DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-2000-02

PARTICIONAMENTO TRANSPARENTE DE AMBIENTES VIRTUAIS DISTRIBUÍDOS

Marcos Alves
Markus Endler

abril de 2000

Particionamento Transparente de Ambientes Virtuais Distribuídos

Marcos Alves e Markus Endler
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
{alves,endler}@ime.usp.br

Resumo

Este artigo trata de uma técnica para a construção de Ambientes Virtuais Distribuídos (*AVDs*) compartilhados por muitos usuários que visa reduzir o número de interações entre os processos distribuídos. A principal contribuição deste trabalho consiste do desenvolvimento de protocolos distribuídos para uma arquitetura de servidores descentralizados que permitem um particionamento de um ambiente virtual em regiões (domínios) regulares, de maneira que a migração de elementos (atores) de um domínio para outro seja praticamente imperceptível para o usuário de um *AVD*.

Palavras-chave: Ambiente Virtual Multi-Usuário e Distribuído, Realidade Virtual, Protocolos Distribuídos, Java.

1 Introdução

Um Ambiente Virtual (*AV*) é um programa que simula em tempo real um mundo (ou espaço) imaginário, no qual usuários se movem em um mundo virtual (*MV*) e interagem com objetos representados neste mundo por elementos gráficos. Um Ambiente Virtual Multi-usuário (*AVM*) é uma aplicação onde vários usuários estão simultaneamente presentes interagindo em um mesmo espaço simulado. Em Ambientes Virtuais Distribuídos (*AVDs*) estes usuários acessam um *AVM* através de computadores interconectados através de uma rede (por exemplo, uma *LAN* ou *WAN*) [5].

Um mundo virtual é tipicamente povoado por objetos animados ou não. Objetos inanimados são representações de artefatos móveis tais como mesas, cadeiras, utensílios, etc. Os objetos animados são chamados de atores, e são controlados por usuários ou por um programa (robôs). Atores podem executar ações no *MV*, como por exemplo modificar a forma ou a posição de objetos inanimados ou interagir com outros atores. Na implementação de um *AV* mantém-se em uma base de dados todas as propriedades correntes de cada objeto (animado ou não), tais como sua posição, velocidade, forma, cor, etc.

1.1 Modelos de Comunicação

A escolha adequada de um modelo de comunicação para um *AVD* é um fator muito importante em seu projeto. É necessário estabelecer um compromisso entre a latência da comunicação e a confiabilidade do sistema, sendo que o primeiro elemento tem influência significativa sobre o tempo de resposta a uma ação do usuário e o segundo tem influência sobre o sincronismo das visões do *MV* entre os usuários do *AVD*. Essencialmente, existem duas alternativas básicas para a implementação de um *AVD* [3].

No modelo centralizado, um servidor centralizado coleta as atualizações referentes aos movimentos e interações de atores controlados por usuários em diferentes computadores, armazena as alterações em uma base de dados centralizada, e envia as atualizações de volta para cada participante, que por sua vez interpreta os dados recebidos. Este modelo, no entanto, não é escalonável e pode apresentar grande

latência na comunicação, uma vez que com a entrada de muitos usuários na simulação, o servidor torna-se um gargalo da comunicação no sistema.

Ao contrário do modelo centralizado, *AVDs* apresentam uma maior escalabilidade. *AVDs* são programas distribuídos, nos quais cada processo (visualizador) controlado por um usuário mantém localmente uma cópia completa da base de dados

A maioria dos trabalhos em *AVDs* adotam a seguinte nomenclatura (figura 1): um usuário está sempre associado a um ator no mundo virtual. Um visualizador é um programa com um módulo para síntese gráfica que representa a visão atual do mundo virtual da perspectiva de um ator. O visualizador implementa a interface com o usuário, rotinas para o gerenciamento do ator associado ao usuário e dos *atores-fantasmas* (correspondentes aos atores dos outros usuários remotos), e rotinas para comunicação com os demais visualizadores. O mundo virtual (*MV*) é representado por estruturas de dados replicadas em cada um dos visualizadores que idealmente deveriam estar perfeitamente sicronizadas. Para isto, é necessário que atualizações do estado do *MV* sejam propagadas para todos os visualizadores que participam de um *AVD*.

