UNIVERSIDADE FEDERAL DE PELOTAS
INSTITUTO DE FÍSICA E MATEMÁTICA
Curso de Ciência da Computação

Uma Contribuição à Comunicação e a Coordenação no EXEHDA

Por

MARCELO AUGUSTO CARDOZO JR

Projeto de Conclusão submetido como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Adenauer Corrêa Yamin, MSc. (UCPel)
Coorientadores: Prof. Jorge Luis Victória Barbosa, PhD. (UCPel)
Prof. Marcello da Rocha Macarthy, MSc. (UFPel)

Pelotas, Outubro de 2002

Este trabalho é dedicado a minha família e aos meus amigos, sem o apoio deles este trabalho jamais alcançaria a sua totalidade.

Quero registrar meus agradecimentos aos meus orientadores que também sempre me ajudaram, tanto nas horas ruins quanto nas boas.

A todos vocês meu muito obrigado.

"The most incomprehensible thing about the universe is that it is comprehensible"
Albert Einstein(1935).

Uma contribuição à comunicação e a coordenação no EXEHDA

Monografia defendida e aprovada, em 25 de outubro de 2002, pela banca examinadora constituída pelos professores:

_____

Prof. Adenauer Corrêa Yamim, MSc. (UCPel) - Orientador

_____

Prof. Marcello da Rocha Macarthy, MSc. - Co-orientador

_____

Prof. José Luís Almada Güntzel, Dr.

_____

Prof. Gil Carlos Rodrigues Medeiros, MSc.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

EXEHDA	Execution Environment for High Distributed Applications
EXEHDA-CC	Execution Environment for High Distributed Applications Communication and Coordination Module
ISAM	Infra-estrutura de Suporte a Mobilidade
RMI -	Remote Method Invocation
PRAM -	Parallel Random Access Memory

Resumo

Este texto apresenta uma proposta de coordenação e comunicação para o EXEHDA(*Execution Enviroment for High Distributed Applications*). A mesma se denomina EXEHDA-CC e foi baseada em estudos relacionados com programação distribuída, comunicações anônimas e assíncronas e espaço de tuplas.

No EXEHDA, é contemplado o tratamento da heterogeneidade, o suporte à mobilidade de hardware e software e o uso do escalonamento em estratégias adaptativas na execução das tarefas. O EXEHDA está integrado às pesquisas HOLOPARADIGMA e ISAMadapt. O HOLOPARADIGMA define o modelo e a linguagem de programação utilizados. O ISAMadapt, por sua vez, especifica as abstrações para expressar a adaptação na codificação de aplicações distribuídas em ambiente pervasivo. Juntamente com estes compõe os esforços do projeto ISAM(Infra-Estrutura de Suporte à Mobilidade).

Na primeira parte do texto, capítulos um, dois e três, o trabalho é contextualizado e também é apresentada uma discussão sobre os fundamentos em espaços de tuplas. Na segunda parte, capítulos quatro, cinco e seis é apresentada e discutida a solução proposta, destacando-se os aspectos da modelagem e da implementação do EXEHDA-CC(*Execution Environment for High Distributed Applications Communication and Coordination Module*).

O modelo proposto foi implementado em Java, linguagem utilizada no projeto como um todo. A título de observar o comportamento do EXEHDA-CC, foram realizados testes contemplando tanto operações locais, como distribuídas envolvendo vários equipamentos. Os resultados foram comparados com uma plataforma bastante difundida com funcionalidade próxima ao EXEHDA-CC.

1 INTRODUÇÃO

Os próximos anos serão caracterizados por elevados níveis de heterogeneidade e pela interação entre dispositivos conectados a redes de abrangência global. Estas redes interligadas utilizarão tanto conexões cabeadas como sem fio. As pesquisas envolvendo sistemas distribuídos em redes *wide-area* responderam a diversas questões pertinentes ao gerenciamento de recursos, contudo existem lacunas no que diz respeito ao tratamento da adaptação da execução de forma dinâmica.

A disseminação crescente da Internet, somada ao aumento da velocidade operacional das redes de computadores e das suas interconexões, levam uma perspectiva de uso unificado dos recursos distribuídos, o qual pode ser realizado a partir de qualquer equipamento das redes interconectadas (*Grid Computing*). De forma similar, a ampliação do conceito de rede-sem-fio conduz à proposta de computação móvel. Na Computação Móvel, o usuário portando dispositivos móveis como palmtops e notebooks, independentemente da sua localização física, terá acesso a uma infra-estrutura de serviços.

Integrando estas duas visões, observa-se um movimento em direção à *Pervasive Computing*, criando aplicações com novas funcionalidades. *Pervasive Computing* é a proposta de um novo paradigma computacional, que permite ao usuário o acesso ao seu ambiente computacional a partir de qualquer lugar, a qualquer tempo, usando vários tipos de dispositivos (móveis ou não).

Em contraste com a premissa dos sistemas distribuídos de fornecer transparência da distribuição para o usuário, esta nova classe de aplicações é *context-aware* e tenta tirar vantagem desta informação. Para alcançar esta consciência, a aplicação ou o ambiente de execução pró-ativamente monitoram e controlam as condições do contexto. A aplicação e/ou o sistema reagem às alterações no ambiente através do processo de adaptação. Este processo requer a existência de múltiplos caminhos de execução para uma aplicação, ou configurações alternativas, as quais exibem diferentes perfis de utilização (históricos) dos elementos computacionais.

Esta visão apresenta uma série de novos (e renovados) desafios, oriundos do dinamismo e heterogeneidade do ambiente, além dos novos requerimentos das aplicações no estilo *follow-me* da *Pervasive Computing*. O dinamismo está tanto na

aplicação quanto no sistema de execução. Ambos operam em um ambiente cujas condições na disponibilidade e no acesso aos recursos são variáveis no tempo e no espaço. Como conseqüência, novos tipos de aplicações estão aparecendo, as quais têm um comportamento determinado pela sua sensibilidade à variação nas condições de alguns elementos do ambiente. O ambiente é definido por elementos computacionais que podem ser medidos, como poder computacional, largura de banda, latência da rede, consumo de energia, localização do usuário, preferências do usuário, entre outros.

A complexidade do tratamento da adaptação introduz custos, tanto no gerenciamento, como na execução da aplicação propriamente dita. Estes custos podem se tornar elevados devido à necessidade de predições e estimativas de comportamentos futuros da aplicação e do próprio sistema. Por outro lado, os ganhos potenciais podem ser altos. Diversos trabalhos apontam na direção de que é somente através de uma gerência da adaptação no ambiente de execução, fornecendo às aplicações informações sobre trocas na sua infra-estrutura de suporte, que se pode operar eficientemente em um ambiente distribuído altamente dinâmico.

Sistemas distribuídos tradicionais são construídos com suposições sobre a infra-estrutura física de execução, como conectividade permanente e disponibilidade de recursos. Porém, essas suposições não são válidas na *Pervasive Computing*.

Na computação móvel, conforme o usuário se movimenta, a localização do seu dispositivo móvel se altera, e conseqüentemente, a configuração da rede de acesso e o centro da atividade computacional também se modificam. Por sua vez, no contexto de *Grid Computing* a disponibilidade dos equipamentos e conexões de rede dependem de aspectos multi-institucionais.

Neste novo cenário, o comportamento adaptativo das aplicações levanta um conjunto de questões: "quais são os domínios de aplicações adequados?"; "como os programas devem ser estruturados?"; "como o sistema básico de suporte deve trabalhar?".

Para tratar destas questões, o projeto EXEHDA em andamento no II/UFRGS propõe um *middleware* voltado para o gerenciamento de recursos em redes heterogêneas, com suporte a mobilidade de software e hardware, adaptação dinâmica e com as aplicações modeladas utilizando componentes. A estratégia consiste de um ambiente integrado que oferece um paradigma de programação direcionado aos

objetivos e seu respectivo ambiente de execução e que também gerencia o processo de adaptação através de um modelo colaborativo multinível, no qual tanto o ambiente de execução, como a aplicação, participam das decisões adaptativas.

Os dois focos que norteiam os trabalhos são:

- como construir interfaces de programação que isolem o usuário da complexidade dos contextos de execução modernos, simplificando ao máximo o esforço de programação;
- como deve ser um *middleware* para potencializar o desempenho do processamento de uma aplicação modelada com estas interfaces de programação, considerando a elevada dinamicidade dos contextos de execução atuais.

O ponto de equilíbrio entre estes dois focos é uma questão central da pesquisa em andamento no projeto EXEHDA.

1.1 Contexto local do trabalho

O Holoparadigma foi desenvolvido com as seguintes motivações: exploração do paralelismo implícito, atuar sobre arquiteturas distribuídas, trabalhar com múltiplos paradigmas e explorar a execução destes de forma paralela e distribuída.

Depois de definidos seus princípios, modelo e linguagem, uma das atuais frentes de pesquisa é o seu ambiente de execução, o EXEHDA (*Execution Environment for High Distributed Applications*).

Este trabalho se enquadra neste contexto e tem por objetivo central cooperar na definição do ambiente de execução para o uso do Holoparadigma no desenvolvimento de aplicações móveis distribuídas.

O foco do trabalho é contribuir na definição dos mecanismos para comunicação e coordenação entre os mecanismos do EXEHDA empregados para dar suporte a execução distribuída do Holoparadigma.

1.2 Principais contribuições do autor

Além de participar das reuniões e das discussões sobre a modelagem do EXEHDA como um todo, destaca-se a atuação do autor nos seguintes aspectos do trabalho:

- contribuição na especificação da gerência para execução distribuída do Holoparadigma;
- participação na concepção dos algoritmos e das estruturas de dados necessárias ao EXEHDA-CC;
- participação na definição das primitivas do EXEHDA-CC necessárias ao EXEHDA;
- definição da política de distribuição dos espaços de objetos manipulados pelo EXEHDA-CC;
- implementação do EXEHDA-CC como uma extensão para o Java;
- participação na integração do EXEHDA-CC com os outros componentes do ambiente de execução;
- participação na definição dos procedimentos de compilação do Holoparadigma (HoloJava) no tocante ao EXEHDA-CC;
- efetuação de testes de desempenho do EXEHDA-CC;
- otimização de código do EXEHDA-CC;
- implementação distribuída do EXEHDA-CC.

1.3 Organização do texto

O presente trabalho foi organizado em sete capítulos contemplando os principais aspectos do trabalho redigido.

O primeiro, e corrente, capítulo é a introdução, onde foram apresentados a motivação, o contexto global e o contexto local do trabalho, assim como as contribuições do autor para o projeto de pesquisa EXEHDA.

O segundo capítulo apresenta o contexto no qual o EXEHDA-CC se insere, descrevendo os aspectos mais importantes do Holoparadigma e o ambiente de execução EXEHDA.

O terceiro capítulo oferece ao leitor fundamentos sobre espaços de objetos, que é a abordagem do EXEHDA-CC para a solução dos problemas de comunicação e coordenação.

O quarto capítulo descreve a modelagem, as dificuldades e soluções do EXEHDA-CC para o ambiente de execução.

O quinto capítulo mostra o modelo proposto e sua implementação.

O sexto capítulo contém as considerações finais deste trabalho.

O sétimo capítulo apresenta a bibliografia utilizada.

2 O CONTEXTO DO TRABALHO

O EXEHDA-CC integra a frente de pesquisa EXEHDA, a qual por sua vez juntamente com o desenvolvimento do Holoparadigma e do ISAMadapt compõem o projeto ISAM(Infra-estrutura de Suporte a Mobilidade). Este capítulo tem por objetivo discorrer sobre as principais motivações do projeto ISAM, bem como sobre as principais características das frentes de pesquisa que o compõem. Uma visão geral do ISAM pode ser encontrada em diversos trabalhos[AUG 2001] [YAM 2002a].

2.1 Arquitetura de software utilizada

A arquitetura de software proposta é organizada em camadas com níveis diferenciados de abstração e está direcionada para a busca da manutenibilidade da qualidade de serviços do ambiente de execução, através do conceito de adaptação. Uma visão da arquitetura do ISAM é apresentada na figura 2.1. Salientam-se dois pontos: (1) a adaptação que permeia todo o sistema, por isto está colocada em destaque; (2) o escalonador, que é o "núcleo adaptativo” da arquitetura.

A camada superior (SUP) da arquitetura é composta pela aplicação móvel distribuída. A construção desta aplicação baseia-se nas abstrações do Holoparadigma, as quais permitem expressar mobilidade, acrescidas de novas abstrações para expressar adaptabilidade proposta pelo ISAMadapt.

A camada intermediária aloja as principais funcionalidades providas pelo EXEHDA, a quais são comentadas nos itens que seguem.

Por sua vez, a camada inferior (INF) é composta pelas tecnologias empregadas nos sistemas distribuídos existentes, tais como sistemas operacionais nativos e a Máquina Virtual Java.

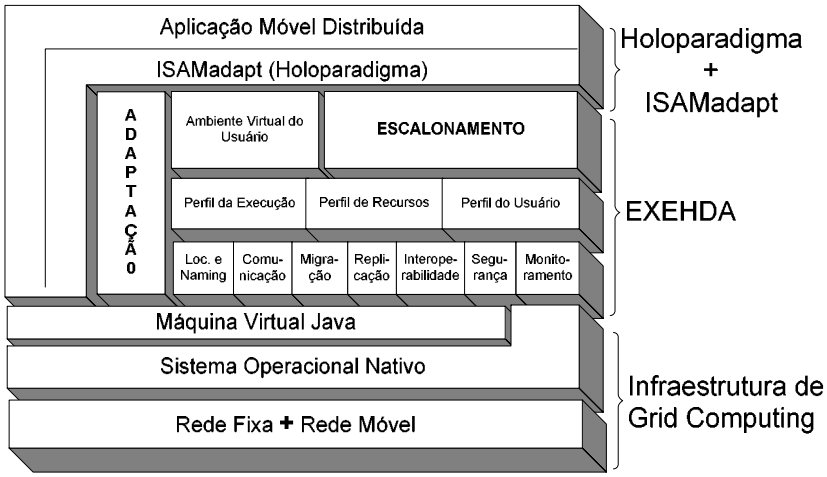

Figura 2.1 – Arquitetura ISAM

2.1.1 Camada intermediária – primeiro nível

A camada intermediária (INTERM) é o núcleo funcional da arquitetura ISAM, sendo formada por três níveis de abstração. O primeiro nível é composto por dois módulos de serviço à aplicação: Escalonamento e Ambiente Virtual do Usuário. O escalonamento, por sua vez, é o componente-chave da adaptação na arquitetura ISAM.

