mathbb{D}}_0$	Processos Elementares Concorrentes	
		$\overline{\mathbb{D}}_0^{\top}$	Processos Elementares Conflitantes	
	$\mathbb{P}_0 - \mathbb{D}_1$	$\mathbb{D}_1$	$\mathbb{P}_0 \coprod (\mathbb{P}_0 \prod \mathbb{P}_0) \coprod (\mathbb{P}_0 \prod_B \mathbb{P}_0)$	$2^1 = 2$ *utc*
		$\mathbb{P}_0 \prod \mathbb{P}_0$	Processos Seqüenciais	
		$\mathbb{P}_0 \prod_B \mathbb{P}_0$	Processos Condicionais	
2^0	$\mathbb{D}_1 - \mathbb{P}_1$	$\mathbb{P}_1$	$\mathbb{D}_1 \coprod \overline{\mathbb{D}}_1 \coprod \overline{\mathbb{D}}_1^{\top}$	$2^1 = 2$ *utc*
		$\mathbb{D}_1$		
		$\overline{\mathbb{D}}_1$		
		$\overline{\mathbb{D}}_1^{\top}$		
	$\mathbb{P}_1 - \mathbb{D}_2$	$\mathbb{D}_2$	$\mathbb{P}_1 \coprod (\mathbb{P}_1 \prod \mathbb{P}_1) \coprod (\mathbb{P}_1 \prod_B \mathbb{P}_1)$	$2^2 = 4$ *utc*
		$\mathbb{P}_0 \prod \mathbb{P}_0$		
		$\mathbb{P}_1 \prod_B \mathbb{P}_1$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n-ésimo	$\mathbb{D}_n - \mathbb{P}_n$	$\mathbb{P}_n$	$\mathbb{D}_n \coprod \overline{\mathbb{D}}_n \coprod \overline{\mathbb{D}}_n^{\top}$	2^n *utc*
		$\mathbb{D}_n$		
		$\overline{\mathbb{D}}_n$		
		$\overline{\mathbb{D}}_n^{\top}$		
	$\mathbb{P}_n - \mathbb{D}_{n+1}$	$\mathbb{D}_{n+1}$	$\mathbb{P}_n \coprod (\mathbb{P}_n \prod \mathbb{P}_n) \coprod (\mathbb{P}_n \prod_B \mathbb{P}_n)$	2^{n+1} *utc*
		$\mathbb{P}_n \prod \mathbb{P}_n$		
		$\mathbb{P}_n \prod_B \mathbb{P}_n$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
comletação	$\mathbb{D}_\infty$		$\mathbb{P}_\infty \prod (\mathbb{P}_\infty \coprod \overline{\mathbb{P}}_n) \prod (\mathbb{P}_\infty \coprod_B \mathbb{P}_\infty)$	2^∞ *utc*

Tabela 3.2: Construção Indutiva da Estrutura Ordenada do Modelo D-GM

desenvolvimento a interpretação semântica de algoritmos básicos para manipulação de estados quânticos, definidos sobre uma estrutura transfinita, tomando valores no conjunto dos números complexos normalizados, envolvendo computações paralelas síncronas e não determinísticas, capazes de interpretar as construções (medidas) do modelo de circuitos quânticos (REISER; COSTA; AMARAL, 2007a,b; AMARAL; REISER; COSTA, 2008).

3.1.2 Conceitos Básicos do Modelo GM

Assim, os domínios $\mathbb{S}$ de estados, o conjunto $\mathbb{B}$ de testes e o conjunto $\mathbb{D}_\infty$ de processos computacionais, rotulados por posições de um espaço geométrico, são espaços coerentes definidos por subconjuntos coerentes (objetos) e funções lineares, garantindo a transformação de um objeto a partir das transformações de suas partes atômicas.

Neste trabalho, para identificação dos processos e estados computacionais, foram considerados os seguintes domínios:

1. $\mathbb{I}$, interpretando uma enumeração das posições em um espaço geométrico (unidimensional e isomorfo aos ordinais);

2. $\mathbb{V}$ interpretando valores computacionais;

3. $\mathbb{F}$ interpretando construtores de processos;

4. $\mathbb{A}$ interpretando expressões de ações computacionais; e

5. $\mathbb{B}$ interpretando testes computacionais (baseado na Álgebra Booleana).

Um processo elementar é caracterizado por alterar uma única posição de memória após o término de sua execução, formalizando assim a unidade de tempo computacional (*utc*) no modelo GM.

A Figura 3.2, mostra a representação de um processo elementar d^k que executa a ação identificada por $d \in \mathbb{A}$ e rotulada por $k \in \mathbb{I}$. Em seqüência, na mesma figura, tem-se a representação gráfica de processo identidade, indicado pela expressão $Skip$, representando a transição onde o estado de entrada coincide com o estado de saída na execução do processo.

A representação gráfica do teste computacional $t \in \mathbb{B}$, caracterizando uma escolha determinística esta representada na Figura 3.2(c). Seguem-se os testes universal e existencial relacionado com a abordagem não-determinística do modelo GM.

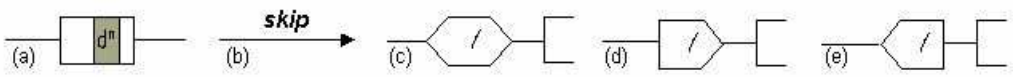

Figura 3.2: Processo elementar d^k, processo Skip e testes computacionais.

Na Figura 3.3, apresentam-se as representações gráficas para os processos resultantes da aplicação dos construtores do conjunto $\mathbb{F}$ representados no modelo GM, considerando-se o segundo nível da estrutura, o subespaço coerente $\mathbb{D}_1$, de acordo com a Figura**??**. Primeiramente, o processo $d^k \cdot e^l$ indicando o produto seqüencial entre os processos elementares d^k e e^l, cuja semântica representa a composição entre as ações $d, e \in \mathbb{A}$. Este processo altera as posições $k, l \in \mathbb{I}$ na memória e é executado em $2utc$:

este produto seqüencial primeiro executa d colocando o resultado na posição k e após o seu término, inicia a execução do processo$e$ relativo à posição l.

Na mesma Figura 3.3, à direita do processo seqüencial, está representado a soma determinística $d^k +_B e^l$ entre os processos elementares d^k e e^l, a partir do teste $b \in \mathbb{B}$, representando uma escolha: o processo elementar d^k é executado se o teste b é verdadeiro, caso contrário, executa-se o processo elementar e^l. A soma não-determinística $|d^k, e^k|$ entre os processos elementares d^k e e^k, mutuamente exclusivos ou conflitantes na posição k de memória, está representada logo após a soma determinística.

Finalizando a seqüência de representações, apresenta-se o produto paralelo $\| d^k, e^l \|$ entre os processos elementares concorrentes d^k e e^l, com $k \neq l$, representando a execução simultânea dos processos elementares d^k e e^l, em 1 *utc*.

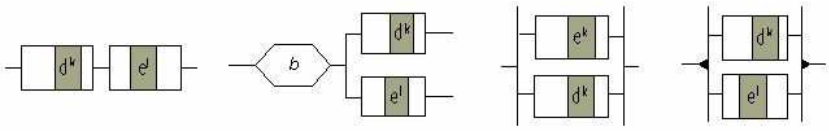

Figura 3.3: Construtores de Processos Aplicados a Processos Elementares.

No modelo GM, dois processos elementares são mutuamente exclusivos se estão definidos pelo mesmo índice *K* (indexados pela mesma posição *k* do espaço geométrico). No caso geral, dois processos são mutuamente exclusivos sempre que existir no mínimo uma mesma posição de memória afetada por ambos. Neste sentido, a arbitrariedade da escolha constitui-se na principal característica da soma não-determinística de processos.

A construção indutiva do modelo GM, garante a representação de processos parciais os quais e facilitam a interpretação semântica associada da transição de estados parciais associado a um processo computacional. A Figura 3.4 mostra a representação gráfica de alguns processos parciais representados no modelo GM, relacionados com os construtores apresentados na Figura 3.3.Processo seqüencial parcial a direita, soma determinística parcial superior, soma não-determinística parcial e produto paralelo parcial, respectivamente.

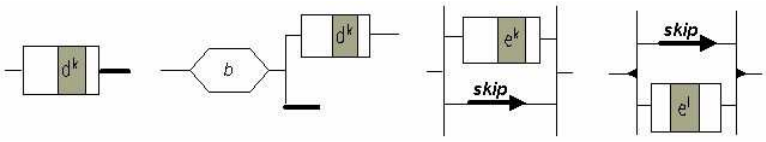

Figura 3.4: Objetos Parciais Representados no Modelo GM.

3.2 Abordagem Visual para o Modelo GM

A melhoria no padrão das linguagens de programação constitui um dos principais objetivos da pesquisa envolvendo as linguagens de programação visuais (MARRIOTT; MEYER, 1998), cujo contexto multidimensional inclui as linguagens textuais.

Visando tornar a programação paralela e/ou distribuída mais acessível a uma audiência específica e buscando auxiliar na validação, correção e organização das correspondentes tarefas de programação, os ambientes de programação visuais vem transferindo

os avanços da pesquisa em programação visual para sua aplicação prática nos modelos e linguagens tradicionais, já familiares aos programadores.

Neste contexto, foi concebida a ferramenta VPE-GM, combinando as tradicionais técnicas das linguagens de programação textual com a construção de um ambiente gráfico/visual, visando a melhoria no acesso, manipulação e compreensão da programação paralela e/ou distribuída no modelo GM, incluindo análise da informação e modelagem da interface dos editores gráficos do sistema.

Na busca de uma semântica bi-dimensional capaz de integrar ambas, a modelagem espacial da memória e as abstrações dos processos modelo GM, a especificação da sintaxe visual para definição da linguagem visual para o modelo GM, indicada por VLGM (*Visual Language for Geometric Machine*) foi introduzida em (REISER et al., 2003a,b). Na especificação da VLGM, consideram-se a definição da sintaxe abstrata, responsável pela estrutura lógica das expressões gráficas, e a definição da sintaxe concreta, orientando o layout da cada expressão para manipulação do usuário programador.

Da mesma forma foi construída a especificação da gramática visual, integrando conceitos da Teoria Algébrica (MILNER, 1980) e da Teoria dos Grafos (WEST, 1996; WILSON, 1996) na representação dos construtores: produto seqüencial e paralelo, somas determinística e não-determinísticas, e as correspondentes construções recursivas. As regras que compõe a gramática visual são definidas em duas etapas, a preparação (caracterizada pela representação de objetos parciais) e a realização (caracterizada pela construção do novo objeto) (CARDOSO et al., 2003) (CARDOSO et al., 2003a).

3.2.1 Especificação da Linguagem Visual para o Modelo GM

Entende-se que as características da programação visual, apresentadas nas seções anteriores, foram relevantes para tornar a programação do paralelismo e não-determinismo no modelo GM mais acessível, incentivando sua aplicação na Computação Científica, em especial na Matemática Intervalar (REISER; COSTA; DIMURO, 2004d) e na Computação Quântica (MONTEIRO et al., 2007, 2006).

A especificação da VLGM baseia-se na abordagem proposta no Projeto GENGED (BARDOHL; ERMEL, 2001) e inclui a especificação do alfabeto visual (Seção3.2.1.1) e da gramática visual (Seção3.2.1.2) para a linguagem VLGM, com ênfase nas construções recursivas. Os resultados apresentados foram aplicados na modelagem do ambiente visual de programação para o modelo de Máquina Geométrica.

3.2.1.1 Alfabeto Visual da VLGM

Nesta seção são apresentados os grafos associados aos processos do tipo identidade e elementar, exemplificando o alfabeto visual. As demais construções podem ser encontradas em (CARDOSO et al., 2003) (CARDOSO et al., 2003a), (CARDOSO et al., 2003b).

- *Processo Identidade*

  O grafo que representa o processo identidade na VLGM é definido por dois nodos: (i) o objeto-processo representado por uma seta na sintaxe concreta, e (ii) o nodo atributo-ação responsável por sua identificação nominal, conforme mostra a Figura 3.5. O atributo-ação é identificado pela palavra `Skip` que pertence a um conjunto Σ de palavras e sua representação é um retângulo com as extremidades arredondadas e dois arcos (conexões): (i) `nome da ação`, com origem no nodo-atributo

e que se liga ao conjunto Σ e (ii) `n-ação`, com origem no nodo-atributo e com destino no objeto-processo. Na sintaxe concreta, a função de inclusão é indicada por um arco tracejado e nomeado `incl_ação`, mapeando nome do atributo para a sua localização no diagrama de representação do processo identidade. Próximo a este objeto fica uma especificação quanto ao tamanho e ao nome da fonte utilizada na sua identificação.

- *Processo Elementar*

  Na construção da sintaxe abstrata, os atributos posição e ação de um processo elementar são objetos-atributo do objeto-processo no grafo que o representa: (i) o atributo-posição é determinado pelo rótulo, elemento do conjunto de rótulos, e sua representação é um retângulo com as extremidades arredondadas; (ii) o atributo-ação é identificado por uma palavra a qual pertence ao alfabeto Σ e sua representação é também um retângulo com as extremidades arredondadas. Cada objeto-atributo têm dois arcos : (i) `n-ação` com origem no nodo-atributo, o qual se liga ao conjunto que define seu nome; (ii) `n-pos` com origem no nodo-atributo e com destino no objeto-processo elementar.

  Na sintaxe concreta, o diagrama de representação do processo elementar é identificado por um retângulo com duas conexões laterais e dividido em partes, que identificam os atributos. Nas duas divisões centrais colocam-se as correspondentes posições e ações. Esta tarefa é formalizada pelas funções de inclusão, indicadas por arcos tracejados e nomeadas `incl_pos` e `incl_ação`. As funções de inclusão mapeiam os nomes dos atributos a sua localização no diagrama de representação do processo elementar. Próximo a estas, tem-se a declaração do tamanho e do nome da fonte utilizada, conforme mostra a Figura 3.6.

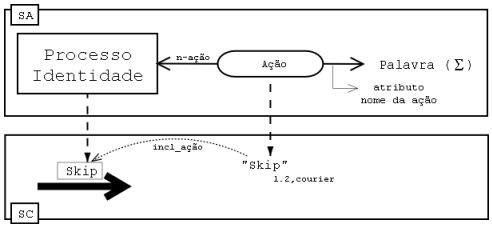

Figura 3.5: Processo Identidade

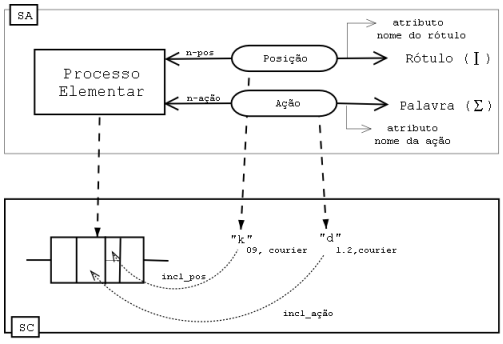

Figura 3.6: Processo Elementar

3.2.1.2 Gramática Visual da VLGM

Nesta seção, apresenta-se a especificação gramatical da regra de construção do produto paralelo. Para os demais construtores a especificação das regras da gramática visual estão descritas em (CARDOSO et al., 2003) (CARDOSO et al., 2003a).

A transformação de grafos que formaliza a regra gramatical para o construtor produto paralelo, envolvendo os objetos que representam os correspondentes processos parciais (superior e inferior) é definida em duas etapas:

- Na etapa de preparação tem-se: (i) sintaxe abstrata para os processos concorrentes são representados por subgrafos identificados pelos arcos de conexão: `fator_sup` e `fato_inf`; (ii) na sintaxe concreta, o produto paralelo é representado por diagramas restringidos por barras verticais paralelas entre os quais estão dois objetos-processo sobrepostos. Estas sobreposições, na sintaxe concreta, são identificados pelas funções: `nomepp_sup`, `nomepp_inf` (identificando os processos parciais) e `inspp_sup`, `inspp_inf` (inserindo os processos parciais). Na Figura 3.7, mostram-se os objetos e as conexões envolvidos na transformação dos grafos desta subetapa e apresentam-se os correspondentes morfismos envolvidos na gramática.

- Na etapa de realização, a transformação é definida pela composição das funções que nomeiam os processos parciais (`nomepp_sup` e `nomepp_inf`) com a função de inclusão (`incluipp`), resultando na função composta denominada `incluipp_nomeia`. Assim também é definida a função composta `combpp_insp`. O diagrama de representação para a regra geral do processo paralelo está descrito na Figura 3.7, tornando explícita a composição que gera a regra de construção de processos concorrentes.

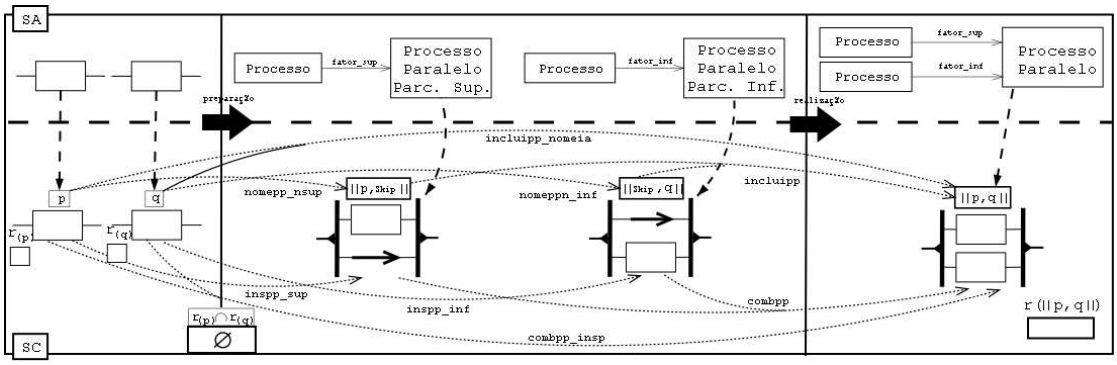

Figura 3.7: Regra Gramatical para a Construção de um Processo Paralelo

3.3 Ferramenta VPE-GM

Nesta sessão são descritas as principais características relacionadas com o Modelo GM que fundamentaram a implementação do ambiente de programação visual VPE-GM (*V*isual Programming Environment for the Geometric Machine Model).

A Máquina Geométrica está caracterizada como um modelo de computação com nível de abstração totalmente explícito do paralelismo (FONSECA, 2006; SKILLICORN; TALIA, 1998). Neste contexto, na ferramenta VPE-GM, as construções como assinalamento e decomposição são especificadas quando da interface gráfica dos editores, considerando para tal a notação posicional dos processos, testes e estados de memória.

De forma análoga, o mapeamento e a sincronização de tarefas, incluindo gerenciamento do conflito de acesso à memória no modelo GM são controlados e interpretados pela função-posição. Neste caso, exigindo conhecimento das interpretações semânticas que podem ser obtidas com a fundamentação, baseada no modelo GM e suas caracterizações de memória, processos e computações.