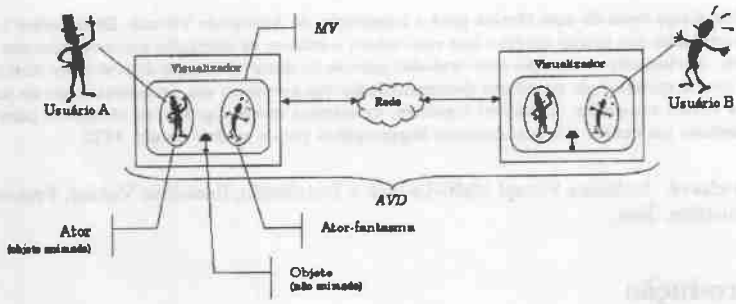

Figura 1: Nomenclatura usada em *AVs*

Para reduzir o número de mensagens enviadas entre os visualizadores, duas técnicas geralmente são empregadas: *multicast* e *Dead Reckoning*.

Multicast é uma difusão de mensagens pela rede, a qual permite que todos os visualizadores de um *AVD* executando na rede recebam cada mensagem enviada. Isto evita que, em uma simulação com *n* usuários, seja necessário estabelecer (*n-1*) conexões ou enviar *n* mensagens a fim de informar cada visualizador associado a um usuário das modificações na base de dados local. A atualização é transmitida uma única vez e todos os visualizadores a recebem praticamente simultaneamente.

Dead Reckoning é uma técnica adicional usada em *AVDs* para reduzir o número de mensagens de atualização. Esta técnica é baseada em uma projeção do movimento futuro de cada ator associado a um usuário. Para reduzir o tráfego de mensagens de atualização, o visualizador de cada usuário envia, além da posição atual de seu ator, um vetor velocidade. Se o ator mantiver seu movimento na direção e sentido indicados por sua velocidade, não há necessidade de enviar qualquer alteração aos demais visualizadores, pois estes são capazes de calcular a posição do ator em movimento a partir da posição anterior e da velocidade atual. Ou seja, com esta técnica, só é necessário difundir uma atualização quando a nova posição de um ator difere significativamente de sua posição projetada. A figura 2 mostra a defasagem entre as trajetórias real e projetada de um ator (representado por um triângulo) em um espaço bidimensional. Podemos observar nesta figura que decorrido um tempo t, a posição real do ator é (7,10), enquanto que a posição projetada para o mesmo tempo *t* é (4,12). Dependendo da especificação do limite do *Desvio Aceitável* (que pode inclusive ser formulado levando em conta o vetor velocidade), um desvio real pode causar a difusão de uma atualização.

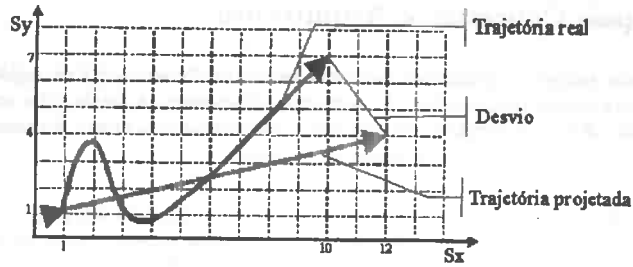

Figura 2: Trajetória real *versus* trajetória projetada

1.2 Motivação

Quando o número de usuários participantes em uma simulação torna-se muito grande, pode ocorrer o problema do aumento considerável de mensagens relativas à atualização dos visualizadores na rede, mesmo utilizando-se comunicação *multicast* e *Dead Reckoning*. Com o objetivo de reduzir o fluxo de mensagens entre os visualizadores em um *AVD*, faz-se necessário limitar o conjunto de visualizadores a serem informados sobre uma atualização usando como critério alguma noção de "proximidade" entre atores. Ou seja, com isto são desconsiderados todos aqueles usuários para os quais a atualização não é relevante.

Uma forma de particionamento do conjunto de atores em grupos (com percepção mútua de movimento) poderia ser aquela determinada pela intersecção das vizinhanças diretas entre os atores. Esta vizinhança pode ser definida diferentemente para cada meio de interação (áudio, vídeo, etc.). No entanto, devido à constante movimentação dos atores, a relação de vizinhança pode variar muito freqüentemente, o que dificulta a sua implementação.