O Ambiente Virtual do Usuário (AVU) compõe-se dos elementos que integram a interface de interação do usuário móvel com o sistema. Este módulo é o responsável por implementar o suporte para que a aplicação que o usuário está executando em uma localização possa ser instanciada e continuada em outra localização sem descontinuidade, permitindo o estilo de aplicações *follow-me*. O modelo foi projetado para suportar a exploração de aplicações contextualizadas (adaptadas aos recursos, serviços e localização corrente) e individualizadas (adaptadas aos interesses e preferências do usuário móvel). O desafio da adaptabilidade é suportar os usuários em diferentes localizações, com diferentes sistemas de interação que demandam diferentes sistemas de apresentação, dentro dos limites da mobilidade. Este módulo deve caracterizar, selecionar e apresentar as informações de acordo com as necessidades e o contexto em que o usuário se encontra. Para realizar estas tarefas, o sistema se baseia num modelo de uso onde as informações sobre o ambiente de trabalho, preferências, padrões de uso, padrões de movimento físico e hardware do usuário são dinamicamente monitoradas e integram o Perfil do Usuário e da Aplicação.

2.1.2 Camada intermediária – segundo nível

Como já caracterizado anteriormente, na proposta ISAM busca-se um conceito flexível de adaptação que está relacionado ao contexto em que a aplicação está inserida. Por sua vez, a mobilidade física introduz a possibilidade de movimentação do usuário durante a execução de uma aplicação. Desta forma, os recursos disponíveis podem se alterar, tanto em função da área de cobertura e heterogeneidade das redes, quanto em função da disponibilidade dos recursos devido à alta dinamicidade do sistema. Assim, a localização corrente do usuário determina o contexto de execução, definido como "toda informação, relevante para a aplicação, que pode ser obtida e usada para definir seu comportamento". Numa análise preliminar, o contexto é determinado através de informações de quem, onde, quando, o que está sendo realizado e com o que está sendo realizado. Obter essas informações é a tarefa do módulo de monitoramento, que atua tanto na parte móvel quanto na parte fixa da rede.

As informações que dirigem as decisões do escalonador e dão suporte à aplicação para sua decisão de adaptação são advindas de quatro fontes: perfil da execução, perfil dos recursos, perfil do usuário e da aplicação (ISAMadapt). O módulo monitoramento do ISAM obtém informações do acompanhamento das aplicações executadas pelo usuário, em um dado tempo e em um dado local, com determinados parâmetros, o que permite determinar a evolução histórica e quantitativa das entidades monitoradas. A interpretação destas informações estabelece o perfil do usuário e das aplicações. Desta forma, as aplicações móveis ISAM poderão se adaptar à dimensão pessoal, além das dimensões temporal e espacial presentes nos demais sistemas móveis.

2.1.3 Camada intermediária – terceiro nível

No terceiro nível da camada intermediária estão os serviços básicos do ambiente de execução ISAM que provêm a funcionalidade necessária para o segundo nível e cobrem vários aspectos, tais como migração – mecanismos para deslocar um ente de uma localização física para outra; replicação otimista – mecanismo para aumentar a disponibilidade e o desempenho do acesso aos dados; localização e *naming* – para dar suporte ao movimento dos dispositivos móveis entre diferentes equipamentos mantendo a execução durante o deslocamento.

2.2 Holoparadigma

Esta seção tem por objetivo resumir os principais aspectos do Holoparadigma (de forma abreviada, Holo). Holo é um modelo multiparadigma que possui uma semântica simples e distribuída. Através dessa semântica, o modelo estimula a exploração automática da distribuição (distribuição implícita).

2.2.1 Tipos de entes

O Holoparadigma estabelece duas **classificações básicas** para organização dos entes: organizacional e funcional. A **classificação organizacional** distingue os entes, de acordo com a sua estrutura, em dois tipos:

- **Ente elementar:** ente sem níveis de composição;
- **Ente composto:** ente formado pela composição de outros entes. Não existe limite para níveis de composição.

Um **ente elementar** (figura 2.2a) é organizado em três partes: interface, comportamento e história. A **interface** descreve suas possíveis relações com os demais entes. O **comportamento** contém **ações** que implementam a funcionalidade de um ente. Holo não estabelece os tipos de ações a serem utilizadas, no entanto, estabelece que existem dois tipos básicos de comportamento:

- **Imperativo**: o comportamento imperativo é composto de ações imperativas que descrevem os caminhos para solução de um problema (enfoque no controle, ou seja, como deve ser realizada a ação). Uma ação imperativa possui uma natureza determinista. O paradigma imperativo é uma alternativa para descrição do comportamento imperativo;
- **Lógico**: o comportamento lógico é composto de ações lógicas que expressam um problema de forma declarativa (enfoque na lógica, ou seja, o que deve ser realizado). Uma ação lógica possui uma natureza não-determinista. O paradigma em lógica é uma alternativa para descrição do comportamento lógico.

A **história** é um espaço de armazenamento compartilhado no interior de um ente. O símbolo é o átomo de informação no Holoparadigma. Holo propõe a utilização do processamento simbólico como principal instrumento para o tratamento de

informações. Esta característica é herdada do paradigma em lógica. Neste sentido, a variável lógica e a unificação são consideradas a base do tratamento de símbolos. Holo estabelece que a história deve ser direcionada para armazenamento e gerenciamento de símbolos. Portanto, o paradigma em lógica torna-se uma alternativa adequada para sua implementação.

Um **ente composto** (figura 2.2b) possui a mesma organização de um ente elementar, no entanto, suporta a existência de outros entes na sua composição (**entes componentes**). Cada ente possui uma história. A história fica encapsulada no ente e, no caso dos entes compostos, é compartilhada pelos entes componentes. Os entes componentes participam do desenvolvimento da história compartilhada e sofrem os reflexos das mudanças históricas. Sendo assim, podem existir vários níveis de encapsulamento da história. Os entes acessam somente a história em um nível. A composição varia de acordo com a mobilidade dos entes em tempo de execução. A figura 2.2c mostra dois níveis de história encapsulada em um ente composto organizado em três níveis. Os comportamentos e interfaces foram omitidos para simplificação da figura.

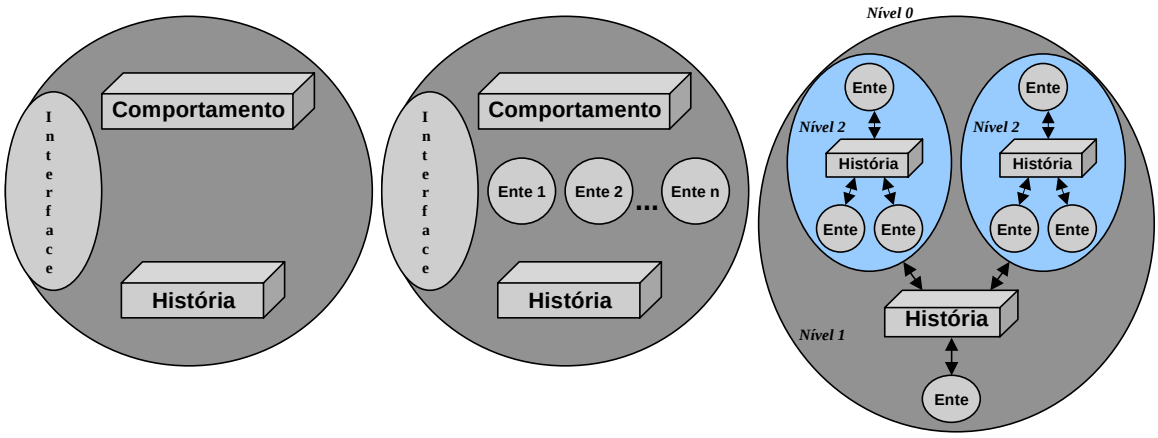

Figura 2.2 – Tipos de Entes

Um ente elementar assemelha-se a um **objeto** do paradigma orientado a objetos. Do ponto de vista estrutural, a principal diferença consiste na história, a qual atua como uma forma alternativa de comunicação e sincronização. Um ente composto assemelha-se a um **grupo**. Neste caso, a história atua como um espaço compartilhado vinculado ao grupo. Lea [LEA 2001] salienta que os conceitos da orientação a objetos, tais como

objetos e classes, tornam-se limitados quando o tratamento de composição envolve aspectos dinâmicos (mudam durante a execução). Lea também propõe a utilização de grupos para solução desta limitação. Por sua vez, Nierstrasz destaca que as linguagens orientadas a objeto enfocam aspectos de programação em detrimento de aspectos de composição. Vários estudos relacionados com arquitetura de software seguem a mesma argumentação. Os entes compostos aliados à mobilidade permitem a composição dinâmica no Holoparadigma. Além disso, a utilização de uma mesma unidade (ente) para manipulação de elementos e grupos, simplifica o modelo e sintetiza conceitos já existentes na ciência da computação.

A **classificação funcional** distingue os entes de acordo com suas funções:

- **Ente estático**: definição estática de um ente. Esta definição estabelece um padrão estático que pode ser utilizado para criação de outros entes através da clonagem. Um ente estático é criado no nível de modelagem e programação. Modelos e programas são compostos de descrições de entes (entes estáticos), as quais estabelecem interfaces, comportamentos e histórias;
- **Ente dinâmico**: ente em execução. Um programa em execução é composto de entes dinâmicos. Estes entes executam ações e interagem de acordo com seus comportamentos e histórias.

A única distinção entre entes estáticos e dinâmicos consiste na sua função. Os entes estáticos são utilizados como matrizes estáticas para criação de outros entes. Além disso, estabelecem um estado inicial para execução de programas. Por sua vez, os dinâmicos representam o estado corrente de uma execução.

2.2.2 Distribuição e mobilidade

O Holoparadigma busca a distribuição implícita através da Holosemântica. Uma discussão neste sentido está apresentada em. Neste escopo, um ente assume dois **estados de distribuição**:

- **Centralizado:** um ente está centralizado quando se localiza em apenas um nodo de um sistema distribuído. Entes elementares estão sempre centralizados. Um ente composto está centralizado se todos os seus entes componentes estão localizados no mesmo nodo;

- **Distribuído:** um ente está distribuído quando se localiza em mais de um nodo de um sistema distribuído. Entes elementares não podem estar distribuídos. Um ente composto está distribuído se um ou mais entes componentes estão distribuídos.

A figura 2.3 exemplifica uma possível distribuição para o ente apresentado na figura 2.2c. O ente encontra-se distribuído em dois nodos da arquitetura distribuída. A história de um ente distribuído é denominada **história distribuída**. A distribuição da história pode ser baseada em técnicas de **memória compartilhada distribuída** ou **espaços distribuídos**.

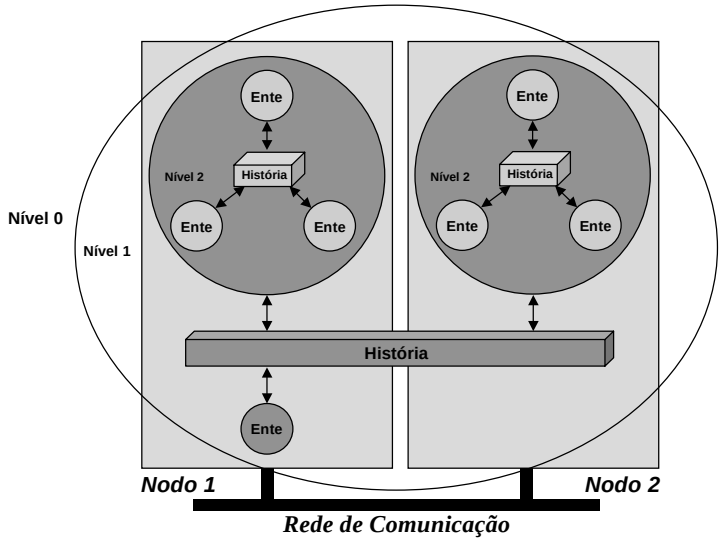

Figura 2.3 – Ente Distribuído

A mobilidade é a capacidade que permite o deslocamento de um ente. No Holoparadigma existem dois tipos de mobilidade:

- **Mobilidade lógica:** a mobilidade lógica relaciona-se com o deslocamento em nível de modelagem, ou seja, sem considerações sobre a plataforma de execução. Neste contexto, um ente se move quando cruza uma ou mais fronteiras de entes;
- **Mobilidade física:** a mobilidade física relaciona-se com o deslocamento entre nós de uma arquitetura distribuída. Neste contexto, um ente se move quando se desloca de um nó para outro.

A figura 2.4 exemplifica uma possível mobilidade lógica no ente apresentado na figura 2.2c. Conforme exemplificado, após o deslocamento, o **ente móvel** não possui mais acesso à história no **ente origem**. No entanto, passa a ter acesso à história no **ente destino**. Neste caso, somente ocorrerá mobilidade física se os entes origem e destino estiverem alocados em diferentes nós de uma arquitetura distribuída.

As mobilidades lógica e física são independentes. A ocorrência de um tipo de deslocamento não implica a ocorrência do outro. Merece atenção o caso da mobilidade física não implicar obrigatoriamente a mobilidade lógica. Considere-se o ente distribuído mostrado na figura 2.3. Se o ente elementar localizado no nível um, realizar um deslocamento físico conforme mostrado na figura 2.4b, não haverá mobilidade lógica apesar de ter ocorrido mobilidade física. Neste caso, o ente movido continua com a mesma visão da história.