A modelagem e implementação da ferramenta computacional VPE-GM está descrita em (PRESTES et al., 2004; PRESTES; REISER; COSTA, 2005) de onde foram

obtidas as informações que fundamentaram os estudos apresentados nesta seção. Quando considerou-se o desenvolvimento do ambiente VPE-GM, as técnicas tradicionais da programação textual foram combinadas com as construções visuais, visando incentivar a programação na Máquina Geométrica e tornando mais fácil a simulação de computações paralelas e não-determinísticas.

A integração do ambiente VPE-GM com a arquitetura VirD-GM proposta neste trabalho, provê representação visual para processos e construtores interpretados no modelo GM. Obtém-se assim, um ambiente de geração e construção de novos processos concorrentes que viabiliza sua execução paralela e/ou distribuída. Justifica-se portanto, uma breve descrição da ferramenta VPE-GM considerada logo a seguir, cujos conceitos fundamentais e ilustrações foram obtidos em (PRESTES; REISER; COSTA, 2005).

3.3.1 Descrição da Ferramenta VPE-GM

Desenvolvida com o objetivo de estimular a simulação de algoritmos paralelos e distribuídos da Computação Científica, esta ferramenta computacional está implementada com base na concepção livre, multi-plataforma, utilizando a linguagem *Python* e aplicando técnicas visuais para construção gráfica de processos e da memória (M.C.BROWN, 2001; ROSSUM; DRAKE, 2003).

A discretização do espaço geométrico é obtida pela estrutura matricial que modela a memória. A indexação dos valores de memória explicitada na representação gráfica, facilita a aplicação e compreensão do paralelismo síncrono e do não-determinismo caracterizado pelo conflito de acesso à memória.

Os principais componentes da ferramenta VPE-GM, e que permanecem quando de sua integração com a VirD-GM para consolidação do Projeto D-GM, são o Editor de Processos, o Editor de Memória e as correspondentes interfaces gráficas que viabilizam o acesso do usuário ao ambiente, incluindo as correspondentes comunicações entre estes componentes. Na Figura 3.8, tem-se um esquema do ambiente com especificação dos módulos e das comunicações entre o Editor de Processos e o Editor de Memórias, incluindo módulo de controle e execução relacionados com a VirD-GM.

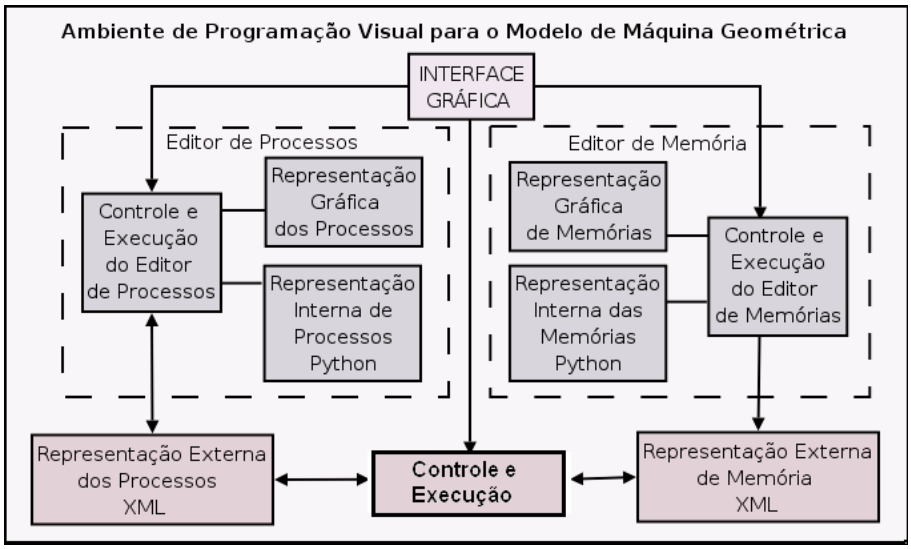

Figura 3.8: Componentes da VPE-GM

3.3.1.1 Editor de Processos

No Editor Gráfico de Processos pode-se efetivar a manipulação de arquivos com opções para criar, deletar, salvar, redimensionar e reconstruir objetos gráficos, obtida pelo estudo e aplicações de eventos e introduzindo o conceito de envelopes: arquivos gerados na linguagem *Python* para representação interna dos processos e caracterizados pelo agrupamento dos objetos gráficos quando da edição de processos.

Estes arquivos são gerados no módulo de Representação Interna dos Processos. De forma análoga, os objetos gráficos também são descritos por arquivos gerados na linguagem XML e armazenados no módulo de Representação Externa dos Processos. A coleção destes arquivos constitui a Biblioteca de Processos, cujo acesso é viabilizado pelo módulo de controle e execução de processos. Este módulo permite a comunicação entre as diferentes representações dos objetos gráficos, ou seja, os correspondentes arquivos em XML e em *Python*. A configuração dos processos enviada ao módulo de controle e execução está especificada nesta descrição em XML dos objetos gráficos.

Na Figura 3.9, apresentam-se as principais construções obtidas a partir de processos elementares, associados às diversas posições de memória. Pela composição dos construtores são graficamente gerados processos mais complexos: (i) produtos seqüenciais (Figura 3.9 (A)); (ii) produtos paralelos (Figura 3.9 (B)); (iii) somas não-determinísticas (Figura 3.9 (C)); (iv) processos iterativos (Figura 3.9 (D)); e (v) somas determinísticas (Figura 3.9 (E)). Consideram-se também a definição de macros, associadas a programas já disponíveis na biblioteca de processos. Além destes processos e construtores também são definidos testes computacionais: teste determinístico e, para os casos de computações não-determinísticas, tem-se os testes universal e existencial. A partir destes testes é possível realizar escolhas, as quais redirecionam o fluxo de dados durante a computação.

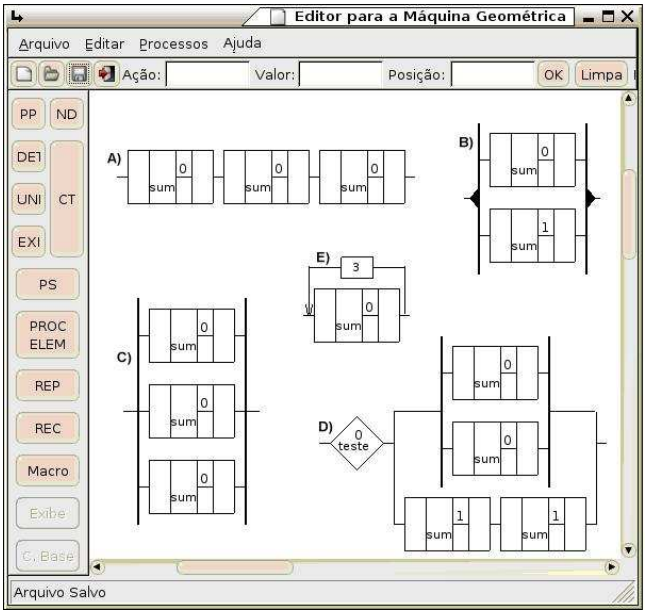

Figura 3.9: Interface do Editor de Processos do Ambiente VPE-GM.

3.3.1.2 Editor de Memória

O Editor de Memória possibilita a representação gráfica da memória, disponibilizando ao usuário, além da escolha da estrutura matricial, possivelmente multidimensional

e adequada ao desenvolvimento de uma aplicação, a visualização das transformações da memória nas computações geradas, viabilizando a validação dos parâmetros que definem os estados e correspondentes processos. A estrutura modular do Editor de Memórias para representação interna e externa de estados foi implementada pela criação de funções de controle e execução. Assim também sua integração com a VirD-GM, foram construídos de forma análoga ao Editor de Processos.

Salienta-se que, pelo módulo de Execução e Controle do Editor de Memórias são exportados os arquivos em *Python* para o módulo de Representação Externa de Memórias, responsável pela geração e armazenamento das correspondentes descrições em XML.

A interface do Editor de Memória do ambiente VPE-GM pode ser visualizada na Figura 3.10, mostrando os parâmetros para configuração e estados computacionais.

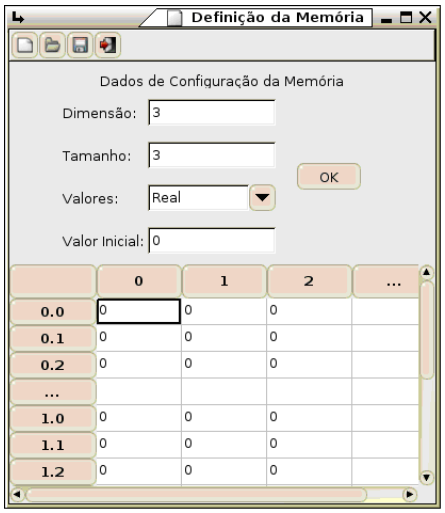

Figura 3.10: Interface do Editor de Memória do Ambiente VPE-GM.

O processo dinâmico de execução ocorre a partir da configuração dos processos e da memória, definidos neste caso, como parâmetros de entrada, os quais são validados nas interfaces gráficas dos correspondentes editores. O módulo de Controle e Execução na VirD-GM importa os arquivos de outras bibliotecas (Computação Científica) provendo os recursos (métodos, técnicas, funções e procedimentos) necessários para posterior execução na VirD-GM. O ambiente VPE-GM também permite que os resultados parciais e finais dessa execução sejam disponibilizados para o usuário pela interface da VPE-GM.

3.3.2 Funcionamento da Ferramenta VPE-GM

No esquema da Figura 3.11, explica-se o funcionamento do ambiente VPE-GM.

Primeiramente, mostra-se a edição de um processo elementar que executa uma soma (`sum`) sobre uma posição de memória `0`. Neste exemplo, considera-se que serão definidos como parâmetros as constantes `3` e `2`, conforme mostra o código XML gerado.

A edição do programa gera um arquivo XML, que o representa através do módulo de Representação Externa dos Processos. Na Figura 3.11, mostra-se um exemplo resumido de código XML que é gerado para o Processo Elementar, indicado pela expressão (`conselem`), identificando o atributo `repr`. Este processo executa uma operação `sum` definida em `acao`, atribuindo o resultado da execução da operação à posição `0` (campo

pos).

No segundo momento, o usuário deve configurar a memória, de forma compatível com o programa criado na interface do Editor de Processos. Na verificação da compatibilidade, entre memória e processos, faz-se necessário que a memória apresente todas as posições às quais os processos fazem referência. Na Figura 3.11, o Processo Elementar escreve o resultado numa posição de memória de valor 0, verificando condições como dimensão e tamanho da memória (no caso, unidimensional e de tamanho 3). De maneira semelhante, é possível gerar o arquivo XML da memória, o qual permite a visualização do resultado obtido, assim como também a apresentação dos dados referentes à memória e aos valores que ela armazena. Estes valores são enviados para o módulo de comando e execução da VirD-GM.

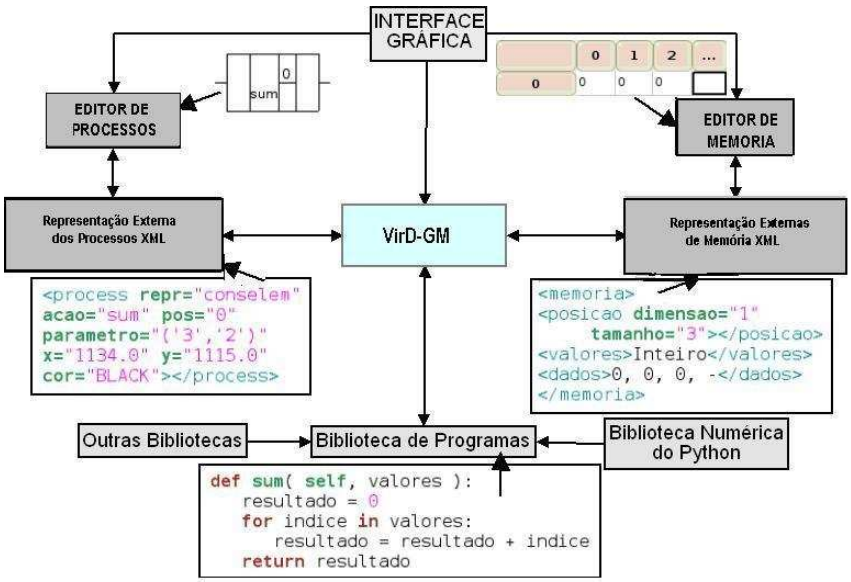

Figura 3.11: Exemplo de Funcionamento do Ambiente APV-GM.

Depois da geração dos arquivos XML, o ambiente VPE-GM realiza um teste de compatibilidade entre os arquivos, validando a configuração de memória definida de forma compatível com os processos gerados. A partir do nome do processo, verifica-se sua identificação em uma Biblioteca de Programas. Neste exemplo, está sendo validada uma função de soma, definida pela expressão sum, aplicando funções contidas na biblioteca numérica padrão do *Python*. Os arquivos são enviados à VirD-GM onde inicia a execução, propriamente dita.

3.4 *Middleware* EXEHDA: Fundamentos e Tecnologias

O EXEHDA (*Execution Environment for High Distributed Applications*) é um *middleware* direcionado a dar suporte à computação das aplicações distribuídas e/ou paralelas em ambientes com elevado nível de distribuição e heterogeneidade de recursos. Este tipo de ambiente atualmente constitui o que a literatura denomina de ambientes pervasivos. O mesmo foi proposto no trabalho (YAMIN, 2004).

Na continuidade são apresentadas as premissas utilizadas na concepção do EXEHDA (YAMIN, 2004; YAMIN et al., 2005), e que foram consideradas na concepção

da VirD-GM.

3.4.1 Organização do EXEHDA

O *middleware* EXEHDA tem por finalidade definir a arquitetura para um ambiente de execução destinado às aplicações distribuídas, no qual as condições de contexto são pró-ativamente monitoradas, e o suporte à execução deve permitir que tanto a aplicação como ele próprio utilizem estas informações na gerência da adaptação de seus aspectos funcionais e não-funcionais. Entende-se por adaptação funcional aquela que implica a modificação do código sendo executado. Por sua vez, adaptação não-funcional, é aquela que atua sobre a gerência da execução distribuída. Também a premissa *siga-me* das aplicações *pervasivas* deverá ser suportada, garantindo a execução da aplicação do usuário em qualquer tempo, lugar e equipamento (YAMIN et al., 2005).

As aplicações-alvo são distribuídas, adaptativas ao contexto em que executam e compreendem a mobilidade lógica e a física. Na perspectiva do EXEHDA, entende-se por mobilidade lógica a movimentação entre equipamentos de artefatos de software e seu contexto, e por mobilidade física o deslocamento do usuário, portando ou não seu equipamento.

3.4.1.1 Arquitetura de Software

A figura 3.12 apresenta uma visão geral da arquitetura de software do EXEDHA. A representação da consciência do contexto nesta figura como um módulo virtual tem por objetivo ressaltar sua importância na arquitetura, bem como caracterizar sua presença na concepção de todos os outros componentes (YAMIN, 2004; YAMIN et al., 2005).

A arquitetura do EXEHDA apresenta uma organização lógica em três camadas: (*sup*) camada de aplicação; (*interm*) camada de suporte e ambiente de execução; (*inf*) camada de sistemas básicos.

Na camada superior (aplicação) está o suporte para o desenvolvimento de aplicações. As aplicações no EXEHDA são programadas em Java com extensões para as bibliotecas que ativam os serviços da arquitetura de software.

Na camada intermediária (EXEHDA) estão os mecanismos de suporte à execução da aplicação. Esta camada é formada por dois níveis: O primeiro nível é composto por três módulos de serviço à aplicação: Acesso Distribuído na perspectiva Pervasiva a Código e Dados, Reconhecimento de Contexto e Ambiente de Execução da Aplicação.

No segundo nível da camada intermediária estão os serviços básicos do EXEHDA, dos quais provem as funcionalidades necessárias para o primeiro nível e cobrem vários aspectos, tais como: (i) *migração*: mecanismos para deslocar um componente de software de uma localização física (equipamento) para outra; (ii) *persistência*: mecanismo para aumentar a disponibilidade e o desempenho do acesso aos dados; (iii) *descoberta de recursos*: para dar suporte ao movimento dos dispositivos móveis e dos componentes entre diferentes células; (iv) *comunicação*: com possibilidade de ser anônima e assíncrona; escalonamento: permite decidir o melhor nodo para criar os componentes da aplicação; (v)*monitoramento*: sensores que fornecem informações sobre o ambiente de execução e aplicação.

A camada inferior da arquitetura é composta pelos sistemas e linguagens nativas que integram o meio físico de execução. Por questões de portabilidade, nesta camada a plataforma base de implementação é a Máquina Virtual Java.

A arquitetura supõe a existência de uma rede de interconexão, que implementa a comunicação entre os diferentes equipamentos que compõem meios físico distribuídos sobre os quais acontece as computações distribuídas e/ou Paralelas.

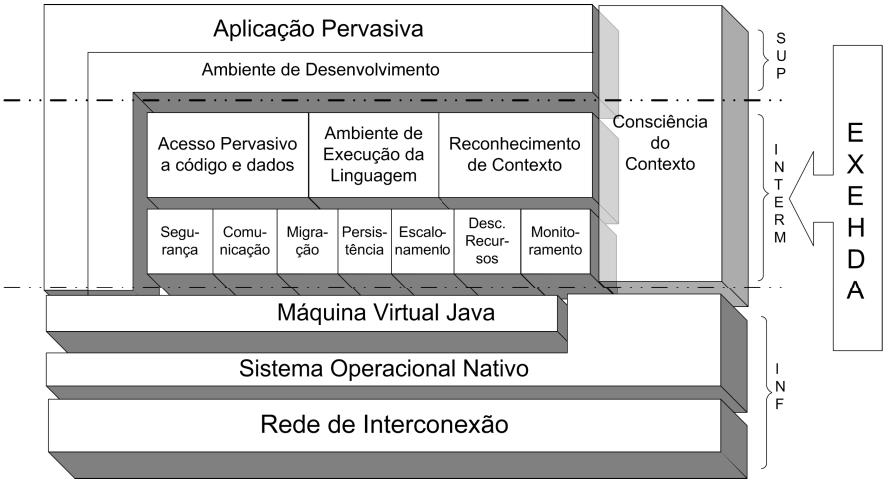

Figura 3.12: Arquitetura de Software do EXEHDA
(YAMIN, 2004)

Os principais requisitos que o EXEHDA deve atender são: (i) gerenciar tanto aspectos não-funcionais como funcionais da aplicação, e de modo independente, (ii) dar suporte à adaptação dinâmica de aplicações; (iii) disponibilizar mecanismos para obter e tratar informações de contexto; (iv) utilizar informações de contexto na tomada de decisões, (v) decidir as ações adaptativas de forma colaborativa com a aplicação e (vi) possibilitar ao usuário tanto o disparo de aplicações, quanto o acesso a dados a partir de qualquer lugar.