Em nossa abordagem optamos por um agrupamento mais simples, baseado em um particionamento estático do *MV* em regiões (domínios), e no qual uma atualização ocorre somente entre os atores localizados em um mesmo domínio, através de *multicast*. Por exemplo, poderia-se usar tal particionamento em um mundo virtual representando uma casa com vários cômodos. A percepção mútua entre os atores estaria portanto restrita aos atores que se encontram em um mesmo cômodo. Quando estes atores se aproximassem de uma porta ou janela, os mesmos passariam a "perceber" também a movimentação de atores no outro cômodo.

Estamos cientes de que esta forma de agrupamento de atores de acordo com sua pertinência a uma região do *MV* (domínio) pode não ser adequada para determinados ambientes simulados, especialmente quando se deseja simular a interação através de vários meios (áudio, vídeo, etc.). No entanto, um dos objetivos do nosso trabalho foi mostrar que tal particionamento de um *MV* pode ser implementado de modo distribuído de forma que a migração de atores de um domínio para outro seja praticamente imperceptível para o usuário. Para tal, desenvolvemos uma arquitetura distribuída que implementa esta abordagem, a qual denominamos Ambiente Virtual Distribuído com Particionamento Transparente em Domínios - *AVD-PTD* [1].

O restante do artigo está organizado da seguinte forma: na seção 2 apresentamos os principais conceitos e a arquitetura distribuída que implementa o particionamento do *MV*, e na seção 3 descrevemos os protocolos utilizados para implementá-la. Na seção 4 apresentamos uma visão geral de um protótipo que desenvolvemos, e na seção 5 os testes realizados com este protótipo. A seção 6 apresenta um resumo de alguns trabalhos relacionados, e na seção 7 finalizamos com algumas conclusões.

2 Principais Conceitos e Arquitetura

Como mencionado, em nosso trabalho adotamos uma técnica de divisão do *MV* em regiões regulares e fixas (domínios), como forma de reduzir o fluxo de mensagens entre atores. A fim de obter um particionamento do mundo virtual (*MV*), que permita uma migração transparente para atores, foi necessário desenvolver um conjunto de protocolos baseados nos seguintes conceitos:

- **Domínio:** É uma região regular e fixa (por exemplo, retângulo em 2D, cubo em 3D, etc.) que faz parte do *MV*. O conceito de domínio, tal como usado neste trabalho, serve única-e-exclusivamente para agrupar os objetos (atores) que devem ter uma percepção mútua, e que portanto precisam ter as suas visões sincronizadas com as dos demais atores presentes no mesmo domínio.

- **Servidor de Pertinência:** A cada domínio do mundo virtual *MV* está associado um Servidor de Pertinência (*SP*). O *SP* é responsável por gerenciar a informação sobre quais atores estão localizados no seu respectivo domínio. A figura 3 mostra um *MV* particionado em quatro domínios (D_1, D_2, D_3, D_4) com os respectivos *SPs* associados e a lista de atores presentes em cada domínio.

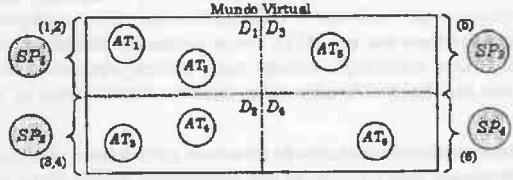

Figura 3: Representação da associação de *SPs* com domínios no *MV*

- **Super-Servidor:** É um elemento central que armazena informações sobre o particionamento estático do *MV* e a associação entre domínios e *SPs*.

- **Servidor de Domínio:** A cada domínio do mundo virtual *MV* está associado também um Servidor de Domínio (*SD*). O papel de servidor de domínio é realizado por um dos visualizadores cujo ator se encontra no domínio ou pelo *SP*. A principal tarefa do *SD* é a de enviar o estado atual do domínio, como por exemplo o conjunto de objetos não animados presentes no domínio, para o visualizador de cada novo ator que entra no domínio. O *SP* determina qual processo exercerá o papel de *SD*.