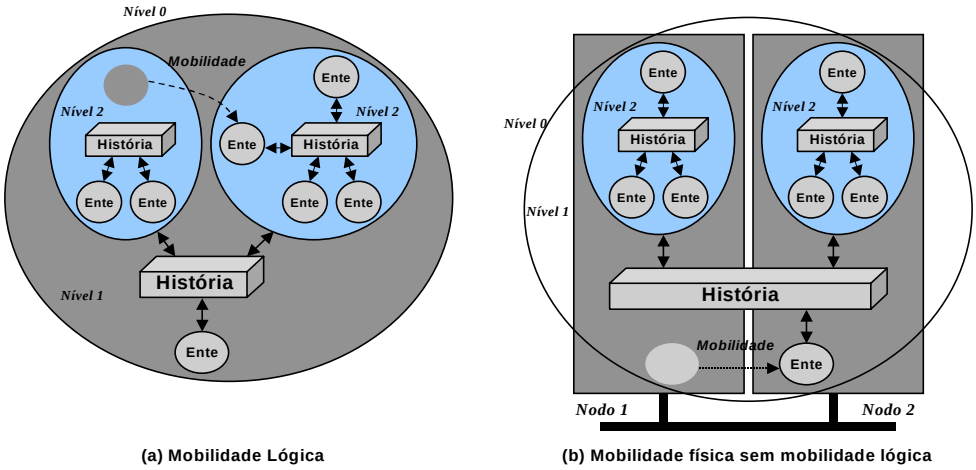

Figura 2.4 – Mobilidade no Holoparadigma

2.3 O EXEHDA

Os próximos anos serão caracterizados por elevados níveis de mobilidade, heterogeneidade e interação entre dispositivos conectados a redes de abrangência global. Estas redes interligadas utilizarão tanto conexões cabeadas como sem fio. As primeiras pesquisas envolvendo sistemas distribuídos em redes *wide-area* responderam a diversas questões pertinentes ao gerenciamento de recursos neste contexto, contudo existem lacunas no que diz respeito ao tratamento da heterogeneidade e da adaptação dinâmica. Trabalhos mais recentes empregando tecnologias como CORBA e Java/Jini

abordam a questão da heterogeneidade, porém não se aprofundam em aspectos pertinentes à adaptabilidade.

A integração do Holoparadigma com o EXEHDA constitui uma abordagem integrada de software e ambiente de execução direcionada às redes heterogêneas, com suporte às mobilidades lógica (software) e física (hardware), voltada à execução de aplicações distribuídas em escala global e baseada em componentes de software. Uma visão de composição da proposta linguagem de programação e seu ambiente de execução pode ser visto na figura 2.5 (uma visão sintética da figura 2.1).

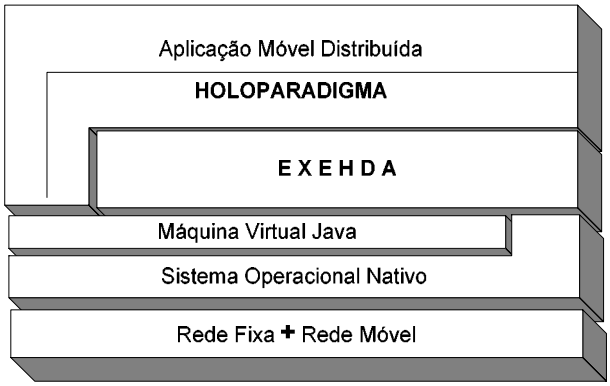

Figura 2.5 – Contexto de Execução

Uma visão global do EXEHDA pode ser vista na figura 2.6. Nesta figura aparecem: a geração do código adaptativo (ISAMadapt), o compilador do Holoparadigma o qual gera código Java extendido e os dois níveis do EXEHDA. Um nível voltado para o nível da aplicação, e outro direcionado para a administração do nível físico.

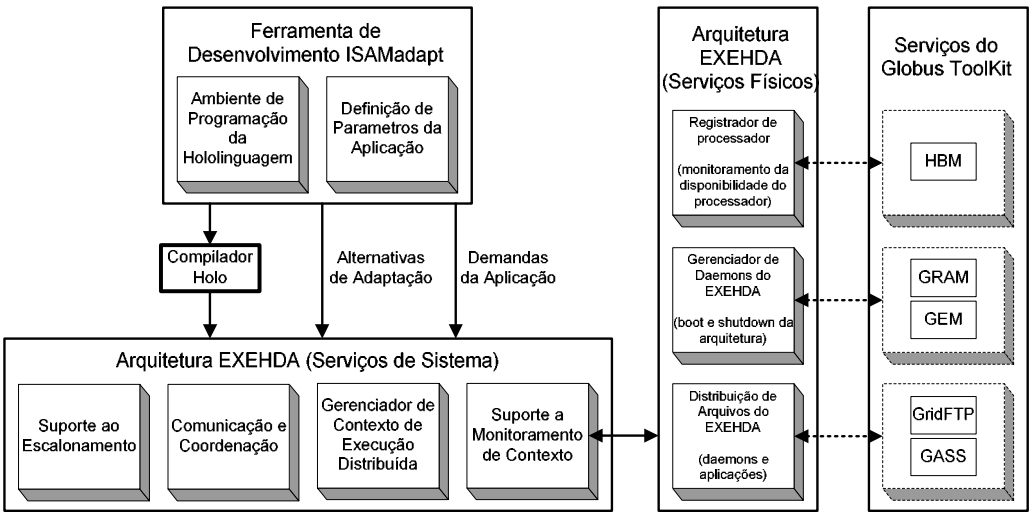

Figura 2.6 – Visão Geral do EXEHDA

A gerência da infra-estrutura física do ambiente de execução ISAM integra com Globus Toolkit sob três aspectos:

- **monitoramento de *hosts* disponíveis**: este módulo do ambiente de execução interage com o serviço Globus HBM para detecção da presença de *hosts*. A informação dos processadores disponíveis é repassada para o serviço de escalonamento. O emprego efetivo do host pela aplicação, será determinado pelo serviço de escalonamento em função de seus índices de ocupação. As métricas e as heurísticas de escalonamento são providas pelo *middleware* sem o emprego de funcionalidades do Globus.
- disparo remoto de serviços do ambiente de execução: este módulo interage com os serviços Globus GRAM e GEM para disparar o suporte para o *middleware* ISAM nos *hosts* da arquitetura distribuída. A informação de disponibilidade de *hosts* é obtida através do primeiro item.
- **instalação dos arquivos necessários ao ambiente de execução:** este módulo interage com os serviços Globus GridFTP e GASS com o objetivo de disseminar nas células (vide figura 2.7) que compõem o ambiente de execução os arquivos executáveis, classes Java e arquivos de configuração necessários ao ambiente de execução. O tamanho da

célula é configurável e os *hosts* EXEHDAbase concentram os serviços de gerência da mesma.

O contexto de *Grid Computing* é mapeado em células de execução como pode ser visto na figura 2.7. Considera-se que os *hosts* móveis devam usufruir da infra-estrutura de rede fixa existente, beneficiando-se de ambientes como o oferecido pela Internet. Este modelo é refletido nos elementos básicos do ambiente de execução do sistema ISAM:

- EXEHDAcél - denota a área de atuação de uma EXEHDAbase, e é composta pela mesma e por HoloSítios;
- EXEHDAbase - é o ponto de contato do *host* móvel com os serviços ISAM residentes na parte fixa da rede. Possui as funções de identificação, autenticação e de ativação das ações básicas do sistema;
- EXEHDAnó - são os nodos do sistema responsáveis pela execução da aplicação móvel distribuída propriamente dita. Nestes também processam serviços de gerenciamento ISAM;
- EXEHDAmob-nó - são os nodos móveis do sistema. Análogos aos EXEHDAnós, atuam na execução das aplicações e em algumas funções de monitoramento de recursos, conforme seu poder computacional. Os de menor porte tratam somente da interação com o usuário;
- EXEHDAhome - é um ponto de referência único por usuário móvel no âmbito de toda rede. Está associado a um EXEHDAnó registrado para tal na arquitetura.

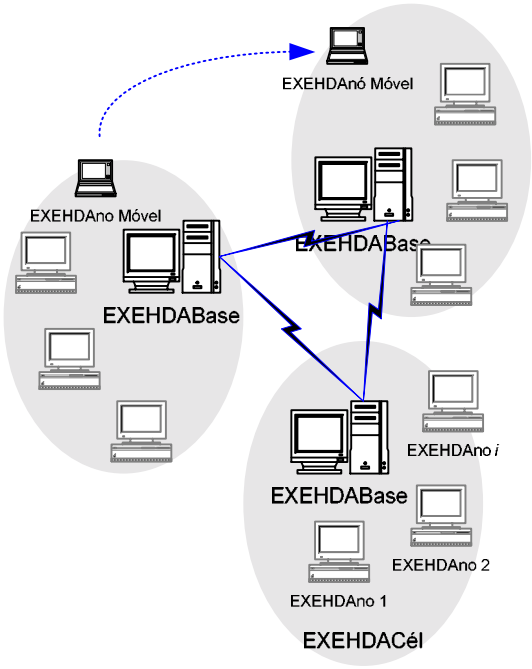

Figura 2.7 – Elementos Básicos do Ambiente de Execução

2.3.1 Características do EXEHDA

Esta seção objetiva destacar os aspectos que caracterizam o EXEHDA. Uma discussão destas características pode ser encontrada em [YAM 2002b]:

- sua operação ocorre sobre o sistema operacional, e sem exigir alteração do mesmo. Isto potencializa a portabilidade;
- pode suportar tanto execuções paralelas como distribuídas. Para tal, interfaces de programação para comunicação interprocessos, tanto síncronas quanto assíncronas, são disponibilizadas;
- não está comprometido com uma heurística de escalonamento em particular. Ao contrário, disponibiliza facilidades para que novas heurísticas sejam implementadas. Esta política também se aplica no que diz respeito aos procedimentos de sensoriamento;
- a heurística de escalonamento/sensoriamento a ser utilizada é selecionada e/ou contextualizada por usuário e aplicação;
- os componentes que tomam decisão são replicados, e são capazes de atividades autônomas e assíncronas. Este aspecto é indispensável para uma operação com elevada escalabilidade. Os servidores que irão

compor a gerência distribuída da HoloTree se enquadram nesta situação;

- as metas de escalonamento são perseguidas em escopos. Cada componente que toma decisão escalona serviços no seu domínio;
- uso intensivo de registro histórico como auxiliar na tomada de decisão.

Algumas dessas características são típicas de propostas de ambiente de execução com escalonamento difuso e voltadas para aplicações com elevada dinamicidade de execução.

No que diz respeito às estratégias para maximização do desempenho da execução de aplicações Holoparadigma, o escalonamento no EXEHDA utiliza os seguintes princípios:

- balanceamento de carga nos nodos responsáveis pelo processamento;
- localização dos recursos (software e hardware) mais próximos (reduzir custo de comunicação);
- emprego de replicação de serviços e de dados;
- disponibilização antecipada, por usuário, da demanda de componentes das aplicações e dos dados;
- otimização no volume de comunicações, utilizando transferências de contextos e componentes de aplicação personalizadas por usuário;
- monitoração da comunicação praticada pelos componentes das aplicações em execução, com intuito de otimizar aspectos de mapeamento;
- uso de uma estratégia colaborativa entre o ambiente de execução e a aplicação na tomada de decisões de escalonamento (adaptação).

O emprego destes procedimentos fica potencializado pela possível alternância do ponto de conexão dos EXEHDAnós móveis no contexto da rede, comportamento este inerente à computação móvel.

2.3.2 Decisões de adaptação do EXEHDA

A adaptação é um processo disparado em resposta a uma situação ou evento externo, resultando na troca de um recurso ou por outro ou pela alteração na qualidade do serviço prestado. A adaptação faz o sistema se acomodar às alterações nos recursos disponíveis para ser capaz de continuar trabalhando com a melhor qualidade possível,

em cada momento. Pesquisas recentes estão começando a tratar desse problema, porém o fazem com a um tipo de aplicação em específico, tais como processamento de imagens ou multimídia. A inovação desta proposta integrada: Holoparadigma, EXEHDA e ISAMadapt está em projetar uma arquitetura com um tratamento uniforme da adaptação e não comprometida com um domínio específico de aplicação. Isto é um fator complicador na proposta, porém justifica-se por se considerar que o potencial de aplicabilidade da computação móvel é amplo. Novos tipos de aplicações estão surgindo derivados do comportamento do usuário móvel (ainda não totalmente entendido/conhecido). Com isto, projetar aplicações móveis é uma tarefa difícil e esta deve ter uma ampla colaboração do sistema. Argumenta-se que para simplificar a tarefa de projetar aplicações móveis distribuídas, o programador deve ter à disposição uma linguagem para expressar aplicações de propósito geral, com abstrações para expressar a mobilidade e a adaptabilidade, e um ambiente de execução que lhe forneça mecanismos para especializar o comportamento móvel adaptativo ao domínio específico da aplicação.

2.3.3 Dimensões e níveis de adaptação

Pelo exposto, a adaptação ocorre como reação às variações no contexto de execução da aplicação. Desta forma, vê-se que as aplicações móveis podem se adaptar ao longo de três dimensões: **temporal** (quando ela ocorre), **espacial** (escolha dos recursos e serviços) e **pessoal** (preferências e comportamento individual do usuário). Por sua vez, em cada dimensão, pode-se pensar na adaptação em três níveis de abrangência: **recursos** - relativo às reações às mudanças em recursos físicos, como banda e latência da rede, alterando à qualidade da informação; **conteúdo** - relativo às alterações na semântica da aplicação, desde que esta pode alterar não somente o formato da informação, mas também seu conteúdo, como as *location-aware applications*; **situação** - relativo às mudanças do comportamento do usuário final em face de um novo contexto. Por exemplo, a funcionalidade da aplicação em execução se altera quando o usuário está em determinado local: escritório, casa ou carro. Nos sistemas móveis atuais somente o nível básico, de recursos, é normalmente presente nas arquiteturas de software analisadas.

2.3.4 Transparência x consciência da mobilidade

Uma grande diferença entre os sistemas distribuídos tradicionais e os novos sistemas para suporte a computação móvel, é que os primeiros procuram fornecer transparência da distribuição ao usuário, enquanto que nos segundos um certo grau de consciência da mobilidade (localização, contexto) é importante para o comportamento adaptativo. Assim, é necessário estabelecer um equilíbrio entre consciência e transparência (em geral, conflitantes) da mobilidade: o sistema deve fornecer informações sobre o ambiente requeridas pela aplicação para que esta possa reagir (adaptar-se) conforme suas necessidades. Os *middleware*s dos ambientes de execução correntemente disponíveis não fornecem facilidades para controlar o grau de transparência requerido pela aplicação móvel. Também neste aspecto, a arquitetura do EXEHDA se diferencia das demais.