3.4.1.2 Organização do Ambiente Distribuído Gerenciado pelo EXEHDA

A composição do ambiente computacional distribuído gerenciado pelo EXEHDA envolve tanto os dispositivos dos usuários, como os equipamentos da infra-estrutura de suporte, todos instanciados pelo seu respectivo perfil de execução do *middleware*. O cenário da computação distribuída é mapeado em uma organização composta pela agregação de células de execução do EXEHDA, conforme pode ser visto na figura 3.13 (YAMIN, 2004).

Os recursos da infra-estrutura física são mapeados para três abstrações básicas, as quais são utilizadas na composição do ambiente distribuído (YAMIN, 2004):

- **EXEHDAcels:** denota a área de atuação de uma EXEHDAbase, e é composta por esta e por EXEHDAnodos. Os principais aspectos considerados na definição da abrangência de uma célula são: o escopo institucional, a proximidade geográfica e o custo de comunicação;

- **EXEHDAbase:** é o ponto de contato para os EXEHDAnodos. É responsável por todos os serviços básicos da célula de execução e, embora constitua uma referência lógica única, seus serviços, sobretudo por aspectos de escalabilidade, poderão estar distribuídos entre vários equipamentos;

- **EXEHDAnodo:** são os equipamentos de processamento disponíveis no ambiente computacional distribuído, sendo responsáveis pela execução das aplicações.

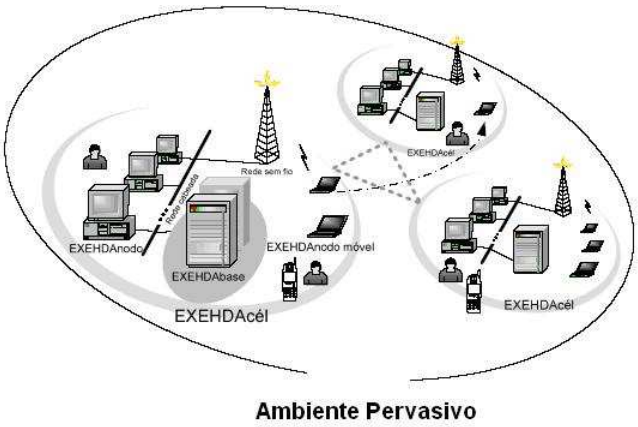

Figura 3.13: ISAM Ambiente Pervasivo Gerenciado pelo EXEHDA (YAMIN, 2004)

O ambiente de computação provido pelo EXEHDA é formado por equipamentos multi-institucionais, o que gera a necessidade de adotar procedimentos de gerência iguais aos utilizados em ambientes de Grade Computacional (*Grid Computing*) (YAMIN et al., 2003). O gerenciamento da organização celular adotada resguarda a autonomia das instituições envolvidas.

3.4.1.3 Organização Baseada em Serviços

O requisito de operação em um ambiente altamente heterogêneo, onde não só o hardware exibe capacidades variadas de processamento e memória, mas também as bibliotecas de software disponíveis em cada dispositivo, motivaram a adoção de uma abordagem na qual um núcleo mínimo do *middleware* tem suas funcionalidades estendidas por serviços carregados sob demanda. Esta organização reflete um padrão de projeto referenciado na literatura como micro-kernel (BUSCHMANN, 1996). Some-se a isto o fato de que esta carga sob demanda tem perfil adaptativo. Deste modo, poderá ser utilizada versão de um determinado serviço, melhor sintonizada às características do dispositivo em questão. Isto é possível porque, na modelagem do EXEHDA, os serviços estão definidos por sua interface, e não pela sua implementação propriamente dita.

A contra-proposta à estratégia micro-kernel de um único binário monolítico, cujas funcionalidades cobrissem todas as combinações de necessidades das aplicações e dispositivos, se mostra impraticável na Computação Pervasiva, cujo ambiente computacional apresenta elevada heterogeneidade de recursos de processamento.

Por sua vez, o requisito do *middleware* de manter-se operacional durante os períodos de desconexão planejada motivou, além da concepção de primitivas de comunicação adequadas a esta situação, a separação dos serviços que implementam operações de natureza distribuída em instâncias locais ao *EXEHDAnodo* (instância nodal), e instâncias locais a *EXEHDAbase* (instância celular). Neste sentido, o relacionamento entre instância de nodo e celular assemelha-se à estratégia de *Proxies*, enquanto que o relacionamento entre instâncias celulares assume um caráter P2P. A abordagem P2P nas operações inter-celulares vai ao encontro do requisito de escalabilidade. Uma organização dos subsistemas do EXEHDA pode ser visto na figura 3.14.

Com isso, os componentes da aplicação em execução em determinado dispositivo podem permanecer operacionais, desde que, para satisfação de uma dada requisição pelo

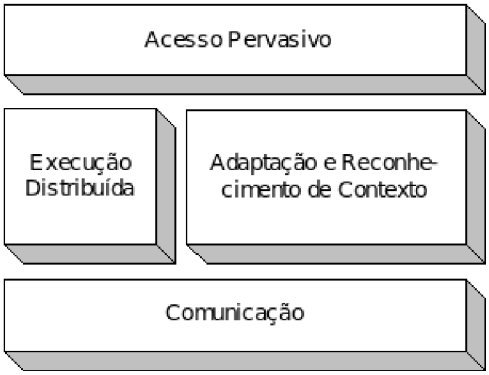

Figura 3.14: Organização dos Subsistemas do EXEHDA
(YAMIN, 2004)

middleware, o acesso a um recurso externo ao dispositivo seja prescindível. Por outro lado, a instância celular, em execução na base da célula, provê uma referência para os outros recursos, no caso da realização de operações que requeiram coordenação distribuída. Neste sentido, observe-se que a *EXEHDAbase* é, por definição, uma entidade estável dentro da *EXEHDAcel*, permitindo que os demais integrantes (recursos) da célula tenham um caráter mais dinâmico no que se refere a sua disponibilidade (presença efetiva) na célula.

3.4.1.4 Composição do Núcleo do EXEHDA

A funcionalidade provida pelo EXEHDA é personalizável no nível de nodo, sendo determinada pelo conjunto de serviços ativos e controlada por meio de perfis de execução. Um perfil de execução define um conjunto de serviços a ser ativado em um *EXEHDAnodo*, associando a cada serviço uma implementação específica dentre as disponíveis, bem como definindo parâmetros para sua execução. Adicionalmente, o perfil de execução também controla a política de carga a ser utilizada para um determinado serviço, a qual se traduz em duas opções: (*i*) quando da ativação do nodo (*bootstrap* do *middleware*) e (*ii*) sob demanda.

Desta maneira, a informação definida nos perfis de execução é também consultada, quando da carga de serviços sob demanda, assim, a estratégia adaptativa para carga dos serviços acontece tanto na inicialização do nodo, quanto após este já estar em operação e precisar instalar um novo serviço. Esta política para carga dos serviços é disponibilizada por um núcleo mínimo do EXEHDA, o qual é instalado em todo *EXEHDAnodo* que for integrado ao ambiente de execução. Este núcleo é formado por dois componentes, conforme figura 3.15:

- **ProfileManager:** interpreta a informação disponível nos perfis de execução e a disponibiliza aos outros serviços do *middleware*. Cada EXEHDAnodo tem um perfil de execução individualizado;

- **ServiceManager:** realiza a ativação dos serviços no *EXEHDAnodo* a partir das informações disponibilizadas pelo *ProfileManager*. Para isto, carrega sob demanda o código dos serviços do *middleware*, a partir do repositório de serviços que pode ser local ou remoto, dependendo da capacidade de armazenamento do *EXEHDAnodo* e da natureza do serviço.

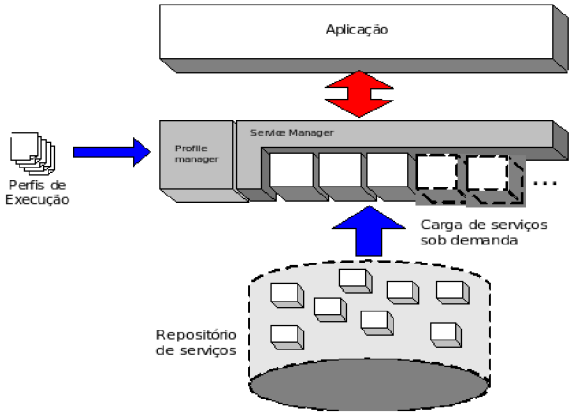

Figura 3.15: Organização do Núcleo do EXEHDA
(YAMIN, 2004)

3.4.2 Funcionalidades dos Subsistemas do EXEHDA

Esta seção resume os estudo feito das funcionalidades dos subsistemas que compõem o EXEHDA. Este estudo foi fundamental no momento da concepção da integração VirD-GM à arquitetura de software do EXEHDA.

3.4.2.1 Subsistema de Execução Distribuída

O Subsistema de Execução Distribuída é responsável pelo suporte ao processamento distribuído no EXEHDA. No intuito de promover uma execução efetivamente distribuída, este subsistema interage com outros subsistemas do EXEHDA. Este subsistema é constituído pelos seguintes componentes:

- *Executor*: serviço que acumula as funções de disparo de aplicações, e de criação e migração dos seus Componentes. Na implementação destas funções é empregsada a instalação de código sob demanda. Para tal, interage com os serviços *CIB* e *BDA*.

- *Cell Information Base - CIB*: serviço que implementa a base de informações da célula. Sua principal funcionalidade está relacionada à manutenção da infra-estrutura distribuída que forma o ambiente de execução.

- *OXManager*: a abstração OX (Objeto eXehda (EXEHDA)), provida pelo *middleware* às aplicações, consiste em uma instância de objeto, criada por intermédio do serviço *Executor*, à qual pode ser associada meta-informação em tempo de execução. No caso da abstração OX, esta meta-informação ocorre na forma de atributos, i.e., pares (nome, valor), sendo que ”nome” é uma cadeia de caracteres ASCII que descreve o atributo, podendo ”valor” ser um conteúdo genérico, inclusive de natureza binária. A gerência e manutenção da meta-informação associada a um OX é atribuição do serviço *OXManager*.

- *Discoverer*: o serviço de descoberta de recursos responsável pela localização de recursos especializados no ambiente de execução a partir de especificações abstratas dos mesmos.

- *ResourceBroker*: o controle da alocação de recursos às aplicações no EXEHDA é desempenhado pelo serviço *ResourceBroker*, o qual atende tanto requisições ori-

ginárias da própria EXEHDAcel quanto oriundas de outras células do ambiente de execução.

- *Gateway*: serviço que faz a intermediação das comunicações entre os nodos externos à célula e os recursos internos a ela. Da sua ação integrada com o *Resource-Broker* decorre o controle de acesso aos recursos de uma EXEHDAcel.

- *StdStreams*: serviço que provê o suporte ao redirecionamento dos *streams* padrões de entrada, saída e erro. Toda aplicação em execução nos EXEHDAnodos possui um componente console individualizado, o qual agrupa os três *streams* padrões.

- *Logger*: este serviço na fase de desenvolvimento, pode ser empregado para depuração de um programa auxiliando a identificar comportamentos errôneos ou imprevistos (gargalos de execução - *bottlenecks*). No tocante a segurança dos sistemas computacionais, esta funcionalidade é freqüentemente empregada para registro de operações importantes e/ou críticas realizadas, facilitando a identificação de situações de intrusão, ou de uso indevido do sistema.

- *Dynamic Configurator - DC*: serviço que tem como objetivo realizar a configuração do perfil de execução do *middleware* em um determinado EXEHDAnodo de forma automatizada.

3.4.2.2 Subsistema de Comunicação

A natureza do sistemas distribuídos em larga escala muitas vezes inviabiliza a interação contínua entre os componentes da aplicação distribuída. O subsistema de comunicação do EXEHDA disponibiliza mecanismos que atendem este aspecto de desconexões. Integram este subsistema os serviços *Dispatcher, WORB, CCManager* os quais contemplam modelos com níveis diferenciados de abstração para as comunicações.

- *Dispatcher*

  Serviço comunicação mais elementar do EXEHDA disponibiliza a troca de mensagens ponto-a-ponto com garantia de entrega e ordenamento das mensagens, o qual é especializado para operação no ambiente pervasivo. As mensagens trafegam entre instâncias do *Dispatcher* localizadas em nodos diferentes através de estruturas denominadas canais. Nesse sentido, quando de sua inicialização, o *Dispatcher* atualiza a informação do *EXEHDAnodo* na *CIB* (Cell Information Base), em específico o atributo *contactAddress*, provendo uma lista de protocolos e endereços que podem ser utilizados para alcançar aquele EXEHDAnodo.

- *WORB*

  Este subsistema é responsável por simplificar a construção de serviços distribuídos, permitindo que os programadores focalizem esforços no refinamento da semântica distribuída associada ao serviço em desenvolvimento, abstraindo aspectos de baixo nível relativos ao tratamento das comunicações em rede. Para tanto, oferece um modelo de comunicação baseado em invocações remotas de método, similar ao RMI, porém sem exigir a manutenção da conexão durante toda a execução da chamada remota.

- *CCManager*

  Os componentes que constituem as aplicações do EXEHDA, precisam de um mecanismo de comunicação com característica de desacoplamento temporal e espacial. Um serviço deste tipo particularmente oportuno, à medida que simplifica a construção de tais aplicações. O CCManenger vem atender a esta demanda, disponibilizando um mecanismo baseado na abstração espaço de *tuplas*, o qual prescinde da coexistência temporal de emissor e receptor. Outro aspecto oportuno desta abstração é a facilidade com que podem ser implementados outros padrões de comunicação, além do ponto-a-ponto.

3.4.2.3 Subsistema de Acesso Pervasivo

A premissa de acesso em qualquer lugar, todo o tempo, a dados e códigos, requer um suporte do *middleware*. Os serviços que compõem este subsistema no EXEHDA são:

- *BDA-Base de Dados pervasiva das Aplicações*

  O EXEHDA organiza um ambiente de Computação que tem por objetivo (*i*) permitir que o usuário dispare aplicações a partir de qualquer nodo integrante do sistema e, após o disparo, (*ii*) a mobilidade parcial ou integral de tais aplicações em resposta a modificações em seu contexto de execução. A implementação de um mecanismo de instalação sob demanda requer a existência de um repositório de código que forneça a mesma visão do software a partir de qualquer dispositivo do ambiente de execução. Outras funcionalidades para este repositório pervasivo também são desejáveis. Considerando a perspectiva de que as diversas aplicações disponibilizadas sigam linhas de evolução independentes, o suporte a controle de versões é oportuno na direção da manutenção da operacionalidade das diferentes aplicações.

- *AVU - Ambiente Virtual do Usuário*

  O serviço AVU tem como finalidade a manutenção do acesso pervasivo ao ambiente computacional provido pelo EXEHDA, garantido acesso aos arquivos do usuário as suas aplicações, bem como as informações que o mesmo personalizou para o disparo de programas e serviços.

- *SessionManager*

  A gerência da sessão de trabalho do usuário, é gerenciada pelo conjunto de aplicações correntemente em execução para aquele usuário. A informação que descreve o estado da sessão de trabalho é armazenada no *AVU*, estando portando disponível de forma distribuída.

- *Gatekeeper*

  Serviço responsável por gerenciar os acessos entre as entidades externas ao ambiente de execução do EXEHDA e os serviços do mesmo, administrando os procedimentos de autenticação necessários.

3.4.2.4 Subsistema de Reconhecimento de Contexto e Adaptação

Este subsistema é composto pelos serviços (*i*) Collector, (*ii*) Deflector, (*iii*) *ContextManager*, (*iv*) AdaptEngine e (*v*) Scheduler. OS serviços, *AdaptEngine* e *Scheduler*

são responsáveis, pelo controle das adaptações de cunho funcional e não-funcional. Na perspectiva do EXEHDA a adaptação funcional implica a modificação do código sendo executado. A adaptação não-funcional, é responsável pela alocação de recursos durante a execução distribuída. Os serviços que compõem Subsistema de Reconhecimento de Contexto e Adaptação estão detalhados a seguir (YAMIN, 2004):

- *Collector*

  Este serviço atua coletando a informação bruta provenientes dos recursos envolvidos que, após distribuída cria os elementos de contexto. Para isto, o serviço *Collector* reúne a informação oriunda de vários componentes monitores *Monitor* e as repassa aos consumidores registrados *MonitoringConsumer*. O *Monitor* por sua vez é responsável pelo conjunto de sensores configuráveis. O principais consumidores da informação extraída pela monitoração são o serviços *ContextManager* e *Scheduler*.

- *Deflector*

  Este serviço cria a abstração de canais de *multicast* para uso na disseminação das informações monitoradas. A presença do serviço *Deflector* é decorrência da busca de escalabilidade para a arquitetura de monitoração do EXEHDA.

- *ContextManager*

  A informação produzida pela monitoração é tratada pelo *ContextManager*, o qual produz informações abstrata referentes aos elementos de contexto *(valores contextualizados)*.

- *AdaptEngine*

  A gerência de comportamentos adaptativos por parte das aplicações é feito pela *AdaptEngine*. Este serviço é responsável pelo controle das adaptações de cunho funcional. O uso do mesmo desobriga o programador de controlar os aspectos de mais baixo nível envolvidos na definição e liberação dos elementos de contexto junto ao *ContextManager*.

- *Scheduler*

  As adaptações de natureza não-funcional no EXEHDA, e que portanto não implicam alteração de código, são controladas pelo serviço Scheduler. O *Scheduler* emprega a informação de monitoração, obtida junto ao serviço *Collector*, para orientar os assinalamentos físicos das computações. Os assinalamentos físicos decorrem de instanciações remotas ou migrações realizadas pelo serviço *Executor*.

3.5 Considerações Finais

Inspirada na teoria dos sistemas dinâmicos, a idéia intuitiva do modelo GM é a modelagem de uma máquina por uma estrutura matemática, onde os conceitos básicos como memória, processadores e testes são construídos a partir de um espaço geométrico. A construção do modelo de Máquina Geométrica é desenvolvida em níveis, onde os construtores de processos são representados por funções lineares e o procedimento de completação garante a solução para equações recursivas.

A abordagem visual incentiva a programação no modelo GM, acrescentando um caráter didático e integrando a teoria com a prática de programação. O desenvolvimento da linguagem visual e a implementação da ferramenta VPE-GM possibilitam a construção gráfica de processos paralelos e distribuídos. A VPE-GM caracteriza o ambiente de desenvolvimento da ferramenta VirD-GM e viabiliza a validação de parâmetros enviados para ambiente de execução.