- **Região de Fronteira e Aura:** A Região de Fronteira (*RF*) é uma faixa no *MV* que percorre os limites da cada domínio. Esta região é usada para detectar quando um ator está mudando de um domínio para outro. Isto é feito, detectando-se uma "colisão" entre a *aura* [4] do ator e a *RF*. Durante o período em que um ator se encontra na RF, o visualizador correspondente envia e recebe atualizações de e para todos os visualizadores de atores presentes tanto no domínio origem quanto no domínio destino. A figura 4 ilustra a aura (nos atores) e a região de fronteira dos domínios do mundo virtual, no momento de uma intersecção entre a aura do ator AT_5 e a *RF* entre D_3 e D_4.

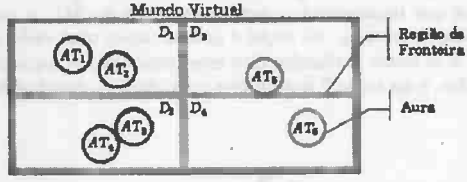

Figura 4: Representação da região de fronteira de domínios e aura de atores

Para a atualização de visões entre atores em um mesmo domínio optou-se por utilizar o *IP-multicast* combinado com a técnica de *Dead Reckoning*. O *IP-multicast* possibilita o envio de um datagrama *IP* para um grupo de *hosts* identificados por um único endereço *IP-multicast* [9]. Em nossa abordagem, cada domínio está associado a um único endereço *IP-multicast*, que é utilizado por todos os visualizadores (com atores no domínio) para a difusão de desvios significativos de posição segundo a técnica de *Dead Reckoning*.

A comunicação entre servidores de pertinência e visualizadores é orientada à conexão (*TCP/IP*), uma vez que ela é usada para troca de dados essenciais para o gerenciamento correto de pertinência.

3 Os Protocolos Distribuídos do AVD-PTD

Nesta seção, apresentamos o conjunto de protocolos usados para a implementação do particionamento transparente do mundo virtual em domínios na arquitetura *AVD-PTD*.

A figura 5 mostra a seqüência de interações entre o Super-servidor (*SServ*), um servidor de pertinência (*SP*), um servidor de domínio (*SD*), o *visualizador* de um novo ator (*NovoAtor*), e demais visualizadores dos atores presentes no domínio. Estas interações ocorrem durante e após a inclusão do *NovoAtor* em um domínio do mundo virtual. A comunicação dos visualizadores entre sí e com o *SP* ocorre através de uma porta *multicast* (*GMC*). O fluxo de mensagens é representado pelas setas com rótulos indicando a seqüência, os tipos e parâmetros das mensagens.

Em linhas gerais, a seqüência (e o tipo) das mensagens (figura 5) é a seguinte:

1. ***NEWCLIENT***: pedido de registro do visualizador de *NovoAtor* (junto ao *SServ*);
2. ***ADDRS_SP***: *SServ* envia referências para endereços (*DomainPort* e *GMC*) do *SP* para o visualizador;
3. ***JOINED***: visualizador se registra no *GMC* e passa a escutar e atualizar sua posição através do mesmo;
4. ***SEND_ADDRS_SD***: visualizador solicita o de envio de referência para a porta *TCP* do *SD*;
5. ***ADDRS_SD***: *SP* envia referência para a porta do *SD*;
6. ***CONEXÃO***: visualizador se conecta ao *SD* para que este possa enviar o estado atual do domínio;
7. ***DOMAIN***: *SD* envia o estado atual do domínio através da conexão *TCP* entre o *SD* e o visualizador do *NovoAtor*.

A seguir, descreveremos os protocolos através de diagramas de eventos no tempo, com indicação dos tipos de mensagens e das referências à portas apresentadas na figura 5.

3.1 Protocolo para Inclusão

O protocolo para inclusão (figura 6) é executado somente quando um ator (*NovoAtor*) entra na simulação do *MV* (em um domínio controlado por *SP*). Neste caso, a tarefa do Super-servidor (*SServ*) é estabelecer a associação entre o *NovoAtor* e o *SP* responsável pelo domínio no qual o *NovoAtor* está ingressando. A partir daí, o visualizador do *NovoAtor* passa a se comunicar diretamente com o *SP* e os visualizadores do *NovoAtor* e dos demais atores presentes no mesmo domínio.