2.3.5 A Organização do escalonamento no EXEHDA

No ISAM, as aplicações solicitam direta ou indiretamente recursos do escalonador. Algumas podem especificar uma determinada necessidade de qualidade de serviço (QoS), outras podem aceitar o “melhor-possível” nos níveis de serviço. Assim, o módulo de escalonamento do *middleware*, tem a estratégia de trabalhar com diferentes políticas de gerenciamento para diferentes aplicações, usuários e/ou domínios de execução. Neste caso, as estratégias de adaptação exigem do mecanismo de escalonamento o tratamento de problemas de otimização utilizando critérios múltiplos.

2.3.6 A Adaptação multinível colaborativa

A proposta ISAM contempla um comportamento adaptativo em dois segmentos: (1) na aplicação, a qual define o comportamento da adaptação (alternativas) e o contexto de seu interesse; (*2*) no ambiente de execução (EXEHDA), o qual tem um comportamento inerentemente adaptativo no momento em que processa a aplicação. Uma visão estrutural da Adaptação Multinível Colaborativa proposta pode ser vista na Figura 2.8.

Na codificação da aplicação, o programador especifica os elementos computacionais que afetam o comportamento da aplicação, os respectivos níveis de variação suportados e codifica comportamentos alternativos para atender à variação nas

condições ambientais (variação nos elementos computacionais no contexto da aplicação).

Estas informações influenciam o comportamento adaptativo a ser adotado pelo ambiente de execução. Por sua vez, o ambiente de execução fornece meios (*a*) para que sejam monitorados elementos computacionais do ambiente, (*b*) para que a aplicação possa registrar seu interesse em determinados elementos, (*c*) para notificar à aplicação das alterações ocorridas, e (*d*) para selecionar o comportamento alternativo mais adequado ao ajuste das novas condições ambientais. Como o sistema gerencia as aplicações, este também pode ter um comportamento pró-ativo, e executar adaptações relativas à administração e desempenho do sistema de forma global. A proposta de adaptação multinível colaborativa está detalhada em [YAM 2002c].

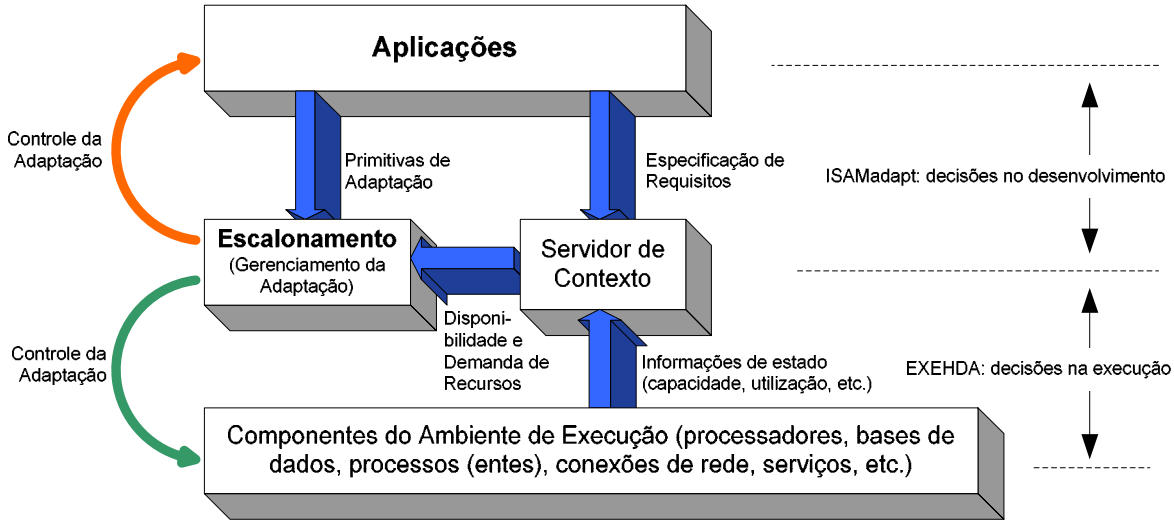

Figura 2.8 – A Adaptação Multinível Colaborativa

Neste sentido, um problema de pesquisa em andamento, é estabelecer o nível de cooperação requerido entre projetistas de sistemas e projetistas de aplicações para criar protocolos aceitáveis para ambos os grupos.

Por outro lado, o ISAM gerencia diversas aplicações que concorrem na utilização dos recursos. A otimização da execução de um subconjunto do total de aplicações não pode comprometer o nível mínimo de QoS necessário para o restante. Desta forma, o escalonador precisa trabalhar com uma visão global das execuções em andamento.

2.3.7 Organização física do escalonamento

A forma como é organizada a distribuição dos equipamentos afeta diretamente todos os serviços de um *middleware*, e naturalmente o seu escalonamento. A organização adotada no EXEHDA é a **celular hierárquica** (veja a figura 2.7). Nesta, os equipamentos pertencentes a uma mesma célula comunicam-se diretamente (utilizando uma organização plana). Nas comunicações com o exterior um equipamento específico atua como fronteira. A característica hierárquica faculta que uma célula possa recursivamente conter outras. Esta proposta de escalonamento como um todo foi apresentada em [YAM 2001].

Esta organização atende a necessidade de confinamento de contexto inerente ao modelo de programação adotado para o EXEHDA, o Holoparadigma. Este modelo é baseado em eventos associados a escopos. Outrossim, a organização celular hierárquica é orgânica com o Holoparadigma, o qual trabalha com o conceito de entes (entidade de modelagem). O conceito de entes também contempla a possibilidade de agrupamento hierárquico. Tal associação se mostra oportuna ao mapeamento e/ou à alocação dinâmica de tarefas.

Em função das características do software e do hardware, o escalonamento no *middleware* ISAM utiliza uma organização fisicamente distribuída e cooperativa representada na figura 2.7. Como principais premissas passíveis de serem atingidas por esta organização tem-se: tolerância a falhas, escalabilidade, autonomia dos domínios administrativos (EXEHDAcel) e suporte a múltiplas políticas de escalonamento.

A proposta está baseada em dois escalonadores (figura 2.9): (1) **EscCel** e (*2*) **EscEnte**:

EscCel: fica localizado no nó EXEHDAbase com atuação entre as EXEHDAcels. O mesmo tem atribuições no gerenciamento global da arquitetura, tais como:

- localizar recursos (hardware e software) mais próximos, para reduzir custos de comunicação;
- decidir quando e onde replicar serviços e/ou componentes de software (entes);
- decidir quando e para onde migrar os componentes de software;
- instanciar o Ambiente Virtual do Usuário nos EXEHDAnós. Esta

instanciação é feita sob duas óticas: (1) balanceamento de carga - neste caso é escolhido o nó menos carregado, (2) aspectos de afinidade da aplicação - exigência de memória, bases de dados, etc.;

- disponibilizar antecipadamente, por usuário, a demanda de componentes das aplicações e dos dados;
- repassar ao escalonador EscEnte a carga de trabalho (componentes de software) proveniente de outras EXEHDAbase.

Pelas suas atribuições, além da consideração de custos de comunicação e balanceamento de carga, o escalonador EscCel atua de forma intensiva sobre aspectos de replicação e migração.

EscEnte: também existente em todas as EXEHDAbases, tem atribuições no gerenciamento interno da EXEHDAcel, tais como:

- efetuar o mapeamento dos componentes da aplicação nos EXEHDAnós da EXEHDAcel. Os critérios utilizados também são balanceamento de carga e de afinidade funcionais;
- dar suporte aos procedimentos de adaptação colaborativa multinível com a aplicação.

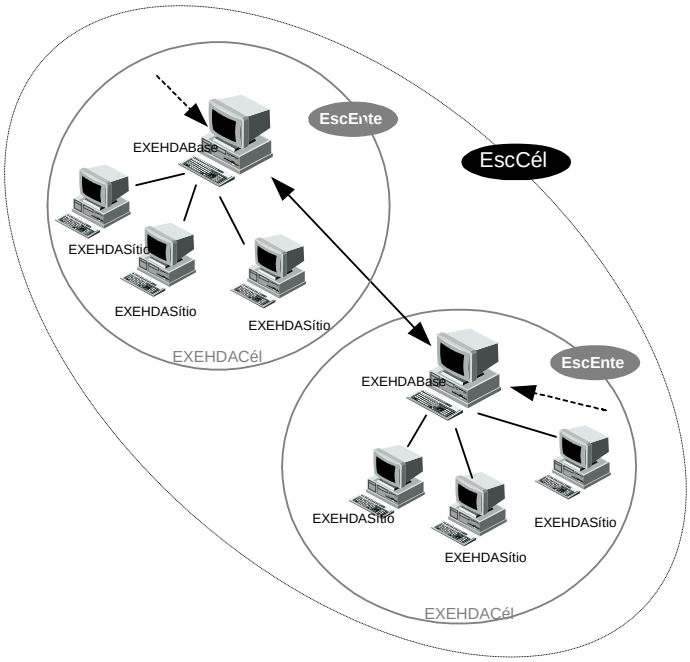

Figura 2.9 – Organização do Escalonamento

Uma estratégia do escalonador EscEnte é associar o contexto (a EXEHDAcel, as aplicações e os usuários) a grupos de escalonamento, onde cada grupo pode definir políticas especificas de balanceamento de carga. Cabe ao mecanismo de escalonamento gerir a evolução da execução das aplicações dos diferentes grupos segundo as políticas por eles selecionadas.

3 ESPAÇOS DE TUPLAS

Esse capítulo é dedicado a introduzir os conceitos sobre espaços de tuplas, suas peculiaridades e seus mecanismos; logo em seguida mostrará uma implementação do espaço de tuplas a qual foi a base para o Exehda-CC.

3.1 Espaço de tuplas: considerações gerais

A inexistência de recursos globalmente compartilhados faz com que as arquiteturas distribuídas sejam intrinsecamente escaláveis. Entretanto, a necessidade de um espaço de interação é difícil de ser prescindido for várias razões, entre elas:

- modelos de computação globais são intrinsecamente mais simples que os modelos locais;
- aplicações locais também necessitam de recursos compartilhados, por exemplo, para criar uma visão única de nomes.

Os problemas mencionados acima podem ser resolvidos através de uma memória distribuída compartilhada [NIT 1991].

O problema de um espaço de interação abstrato é, novamente, a não escabilidade: uma memória distribuída compartilhada, em particular, pode produzir tempos de latência inaceitáveis.

Geralmente aplicações distribuídas seguem alguns padrões predeterminados de interações em suas atividades. Esses padrões identificam, dentro da aplicação, um bem definido escopo de referências, da mesma maneira que variáveis locais tem um escopo restrito na programação seqüencial.

O conceito da memória compartilhada é bem aceito por causa de vários anos de experiência em programação paralela e distribuída. Além disso, modelos computacionais abstratos como a arquitetura PRAM (*Paralell Random Access Memory*) [SKI 1990], produziram e difundiram *know-how* em algoritmos de computação paralela baseados em arquiteturas de memória distribuída.

Na programação paralela e distribuída, a memória compartilhada tem novos objetivos, tais como:

- sistema de comunicação: comunicação entre processos distribuídos pode

operar a base de memória compartilhada, facilitando a implementação de vários tipos de comunicação.

- armazenamento para informação de monitoramento ou debug: em um ambiente local, pode ser difícil e até mesmo oneroso obter informações globais sobre o comportamento do sistema e das aplicações, apesar dessa informação ser essencial.
- memória virtual: memória compartilhada permite que processos acessem uma determinada quantidade de memória que eles necessitam.

Os modelos de memória compartilhada e de troca de mensagens não são mutuamente exclusivos [KRA 1993]. Como exemplo, programação paralela utilizando ambientes de orientação a objeto, os quais o modelo computacional define um cenário de troca de mensagens sem compartilhar recursos, podem se beneficiar da presença de memória compartilhada [MAT 1988].

Entretanto, a implementação física da memória compartilhada apresenta inúmeros problemas [DUN 1990]: entre eles introduz uma centralização que limita a escalabilidade do sistema, o que acaba virando um gargalo quando o tamanho do sistema aumenta muito. Uma solução para esse problema seria a utilização de uma *memória compartilhada distribuída*, onde a memória compartilhada é implementada como uma abstração de software em uma arquitetura de memória distribuída.

Em memórias compartilhadas distribuídas, o espaço de memória lógico é distribuído entre diferentes nodos, cada um com seu espaço de memória local e execução autônoma. A informação contida nessa memória é disponível, de quaisquer nodos em que ela se encontre, para qualquer nodo do sistema, e isso é feito de maneira completamente transparente. A replicação de dados pode fazer com que os tempos de acesso fiquem menores, explorando a localidade desses dados de maneira semelhante a uma memória cache [STE 1990]. Infelizmente, isso introduz o problema de consistência de dados, ou coerência entre as diversas réplicas, fazendo com que seja necessário um protocolo que pode introduzir custos elevados de processamento e comunicação para garantir essa consistência [STU 1990].

3.2 Espaço de tuplas: conceitos básicos

Um espaço de tuplas é essencialmente uma abstração de uma memória

compartilhada, garantindo que vários processos possam acessar o espaço de maneira anônima e assíncrona, sem introduzir os ônus inerentes a memória distribuída.

O espaço de tuplas é baseado em mensagens deixadas pelos processos que compartilham esse espaço, essas mensagens são chamadas de *tuplas*. Cada tupla é formada por uma seqüência de argumentos tipados, por exemplo, <”Tupla”,”Exehda-CC”,1.0> é uma tupla a qual contém a informação que está sendo inserida no espaço.