Dentre as propriedades do ambiente VPE-GM, salientam-se: (i)facilidade provida pela abordagem visual, com manipulação direta dos objetos gráficos e construtores; (ii) composição funcional dos construtores (produto seqüencial e paralelo, processos condicionais e somas não-determinística) quando do desenvolvimento de programas; (iii) metodologia indutiva, onde construtores são aplicados sobre processos definidos por ações indexadas por posições da memória global e compartilhada, viabilizando uma medida de custos computacionais; (iv) independência arquitetural em relação às aplicações do modelo D-GM; (v) semântica explícita e inerente ao ambiente de programação visual, apresentando aspectos da semântica do modelo D-GM embora não formalmente solicitados pelo programador.

Ainda neste capítulo foram exploradas as características e funcionalidades do *middleware* EXEHDA. O EXEHDA é estruturado em um núcleo mínimo e em serviços carregados sob demanda. Os principais serviços fornecidos estão organizados em subsistemas que gerenciam: (a) a execução distribuída; (b) a comunicação; (c) o reconhecimento do contexto; (d) a adaptação; (e) o acesso distribuído aos recursos e serviços; (f) a descoberta e (g) o gerenciamento de recursos. No EXEHDA, aplicações tanto do domínio da Computação em Clusters, em Grade, quanto da Computação Distribuída na perspectiva de elevada pervasividade, podem ser programadas e executadas sob seu gerenciamento.

Com base nos princípios discutidos neste capítulo, no capítulo 4 será apresentada a modelagem proposta para a VirD-GM, bem como os protótipos desenvolvidos e os testes realizados.

4 VIRD-GM: MODELAGEM E IMPLEMENTAÇÃO

Neste capítulo, os aspectos mais importantes de modelagem do ambiente de gerenciamento e execução do modelo D-GM, centrado na VirD-GM, serão apresentados.

Os aspectos da modelagem da VirD-GM foram descritos em etapas cujas atividades caracterizam sua metodologia de desenvolvimento. Primeiramente, apresenta-se a visão arquitetural da VirD-GM, cuja estrutura define sua organização física. Segue-se a descrição dos processos de instalação, configuração e execução da VirD-GM. Neste contexto, fez-se necessário implementar diversos componentes de software, os quais são apresentados nas próximas seções em uma abordagem integrando aspectos funcionais e lógicos. Finalizando este capítulo, descreve-se o fluxo de execução na VirD-GM.

A Figura 4.1 apresenta a relação entre os componentes do modelo D-GM, caracterizando os papéis do ambiente de desenvolvimento (VPE-GM) e do ambiente de execução (VirD-GM). O ambiente de desenvolvimento no modelo D-GM está centrado nos Editores de Processos e de Memória, que compõem os principais recursos da ferramenta VPE-GM, descrita no capitulo 3.3. Esta ferramenta foi desenvolvida com a intenção de tornar mais cômoda a sincronização e a modelagem dos conflitos de acesso à memória compartilhada, valendo-se para isto dos recursos da programação visual.

Figura 4.1: Visão Funcional do Modelo D-GM

4.1 Visão Arquitetural da D-GM

O desenvolvimento da ferramenta VirD-GM tem como premissa o uso do EXEHDA como ambiente de execução para os processos graficamente construídos nos editores do ambiente visual VPE-GM, facultando assim uma execução efetivamente distribuída e paralela para os códigos gerados a partir dos mesmos.

O principal objetivo de empregar o EXEHDA na composição da arquitetura da D-GM é diminuir o custo de especificar os aspectos e serviços básicos necessários para o tratamento da semântica que está agregada a mobilidade lógica e física da distribuição das computações, sua comunicação e sincronização. Os mecanismos utilizados para gerenciar tais aspectos são oferecidos na forma de uma biblioteca de software, a qual integra a API (*Application Programming Interface*) do EXEHDA. Através desta biblioteca é realizado o uso da arquitetura de software do EXEHDA.

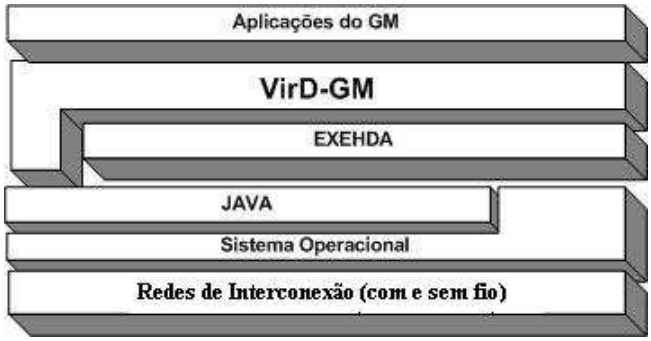

Figura 4.2: Visão Arquitetural do Modelo D-GM

A arquitetura da D-GM está modelada considerando esta perspectiva de integração com o EXEHDA, e apresenta uma organização lógica em três camadas, conforme Figura 4.2, destacando-se a seguinte estrutura organizacional:

- camada de aplicação (nível superior): onde ficam disponibilizadas as abstrações para programação de aplicações distribuídas e/ou paralelas, cuja natureza poderá compreender computações estocástica, intervalares e quânticas;

- camada de suporte e ambiente de execução (nível intermediário): estão os serviços da VirD-GM, que estendem as funcionalidades do EXEHDA para atendimento da especificações do modelo D-GM;

- camada de sistemas básicos (nível inferior): composta pelos sistemas e linguagens nativas que integram o meio físico de execução, consistindo na plataforma base de implementação é a Máquina Virtual Java, sistemas operacionais e redes de interconexão.

4.2 Organização da VirD-GM

Na perspectiva da D-GM, os recursos de infra-estrutura física estão mapeados por três abstrações decorrentes da organização do EXEHDA:

- **VirD-cel**: constitui a área de atuação de uma VirD-base e dos seus VirD-nodos. É no âmbito de uma VirD-Cel que acontecem as computações distribuídas e/ou paralelas concebidas seguido o modelo D-GM;

- **VirD-base**: equipamento responsável pela gerência da arquitetura da D-GM como um todo ou ainda a infra-estrutura para operação do VirD - nodo;

- **VirD-nodo**: são os equipamentos de processamento disponíveis, sendo responsáveis pela execução das computações da D-GM.

4.2.1 VirD-GM: Principais Subsistemas do EXEHDA Utilizados

Estão relacionados, a seguir, os subsistemas do EXEHDA (YAMIN et al., 2005) que são empregados pela VirD-GM, com uma breve descrição dos serviços utilizados.

- Serviços do Subsistema de Execução Distribuída

  - O serviço *Executor* acumula as funções de disparo de aplicações e de criação de objetos. Na implementação destas funções é empregada a instalação de código sob demanda. Para tal, interage com os serviços *CIB* e *BDA* (YAMIN et al., 2005).
  - O serviço *Cell Information Base* (CIB) implementa a base de informações da VirD-cel. Sua principal funcionalidade está relacionada à manutenção de informações referentes a infra-estrutura distribuída que forma o ambiente de suporte à execução.

- Serviços do Subsistema de Comunicação

  - O serviço WORB tem como objetivo permitir que programador possa abstrair aspectos de baixo nível relativos ao tratamento das comunicações em rede. Portanto, o WORB auxilia os programadores a concentrarem esforços no refinamento da semântica distribuída associada a aplicação em desenvolvimento. Para tanto, oferece um modelo de comunicação baseado em invocações remotas de método (similar ao RMI) porém sem exigir a manutenção da conexão durante toda a execução da chamada remota.
  - O serviço *CCManager* dá suporte ao mecanismo de comunicação, disponibilizando um mecanismo baseado na abstração espaço de *tuplas* [GEL 85], o qual prescinde da coexistência temporal de emissor e receptor. Este suporte é particularmente oportuno, à medida que este simplifica a construção de tais aplicações.

- Serviços do Subsistema de Acesso Pervasivo

  - O serviço BDA viabiliza a instalação de aplicações da VirD-GM sob demanda, ou seja, permite ao usuário disparar aplicações a partir de qualquer VirD-nodo integrante da VirD-cel.

4.2.2 Processo de Instalação, Configuração e Execução da VirD-GM

Nesta seção, descreve-se a seqüência de ações para instalação, configuração e execução da VirD-GM.

A primeira parte, caracterizada pelo processo de verificação de requisitos na camada básica consiste na seqüencialização de vários módulos que antecedem a instalação do *middleware* EXEHDA.

Estes módulos analisam a compatibilidade tanto com o sistema operacional instalado no VirD-nodo/VirD-base como com a rede de interconexão disponível, incluindo a análise da atual versão da Máquina Virtual Java. No diagrama da Figura 4.3 estão apresentados os componentes responsáveis pela instalação do sistema de suporte à execução na VirD-GM.

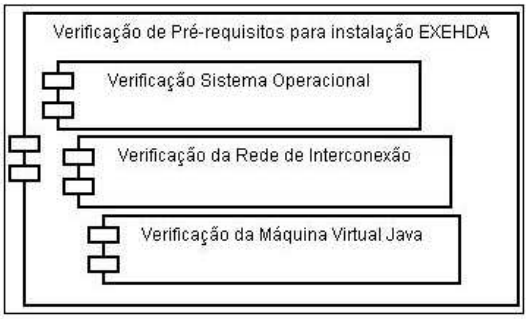

Figura 4.3: Componentes da Camada de Sistemas Básicos da Arquitetura D-GM

O Diagrama de Seqüência da Figura 4.4 apresenta a modelagem em UML da seqüência de eventos que devem ocorrer quando do processo de verificação de requisitos do sistema básico. Neste caso, tem-se como objetos os seguintes componentes: Interface do Sistema, Sistema Operacional, Controle da Interconexão e Máquina Virtual Java.

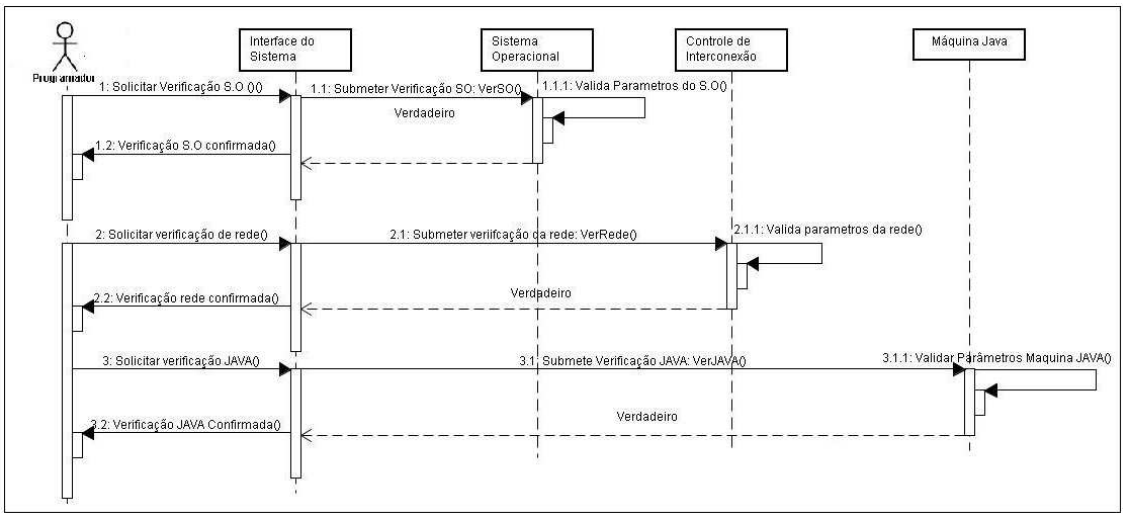

Figura 4.4: Processo de Verificação de Requisitos do Sistema Básico

Consolidada a validação dos pré-requisitos da primeira etapa, o sistema está pronto para a etapa de instalação do *middleware* EXEHDA,o qual consiste na verificação das permissões de usuário, da seleção do diretório de instalação e da transferência das bibliotecas e ferramentas do *middleware* para este diretório.

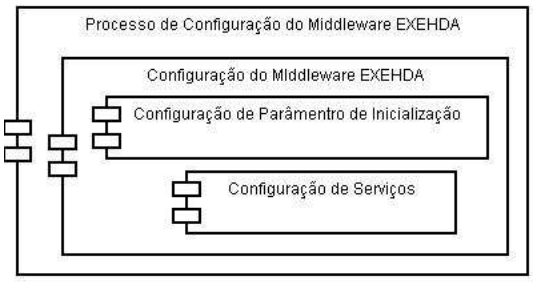

Figura 4.5: Componentes do Processo de Instalação do EXEHDA

O Diagrama de Seqüência na Figura 4.6 apresenta a seqüência de eventos que devem ocorrer quando do processo de instalação do *middleware* EXEHDA.

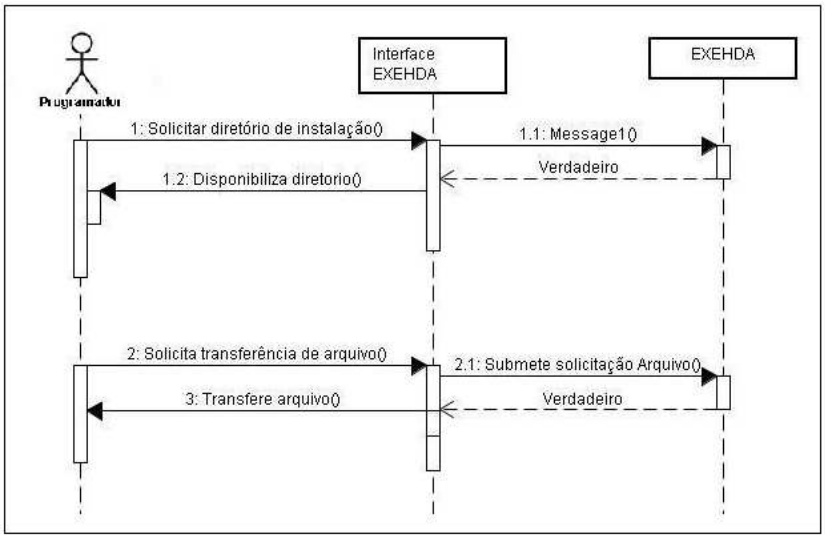

Figura 4.6: Seqüêncialização do Processo de Instalação do EXEHDA

A seguir, tem-se a efetivação da configuração do *middleware* EXEHDA, consistindo principalmente, na definição dos parâmetros de inicialização, tais como os parâmetros de identificação da VirD-Base e sua localização na rede de interconexão. A Figura 4.5 apresenta a configuração de parâmetros de *software* e a configuração de serviços como os principais componentes que implementam a etapa de configuração. Salienta-se também que, a configuração dos serviços pode ser efetivada por demanda ou quando da inicialização do *middleware* EXEHDA. Conforme descrito na Seção 4.3, a carga dos serviços para execução da VirD-GM estrutura-se, essencialmente, nos serviços de execução distribuída e de comunicação.

Após a instalação e configuração do EXEHDA, é permitido ao usuário proceder com a instalação da VirD-GM, etapa que se concretiza com a transferência das bibliotecas da VirD-GM para a base pervasiva de dados, a qual ocorre pela chamada do serviço BDA (YAMIN et al., 2005). Segue-se a etapa de configuração da VirD-GM, obtida com base nos parâmetros de configuração do *middleware* EXEHDA.

As últimas etapas, que antecedem a execução de aplicações, correspondem as inicializações do *middleware* EXEHDA e da VirD-GM, sendo que, no contexto deste trabalho, sua execução acontece em *background* como o sistema operacional.

Na inicialização do EXEHDA, ocorre a leitura do arquivo XML com as especificações previamente descritas quando de sua configuração. De forma análoga, e na sequencia da inicialização do *middleware* EXEHDA, podem ser inicializadas as aplicações já instaladas pelo serviço BDA, em especial, a VirD-GM.

4.3 Modelagem dos Componentes da VirD-GM

Nesta seção é apresentada a visão dos componentes das VirD-GM, possibilitando uma maior abstração do processo de modelagem. Especificando esta modelagem, são apresentados os diagramas de classes, que iteram a modelagem da VirD-GM.

4.3.1 Principais Componentes de Software da VirD-GM

As etapas mais significativas desenvolvidas para alcançar a execução dos processos no modelo D-GM são descritas na Figura 4.7. O diagrama de componentes, com suas correspondentes interações, mostra uma visão mais abrangente dos principais componentes da VirD-GM.

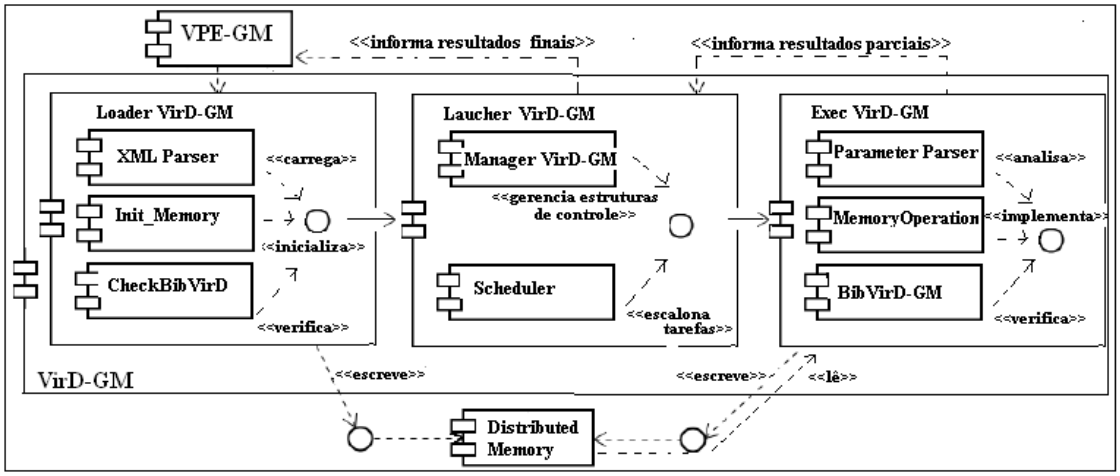

Figura 4.7: Diagrama de Componentes de Software da VirD-GM

4.3.1.1 Loader VirD-GM

O módulo *Loader VirD-GM* recebe os arquivos descritos dos processos e das memórias da aplicação exportados pelo ambiente VPE-GM. Este módulo, além da leitura das estruturas de processos e de memória, também lê os parâmetros para execução. Integram este módulo os seguintes componentes de software:

- *XML Parser:* que realiza o mapeamento dos arquivos descritores em XML para estruturas de dados internos da arquitetura;

- *InitMemory*: responsável por instanciar e inicializar a memória compartilhada. A memória é atualizada com os valores obtidos após o mapeamento do arquivo XML descritor da memória utilizada pela aplicação;

- *CheckBib*: verifica a disponibilidade de bibliotecas auxiliares nos repositórios de funções, as quais são requeridas pelas computações descritas no arquivo XML descritor dos processos da aplicação.

Consideram-se também inclusas neste módulo as atividades de preparação da arquitetura assim como a criação da matriz de adjacência utilizada no controle do fluxo de execução.

4.3.1.2 Launcher VirD-GM

O módulo *Launcher VirD-GM* tem como principal funcionalidade o gerenciamento do disparo da aplicação e da execução bem como do seu processamento. Este módulo inicia sua operação após a atuação no módulo *Loader VirD-GM*.

Dentre seus componentes, destacam-se:

- o componente *Manager VirD-GM*, atua gerenciando o controle da execução da computação baseado nos dados obtidos quando do acesso à matriz de adjacências; em seqüência, após esta verificação, o *Manager VirD-GM* insere os processos na lista de execução;

- o componente *Scheduler VirD-GM*, responsável pelo mapeamento físico dos processos já inclusos na referida lista de execução.

Este escalonamento está baseado em políticas previamente definidas quando da definição dos parâmetros operacionais da arquitetura de software do modelo D-GM.

4.3.1.3 Exec VirD-GM

A funcionalidade do módulo *Exec VirD-GM* consiste no disparo das computações paralelas nos nodos da célula, pela utilização de serviços do *middleware* EXEHDA.

Para tal, fez-se necessário o uso de operações de acesso à memória compartilhada, sendo que os principais componentes modelados e implementados são resumidos logo a seguir.

- O componente *Parameter Parser VirD-GM* é um analisador dos parâmetros de entrada e saída dos processos, que atua de acordo com a sintaxe associada a cada tipo de aplicação modelada na VirD-GM.

- O componente *Memory Operation VirD-GM*, tendo como funcionalidade o acesso à memória distribuída e a realização de operações de leitura e escrita, de acordo com as posições definidas nos parâmetros de entrada e saída.

- O componente *Bib VirD-GM*, o qual permite acesso ao repositório de funções associadas às computações realizadas pela aplicação nos VirD-Nodos.

4.3.2 Diagramas de Classes de Implementação da VirD-GM

A definição das classes de implementação está baseada na visão dos componentes apresentados na Seção 4.3.1. A Figura 4.8 apresenta o Diagrama UML correspondente as classes que integram a VirD-GM.

Utiliza-se uma visão simplificada do diagrama de classes gerados pela ferramenta *NetBeans* (NETBEANS, 2007), a partir do código já implementado da VirD-GM. As funcionalidades das correspondentes classes de implementação apresentadas no diagrama da Figura 4.8 são brevemente descritas a seguir.

- Na classe *VirDProc-Loader*, que implementa o componente *Loader VirD-GM*, é construído o grafo dirigido que representa o fluxo de execução de processos, viabilizando o mapeamento do arquivo de entrada XML. Após a construção deste grafo, ocorre a construção de sua representação como matriz de adjacência, consistindo na estrutura de ordenação do fluxo de execução. O atributo *inputFile* representa o arquivo a ser carregado, o qual está associado ao construtor da classe *VirDProc-Loader*.

- *VirDMemLoader* consiste na classe da VirD-GM que estrutura a memória, de acordo com o arquivo descritor de memória.

- A classe *VirDGraph* representa uma estrutura de grafo, em concordância com os requisitos da arquitetura VirD-GM. A ação inicial do construtor da classe é a geração de um grafo base, cujos nodos são dinamicamente gerados pelo arquivo de entrada.

- A classe *VirDNode* define um nodo do grafo da VirD-GM, representando um processo elementar no modelo D-GM. Caracteriza-se como uma agregação da classe *VirDGraph*, ou seja, os objetos da primeira completam as informações referentes os objetos da segunda.

- A classe de implementação *VirDLauncher*, que implementa o componente *Launcher VirD-GM*, recebe como parâmetro a estrutura da matriz de adjacência e, assim, ocorre a verificação de quais processos podem ser executados, respeitando-se as dependências e a exclusão mútua no acesso à memória. Os processos liberados são inseridos em uma lista de processos para execução, a qual será lida pelo escalonador. O escalonamento é definido através de políticas pré-definidas. Esta classe utiliza os serviços do *middleware* EXEHDA, para criação de objetos remotos (*Executor*) e para chamada destes métodos (*Worb*).

- A classe abstrata *VirDExec*, associada ao componente *Exec VirD-GM*, define métodos que devem ser, obrigatoriamente, implementados por suas classes filhas.

- A classe *VirDExecImpl*, implementando a classe abstrata *VirDExec*, é responsável pela análise dos parâmetros de entrada e saída dos processos e pelo posterior acionamento das funções presentes na biblioteca da VirD-GM. Além destes, para garantir execução das computações, são também consideradas as operações de leitura e escrita na memória compartilhada.

- A classe *VirDProc-Elem* provê representação para um processo elementar, cujos atributos estão associados a correspondente definição no modelo D-GM, a ação (operação aritmética) e uma posição de escrita na memória (modelada por um ponto no espaço geométrico).

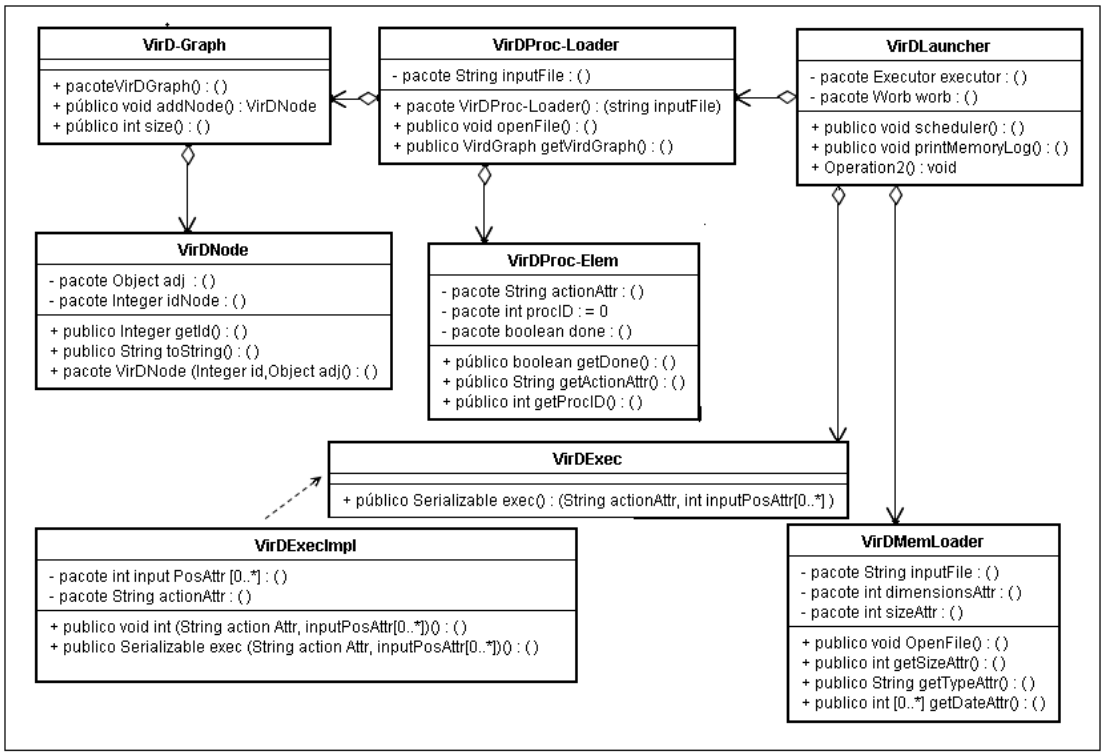

Figura 4.8: Diagrama de Classes da VirD-GM

4.4 Controle do Fluxo de Execução na VirD-GM

A modelagem do fluxo de execução dos processos é obtida através do arquivo descritor dos processos da aplicação. Esse arquivo gerado no VPE-GM, descreve todas as informações necessárias para a execução da computação, contendo construções seqüenciais e paralelas.

Para garantir a sincronização destes processos, são definidos grafos dirigidos, onde cada nodo representa um processo do programa a ser executado. As funções empregadas pelos processos estão previamente definidas em bibliotecas da VirD-GM localizadas na BDA.

A seqüencia do programa no grafo é determinada pelas arestas, representadas por arcos dirigidos interligando nodos. Todo nodo aponta para outro nodo, com exceção do último, indicando deste modo a execução. Desta forma, um processo pode ser executado somente se todos os seus predecessores já tiverem sido executados. Este procedimento garante a exclusão mútua no acesso as áreas de memória comum, evitando conflitos em operações de leitura e/ou escrita.

O grafo característico de uma aplicação é representado na VirD-GM por uma matriz $A = (a_{ij})_{n \times n}$ de adjacências, onde n indica o número de processos a serem executados. Cada linha (i) da matriz representa um processo e cada coluna (j) representa a dependência entre processos identificados pelos índices i e j, quando do conflito de acesso à posições de memórias. O valor booleano 1 a posição a_{ij} indica que existe uma dependência, e portanto uma seqüencialidade na execução. Caso contrário, se a marcação contém o valor booleano 0 indica que não há dependência, ocorrendo a sincronização dos processos.

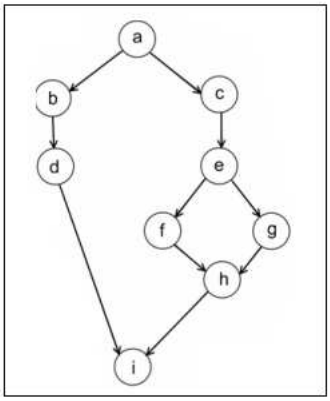

Figura 4.9: Grafo Dirigido Característico de uma Aplicação

A matriz correspondendo ao grafo da Figura 4.9, está apresentada na Figura 4.10.

Atualização →
Teste ↓

	a	b	c	d	e	f	g	h	i
a	0	1	1	0	0	0	0	0	0
b	0	0	0	1	0	0	0	0	0
c	0	0	0	0	1	0	0	0	0
d	0	0	0	0	0	0	0	0	1
e	0	0	0	0	0	1	1	0	0
f	0	0	0	0	0	0	0	1	0
g	0	0	0	0	0	0	0	1	0
h	0	0	0	0	0	0	0	0	1
i	0	0	0	0	0	0	0	0	0

Figura 4.10: Matriz de Adjacência para Controle do Fluxo de Execução na VirD-GM

Ao final da execução de um processo, ocorre a liberação de suas dependências, ou seja, a matriz de adjacências é atualizada com valor 0 na linha correspondente.

Na interface do VPE-GM, apresentada na Figura 4.11, é possível ver o Diagrama Processo composto de operações de soma, subtração, assim como a paralelização dos processos elementares identificados pelas expressões sum^5 e sub^6. O arquivo XML, descritor da aplicação, gerado a partir desta especificação está mostrado na Figura 4.12, e inclui todas as informações necessárias para gerência da execução paralela. Estas informações consideram o tipo de processo, a área de memória alterada e as dependências entre processos para uso quando do processamento.

O grafo de dependências é construído a partir da busca recursiva de processos (elementares) sobre o arquivo descritor de programas escrito em XML, modelado por componentes denominados envelopes. A estrutura deste arquivo XML pode ser vista como uma árvore, cujo número de ramos varia de acordo com a especificação do programa editado no ambiente VPE-GM. O código em XML correspondendo ao diagrama da Figura 4.11 é apresentado na Figura 4.12.

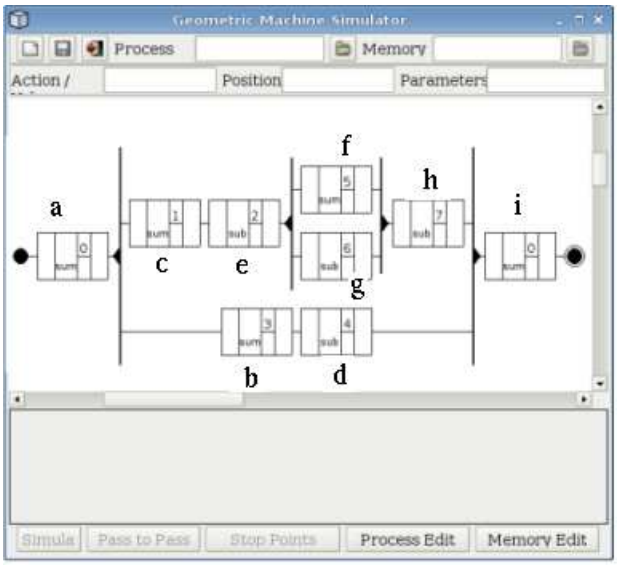

Figura 4.11: Diagrama Processo no VPE-GM

```
<process repr="env" tipo="seq" parametro="" pos="0, 1, 2, 5, 6, 7, 3, 4, 8">
<process repr="conselem" acao="sum" pos="0" parametro="(9,10)"></process>
<process repr="env" tipo="paralelo" parametro="" pos="1, 2, 5, 6, 7, 3, 4">
<process repr="env" tipo="seq" parametro="" pos="1, 2, 5, 6, 7">
<process repr="conselem" acao="sum" pos="1" parametro="(11,12)"></process>
<process repr="env" tipo="seq" parametro="" pos="2, 5, 6, 7">
<process repr="conselem" acao="sub" pos="2" parametro="(15,16)"></process>
<process repr="env" tipo="paralelo" parametro="" pos="5, 6">
<process repr="conselem" acao="sum" pos="5" parametro="(19,20)"></process>
<process repr="conselem" acao="sub" pos="6" parametro="(21,22)"></process>
</process>
<process repr="conselem" acao="sub" pos="7" parametro="(23,24)"></process>
</process>
</process>
<process repr="env" tipo="seq" parametro="" pos="3, 4">
<process repr="conselem" acao="sum" pos="3" parametro="(13,14)"></process>
<process repr="conselem" acao="sum" pos="4" parametro="(17,18)"></process>
</process>
</process>
<process repr="conselem" acao="sum" pos="8" parametro="(25,26)"></process>
</process>
```

Figura 4.12: Código XML correspondente ao Diagrama Processo

A biblioteca DOM (*Document Object Model*) (CHAMPION et al., 1997) foi utilizada para a implementação do *parser* de arquivos XML.

A função que realiza o *parser* do arquivo descritor dos processos da aplicação, seleciona os processos que serão adicionados à lista de computações a serem realizadas, incluindo as correspondentes dependências. Se o nodo atual não for um processo elementar, então tem-se um envelope, que pode conter outros envelopes ou processos. A presença de envelopes determina sucessivas chamadas recursivas até que restem apenas processos elementares na estrutura de árvore. A partir da relação de processos liberados para execução pela matriz de adjacências é realizado o escalonamento dos mesmos para os nodos da VirD-cel. Para cada processo é criada uma *thread*, que fará a execução efetiva do processo e a atualização da sua matriz de adjacências.

4.5 Considerações Finais

O estudo, a modelagem e correspondente implementação do paralelismo no modelo D-GM com base na aplicação de grafos de dependências pode ser resumido na descrição do exemplo Diagrama Processo, explorado na Seção 4.4.

Com base neste estudo, consolidou-se a definição e implementação da arquitetura de software VirD-GM, que viabiliza a execução distribuída e concorrente dos programas concebidas segundo o modelo D-GM, usando como suporte o *middleware* EXEHDA.

Os resultados alcançados com a modelagem e implementação da VirD-GM apontam em outros desafios, podendo-se considerar como outras atividades relacionadas com extensões possíveis ao ambiente: (i) a otimização dos algoritmos gerados, com análise de custo; (ii) a implementação dos demais operadores do modelo D-GM, incluindo as computações não-determinísticas; e (iii) as otimizações para arquiteturas de memória compartilhada.

5 VIRD-GM: APLICAÇÕES DE TESTE

Este capítulo resume o comportamento observado com as aplicações de teste desenvolvidas para avaliação da VirD-GM. O foco buscado foi explorar as funcionalidades de disparo e gerência das aplicações distribuídas e/ou paralelas desenvolvidas utilizando o ambiente de desenvolvimento gráfico do modelo D-GM (VPE-GM).

As aplicações são do tipo sintética e não contemplam otimizações quanto ao desempenho, sua modelagem tem por objetivo sobretudo verificar a correteza de comportamento do mecanismo de comunicação com ambiente de desenvolvimento, bem como do mecanismo para controle do fluxo de execução.

Salienta-se que, para os testes descritos neste capítulo, foram realizadas ao todo 5 medições para cada valor medido, observando-se pequena flutuação, da ordem de $0,15\%$, nos resultados obtidos. Os valores apresentados nas Seções 5.1.2 e 5.2.2 correspondem a uma média obtida a partir da medição realizada. A avaliação dos resultados ocorreu em um *cluster* que disponibiliza 10 unidades de processamento na sua totalidade, possuindo um conjunto de nodos homogêneos quanto a sua capacidade de processamento. Cada nodo consiste de um processador ATM-Athlon de 2020 MHz, com 256 MBytes de memória RAM.

5.1 Aplicação para o Cálculo do Número π pelo Método de Monte Carlo

Esta aplicação tem como objetivo obter o valor do número π pela implementação de um método numérico baseado na proposta de Monte Carlo (PRESS et al., 1992). Do ponto de vista da abordagem computacional, tem-se uma computação paralela do tipo *bag-of-tasks*, cujo número de computações paralelas é controlado pelo programador quando na etapa de desenvolvimento da aplicação.

5.1.1 Metodologia de Implementação

A metodologia para o cálculo do número π, utilizando o método de Monte Carlo, tem a sua interpretação geométrica baseada na Figura 5.1:

A área sombreada na Figura 5.1 indica a quarta parte da área determinada pela circunferência de raio l, pode ser calculada pela expressão:

$$A = \frac{1}{4} * \pi * l^2 \tag{5.1}$$

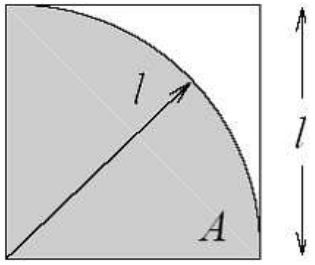

Figura 5.1: Interpretação Geométrica utilizada no Cálculo do π

Para aplicação do Método de Monte Carlo, considera-se P_{sum} como o cardinal do conjunto $\mathbb{P}_{sum}$ de pontos (x, y) gerados aleatoriamente, os quais estão limitados pelo quadrado apresentado na Figura 5.1, ou seja $(x, y) \in \mathbb{P}_{sum}$, sempre que $0 \leq x, y \leq 1$.
De forma análoga, P_{in} indica o cardinal do subconjunto de $\mathbb{P}_{in} \subseteq \mathbb{P}_{sum}$, cuja localização no processo aleatório de geração de pontos está compreendida pela área sombreada da Figura 5.1. Assim, a área limitada pela região sombreada é dada pela expressão:

$$A = l^2 * \frac{P_{in}}{P_{sum}} \tag{5.2}$$

A igualdade entre as Equações (5.1) e (5.2), resulta na expressão:

$$\frac{1}{4} * \pi * l^2 = l^2 * \frac{P_{in}}{P_{sum}}. \tag{5.3}$$

E assim, isolando π na equação (5.3), obtém-se uma expressão de aproximação para o número π:

$$\pi = 4 * \frac{P_{in}}{P_{sum}}. \tag{5.4}$$

A representação gráfica do algoritmo apresentado na Figura 5.3 implementa esta metodologia, e consiste essencialmente, na geração aleatória de pontos $(x, y) \in \mathbb{P}_{sum}, 0 \leq x, y \leq 1$, seguida pela identificação e contagem dos correspondentes pontos para posterior aplicação da expressão de aproximação para o número π indicada na Equação(5.3).