O espaço de tuplas exige uma maneira peculiar de procura por tuplas nele inseridas. Quando desejamos ler alguma tupla, seja para remoção ou apenas a simples leitura temos que criar um *template* da tupla que estamos procurando. Utilizando ainda o exemplo do segundo parágrafo alguns *templates* que fariam o espaço retornar a tupla seriam: <”Tupla”, ?String, ?Float> , <?String, ?String”1.0”> , etc.

No caso da leitura, o espaço de tuplas faz uma comparação entre as tuplas nele contidas e a *template* passada, no primeiro exemplo do parágrafo anterior temos “Tupla” como valor atual, ?String e ?Float como valores formais. Desta maneira após comparar a tupla de pesquisa com as tuplas nele contidas, o espaço retorna uma tupla de maneira não-determinística. Caso não encontre nenhuma, o espaço tomará duas possíveis ações dependendo do tipo de primitiva utilizada pelo programador:

- se for utilizada a primitiva bloqueante, o espaço aguarda até que seja inserida nele uma tupla que combine com o template utilizado, efetivamente assim bloqueando o processo requisitor;
- se for utilizada a primitiva de procura (probe), o espaço retorna um valor nulo, não bloqueando o processo requisitor.

3.3 Vantagens do espaço de tuplas sobre troca de mensagens

O método mais comum de comunicação entre processos é a troca de mensagens. Nessa forma de comunicação ambas as partes, o emissor e o receptor, tem que ser previamente identificados antes da conexão entre eles seja estabelecida, após essa conexão o emissor passa a mensagem e espera uma confirmação de recebimento do receptor antes de fechar a conexão, como vemos na figura 3.1.

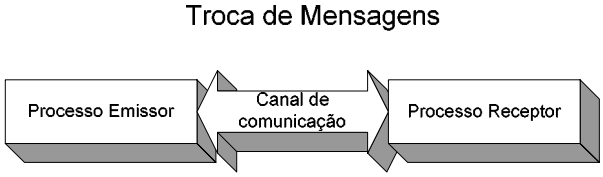

Figura 3.1 - Troca de mensagens.

Se utilizarmos um espaço de tuplas vemos algumas vantagens sobre a troca de mensagens, uma dessas vantagens é o anonimato. O anonimato é atingido já que o espaço de tuplas não guarda nenhuma informação sobre o processo que depositou a tupla como também não guarda informação do processo que retirou essa tupla. Esse tipo de assincronia é à base das vantagens do espaço de tupla sobre a troca de mensagens.

Assincronia temporal é outra vantagem que o espaço de tuplas tem sobre alguns modelos de troca de mensagens, e refere ao fato que o "emissor" não precisa esperar pela confirmação de recebimento do receptor; tudo que o "emissor" faz é inserir uma tupla no espaço. O processo que recupera essa tupla pode nem sequer existir nesse momento, pois não existe um limite de tempo para uma tupla permanecer no espaço. Com isso, um processo pode inserir todas as tuplas que necessita no espaço e seguir adiante com seu processamento, não se preocupando com a recepção dessas tuplas pelos outros processos. Vemos isso ilustrado na figura 3.2.

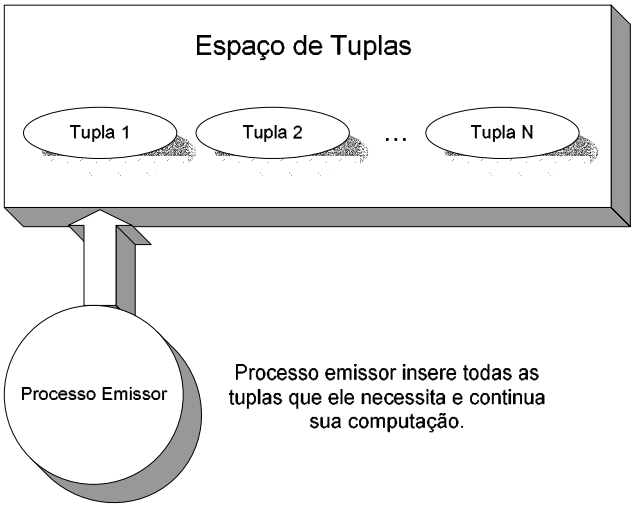

Figura 3.2 - Assincronia Temporal (a).

Outro tipo de assincronia temporal se dá quando um processo requisita uma tupla e o processo que deveria ter inserido essa tupla no espaço ainda não a inseriu, ou

sequer existe ainda. Nesse caso o processo que requisitou a tupla ficará bloqueado enquanto a tupla não for inserida no espaço. Com isso temos uma maneira de sincronizar processos que não foram criados ao mesmo tempo e nem sequer sabem da existência um do outro, vemos isso na figura 3.3.

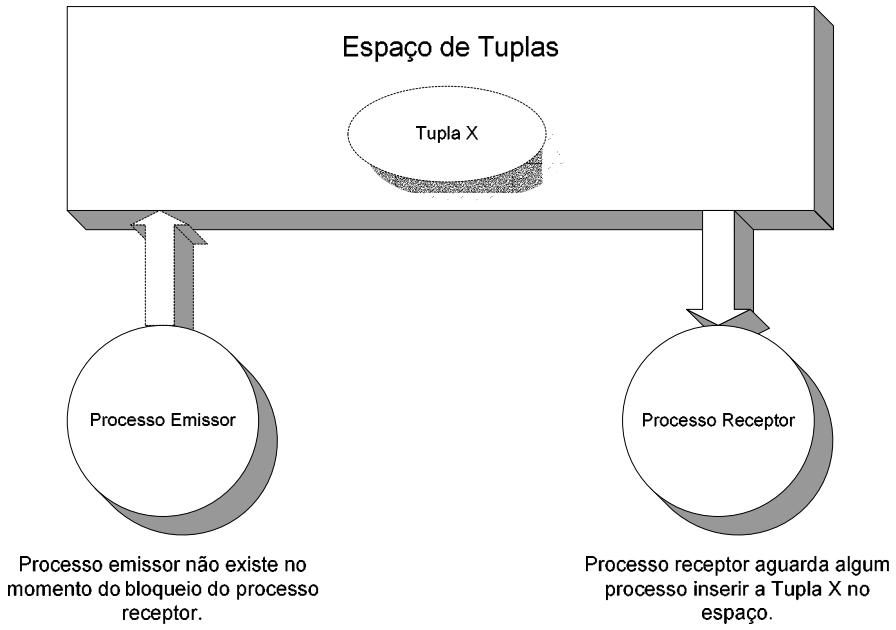

Figura 3.3 - Assincronia Temporal (b).

Assincronia espacial é outra grande vantagem, que ocorre devido ao fato de que processos não sabem quais processos retirarão suas tuplas do espaço mais tarde. Similarmente um processo que insere um determinado tipo de tupla pode nem sempre ser o mesmo. Isso dá suporte a implementação de um método transparente de compartilhamento de carga computacional, sem ter que implementar qualquer mecanismo extra, além do espaço de tuplas. Vários processos designados para atender tarefas podem aceitar tuplas contendo as tarefas a serem realizadas, já que o processo requerendo a tarefa não se importa em quem a realiza, apenas se a realiza corretamente, vemos isso mais claramente na figura 3.4.

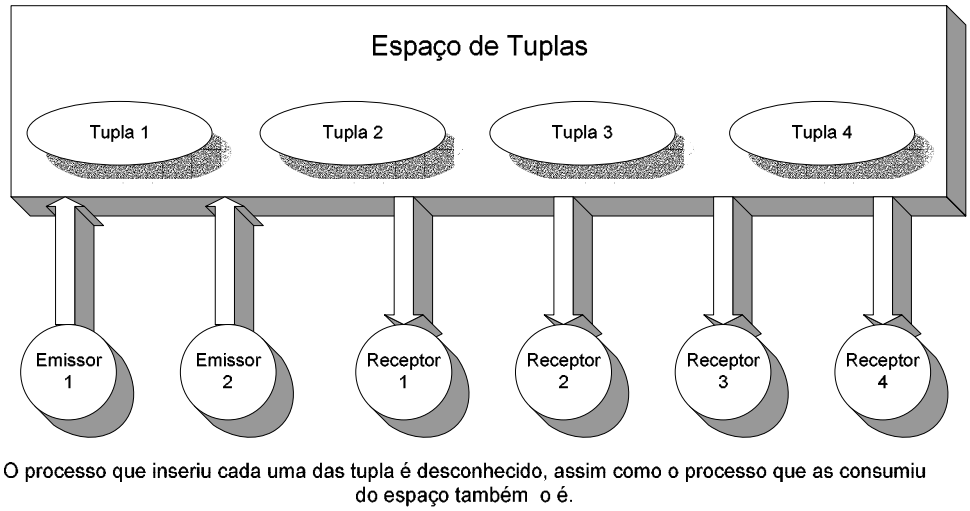

Figura 3.4 - Assincronia Espacial.

Outra vantagem da assincronia espacial é que qualquer processo travado pode ser reiniciado, criando outro processo exatamente igual a aquele que travou e examinando os conteúdos das tuplas existentes no espaço (para se assegurar que as tuplas não interferirão no correto andamento do novo processo); os demais processos não precisarão serem informados de nenhuma mudança.

3.4 O modelo LINDA

O modelo LINDA segue o modelo genérico do espaço de tuplas, veremos agora como os mecanismos deste modelo funcionam.

LINDA é um modelo de programação concorrente desenvolvido na Universidade de Yale que tem como tema central uma alternativa aos dois tradicionais métodos de processamento paralelo: o de memória compartilhada e o de troca de mensagens.

Do ponto de vista da aplicação, LINDA é uma extensão de linguagem de programação que permite uma comunicação anônima e assíncrona. Basicamente o modelo LINDA é uma abstração de memória distribuída associativa, a qual chamamos de espaço de tuplas, que provê mecanismos para a comunicação entre processos sem requerer que uma memória seja fisicamente distribuída.

Uma visão geral do modelo encontramos na figura 3.5.

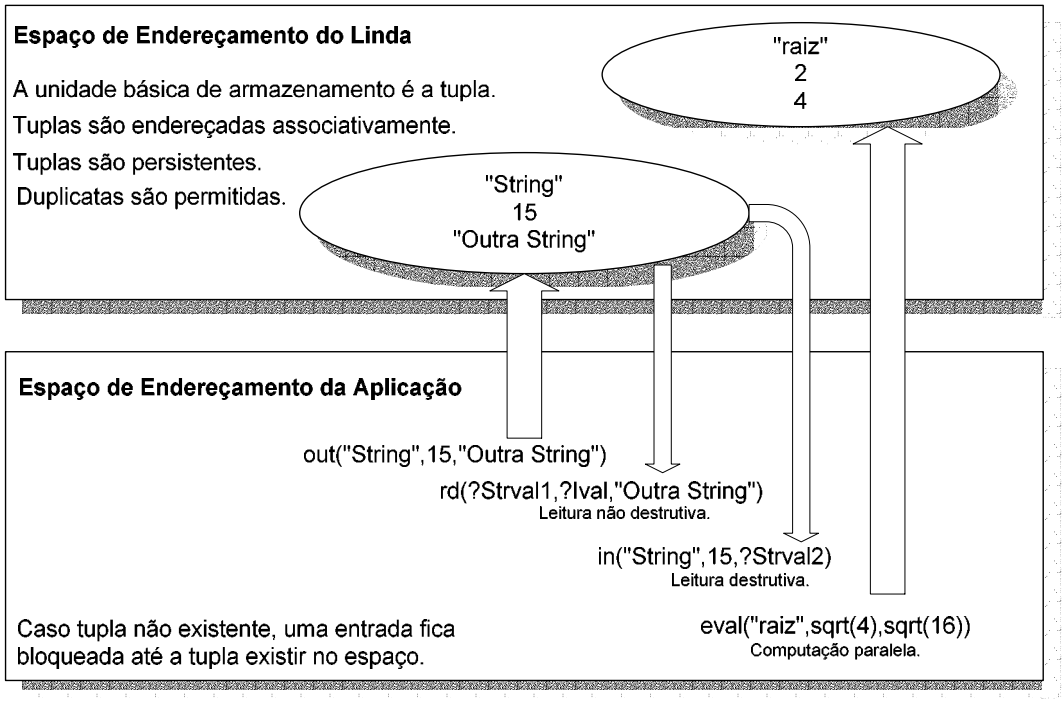

Figura 3.5 - O Modelo Linda.

O modelo LINDA tem quatro primitivas básicas;

- Rd(t): faz a leitura não destrutiva de uma tupla t no espaço;
- Out(t): insere uma tupla t no espaço;
- In(t): faz uma leitura destrutiva de uma tupla t no espaço;
- Eval(expressão): cria uma tupla nova com os campos da expressão dinamicamente calculados por processos independentes.

A utilização dessas primitivas já foi mostrada na figura 3.5, entretanto agora será explicado com maiores detalhes.

A primitiva out(“Tupla”,1.0,1701,”Exehda-CC”) colocaria uma tupla no espaço com essas características e poderia ser lida pelas seguintes primitivas rd();

- rd(“Tupla”, ?fval, ?ival, ?strval);
- rd(?strval, 1.0, ?ival, “Exehda-CC”);
- rd(?strval-1, ?fval, 1701, ?strval-2).

O operador ? designa um valor qualquer que combine com o argumento formal passado e não participam do processo de procura da memória associativa. Qualquer um dos comandos citados retornaria a tupla inserida no parágrafo anterior e não a retiraria da memória, entretanto se a primitiva utilizada fosse o in() esta removeria a tupla do

espaço.

Para ilustrar a primitiva eval() considere o seguinte exemplo:

- eval(“raiz”,sqrt(4),sqrt(16))

As funções sqrt (square root, raiz quadrada) são computadas por processos independentes do processo original, quando os três valores da tupla são obtidos, a primitiva eval() combina os resultados na ordem original e insere a tupla resultante no espaço.

4 MODELAGEM DO EXEHDA-CC

A modelagem do Exehda-CC foi feita para que ele atingisse alguns objetivos específicos, necessários pela arquitetura EXEHDA. Neste capitulo, será mostrado como ocorreu seu modelo e porque algumas decisões foram tomadas.