Considerando as características do algoritmo é possível definir sua estruturação em três etapas: (i) particionamento das tarefas entre vários processos do Modelo D-GM; (ii) cálculo do número π pelo Método de Monte Carlo para cada processo do Modelo D-GM; e (iii) agrupamento dos resultados e definição do valor aproximado do número π.

As três etapas da aplicação foram realizadas no ambiente de desenvolvimento do Modelo D-GM. A visualização das etapas em que o algoritmo foi estruturado pode ser visto no diagrama produzido pela ferramenta VPE-GM apresentado na Figura 5.2.

O diagrama apresentado na Figura 5.2 foi exportado para os arquivos descritores da aplicação e da memória. O código gerado pela interface da VPE-GM para os arquivos descritores pode ser visualizado nas Figura 5.3 e Figura 5.4.

Descreve-se agora os diferentes fatores que compõem a aplicação apresentada no diagrama da Figura 5.2o qual foi gerado na interface da VPE-GM.

1. A primeira etapa realiza o particionamento das tarefas e contempla dois processos:

   - `SetNum`^{i}, processo que recebe um valor numérico como parâmetro de entrada

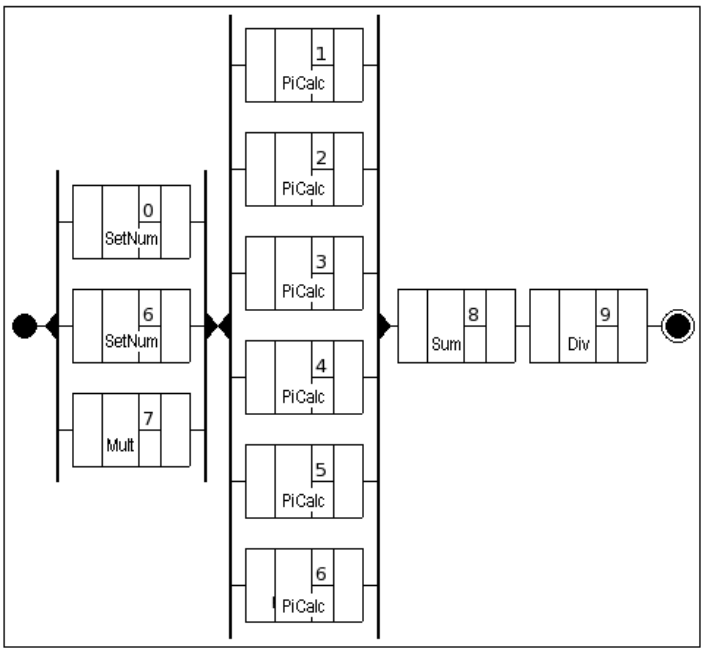

Figura 5.2: Aplicação πCalc Modelada Graficamente na Ferramenta VPE-GM

```
- <save arq="/home/felipe/workspace/VirD-GM/calcPI6.xml" pos="0, 6, 7, 1, 2, 3, 4, 5, 8, 9">
  - <process repr="env" tipo="seq" parametro="" pos="0, 6, 7, 1, 2, 3, 4, 5, 8, 9">
      <process repr="terminicio" x="1225.0" y="1300.0" cor="BLACK" />
    - <process repr="env" tipo="seq" parametro="" pos="0, 6, 7, 1, 2, 3, 4, 5, 8, 9">
      - <process repr="env" tipo="paralelo" parametro="" pos="0, 6, 7">
          <process repr="conselem" acao="setNum" pos="0" parametro="1497618370" x="1298.0" y="1229.0" cor="BLACK" />
          <process repr="conselem" acao="setNum" pos="6" parametro="6" x="1298.0" y="1300.0" cor="BLACK" />
          <process repr="conselem" acao="mul" pos="7" parametro="0,6" x="1298.0" y="1371.0" cor="BLACK" />
        </process>
      - <process repr="env" tipo="paralelo" parametro="" pos="1, 2, 3, 4, 5, 6">
          <process repr="conselem" acao="piCalc" pos="1" parametro="0" x="1416.0" y="1122.0" cor="BLACK" />
          <process repr="conselem" acao="piCalc" pos="2" parametro="0" x="1416.0" y="1193.0" cor="BLACK" />
          <process repr="conselem" acao="piCalc" pos="3" parametro="0" x="1416.0" y="1264.0" cor="BLACK" />
          <process repr="conselem" acao="piCalc" pos="4" parametro="0" x="1416.0" y="1335.0" cor="BLACK" />
          <process repr="conselem" acao="piCalc" pos="5" parametro="0" x="1416.0" y="1406.0" cor="BLACK" />
          <process repr="conselem" acao="piCalc" pos="6" parametro="0" x="1416.0" y="1477.0" cor="BLACK" />
        </process>
      - <process repr="env" tipo="seq" parametro="" pos="8, 9">
          <process repr="conselem" acao="sum" pos="8" parametro="1,2,3,4,5,6" x="1520.0" y="1300.0" cor="BLACK" />
          <process repr="conselem" acao="div" pos="9" parametro="8,7" x="1610.0" y="1300.0" cor="BLACK" />
        </process>
      </process>
      <process repr="termfim" x="1671.0" y="1300.0" cor="BLACK" />
    </process>
  </save>
```

Figura 5.3: Código XML do Arquivo Descritor de Processos da Aplicação *PiCalc*

```
<memory>
        <position dimension="1" size="30"></position>
        <values>Integer</values>
        <data>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</data>
</memory>
```

Figura 5.4: Código XML do Arquivo de Configuração de Memória da Aplicação *PiCalc*

atribuindo este valor à posição de memória i;

- `Mult`^{i}, processo que recebe uma lista de valores numéricos como parâmetros

de entrada, efetuando a correspondente multiplicação destes valores, atribuindo o valor do produto à posição de memória i;

2. A segunda etapa consiste na aplicação do Método de Monte Carlo, sendo então modeladas seis computações paralelas para cálculos *piCalc*^{i}:

   - ao receber um valor inteiro indicando uma posição de memória, o processo realiza uma operação de leitura nesta posição e, a partir deste valor efetua o cálculo do número π pelo Método de Monte Carlo segundo a equação 5.4; segue-se a atribuição do valor calculado à posição de memória i.

     Na Figura 5.5 o algoritmo que implementa o processo *PiCalc*, está descrito em meta-linguagem:

Entrada: número de iterações da execução em n
Resultado: valor estimado do π em PI

1: **for** $P_{total} = 1$ to n **do**
2: $x \leftarrow$ número aleatório entre 0 e 1
3: $y \leftarrow$ número aleatório entre 0 e 1
4: **if** $(x^2 + y^2 \leq 1)$ **then**
5: $P_{in} \leftarrow P_{in} + 1$
6: **end if**
7: **end for**
8: PI $\leftarrow 4 * \frac{P_{in}}{P_{total}}$

Figura 5.5: Algoritmo que Implementa o Processo *PiCalc*

3. Na terceira etapa do diagrama da Figura 5.2, obtém-se o valor aproximado do número π. Esta etapa contempla os seguintes processos:

   - `Sum`^{i}, processo que recebe uma lista de valores numéricos como parâmetros de entrada, efetuando o somatório destes valores e atribuindo o valor da soma à posição de memória i;
   - `Div`^{i}, processo que recebe uma lista de valores numéricos como parâmetros de entrada, efetuando a divisão destes valores, atribuindo o valor do quociente à posição de memória i;

Faz-se necessário o uso de barreiras de sincronização entre cada uma das três etapas, caracterizando o início e término das mesmas. Deste modo, o algoritmo não permite a execução dos processos de cálculo do número π sem antes finalizar o processo de particionamento. Da mesma forma, não é possível executar a etapa de agrupamento dos resultados sem antes terminar a computação baseada no o método de Monte Carlo.

Os processos que compõem cada uma das três etapas podem ser executados concorrentemente, ou seja, são processos sem dependência de dados, podendo ser executados separadamente e em cada um dos VirD-nodos. Neste contexto, a execução simultânea é dependente da disponibilidade de unidades computacionais da célula, naquele instante de processamento.

Salienta-se ainda que, a escolha do local onde utilizar as barreiras de sincronização são de responsabilidade do desenvolvedor do programa, caracterizando assim as etapas de assinalamento e decomposição explícitas, de acordo com a classificação proposta em (B.WILKINSON; ALLEN, 2004), quando da aplicação do paralelismo.

Tabela 5.1: Avaliação da Aplicação *PiCalc*

Execução.	***N proc.***	***Tempo de Execução (s)***	***Speedup***
A	1	3595.235	-
B	2	1835.356	1,96
C	3	1241.500	2,90
D	4	1242.573	2,89
E	5	1242.530	2,89
F	6	612.079	5,87

5.1.2 Resultados Obtidos

Pela definição da arquitetura da VirD-GM, um dos nodos do *cluster* homogêneo, distingue-se dos demais, caracterizado como nodo VirD-base, e sendo responsável pelo escalonamento das tarefas e o controle do fluxo de dados. Neste nodo são submetidos os processos da aplicação.

Foram realizados seis casos de teste, e para cada um destes, foi obtida uma média dos resultados obtidos. A diferenciação na execução de cada caso de teste dá-se pelo número de nodos utilizados nas computações paralelas, sendo que os arquivos descritores de memória e do programa, ambos gerados nas interfaces da VPE-GM, são idênticos para todos os casos de testes. A modelagem da VirD-GM possibilita que a alocação dinâmica do números de nodos seja passada como parâmetro de entrada.

A Tabela 5.1 tem como principal objetivo resumir os resultados em termos do *speedup* da aplicação referente ao cálculo do número π.

Nesta aplicação, a maior carga computacional concentrou-se nos seis processos *PiCalc* que realizam o cálculo computacional do número π. A seguir temos a descrição das execuçõe:

- **Execução_A**: utiliza um nodo para processamento, obtendo um tempo de execução de 3595.235 segundos, caracterizando um processamento seqüencial;

- **Execução_B**, utiliza dois nodos para processamento, obtendo um tempo de execução de 1835.356 segundos;

- **Execução_C, Execução_D e Execução_E**, foram realizadas com cinco, quatro e três nodos, respectivamente, sendo obtido um tempo de processamento ao redor de 1242.573 segundos para três execuções. Nestas execuções o numero de processos era maior que o número de nodos, tendo sido necessário duas fases em sequencia para conclusão do processamento. Por exemplo no caso de cinco nodos (Execução_E), cinco computações foram processadas em paralelo, seguidas do processamento de uma computação, caracterizando uma organização (5:1), no caso de quatro (Execução_D) nodos ocorreu uma organização (4:2) e no caso de três nodos (Execução_C) ocorreu uma organização (3:3);

- **Execução_F**, utiliza seis nodos para processamento, tendo sido cada computação processada individualmente em um VirD-nodo, obtendo um tempo de execução de 612.079 segundos.

Tabela 5.2: Estado Final da Memória do Cálculo do π

P-N	6	5	4
0	$1.49761837E9$	$1.49761837E9$	$1.49761837E9$
1	$4.70479724E9$	$4.704917404E9$	$4.70491432E9$
2	$4.704920508E9$	$4.704987596E9$	$4.704965756E9$
3	$4.70482914E9$	$4.704850188E9$	$4.704915892E9$
4	$4.704867992E9$	$4.704870512E9$	$4.70492078E9$
5	$4.704833968E9$	$4.704861096E9$	$4.704959872E9$
6	$4.704940032E9$	$4.704911888E9$	$4.704854192E9$
7	$8.98571022E9$	$8.98571022E9$	$8.98571022E9$
8	$2.822918888E10$	$2.8229398684E10$	$2.8229530812E10$
9	3.141564571843048	3.1415879204704646	3.1416026247060524

P-N	3	2	1
0	$1.49761837E9$	$1.49761837E9$	$1.49761837E9$
1	$4.70501216E9$	$4.704919804E9$	$4.70493868E9$
2	$4.704760028E9$	$4.704860928E9$	$4.704876944E9$
3	$4.70483038E9$	$4.704905032E9$	$4.7049015E9$
4	$4.704829396E9$	$4.704996368E9$	$4.7049015E9$
5	$4.70498938E9$	$4.704996368E9$	$4.704907292E9$
6	$4.704914776E9$	$4.704936976E9$	$4.704907292E9$
7	$8.98571022E9$	$4.704815956E9$	$4.704922676E9$
8	$2.822933612E10$	$8.98571022E9$	$4.704911748E9$
9	3.141580957860001	3.141591969120945	3.141594615099885

A tabela 5.2 apresenta os valores finais, nas posições de memória associadas a cada um dos processos que estão definidos, graficamente, no diagrama da Figura 5.2. Por exemplo, a posição de memória 0 armazena o número de pontos (`PSum`) que será utilizado como parâmetro de entrada para o cálculo do número π. Este valor inicializa o algoritmo de Monte Carlo, e por esta razão é o mesmo para todos os testes. Nas posições de memória de 1 a 6, ficam armazenados os valores obtidos com a execução dos processos *PiCalc*. Na posição de memória 7 está armazenado o total de pontos utilizados na computação do π. Por fim, a posição de memória 9 recebe o valor aproximado do número π, que consiste no resultado final de cada teste, conforme mostra a última linha da tabulação apresentada na tabela 5.2.

5.2 Aplicação para Quebra de Senhas pelo Método da Força Bruta

A segunda aplicação considerada neste trabalho denominada *PassBreak* tem como objetivo a quebra de senhas codificadas pela aplicação do algoritmo MD5 (*Message-Digest algorithm 5*), cuja definição está baseada em uma função *hashing*, desenvolvido pela RSA Data Security, Inc., descrito na RFC 1321, e muito utilizado por softwares responsáveis por sistemas de autenticação.

O MD5 como algoritmo unidirecional não permite a transformação inversa que resultaria no texto que lhe deu origem. O método de verificação é, então, feito pela comparação de duas strings geradas pela função *hashing*, uma senha a ser quebrada armazenada na memória, e a outra gerada na tentativa de decodificação através da varredura de um espaço de busca.

5.2.1 Metodologia de Implementação

Apresenta-se a seguir uma breve descrição da metodologia utilizada na aplicação. Aplica-se o Método de Força Bruta, definindo o espaço de busca contendo o conjunto de caracteres que podem ocorrer na geração de senhas e a função de concatenação que

permite a geração de *strings*, as quais constituem as entradas para a função *hashing*.

A cada geração da função, ocorre a correspondente comparação com o valor armazenado na memória. Tem-se então dois casos a considerar: (i) se o resultado da comparação for verdadeiro, significa que a senha foi encontrada e o algoritmo é finalizado; e (ii) caso contrário, o algoritmo continua a procura no espaço de busca.

A aplicação *PassBreak* considerou a possibilidade da ocorrência de até seis computações concorrentes.

A estruturação da aplicação *PassBreak* foi desenvolvida considerando três etapas, cuja diferenciação está diretamente relacionada com o espaço de busca, ou seja, na primeira etapa tem-se como meta a quebra de todas as senhas com tamanho de 3 caracteres, seguida da etapa que busca a quebra das senhas com tamanho de 4 caracteres e por fim, a etapa responsável pela quebra das maiores senhas com tamanho de 5 caracteres. Esta estratégia de modelagem busca minimizar a complexidade associada à aplicação *PassBreak* e otimizar as comparações geradas pela função *hashing*.

Estas etapas podem ser visualizadas no diagrama gerado na interface da VPE-GM, apresentado na Figura 5.6. Descreve-se a seguir, os processos que compõem a construção das três etapas da aplicação *PassBreak*.

1. Para alcançar a quebra de senhas de tamanho 3 na primeira etapa, foram sincronizados seis processos elementares identificados pela expressão *PassBreak3* ^{i}, onde i indica a posição na memória unidimensional que recebe uma *string* correspondendo a entrada da função *hashing* se a comparação é satisfeita e, portanto, uma senha foi encontrada; caso contrário, recebe uma *string* vazia.

2. A segunda etapa é modelada de forma análoga à primeira, considerando-se apenas que o espaço de busca compreende senhas de quatro caracteres, e as computações realizadas pelos processos *PassBreak*.

3. Para completar a modelagem, considera-se na terceira etapa que o espaço de busca compreende senhas de cinco caracteres, sendo as computações realizadas pelos processos *PassBreak*.

O diagrama da Figura 5.6 deu origem ao arquivo descritor da aplicação, cujo código gerado pelo VPE-GM pode ser visualizado na Figura 5.8.

Os resultados de *speedup* obtidos na Tabela 5.3 são decorrentes do número de caracteres que compões as senhas, bem como de sua composição. Neste sentido, senhas que exigem um esforço de busca maior no alfabeto de caracteres impõem um custo computacional maior, exigindo um tempo maior de processamento.

No caso das execuções A e I, indicadas na Tabela 5.3, como somente foram computadas senhas de mesmo tamanho, a computação aconteceu em uma única fase de processamento, justificando-se assim a potencialização do *speedup*, o qual se mostra mais próximo do número de processadores utilizados.

5.2.2 Resultados Obtidos

Na apresentação dos resultados, consideram-se as características da aplicação descrita e a correspondente metodologia de implementação. Nesta aplicação, em cada etapa todos os nodos foram alocados dinamicamente pela aplicação.

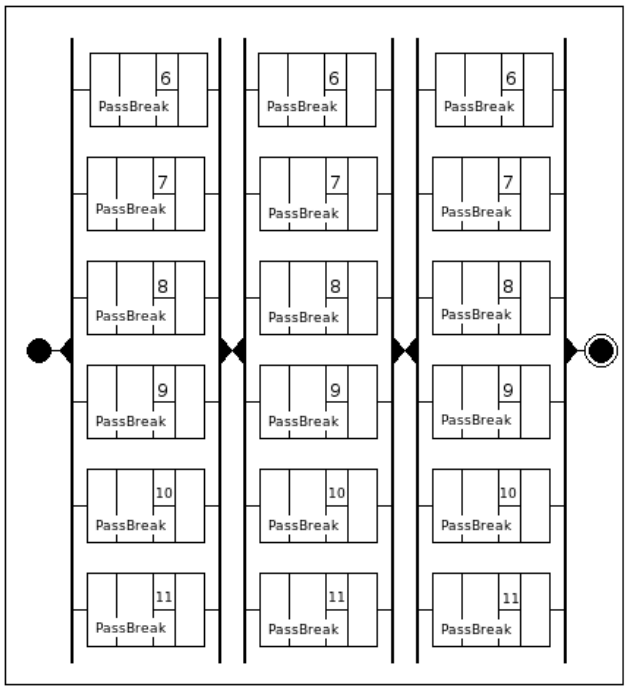

Figura 5.6: Aplicação *PassBreak* Modelada no VPE-GM

```
- <save arq="passBreakerTeste.xml" pos="6, 7, 8, 9, 10, 11">
  - <process repr="env" tipo="seq" parametro="" pos="6, 7, 8, 9, 10, 11">
      <process repr="terminicio"/>
    - <process repr="env" tipo="seq" parametro="" pos="6, 7, 8, 9, 10, 11">
      - <process repr="env" tipo="paralelo" parametro="" pos="6, 7, 8, 9, 10, 11">
          <process repr="conselem" acao="passBreak3" pos="6" parametro="0"/>
          <process repr="conselem" acao="passBreak3" pos="7" parametro="1"/>
          <process repr="conselem" acao="passBreak3" pos="8" parametro="2"/>
          <process repr="conselem" acao="passBreak3" pos="9" parametro="3"/>
          <process repr="conselem" acao="passBreak3" pos="10" parametro="4"/>
          <process repr="conselem" acao="passBreak3" pos="11" parametro="5"/>
        </process>
      - <process repr="env" tipo="paralelo" parametro="" pos="6, 7, 8, 9, 10, 11">
          <process repr="conselem" acao="passBreak4" pos="6" parametro="0"/>
          <process repr="conselem" acao="passBreak4" pos="7" parametro="1"/>
          <process repr="conselem" acao="passBreak4" pos="8" parametro="2"/>
          <process repr="conselem" acao="passBreak4" pos="9" parametro="3"/>
          <process repr="conselem" acao="passBreak4" pos="10" parametro="4"/>
          <process repr="conselem" acao="passBreak4" pos="11" parametro="5"/>
        </process>
      - <process repr="env" tipo="paralelo" parametro="" pos="6, 7, 8, 9, 10, 11">
          <process repr="conselem" acao="passBreak5" pos="6" parametro="0"/>
          <process repr="conselem" acao="passBreak5" pos="7" parametro="1"/>
          <process repr="conselem" acao="passBreak5" pos="8" parametro="2"/>
          <process repr="conselem" acao="passBreak5" pos="9" parametro="3"/>
          <process repr="conselem" acao="passBreak5" pos="10" parametro="4"/>
          <process repr="conselem" acao="passBreak5" pos="11" parametro="5"/>
        </process>
      </process>
      <process repr="termfim"/>
    </process>
</save>
```

Figura 5.7: Código XML do Arquivo Descritor de Processos da Aplicação *PassBreak*

Na implementação das três etapas da aplicação *PassBreak* foram verificadas as especificações modeladas nas atividades metodológicas descritas na seção 5.2.1.

Consideram-se os seguintes parâmetros:

- t_i^n como o tempo que um processador utiliza para executar o processo i referente a

```
<memory>
        <position dimension="1" size="6"></position>
        <values>String</values>
        <data>  aeec645249be520c2b20c9d4b33912ee,
                f947e6c5ca751155df70628f65c28967,
                ae2f866194753ab18a3a38b426b41958,
                d0cbddd770c023ee20170c7dc1741d42,
                d65299a5ae9e14fd5cd4ff0a59d2d041,
                c295e2d817b939ed7cd97fede0a233ab
        </data>
</memory>
```

Figura 5.8: Código XML do Arquivo de Configuração de Memória da Aplicação *PassBreak*

Tabela 5.3: Avaliação da Aplicação *PassBreak*

Exec	***S3***	***S4***	***S5***	T_{Seq}	T_{Par}	***Speedup***
A	6	0	0	4693	869	5.4
B	5	1	0	6466	3336	1.9
C	4	1	1	16265	14056	1.2
D	3	2	1	18796	14814	1.3
E	2	2	2	28340	15917	1.8
F	1	2	3	42497	17429	2.4
G	1	1	4	50057	17429	2.9
H	0	1	5	59915	16567	3.6
I	0	0	6	69101	13872	5.0

uma senha de tamanho n.

- $T_k^n = max_{1 \leq i \leq k}\{t_i^n\}$.

A expressão algébrica usada para calcular o tempo seqüencial, tempo paralelo e o *speedup*, referente às execuções A e I na Tabela 5.3, são dados, respectivamente, por:

$$T_{Seq}(A) = \sum_{i=1}^{6} t_i^3, \quad T_{Par}(A) = T_6^3 = max_{1 \leq i \leq 6}\{t_i^3\}, \quad S = \frac{T_{Seq}(A)}{T_{Par}(A)};$$

$$T_{Seq}(I) = \sum_{i=1}^{6} t_i^5, \quad T_{Par}(I) = T_6^5 = max_{1 \leq i \leq 6}\{t_i^5\}, \quad S = \frac{T_{Seq}(I)}{T_{Par}(I)}.$$

De forma análoga, a expressão algébrica usada para calcular o tempo seqüencial, o tempo paralelo e o *speedup*, referente à execução E, na Tabela 5.3, é dada por:

$$T_{Seq}(E) = \sum_{i=1}^{2} t_i^3 + \sum_{i=1}^{2} t_i^4 + \sum_{i=1}^{2} t_i^5, \quad T_{Par}(E) = T_2^3 + T_2^4 + T_2^5, \quad S = \frac{T_{Seq}(E)}{T_{Par}(E)}.$$

Os resultados do speedup apresentados na Tabela 5.3 são decorrentes tanto do número de caracteres que as compõem, como também da composição das senhas no que

diz respeito a posição dos caracteres no alfabeto de busca, isto é, senhas que exigem um esforço de busca maior no alfabeto impõe um custo computacional correspondentemente maior.

Nos casos de execução A e I da Tabela 5.3 foram quebradas senhas de um mesmo tamanho, três e seis caracteres respectivamente, em uma única fase de processamento. O fato do processamento se concentrar em uma única fase, potencializou o speed obtido, pois o número de processos paralelos atingiu o máximo possível, que no caso o limite seria seis computações concorrentes.

5.3 Considerações Finais

Os resultados obtidos caracterizaram que a VirD-GM apresentou o comportamento previsto, respeitando as devidas barreiras de sincronização inseridas entre as tarefas e seus respectivos acessos à memória através de operações de leitura e escrita.

A aplicação *PiCalc* consiste na utilização do Método de Monte Carlo para aproximar o valor do número π. O Método de Monte Carlo é baseado na estatística para a resolução de problemas numéricos, através da verificação do número de vezes que um evento é verdadeiro com relação ao número total de vezes que esse evento ocorreu.

Devido as características lineares do Método de Monte Carlo os tempos de execução da aplicação *PiCalc*, mostraram uma coerência quanto ao seu *speedup*, nas diferentes configurações de VirD-Nodos utilizados nos testes descritos.

A segunda aplicação, cujo objetivo é a descoberta de um conjunto de senhas codificada pela aplicação do algoritmo MD5, segundo método de Força Bruta.

Outras aplicações mais diretamente relacionadas às implementações paralelas e/ou distribuídas de algoritmos da Computação Científica, caracterizados por manipulação do fluxo de dados, podem ser executados na VirD-GM, na continuidade do trabalho.

6 CONSIDERAÇÕES FINAIS

A abordagem do texto já contempla considerações conclusivas ao longo do mesmo, além das específicas ao final de cada capítulo. Entretanto, nestas considerações finais destacam-se as principais conclusões, resume-se as contribuições mais relevantes e apresenta-se as principais publicações obtidas com o desenvolvimento do trabalho. Com base nos resultados obtidos, também são apontadas novas perspectivas ao Projeto D-GM.

O Projeto D-GM foi concebido para viabilizar uma execução efetivamente distribuída e paralela para os códigos gerados pelo modelo GM, cuja a execução acontece através da VirD-GM. Esta dissertação teve como foco do seu esforço de pesquisa a modelagem , implementação e avaliação da VirD-GM.

As atividades de pesquisas desenvolvidas ao longo deste trabalho, especialmente nos estudos realizados quando da modelagem e da construção da VirD-GM, permitiram alcançar algumas constatações, como:

- a proposta de integração de uma abordagem teórica, com uma infra-estrutura operacional para processamento paralelo e distribuído mostra-se atual e de relevância para a pesquisa na área.

- a abordagem visual para ambientes de desenvolvimento mostra-se oportuna para a modelagem de aplicações paralelas e distribuídas, dentre os diversos aspectos que corroboram neste sentido destaca-se:

  - oferece facilidades de manipulação dos componentes de software;
  - provê validação dos parâmetros que constituem às configurações enviadas ao ambiente de execução;
  - possibilita o uso de uma semântica multidimensional para a computação paralela e distribuída;
  - introduz um novo instrumento didático, motivando o ensino de conceitos fundamentais da computação concorrente.

Por outro lado, a crescente evolução tecnológica promove o surgimento continuado de novas arquiteturas distribuídas e/ou paralelas, o que potencializa a necessidade de produzir código com a maior portabilidade possível, de modo que fique reduzido os custos de portabilidade para os diferentes hardwares disponíveis.

6.1 Principais Contribuições

Identificam-se contribuições de duas naturezas. A seguir estas contribuições estão detalhadas, sendo caracterizadas as suas contribuições técnicas especificas.

- Concepção e modelagem da integração entre um modelo abstrato de computação (GM) e um ambiente computacional para execução paralela e distribuída (EXEHDA), tendo como ênfase o desenvolvimento e execução concorrente de algoritmos para computação científica via aplicação da programação visual. Esta contribuição pode ser caracterizada pelos seguintes aspectos específicos:

  - Identificação dos principais ambientes de programação paralela e/ou distribuída que utilizam a abordagem visual;
  - Estudo da proposta do projeto D-GM, com ênfase na compreensão das noções de concorrência e conflito intermitentes com as noções de comunicação e sincronização de processos, os quais relacionam-se diretamente com a estrutura espaço-temporal do modelo GM;
  - Revisão do ambiente de programação visual VPE-GM e das definições dos construtores de processos (produtos paralelo e seqüencial, somas determinística e não determinística) e as possibilidades de configuração dos estados computacionais;
  - Identificação da organização física e lógica, dos recursos e serviços e demais funcionalidades do EXHEDA que constituem o suporte à execução paralela e/ou distribuída para o projeto D-GM.

- Modelagem e implementação da VirD-GM, consolidando a integração da visão arquitetural (centrada na concepção e construção da VirD-GM) com a visão funcional (caracterizada pela extensão do ambiente VPE-GM), ambas estruturas constituem as partes essenciais do Projeto D-GM. Por sua vez, esta contribuição pode ser caracterizada pelos seguintes aspectos específicos:

  - A integração dos editores da ferramenta APV-GM com o módulo de controle e execução da computação paralela e distribuída da VirD-GM;
  - Aplicação e customização dos serviços disponibilizados pelo *middleware* EXEHDA para atendimento às demandas computacionais da VirD-GM;
  - Concepção da arquitetura de software denominada VirD-GM, proposta como gerente da computações paralelas definidas no Projeto D-GM, e modelagem das camadas de aplicação, de suporte ao ambiente de execução e de sistemas básicos;
  - Especificação e implementação do controle do fluxo de dados e manipulação das dependências pelo uso de matrizes de adjacências, estruturação de barreiras de sincronização visando a corretude da execução;
  - Avaliação da execução da concorrência síncrona pela realização de testes envolvendo o *middleware* e as aplicações.

6.2 Outras Contribuições

Além dos resultados técnicos descritos seção anterior, destacam-se duas outras contribuições.

6.2.1 Suporte à Simulação da Programação Distribuída e Paralela com Ênfase no Desenvolvimento de Software Livre

Ao longo do desenvolvimento do trabalho ficou identificada a possibilidade de exploração do paralelismo no modelo GM de modo síncrono, sobre sistemas distribuídos e/ou paralelos, tendo a VirD-GM atuado como integradora entre as aplicações desenvolvidas no ambiente visual de programação VPE-GM e o *middleware* EXEHDA.

A construção do protótipo baseado na arquitetura proposta para a VirD-GM realizou a integração entre o ambiente de programação visual VPE-GM e o *middleware* EXHEDA.

Na prototipação da ferramenta VirD-GM, alguns dos resultados obtidos foram:

- consolidação do ambiente de desenvolvimento, pela integração entre a edição gráfica de processos e a configuração espacial da memória e a geração da entrada de dados para o ambiente de execução VirD-GM;
- suporte à simulação e à execução de algoritmos paralelos integrando bibliotecas específicas para aplicações da computação científica;
- suporte a especificação de relacionamentos entre módulos de programas paralelos e distribuídos, potencializando as práticas de reutilização de componentes de software existente.

Na concepção e implementação do modelo D-GM, e a validação do protótipo baseado na especificação arquitetural da VirD-GM está centrada no paradigma do software livre. O ambiente de programação visual para o modelo GM (*Visual Programming Environment for the Geometric Machine Model* - VPE-GM) utiliza a linguagem *Python* no desenvolvimento de sua interface gráfica, visando uma interação do programa com o usuário.

As bibliotecas gráficas da linguagem *Python* (*wxPython*) viabilizam a criação de interfaces gráficas para construção de processos e especificação de memórias, assegurando também a escolha da estrutura e dimensão do espaço geométrico. A linguagem XML suporta as atribuições dos editores possibilitando o gerenciamento dos arquivos gráficos gerados nas interfaces durante a edição, e posterior apresentação dos resultados ao ambiente de execução, denominado VirD-GM.

O protótipo da VirD-GM foi implementado em Java. No que diz respeito à gerência da execução distribuída e paralela, a proposta em desenvolvimento considera uma arquitetura, onde o sistema suporte está caracterizado pelo *middleware* EXEHDA, definida especificamente para esta finalidade. O módulo de gerenciamento da VirD-GM é capaz de prover estrutura para apoio à comunicação entre o ambiente de desenvolvimento (VPE-GM) e ambiente de execução, tendo como entrada o arquivo descritor da aplicação distribuída e/ou paralela em XML a ser executada sobre a arquitetura de software da VirD-GM.

Neste sentido, o trabalho desenvolvido colaborou para:

- Produção e organização da documentação tanto do *software* do VirD-GM, como das aplicações nele desenvolvidas;
- .

6.2.2 Consolidação da Integração entre os Grupos de Pesquisa GFMC e G3PD

A modelagem das abstrações da D-GM (*Distributed Geometric Machine*) em ambiente de execução distribuída, denominado VirD-GM(*Virtual Distributed Geometric Machine*), consolida a proposta de integração entre dois grupos de pesquisa da Universidade Católica de Pelotas (UCPEL): Grupo de Matemática e Fundamentos da Computação (GFMC) e o Grupo de Pesquisa em Processamento Paralelo e Distribuído (G3PD).

A realização do Projeto proporcionou o desenvolvimento teórico-prático do trabalho na área de programação distribuída, envolvendo principalmente, o paralelismo síncrono e o não-determinismo. Desta forma, este trabalho colaborou para que se tornasse efetiva a cooperação entre as linhas de pesquisa, as disciplinas e os projetos referentes a estes grupos, consolidando-os junto ao recém aprovado Mestrado em Ciência da Computação, no Programa de Pós-Graduação em Informática da UCPEL, e às instituições nacionais de apoio à pesquisa e ao desenvolvimento tecnológico.

Assim, com base nos resultados alcançados e já publicados quando do desenvolvido desta dissertação, outras etapas foram concebidas como continuidade ao desenvolvimento do Projeto D-GM e poderão contar com apoio e suporte conquistado junto ao Edital MCT/CNPq 15/2007 Universal.

Neste contexto, tem-se ainda:

- Divulgação dos resultados do trabalho realizado durante o desenvolvimento da dissertação através de publicações. As principais estão relacionadas na Seção 6.3.
- Disseminação no âmbito da UCPEL e da UFPel de conhecimento pertinente às áreas de Processamento Paralelo e Distribuído
- Repasse do conhecimento e das tecnologias associadas quando da implementação do protótipo da VirD-G em um site na Internet. A URL do site é https://olaria.ucpel.tche.br/d-gm, sendo disponibilizado no mesmo, além de documentação pertinente, ferramentas de software, bem como códigos desenvolvidos.

6.3 Publicações Realizadas

- Publicações em Congressos
  - Congresso latino-americano
    - FONSECA, Vanessa Souza da ; REISER, Renata Hax Sander ; YAMIN, Adenauer Correa ; COSTA, Antônio Carlos da Rocha ; PILLA, Maurício. *VirD-GM: Introducing a Modelling and Execution Environment for the Distributed Geometric Machine Machine.* In: Conferencia Latinoamericana de Informática, 2007, San José. Anais da Conferencia Latinoamericana de Informática - CLEI 2007, 2007. p. 1-10.

- Congresso Internacional

  * FONSECA, Vanessa Souza da ; REISER, Renata Hax Sander ; YAMIN, Adenauer Correa ; PILLA, Maurício ; COSTA, Antônio Carlos da Rocha . *VirD-GM: Towards to a Grid Computing Environment*. In: Seventh IEEE International Symposium on Cluster Computing and the Grid CCGrid 2007, 2007, Rio de Janeiro. Web Proceedings on the CCGrid. Rio de Janeiro : IEEE Computer Society, 2007. v. 1. p. 1-5.

- Congressos Nacionais

  * MUNHOZ, Felipe Natale ; FONSECA, Vanessa Souza da; REISER, Renata Hax Sander; PILLA, Maurício; YAMIN, Adenauer Correa . *Paralelismo no modelo D-GM: Emprego de Grafos de Dependências.* In: Workshop de Sistemas Computacionais de Alto desempenho - Gramado. Anais do CTIC-WSCAD 2007, p. 1-4.

- Congresso Regional

  * MUNHOZ, F. N.; FONSECA, V. S.; VIVAN, J.; REISER, R. H. S.; YAMIN, A. C. *Arquitetura de Software da VirD-GM: Modelagem e Funcionalidades*, Proceedings of ERAD 2008: VIII Escola Regional de Alto Desempenho, 2008, UNISC - Santa Cruz, 209–212.

- Resumos publicados em Congressos Nacionais

  - FONSECA, Vanessa Souza; REISER, Renata Hax Sander; YAMIN, Adenauer Correa; PILLA, Maurício . *O Modelo de Máquina Geométrica: Revisão das Potencialidades do Paralelismo.* In: VII Escola Regional de Alto Desempenho, 2007, Porto Alegre. Anais VII Escola Regional de Alto Desempenho. Porto Alegre : Instituto de Informática da UFRGS, 2007. v. 1. p. 63-64.

  - MUNHOZ, F.; FONSECA, Vanessa Souza; REISER, Renata Hax Sander; YAMIN, Adenauer Correa. *Arquitetura de Software da VirD-GM: Modelagem e Funcionalidades.* In: VIII Escola Regional de Alto Desempenho, 2008, Santa Cruz. Anais VIII Escola Regional de Alto Desempenho. Santa Cruz: UNISC, 2008. v. 1. p. 1-4.