4.1 Objetivos do EXEHDA-CC

Devido ser um módulo integrante de um sistema, o Exehda-CC teve que ser criado com certos objetivos em mente, tais como:

- integrável: como parte de um sistema, esse módulo tem que ser completamente integrável no restante, assim como deve ser compatível com o software já existente da arquitetura EXEHDA;
- portável: devido à proposta do EXEHDA ser um ambiente de execução portável, assim deve ser todos os seus módulos e serviços;
- suporte a múltiplos espaços de tuplas: como o Holoparadigma utiliza o conceito de cada ente ter sua própria história, e apenas os filhos podem ver a história do pai, é imprescindível o Exehda-CC dar o suporte necessário a essa característica;
- transparência: uma das principais características do Exehda é ser transparente ao programador;
- escabilidade: o Exehda-CC deve ser escalável para suportar o crescimento do sistema.

Esses objetivos ditaram todo o progresso do exehda-cc na sua fase de modelagem, e foi concebida de forma gradual e evolucionária, a qual pode ser dividida em três fases:

- fase de acesso local;
- fase de múltiplos espaços;
- fase de acesso remoto.

4.2 Fase de acesso local

Na fase de acesso local foram levados todos os objetivos relevantes a um acesso

local de ente a sua história, assim como os acessos da arquitetura a seu espaço compartilhado para tomada de decisão.

Logo no início, foi visto que o espaço de tuplas era muito rígido a nível conceitual para que os programadores, e até mesmo a própria arquitetura, o pudessem utilizar de maneira eficiente. Para resolver esse problema foi adotado que o Exehda-CC utilizaria uma pequena evolução do conceito de espaço de tuplas, o *espaço de objetos*.

4.2.1 O espaço de objetos do Exehda-CC

O espaço de objetos segue os mesmos objetivos do espaço de tuplas, porém, não armazena, em seu espaço, tuplas contendo argumentos tipados, mas sim objetos, a diferença pode não parecer óbvia a primeira vista, mas a nível de implementação é significativa, isso será discutido no capítulo 5.

A evolução na abstração permitiu um melhor modelo, já que foi utilizada uma técnica de modelagem de projeto orientado a objetos, e também uma maior liberdade ao que os programadores podem armazenar no espaço, como armazenar e como localizar o objeto desejado.

4.2.2 Modelando o acesso local

Devido ao EXEHDA inicialmente utilizar o Jada[ROS 2002] como uma solução de espaço de tuplas, foi optado que, para manter uma compatibilidade com as aplicações já prontas, que o Exehda-CC utilizasse a mesma API do Jada. Isso nos da já as primeiras primitivas que a classe criada para o acesso local iria implementar: o in() e o out().

Entretanto, para que se tivesse um maior controle de como os objetos são armazenados no espaço, uma classe básica de objeto a ser armazenado deveria ser criada, em outras palavras, a mínima unidade do espaço de objetos.

Então se chegou a classe ObjectSpace, que contém os seguintes métodos:

- create(): este método, quando chamado, cria o objeto e armazena nele o objeto passado para o espaço;
- read(): este método retorna o objeto armazenado pela classe.

Assim, obtemos o diagrama UML da figura 4.1.

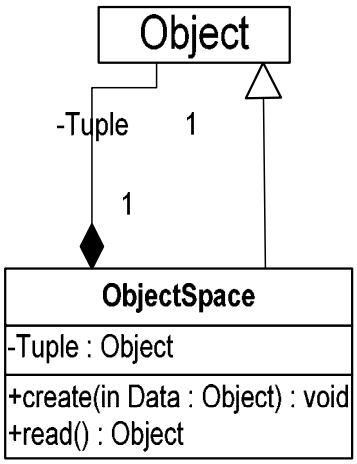

Figura 4.1 - UML da classe ObjectSpace.

Após definido o objeto base do nosso espaço, então se começou a modelagem do espaço de objetos local. Seguindo os objetivos descritos na sessão 4.1, se obteve uma classe com as seguintes propriedades:

- rápido acesso a objetos nela armazenados;
- possibilidade de duplicidade de objetos.

Então foi tomada a decisão de utilizar uma tabela *hash* para o armazenamento principal, auxiliada por um vetor para os objetos duplicados. Assim respeitamos, dentro de limites de implementação, as propriedades acima, assim como os objetivos mencionados na sessão 4.1. O nome escolhido para essa classe foi *ObjectSpaceCoordenator*, devido a ela coordenar todo o acesso ao espaço de objetos local. A API do Exehda-CC também começa nessa classe, portanto as primitivas são homônimas com as do LINDA, já que o Jada também se baseou nele para a sua criação. Ficando, portanto, da seguinte maneira os métodos públicos dessa classe:

- out(objeto);
- in(objeto);
- rd(objeto).

Foram necessários dois métodos auxiliares para simplificar a procura e remoção dos objetos do espaço, são eles respectivamente:

- findObject(objeto);
- remove(objeto).

Obtendo assim o UML da figura 4.2.

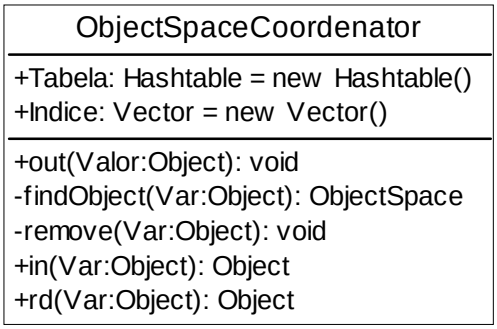

Figura 4.2 - UML da classe *ObjectSpaceCoordenator*.

4.3 Modelando os múltiplos espaços

Após obter um espaço de objetos local funcional, começaram os estudos pertinentes aos múltiplos espaços. A solução básica era óbvia, bastava ter múltiplas instâncias do *ObjectSpaceCoordenator* e já se obteria múltiplos espaços, mas não é assim tão simples, tem também o problema de endereçamento dos espaços, como fazer para os entes endereçarem o espaço desejado, tendo esse problema em mente, chegou-se a idéia de um coordenador de múltiplos espaços. Para isso ele apenas precisava ter as seguintes características:

- instanciar múltiplos *ObjectSpaceCoordenator* de maneira ordenada;
- fazer o endereçamento correto do *ObjectSpaceCoodenator* de cada ente;

Para atender múltiplos entes ao mesmo tempo o coordenador de múltiplos espaços deveria ser multiprocessado, então surgiu a idéia de usar um servidor RMI(*Remote Method Invocation*) para tal função, o que ao mesmo tempo iria poder prover o acesso remoto ao espaço de objetos, portanto a partir do momento da adoção do RMI como solução do problema do multiprocessamento do coordenador, também começou a modelagem do acesso remoto.

4.4 Modelando o acesso remoto

Devido a característica do RMI exigir uma interface para os métodos de cada classe que fosse possível acessar remotamente, tanto o coordenador de acesso remoto quanto o coordenador de múltiplos espaços tem suas interfaces, isso faz com que um *stub* seja criado para as classes remotas e os entes pudessem acessar o coordenador com facilidade.

O nome escolhido para a interface de comunicação com o coordenador de múltiplos espaços foi *ObjectServer* devido ao fato de ser um servidor remoto na maioria dos casos, exceto para os entes criados na EXEHDABase, nela estão contidos todos os métodos que são acessados remotamente pelos entes. Nessa fase do modelamento foram escolhido mais dois métodos a serem inseridos nas primitivas básicas do EXEHDA-CC, são eles:

- inp(objeto): variação do método in() no modo *probe*, ou seja, caso não encontre nenhum objeto no espaço ele retorna um valor nulo ao processo requisitor.
- rdp(objeto): variação do método rd() no modo *probe*.

Com a adição desses métodos obtemos o UML para a interface como na figura 4.3.

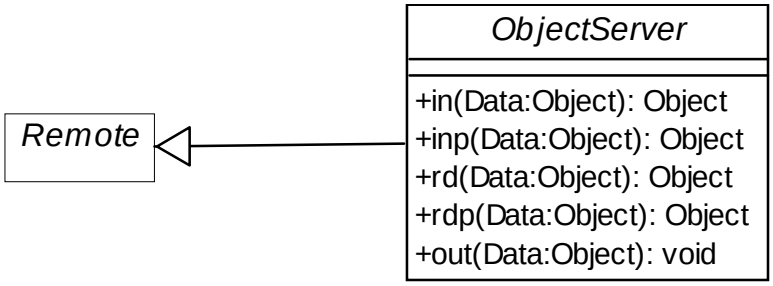

Figura 4.3 - UML da classe *ObjectServer*.

Com o modelo do acesso remoto já definido apenas é necessário modelar a classe que atendera as requisições remotas, em outras palavras, a classe que implementa a interface. Essa classe foi chamada de *ObjectServerImplementation* e pode ser vista no UML da figura 4.4.

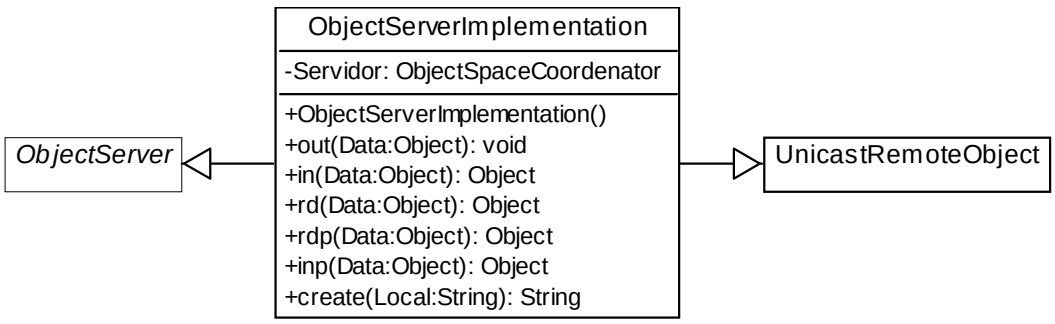

Figura 4.4 - UML da classe *ObjectServerImplementation*.

Com isso foi resolvido o problema do acesso remoto, mas o problema da gerência de múltiplos espaços necessita de mais uma camada RMI, nela foi atribuída a função de criação dos múltiplos espaços, assim como o desligamento total do servidor, quando não mais necessário.

Novamente necessitamos obter uma interface para esse objeto, então foi definida a interface mostrada na figura 4.5.

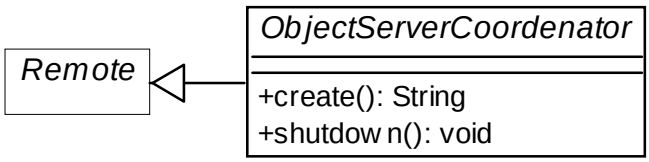

Figura 4.5 - UML da classe *ObjectServerCoordenator*.

A implementação dessa interface é de fácil modelagem já que é bem direta, entretanto convém lembrar que essa classe necessita fazer o correto endereçamento de cada espaço de objetos, uma maneira fácil de fazer isso foi fazer com que quando um espaço é criado, o coordenador retorna o endereço RMI do servidor recém criado para atender esse espaço. Isso contribuiu para a legibilidade do modelo, pois a classe que implementa o coordenador de múltiplos espaços, de agora em diante referida como *ObjectServerCoordenatorImp*, cria um novo espaço e retorna o endereço desse espaço para o objeto que requisitou essa criação, em seguida o objeto requisitante se liga a esse servidor e começa a o utilizar de maneira remota através da interface *ObjectServer*. Portanto o *ObjectServerCoordenatorImp* necessita apenas de uma propriedade que é o número, indicando um número seqüencial de espaços de objetos aberto, essa propriedade é utilizada para nomear um novo espaço no processo de criação.

Podemos ver o UML do *ObjectServerCoordenatorImp* na figura 4.6.

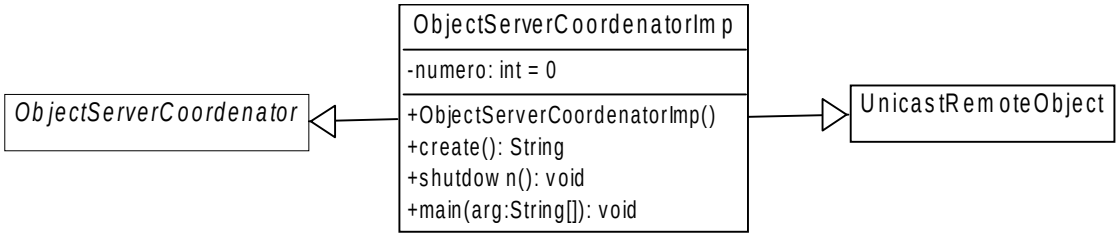

Figura 4.6 - UML da classe *ObjectServerCoordenatorImp*.

5 A IMPLEMENTAÇÃO E TESTES DO EXEHDA-CC

Nesse capítulo serão caracterizados os principais aspectos relevantes a implementação do EXEHDA-CC, a linguagem utilizada, as principais tecnologias relacionadas e, por fim, será feita uma análise de resultados atingidos.

5.1 A linguagem Java

Java é uma linguagem que foi criada pela *Sun Microsystems* a partir de uma pesquisa corporativa interna com o codinome *Green* e que ocorreu no ano de 1991. Deste projeto resultou a linguagem que é baseada em C e C++ e que seu criador, James Gosling, chamou de *Oak* (carvalho) em homenagem a uma árvore que dava para a janela de seu escritório na Sun. Descobriu-se mais tarde que já havia uma linguagem de programação que se chamava *Oak*. Quando uma equipe da Sun visitou uma cafeteria local, o nome Java (cidade de origem de um tipo de café importado) foi sugerido e pegou.

Mas o projeto *Green* atravessava algumas dificuldades. O mercado para dispositivos eletrônicos inteligentes destinados ao consumidor final não estava se desenvolvendo tão rapidamente como a Sun tinha previsto. Pior ainda, um contrato importante pelo qual a Sun competia fora concedido à outra empresa. Então o projeto estava em risco de cancelamento. Por pura sorte, a *World Wide Web* explodiu em popularidade em 1993 e as pessoas da Sun viram imediato potencial de utilizar Java para criar páginas da *Web* com o chamado conteúdo dinâmico. Isso deu nova vida ao projeto.

Java apresenta inúmeras vantagens quando utilizada para a programação de um sistema distribuído como, por exemplo, a portabilidade de seu código que uma vez compilado e gerado o *bytecode*, este poderá ser executado em qualquer arquitetura que possua uma **Máquina Virtual Java** (JVM). Outra vantagem relacionada à portabilidade é ela implementar primitivas de tipo padronizadas como, por exemplo, o *integer* ter 32 bits em todas as máquinas.

Dentre outras vantagens proporcionadas por Java estão: a eliminação dos principais geradores de erros (ponteiros e gerenciamento de memória via código), a sua alta performance com *multithreading* interno para períodos de inatividade do

processador e também facilitando a paralelização das aplicações, além de que quando utilizada para implementar sistemas por um grupo de pessoas, devido a ela ser orientada a objeto, ela facilita o encapsulamento do código em entidades as quais podem ser tratadas independentemente facilitando assim também a reutilização de código.

5.2 O *Remote Method Invocation*

A Java *Remote Method Invocation* (RMI) permite que objetos Java executando no mesmo computador ou em computadores diferentes se comuniquem entre si via **chamadas de método remoto**. Essas chamadas de método são semelhantes àquelas que operam em objetos no mesmo programa.

A RMI está baseada em uma tecnologia anterior semelhante para programação procedural, chamada de chamadas de procedimento remoto (*remote procedure calls* – RPCs), desenvolvida nos anos de 1980. A RPC permite que um programa procedural (isto é, um programa escrito em C ou outra linguagem de programação procedural) chame uma função que reside em outro computador tão convenientemente como se essa função fosse parte do mesmo programa que executa no mesmo computador. Um objetivo da RPC foi permitir aos programadores se concentrar nas tarefas exigidas de um aplicativo chamando funções e, ao mesmo tempo, tornar transparente para o programador o mecanismo que permite que as partes do aplicativo se comuniquem através de uma rede. A RPC realiza todas as funções de rede e ordenação (*marshalling*) dos dados (isto é, empacotamento de argumentos de função e valores de retorno para transmissão através de uma rede). Uma desvantagem de RPC é que ela suporta um conjunto limitado de tipos de dados simples. Portanto, a RPC não é adequada para passar e retornar objetos Java. Outra desvantagem da RPC é ela que exige do programador aprender uma linguagem de definição de interface (*interface definition language* – IDL) especial para descrever as funções que podem ser invocadas remotamente.

A RMI é implementação da RPC por Java para comunicação distribuída de um objeto Java com outro. Uma vez que um método (ou serviço) de um objeto Java é registrado como sendo remotamente acessível, um cliente pode “pesquisar” (“*lookup*”) esse serviço e receber uma referência que permita ao cliente utilizar esse serviço (isto é, chamar o método). A sintaxe da chamada de método é idêntica àquela de uma chamada

para um método de outro objeto do mesmo programa. Como com a RPC, a ordenação (*marshalling*) dos dados é tratada pela RMI. Entretanto, a RMI oferece transferência de objetos de tipos de dados complexos via o mecanismo de serialização de objeto. Essas razões fizeram que o RMI fosse escolhido como o método de comunicação entre os objetos EXEHDA-CC.

5.3 A Topologia de Interconexão Empregada no EXEHDA-CC

No caso do EXEHDA, o processamento ocorre organizado em células. A abrangência de uma célula é determinada tendo por critério o custo de comunicação entre os nós processadores envolvidos. Este critério é subjetivo, e sua aplicação é feita pelo administrador do sistema.

Os equipamentos EXEHDAbase constituem o centro administrativo da célula, e dentre outras atribuições alojam o módulo distribuído EXEHDA-CC que se interligam através de um mecanismo de pesquisa remota.

Este módulo EXEHDA-CC responde por requisições RMI feitas pelos entes que estão trabalhando em sua célula, em casos que há mobilidade física do ente, ou seja, ele migrar de célula, a EXEHDAbase local recebe suas requisições, e se esse ente estende um espaço de objetos de outro EXEHDAbase, então há uma comunicação entre as bases para atender essa requisição.

5.4 Espaço de objetos x espaço de tuplas

A escolha do espaço de objetos foi tomada pensando na implementação. Devido a linguagem de implementação escolhida ser o Java, isso faz com que tenhamos toda uma gama de classes e métodos para manipulação de objetos, assim a utilização dessas funções intrínsecas da linguagem torna sua programação mais fácil e elegante.

Também foi tomada levando em consideração a alta abstração do espaço com o que se esta inserindo nele, permitindo que o programador decida a estrutura de seu objeto a ser armazenado no espaço. Lembrando que no espaço de tuplas os argumentos são fortemente tipados, no espaço de objetos há uma maior liberdade de onde se por o objeto de pesquisa, pois a procura associativa do EXEHDA-CC permite que todos os objetos na mais alta hierarquia sejam comparados com o objeto de pesquisa, ou seja, se o objeto armazenado for um vetor de vetores, apenas os objetos do vetor “externo”

serão comparados com o padrão de pesquisa. O espaço de objetos também simula um espaço de tuplas perfeitamente, desde que esse seja na verdade um espaço de vetores, onde cada vetor simula uma tupla.

Assim temos um espaço que permite ao programador uma maior liberdade quanto a organização de seus objetos a serem armazenados, que permite uma fácil simulação de um espaço de tuplas caso necessário e ainda é de fácil implementação, tendo um código fonte de fácil compreensão para suas eventuais atualizações devido ao crescimento do ambiente de execução EXEHDA.

5.5 Testes: comparação com o Jada

Nessa sessão mostraremos alguns testes de desempenho da implementação do Exehda-CC com uma implementação de espaço de tuplas chamada Jada.

O Jada foi criado como parte integrante do projeto *PageSpace*, com o intuito de prover um sólido espaço de tuplas. Antes do desenvolvimento do Exehda-CC, o Jada foi utilizado pelo EXEHDA/Holoparadigma para fazer as simulações de sincronismo entre processos. Nada mais natural do que comparar o EXEHDA-CC com seu antecessor.

Os testes foram realizados nas máquinas do G3PD, que tem a seguinte configuração:

- Processador Intel Pentium® II 266 MHZ.
- 64 Megabytes de memória RAM.
- Sistema Operacional *Linux Debian Woody*.
- Java 2 SDK da *Sun Microsystems* versão 1.4.0.

5.5.1 Teste 1: Sincronismo de processos distribuídos

Esse teste mostra o desempenho do espaço de objetos ma função de manter sincronismo entre dois processos distribuídos em uma EXEHDACél.

O teste foi feito de maneira que o primeiro processo envia um sinal de sincronismo e aguarda a resposta do outro processo, o segundo processo recebe os sinais enviados pelo primeiro e então devolve uma resposta. Deste modo são manipulados pelo espaço de objetos dois objetos a cada ciclo de repetição.

Na tabela 5.1 apresentamos os resultados atingidos pelo EXEHDA-CC, na tabela 5.3 apresentamos os resultados atingidos pelo Jada, e por fim os resultados computados de forma gráfica na figura 5.1.

Tabela 5.1 - Resultados obtidos pelo EXEHDA-CC (teste 1).

Repe-tições	Medição 1	Medição 2	Medição 3	Medição 4	Medição 5	Medição 6	Medição 7	Medição 8
1000	11887	12108	11767	12178	11918	11957	12458	11997
1500	16123	15813	16324	15803	16884	15542	16704	15582
2000	19418	19498	19338	19798	19748	19729	19528	19898
2500	23224	23423	23384	24004	23293	22943	23163	23764
3000	26909	26718	26328	26458	26458	27079	27390	25897
3500	31084	30454	29913	29863	29893	29663	29212	29182
4000	33107	33927	32857	32817	33368	32497	32877	33368
4500	35661	35781	36272	36172	35972	35831	36432	36102
5000	39096	40208	40398	39037	39156	39066	39347	39326

Tabela 5.2 - Tempos médios e desvio padrão atingidos pelo EXEHDA-CC (teste 1).

Repetições	média	Std	Std%
1000	12033,75	213,6885	1,775743
1500	16096,88	504,3192	3,133025
2000	19619,38	200,3233	1,021048
2500	23399,75	339,2301	1,449717
3000	26654,63	470,1969	1,764035
3500	29908	627,3395	2,097564
4000	33102,25	444,0176	1,341352
4500	36027,88	263,6575	0,731815
5000	39454,25	538,3686	1,364539

Tabela 5.3 - Resultados atingidos pelo Jada (teste 1).

Repe-tições	Medição 1	Medição 2	Medição 3	Medição 4	Medição 5	Medição 6	Medição 7	Medição 8
1000	19448	19358	19317	19688	19418	19918	19428	19708
1500	29733	27790	29553	28451	28161	27810	28191	27620
2000	36753	36833	36703	36773	36762	37804	37133	36943
2500	46107	46737	44584	44543	44744	44444	45376	45275
3000	53297	54319	54618	53226	53737	53307	53837	53958
3500	64433	62149	61929	62850	62920	61489	62971	62980
4000	70942	71633	71022	70892	70812	71353	71753	71964
4500	80486	78282	78703	80166	80155	80796	79985	80746
5000	87706	86765	88057	87366	85042	85017	85280	86016

Tabela 5.4 - Tempos médios e desvio padrão atingidos pelo Jada (teste 1).

Repetições	média	Std	Std%
1000	19535,38	210,9793	1,079986
1500	28413,63	805,0569	2,833348
2000	36963	366,7779	0,992284
2500	45226,25	828,655	1,832243
3000	53787,38	506,1121	0,94095
3500	62715,13	893,5293	1,424743
4000	71296,38	442,3563	0,620447
4500	79914,88	929,969	1,1637
5000	86406,13	1234,971	1,429263

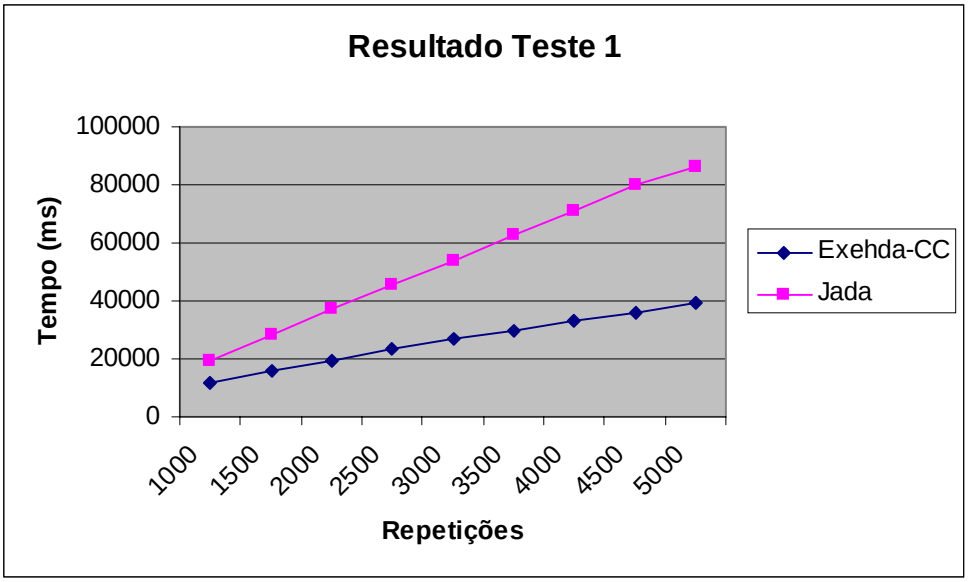

Figura 5.1 - Resultados do teste 1.

Os dados apontam que o EXEHDA-CC é mais rápido que o Jada quando se trata de enviar sinais de sincronismo entre processos. Também percebe-se que ambos têm um crescimento linear a medida que mais sinais são enviados, com o EXEHDA-CC possuindo um grau de crescimento menor do que o Jada. Isso faz com que o EXEHDA-CC atenda melhor o EXEHDA.

5.5.2 Teste 2: comunicação entre processos

Esse teste têm o intuito de mostrar o desempenho do espaço quando fazendo o papel de meio de comunicação entre processos. O teste se deu com um processo A criando uma mensagem de 512 bytes, enviando esta mensagem para o processo B via espaço de objetos e aguardando uma confirmação de retirada da mensagem do espaço.

Somente após o recebimento dessa mensagem é que o próximo ciclo na repetição se daria.

Temos então a tabela 5.5 contendo os resultados obtidos pelo EXEHDA-CC, a tabela 5.7 contendo os resultados obtidos pelo Jada , e na figura 5.2 temos uma análise gráfica dos resultados obtidos.

Tabela 5.5 - Resultados obtidos pelo EXEHDA-CC (teste 2).

Repe-tições	Medição 1	Medição 2	Medição 3	Medição 4	Medição 5	Medição 6	Medição 7	Medição 8
1000	15503	15252	15272	15202	15292	15232	15422	15252
1500	20670	20500	20089	20870	20279	20109	20159	19929
2000	24786	24726	24695	24776	24655	24806	24836	24776
2500	29252	29142	29202	29643	29202	29632	29052	29662
3000	33539	33338	33388	33458	33799	33548	33548	33398
3500	38245	37464	37985	37454	37534	37835	37694	38736
4000	41870	42251	42230	42251	41890	42061	42281	42101
4500	47599	46687	46557	46918	46928	46728	46817	46447
5000	51174	51183	51134	50713	50853	51003	51564	50443

Tabela 5.6 - Tempos médios e desvio padrão obtidos pelo EXEHDA-CC (teste 2).

Repetições	média	Std	Std%
1000	15303,38	103,9821	0,679471
1500	20325,63	324,1498	1,594784
2000	24757	60,25422	0,243383
2500	29348,38	253,0319	0,862167
3000	33502	144,6444	0,431749
3500	37868,38	445,0961	1,175377
4000	42116,88	165,4029	0,392724
4500	46835,13	350,6604	0,748712
5000	51008,38	340,9005	0,668323

Tabela 5.7 - Resultados obtidos pelo Jada (teste 2).