  - MONTEIRO, Eduarda Rodrigues; MARON, K. Adriano; MUNHOZ, Felipe Natale; FONSECA, Vanessa S.; AMARAL, Rafael B.; REISER, Renata H. S. *Abordagem Visual para Computações no qGM.* In XIX CONGRESSO DE MATEMÁTICA APLICADA E COMPUTACIONAL, 2007, SBMAC. Santa Catarina: SBMAC/UFSC, 2007. v.1, p. 1-1.

  - MUNHOZ, Felipe Natale; MONTEIRO, Eduarda Rodrigues; MARON, K. Adriano; FONSECA, Vanessa S.; AMARAL, Rafael B.; REISER, Renata H. S.; PILLA, Maurício ; YAMIN, Adenauer Correa. *Abordagem Visual para Computações no qGM.* In XIX CONGRESSO DE MATEMÁTICA APLICADA E COMPUTACIONAL, 2007, SBMAC. Santa Catarina: SBMAC/UFSC, 2007. v.1, p. 1-1.

6.4 Trabalhos Futuros

Os resultados atigidos até o momento no projeto D-GM, viabilizaram um protótipo para a VirD-GM (FONSECA, 2006; FONSECA et al., 2007, 2008, 2005), cuja concepção tem-se mostrado interessante para análise do paralelismo, segundo abstrações do modelo GM. Dentre as alternativas para continuidade das pesquisas destacam-se como significativas as descritas a seguir:

- Avaliação dos resultados obtidos com a implementação do protótipo da Virdt-GM com ênfase no desenvolvimento e execução concorrente de algoritmos para a computação científica.
  - Simulação e análise via *software* do paralelismo quântico, segundo abstrações da extensão quântica do modelo GM (qGM).
  - Abordagem visual para simulação de algoritmos quânticos pela integração do editor de circuitos quânticos *qEdit* (MONTEIRO et al., 2006) à ferramenta VirD-GM.
  - Integração da biblioteca *qGM-Analyzer* (MARON et al., 2007; MONTEIRO et al., 2007) ao ambiente de execução da VirD-GM.
  - Desenvolvimento de aplicações voltadas principalmente para a Computação Intervalar, incluindo execução de algoritmos intervalares.
- Revisões e otimizações na arquitetura de *software* VirD-GM.
  - Extensão do estudo das noções de conflito, diretamente relacionadas com a estrutura não determinística do modelo GM.
  - Revisão das noções de comunicação e sincronização de processos, considerando diferentes arquiteturas, como as arquiteturas Multi-Core.
  - Otimização dos mecanismos e estruturas de dados utilizados na comunicação entre os ambientes de desenvolvimento e de execução da VirD-GM.
  - Proposição, implementação e avaliação de políticas de escalonamento para o módulo de gerenciamento e execução da VirD-GM.
- Reavaliação da interface gráfica considerando as demandas do Projeto D-GM, e as aplicações científicas em desenvolvimento.
- Organização da produção técnica gerada visando sua disponibilização como *software* livre.

REFERÊNCIAS

AMARAL, R. B.; REISER, R.; COSTA, A. Interpretações do Interferômetro de Mach-Zehnder no Modelo qMG. In: XXXI CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2008. **Anais...** SBMAC/ANAMA/PA, 2008. p.1–10. meio digital.

BARDOHL, R.; ERMEL, C. Visual Specification and Parsing of a Statechart Variant using GENGED. In: SYMPOSIUM ON VISUAL LANGUAGES AND FORMAL METHODS, 2001. **Proceedings...** [S.l.: s.n.], 2001. p.5–7.

BEGUELIN, A.; DONGARRA, J.; GEIST, A.; MANCHEK, R.; MOORE, K.; NEWTON, P.; SUNDERAM, V. **HeNCE**: A Users' Guide Version 2.0. (disponível via ftp://netlib2.cs.utk.edu/hence/HeNCE-2.0-doc.ps.gz).

BOAS, P. E. **Machine Models and Simulations**. [S.l.]: HandBook of theoretical Computer Science, 1990. 1-66p.

BOVET, D. P.; CRESCENZI, P. **Introduction to the Theory of Complexity**. [S.l.]: Prentice Hall, 1994. 282p.

BUSCHMANN, F. **Pattern-oriented software arquitecture** : a system of patterns. New York: [s.n.], 1996. John Wiley.

B.WILKINSON; ALLEN, M. **Parallel Programming**: Techniques and Applications Using Networked Workstations and Parallel Computers. [S.l.]: Prentice Hall, 2004. 496p.

CARDOSO, M. B.; REISER, R. H. S.; A. C. R. COSTA, L. L. B.; PRESTES, D. G. Introduzindo um Ambiente de Programação Visual para a Máquina Geométrica. In: II SIMPóSIO DE INFORMáTICA DA REGIãO CENTRO DO RS, 2003, Santa Maria, Brasil. **Anais...** [S.l.: s.n.], 2003. p.1–8. disponível em CD do simpósio.

CARDOSO, M. B.; REISER, R. H. S.; COSTA, A. C. R.; DIMURO, G. P. Especificando a Recursão na Linuagem Visual para o Modelo de Máquina Geométrica. In: XVIII CONGRESSO REGIONAL DE INICIAçãO CIENTíFICA E TECNOLóGICA EM ENGENHARIA, 2003, Itajaí, SC. **Anais...** [S.l.: s.n.], 2003. p.1–5. disponível em CD do congresso.

CARDOSO, M. B.; REISER, R. H. S.; COSTA, A. C. R.; DIMURO, G. P. Ambiente de Programação Visual para a Máquina Geométrica. In: VIII SIMPóSIO DE INFORMáTICA, 2003, Uruguaiana, RS. **Anais...** [S.l.: s.n.], 2003. p.1–5. disponível em CD do congresso.

CARDOSO, M.; REISER, R.; COSTA, A. Uma Abordagem Funcional na Implementação do Algoritmo de teleportação Quântica. In: XXVIII CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL, 2005, Rio de Janeiro,SBMAC. **Anais...** [S.l.: s.n.], 2005. v.1, p.1–6.

CARDOSO, M.; REISER, R.; COSTA, A.; MUNHOZ, F. **Explaining Basic Aspects of Quantum Computation Based on Functional Approach**. submetido.

CHAMPION, M.; BYRNE, S.; NICOL, G.; WOOD, L. **Document Object Model (Core) Level 1**.

DAVOLI, R.; LA.GIACHINI; O.BABAOGLU; ALVISI, L. Parallel Computing in Networks of Workstations with Paralex. In: **IEEE Transactions on Parallel and Distributed Systems**. [S.l.: s.n.], 1996. p.371–384.

DEHNE, F. K. H. A.; FABRI, A.; RAU-CHAPLIN, A. Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers. In: SYMPOSIUM ON COMPUTATIONAL GEOMETRY, 1993. **Anais...** [S.l.: s.n.], 1993. p.298–307.

FONSECA, V. S. **Um Estudo das Potencilidades e da Exploração do Paralelismo com Aplicações no Modelo de Máquina Geométrica**. [S.l.]: Universidade Católica de Pelotas, 2006. (Trabalho Individual 2006/1-009).

FONSECA, V. S. da; REISER, R.; COSTA, A.; YAMIN, A. C.; PILLA, M. VirD-GM: Introducing a Modelling and Execution Environment for the Distributed Geometric Machine Machine. In: CLEI2007: XXXII CONFERêNCIA LATINOAMERICANA DE INFORMáTICA, 2005, San José. **Anais...** Universidad Javeriana, 2005. p.1–10. meio digital.

FONSECA, V. S.; REISER, R. H. S.; YAMIN, A. C.; PILLA, M. L. VirD-GM: Towards a Grid Computing Environment. In: CCGRID 2007, 2007. **Proceedings...** [S.l.: s.n.], 2007. p.1–6.

FONSECA, V. S.; REISER, R. H. S.; YAMIN, A. C.; PILLA, M. L. Modelando o Paralelismo na Arquitetura VirD-GM. In: ERAD 2008, 2008. **Proceedings...** [S.l.: s.n.], 2008. p.139–140.

GIRARD, J. **Between logic and quantic**: a tract, manuscript. [S.l.: s.n.], 2003.

GIRARD, J.; LAFONT, Y.; TAYLOR, P. M. **Visual Object-Oriented Programming**. Cambridge: Cambridge University Press, 1989. 176p.

GIRARD, J. Y. The system F of variable types, fifteen years later. In: **Theoretical Computer Science**. [S.l.: s.n.], 1986. n.45, p.159–192.

GIRARD, J. Y. Linear logic. In: **Theoretical Computer Science**. [S.l.: s.n.], 1987. p.1–102.

JUNIOR, A. A. C.; NASU, C. Y.; CáCERES, E.; MONGELLI, H. Modelos realísticos de Computação Paralela. **INFOCOMP Journal of Computer Science**, [S.l.], v.2, n.1, p.16–20, 2000.

LINHALIS, F. **Interface Básica para um Servidor Universal**. 2000. 185p. Tese (Doutorado em Ciência da Computação) — Universidade São Paulo/ICMC-USP - São Carlos - SP.

MALACARNE, J.; GAYER, C. Ambiente de Programação Visual com Objetos Distríbuidos em Java. **SBRC 2001 - 19º Simpósio Brasileiro de Redes de Computadores**, Florianópolis, SC, p.1–16, 2001.

MARKOV, S. On quasivector spaces of convex bodies and zonotopes. **Numerical Algorithms**, Netherlands, v.1, p.475–488, 2004.

MARKOV, S.; ALT, R. Stochastic Aritmetic, Addition and Multiplication by Scalars. **Appl. Ner. Math.**, [S.l.], v.1, p.475–488, 2004.

MARON, A.; MUNHOZ, F.; MONTEIRO, E.; AMARAL, R.; ; REISER, R. qGM-Analyser: Ferramenta Computacional para Análise das Computações Quânticas Obtidas no Modelo qGM. In: XXX CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2007. **Anais...** SBMAC/UFSC/SC, 2007. p.1–2. meio digital.

MARRIOTT, K.; MEYER, B. **Visual Language Theory**. [S.l.]: Springer, 1998.

M.C.BROWN. **Python**:The Complete Reference. Berkeley: Osborne/McGraw-Hill, 2001.

MILNER, R. A Calculus of Communicating Systems. In: **Lecture Notes in Computer Science**. Berlin: Springer-Verlag, 1980. v.92.

MINSKY, M. **Computation, Finite and Infinite Machines**. New York: Prendice Hall, 1967.

MONTEIRO, E.; MARON, A.; MUNHOZ, F.; AMARAL, R.; ; REISER, R. Abordagem Visual para Computações no Modelo qGM. In: XXX CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2007. **Anais...** SBMAC/UFSC/SC, 2007. p.1–2. meio digital.

MONTEIRO, E.; MUNHOZ, F.; PORTO, D.; VIZZOTTO, J.; REISER, R. Construção de Circuitos Quânticos Integrada a Ambiente de Programação Visual. In: XIX CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2006. **Anais...** SBMAC/UNICAMP, 2006. p.1–2. meio digital.

MONTEIRO, E.; MUNHOZ, F.; VIZZOTTO, J.; REISER, R.; COSTA, A. Estudo sobre uma simulação em linguagem de programação funcional do algoritmo de criptografia quântica BB84. In: WORKSHOP ESCOLA DE INFORMAçãO E COMPUTAçãO QUâNTICA, 2006. **Anais...** EDUCAT/UCPEL, 2006. p.1–2.

NETBEANS, I. **NetBeans**. (disponível via WWW em http://www.netbeans.org).

P.KACSUK; LOURENÇO, J. C.; DÓZSA, J.; FADGYAS, T. A graphical development and debugging environment for parallel programs. **Parallel Computing, Parallel Computing Journal, Elsevier**, [S.l.], v.22, n.13, p.1747–1770, 1997.

PRADO, F.; LUCREDIO, D. **Ferramenta MVCASE**. 2001. 100p. Tese (Doutorado em Ciência da Computação) — Universidade Federal de São Carlos.

PRESS, W.; FLANNERY, B. P.; TEUKOLSKY, S. A.; VETTERLING, W. T. **Numerical Recipes in C**: The Art of Scientific Computing. [S.l.]: Cambridge University Press, 1992.

PRESTES, D.; CARDOSO, M.; REISER, R.; COSTA, A. Implementando o Ambiente de Programação Visual para a Máquina Geométrica. **I Workcomp Sul - Workshop de Ciências da Computação e Sistemas de Informação da Região Sul**, Itajaí, SC, p.1–11, 2004.

PRESTES, D.; REISER, R.; COSTA, A. O Simulador para o Ambiente de Programação Visual para o Modelo de Máquina Geométrica. In: CNMAC 2005 - XXVIII CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2005, Santo Amaro, SP. **Anais...** SBMAC-RJ, 2005. p.175–176. meio digital.

REISER, R. **A máquina geométrica - um modelo computacional para concorrência e não-determinismo usando como estrutura os Espaços Coerentes**. 2002. 254p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática/UFRGS - Porto Alegre.

REISER, R.; COSTA, A.; AMARAL, R. Leading to Quantum Semantic Interpretations based on Coherence Spaces. In: NANOBIO 2007 - II WORKSHOP EM NANOTECNOLOGIA E COMPUTAçãO INSPIRADA EM BIOLOGIA, 2007. **Anais...** PUC/RJ, 2007. p.1–6. meio digital.

REISER, R.; COSTA, A.; AMARAL, R. Quantum Computing: Computation in Coherence Spaces. In: PROCEEDINGS OF WECIQ WORKSHOP AND SCHOOL OF QUANTUM INFORMATION AND COMMUNICATION, 2., 2007. **Anais...** UFCG - PR, 2007. p.1–10.

REISER, R.; COSTA, A. C. R.; DIMURO, G. Distributed approach for the Geometric Machine Model. In: PARA 2004 - WORKSHOP ON STATE-OF-THE-ART IN SCIENTIFIC COMPUTING VALIDETED, 2004, Ligby. **Anais...** [S.l.: s.n.], 2004. n.1, p.106–114.

REISER, R.; COSTA, A. C. R.; DIMURO, G. The Quantum Geometric Machine. In: TH GAMM/IMACS - INTERNATIONAL SYMPOSIUM ON SCIENTIFIC COMPUTING, COMPUTER ARITHMETIC AND VALIDATES NUMERICS, 11., 2004, Fukuoka, Japão. **Anais...** [S.l.: s.n.], 2004. p.132–132.

REISER, R.; COSTA, A.; DIMURO, G. First Steps in the Construction of the Geometric Machine Model. In: **TEMA - Tendências em Matemática Aplicada e Computacional**. [S.l.: s.n.], 2002. v.3, n.1, p.183–192.

REISER, R.; COSTA, A.; DIMURO, G. The distributed version of the geometric machine model. In: XXVI CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2003. **Anais...** SBMAC-RJ, 2003. p.493–493. meio digital.

REISER, R.; COSTA, A.; DIMURO, G. A programming language for the Interval Geometric Machine Model. In: **Eletronic Notes in Theoretical Computer Science**. [S.l.: s.n.], 2003. v.84, p.1–12.

REISER, R.; COSTA, A.; DIMURO, G. O modelo de Máquina Geométrica Intervalar. In: **TEMA - Tendências em Matemática Aplicada e Computacional**. [S.l.: s.n.], 2003. v.4, n.1, p.109–118.

REISER, R.; COSTA, A.; DIMURO, G. The Stochastic Geometric Machine Model. In: TEMA TENDêNCIAS EM MATEMáTICA APLICADA E COMPUTACIONAL, 2004, Rio de Janeiro, Brazil. **Anais...** SBMAC/RJ, 2004. v.5, n.2, p.305–314.

REISER, R.; COSTA, A.; DIMURO, G. The Interval Geometric Machine. In: **Numerical Algorithms**. Dordrecht: Kluwer, 2004. v.37, p.357–366.

REISER, R.; COSTA, A.; DIMURO, G. The Distributed Interval Geometric Machine Model. In: LECTURE NOTES IN COMPUTER SCIENCE, 2005. **Anais...** Springer, 2005. n.3732, p.179–188.

REISER, R.; COSTA, A.; DIMURO, G. Programming in the Quantum Geometric Machine Model. In: XXVIII CONGRESSO NACIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 2005. **Anais...** SBMAC/SENAC-RJ, 2005. p.1–6. meio digital.

REISER, R.; COSTA, A.; DIMURO, G.; CARDOSO, M. Specifying the Geometric Machine Visual Language. In: **IEEE Symposium on Visual Languages and Formal Methods**. [S.l.: s.n.], 2003. p.1–3.

REISER, R.; COSTA, A.; DIMURO, G.; CARDOSO, M. Utilizando a Programação Visual no Modelo de Máquina Geométrica. In: CLEI 2003 CONFERêNCIA LATINOAMERICANA DE INFORMáTICA, 2003, La Paz, Bolívia. **Anais...** Universidad Major san Andrés, 2003. p.1–10.

ROSSUM, G.; DRAKE, L. **Python Tutorial**. (disponível via WWW em http://www.pythonorg/doc/current/tut/tut.html).

SCOTT, D. The lattice of flow diagrams. In: **Lecture Notes in Mathematics**. [S.l.]: Springer Verlag, 1971. p.311–372.

SKILLICORN, D. B.; TALIA, D. Models and Languages for Parallel Computation. **ACM Computing Surveys**, New York, v.30, n.2, p.123–169, june 1998.

VALIANT, L. G. A Bridging Model for Parallel Computation. **Communications of the ACM**, New york, v.33, n.8, p.103–111, august 1990.

WEST, D. **Introduction to Graph Theory**. [S.l.]: Birkhauser, 1996.

WILSON, R. **Introduction to Graph Theory**. [S.l.]: Addison Wesley, 1996.

WORSCH, T. On parallel Turing machines with multi-head control units. **Parallel Computing**, [S.l.], v.23, n.11, p.1683–1697, 1997.

WORSCH, T. Parallel Turing machines with one-head control units and cellular automata. **Theoretical Computer Science**, [S.l.], v.217, n.1, p.3–30, 1999.

YAMIN, A. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva**. 2004. 195p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, RS.

YAMIN, A. C.; AUGUSTIN, I.; BARBOSA, J.; SILVA, L.; REAL, R.; CAVALHEIRO, L.; GEYER, C. Towards Merging Context-aware, Mobile and Grid Computing. **Journal of High Performance Computing Applications**, London, v.17, n.2, p.191–203, 2003.

YAMIN, A. C.; AUGUSTIN, I.; BARBOSA, J.; SILVA, L.; REAL, R.; SCHAFFER, A.; GEYER, C. EXEHDA: adaptive middleware for building a pervasive grid environment. In: CZAP, H.; UNLAND, R.; BRANKI, C.; TIANFIELD, H. (Ed.). **Frontiers in Artificial Intelligence and Applications - Self-Organization and Autonomic Informatics**. [S.l.]: IOS Press, 2005. v.135, p.203–219.