Repe-tições	Medição 1	Medição 2	Medição 3	Medição 4	Medição 5	Medição 6	Medição 7	Medição 8
1000	19468	19237	19187	18957	19398	19198	19158	19438
1500	27830	28140	27921	27700	29272	28622	28521	28030
2000	37034	37073	36963	37574	37253	37304	36682	36752
2500	48329	48630	48730	48750	48200	48670	48519	48750
3000	57703	57683	57763	57463	57152	57924	57343	57072
3500	67657	66536	67730	67913	66823	67609	67296	67518
4000	78663	78192	78560	78360	78313	78879	78758	78965
4500	86845	86654	86943	86153	86530	86638	87120	86909
5000	97340	96780	96698	96443	96768	96899	96448	97053

Tabela 5.8 - Tempos médios e desvio padrão obtido pelo Jada (teste 2).

Repetições	média	Std	Std%
1000	19255,13	171,4297	0,890307
1500	28254,5	521,5072	1,845749
2000	37079,38	294,2253	0,793501
2500	48572,25	207,5509	0,427303
3000	57512,88	305,3389	0,530905
3500	67385,25	475,7285	0,705983
4000	78586,25	279,4176	0,355555
4500	86724	299,9086	0,34582
5000	96803,63	299,6393	0,309533

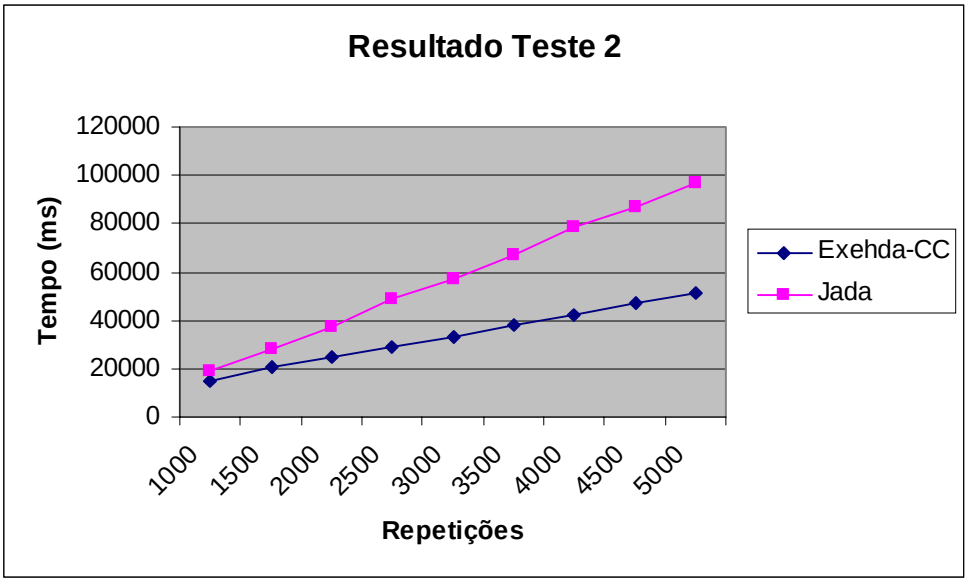

Figura 5.2 - Resultados do teste2.

Novamente o EXEHDA-CC mostrou-se superior ao Jada quando se analisou a comunicação de dados entre processos. Ambos os espaços mostraram crescimento linear, com o EXEHDA-CC tendo um crescimento menor que o Jada. Isso se mostra fundamental, já que o EXEHDA-CC também é utilizado pelo EXEHDA como meio de comunicação entre vários processos.

5.5.3 Teste 3: contenção

Esse teste tem por objetivo verificar o impacto que vários processos simultâneos causam na performance do espaço de objetos. Foi-se estabelecido que para melhor compreensão dos dados que esse teste fosse relativo a performance do espaço quando de apenas um processo acessando-o. Na tabela 5.9 temos os resultados obtidos com o EXEHDA-CC, na tabela 5.11 temos os resultados obtidos com o Jada e temos o na

figura 5.3 o gráfico da performance relativa que cada espaço obteve.

Tabela 5.9 - Resultados obtidos pelo EXEHDA-CC (teste 3).

Proces-sos	Medição 1	Medição 2	Medição 3	Medição 4	Medição 5	Medição 6	Medição 7	Medição 8
1	6336	6331	6253	6292	6219	6317	6321	6260
2	7381	7307	7228	7284	7266	7331	7331	7151
3	9035	8725	8434	8634	8464	8332	8311	8543
4	10554	10361	10478	10313	10464	10611	10564	10142

Tabela 5.10 - Tempos médios e desvio padrão obtidos pelo EXEHDA-CC (teste 3).

Repetições	média	Std	Std%
1	6291,125	42,74655	0,679474
2	7284,875	71,21283	0,977544
3	8559,75	238,3549	2,784602
4	10435,88	156,1459	1,496242

Tabela 5.11 - Resultados obtidos pelo Jada (teste 3).

Repe-tições	Medição 1	Medição 2	Medição 3	Medição 4	Medição 5	Medição 6	Medição 7	Medição 8
1	9122	9123	9309	10102	10209	9169	9143	9339
2	11219	11282	11365	11322	11305	11322	11362	12365
3	11729	11684	11525	11505	11686	12284	11784	11566
4	12333	12313	12505	12323	12455	12351	11943	11523

Tabela 5.12 - Tempos médios e desvio padrão obtidos pelo Jada (teste 3).

Repetições	média	Std	Std%
1	9439,5	450,4328	4,771787
2	11442,75	375,5285	3,281803
3	11720,38	248,6219	2,121279
4	12218,25	326,9232	2,675696

Tabela 5.13 - Tabela resumo dos tempos médios obtidos (performance relativa).

Resumo	1	2	3	4
Exehda-CC	1	1,1579606	1,3606072	1,6588249
Jada	1	1,2122199	1,2416309	1,2943747

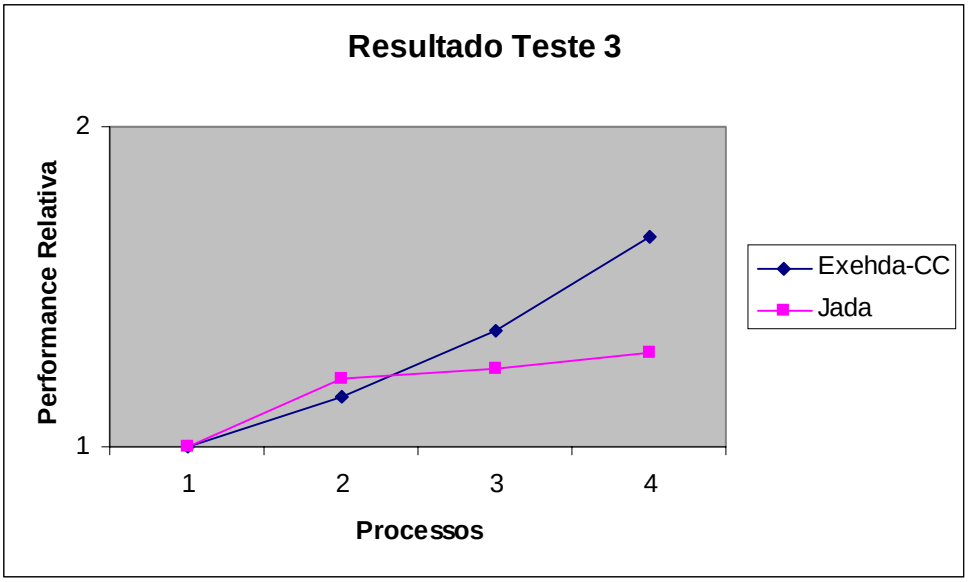

Figura 5.3 - Resultados do teste 3.

Como podemos ver pela figura 5.3 o Jada obteve melhor desempenho, obtendo apenas um decréscimo de 29% na performance, quando de quatro processos simultâneos fazendo acessos, contra o decréscimo de 65% da performance do EXEHDA-CC em iguais condições. Entretanto, olhando todos os requisitos do EXEHDA, o EXEHDA-CC ainda assim é melhor no papel de um sistema de comunicação e coordenação de processos. Tendo como principal atrativo, sua alta abstração no código fonte dos programas do usuário.

6 CONCLUSÃO

O rápido crescimento das redes de computadores nos leva a uma perspectiva de utilização de recursos distribuídos. Temos também um crescimento significativo do uso de computação móvel e nômade. Para o aproveitamento desse poder computacional surgiu o conceito de *Pervasive Computing*. O EXEHDA é um *middleware* que tem por objetivo criar um ambiente de execução para dar suporte ao Holoparadigma. Para manter um alto nível de abstração de programação, o EXEHDA tem que prover uma gama de serviços de baixo nível que abstraem vários conceitos. Também se mostrou de grande importância ter um meio de comunicação entre processos de alto nível, que tivesse as facilidades de uma memória compartilhada distribuída, mas não fosse tão onerosa. Para solucionar esse problema surgiu o EXEHDA-CC, que utilizou o conceito de espaço de objetos para atingir o objetivo desejado.

Como o conceito de espaço de objetos é anônimo, assíncrono e atemporal, temos uma série de considerações a fazer quando da etapa de modelagem. Como o EXEHDA trabalha com múltiplos espaços em múltiplas EXEHDACéls, o tamanho médio de um espaço de objetos tende a ser pequeno, portanto foi escolhido o método de tabela de *hash* para indexação de objetos a baixo nível. Isso deu ao EXEHDA-CC a performance desejada pelo ambiente.

A implementação de um ambiente de execução, como o EXEHDA, é altamente complexa e exige um elevado nível de integração entre os módulos para se atingir uma performance aceitável. Nesse ponto, a linguagem Java facilitou consideravelmente tanto a modelagem, quanto a implementação dos módulos. Possuindo vários objetos e métodos já bem otimizados, isto poupou esforços em implementação e otimização de estruturas de dados já conhecidas e amplamente utilizadas. Como conseqüência, o EXEHDA-CC tem um modelo enxuto, de fácil entendimento, o que permitirá a evolução do módulo, a medida que o EXEHDA também evolua.

É possível concluir que a procura por mais refinadas técnicas de computação paralela e distribuída é uma tendência que devemos seguir, pois nos dá um poder de processamento muito maior a baixo custo. Tendo um maior entendimento dos problemas inerentes desse tipo de computação, podemos criar *hardware* e *softwares* capazes de melhor lidar com eles.

Também pode ser concluído que o espaço de objetos é uma técnica poderosa de abstração de memória distribuída, nos dando uma maior liberdade para sincronizar processos, e guardar informações. Isso deu ao EXEHDA-CC uma grande capacidade de sincronizar processos atemporais. Pelos métodos de sincronização convencionais isso jamais seria possível.

Como trabalhos futuros na linha de pesquisa do EXEHDA-CC, destaca-se:

- adição do suporte a tuplas reativas, dando a ele uma capacidade ainda maior de reagir e se adequar as mudanças no ambiente;
- aprimoramento dos algoritmos de busca e inserção de tuplas;
- integração do EXEHDA-CC com outros módulos do EXEHDA.

7 Bibliografia

[AUG 2001] AUGUSTIN, Iara; YAMIN, Adenauer; BARBOSA, Jorge; GEYER, Cláudio. Requisitos para o Projeto de Aplicações Móveis Distribuídas. VIII CACIC CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN. Santa Cruz, Argentina. Oct, 2001.

[AVE 2000] AVEDAL, Karl. Distributed Computing Using RMI. In: Java Server Programming J2EE Edition. Chicago: Wrox Press, 2000. v.1, chap. 2, p.35-97.

[BAR 2002] BARBOSA, Jorge L. V. Holoparadigma: Um Modelo Multiparadigma Orientado ao Desenvolvimento de Software Distribuído. 2002. 215p. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[YAM 2001] YAMIN, Adenauer; AUGUSTIN, Iara; BARBOSA, Jorge. SILVA, Luciano; GEYER, Cláudio. Explorando o Escalonamento no Desempenho de Aplicações Móveis Distribuídas. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, II, WSCAD 2001. Anais . Pirenópolis, set., 2001.

[YAM 2002a] YAMIN, Adenauer Corrêa et al. ISAM: a Pervasive View in Distributed Mobile Computing. In: NETWORK CONTROL AND ENGINEERING FOR QOS, SECURITY AND MOBILITY WITH FOCUS ON POLICY-BASED NETWORKING, 2002, Paris, France. **Proceedings…** Paris: IEEE/IFIP, October 2002.

[YAM 2002b] YAMIN, Adenauer Corrêa et al. A Framework for Exploiting Adaptation in High Heterogeneous Distributed Processing. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 14., 2002, Vitória, Brasil. **Proceedings…** New York IEEE Press, October 2002.

[YAM 2002c] YAMIN, Adenauer Corrêa et al. Collaborative Multilevel Adaptation in Distributed Mobile Applications. In: International Conference of the Chilean Computer Science Society (SCCC 2002), 12., 2002, Atacama, Chile. **Proceedings…** New York IEEE Press. Atacama,. November 2002.

[NIT 1991] B.NITZBERG, V.Lo. Distributed Shared Memory: A Survey of Issues and Algorithms, IEEE Computer, Vol 24, No. 8, Aug 1991.

[SKI 1990] SKILLICORN, D.B. Architecture-Independent Parallel Computation, IEEE Computer, Dec. 1990.

[KRA 1993] KRANZ, D. et at. Integrating Message-Passing and Shared-Memory: Early Experience, ACM Sigplan Notices, Vol. 28, No. 7, July 1993.

[MAT 1988] MATSUOKA, S.; KAWAI, S. Using Tuple Space Communication in Distributed Object-Oriented Languages, OOPSLA ’88 Conference Proceedings, San Diego (CA), Sept. 1988.

[DUN 1990] DUNCAN, R. A survey of Parallel Computer Architectures, IEEE Computer, Feb. 1990.

[STE 1990] STENSTROM, P. A Survey of Cache Coherence Schemes for Multiprocessors, IEEE Computer, June 1990.

[STU 1990] STUMM, M.; ZHOU S. Algorithms Implementing Distributed Shared Memory, IEEE Computer, May 1990.

[LEA 2001] LEA, Doug. Objects in Groups. Disponível em: <http://gee.cs.oswego.edu/dl/groups/>. Acesso em: novembro 2001.

[ROS 2002] ROSSI, Davide. Jada, Disponível em: <http://www.cs.unibo.it/~rossi/jada/>. Acesso em: Outubro 2002.