Como se pode verificar na figura 6, inicialmente um pedido de registro é enviado (e_1) pelo visualizador do *NovoAtor* para o *SServ*. Este pedido é feito através de uma mensagem do tipo *NEWCLIENT*, cujo conteúdo é uma referência a uma porta (*Mngt*) do visualizador do *NovoAtor* e sua posição inicial. Ao receber esta, o *SServ* determina, a partir da posição inicial do *NovoAtor* qual é o *SP* responsável pelo domínio no qual o *NovoAtor* deseja entrar (e_2). O *SServ* então envia mensagem (e_3) do tipo *ADDRS_SP*

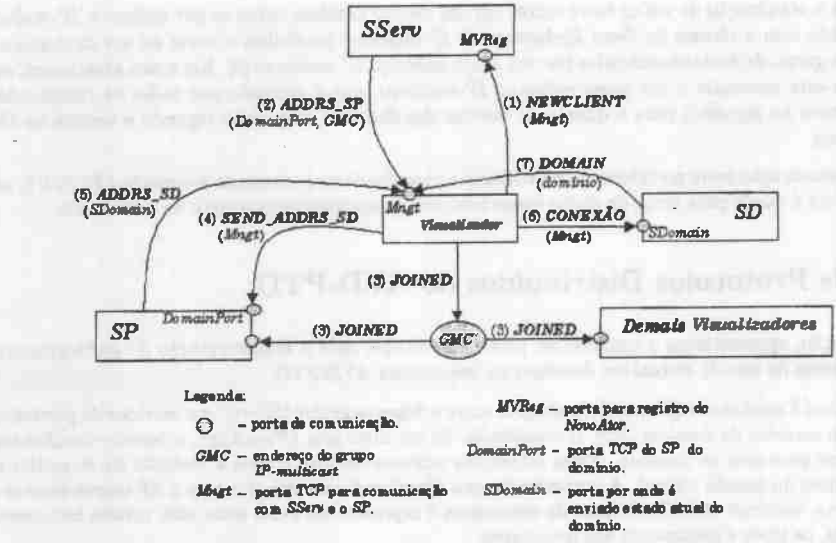

Figura 5: Linhas de comunicação no *MV*

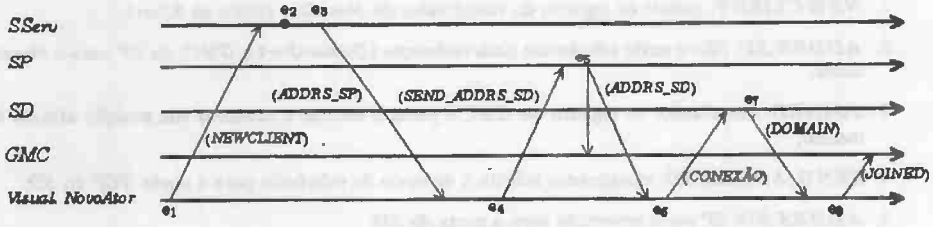

Figura 6: Inclusão de um novo ator

ao visualizador do *NovoAtor*, contendo uma referência para a porta *DomainPort* no *SP* e a porta *IP-multicast* (*GMC*), bem como as coordenadas das fronteiras do domínio em questão. O visualizador então estabelce uma conexão com *DomainPort* e envia (e_4) a mensagem do tipo *SEND_ADDRS_SD* para o *SP*, solicitando o endereço do *SD*. Esta mensagem contém uma referência a uma porta (*Mngt*), a qual o *SP* utilizará para se comunicar com o visualizador. A mensagem de resposta (e_5) do tipo *ADDRS_SD* enviada ao visualizador pode ter um conteúdo diferente, dependendo das seguintes situações:

- Se *NovoAtor* é o único ator no domínio, a mensagem enviada pelo *SP* tem como conteúdo o endereço (*Mngt*) do visualizador de *NovoAtor*, indicando assim que o próprio visualizador será o novo servidor de domínio (*SD*). Neste caso, o visualizador recebe o estado do domínio do próprio *SP*. A informação sobre a identificação do *SD* é guardada pelo *SP* (neste caso, os eventos e_6 e e_7 são desconsiderados).
- Se já existir um *SD*, o *SP* envia mensagem contendo uma referência para a porta *SDomain* do *SD*. O visualizador do *NovoAtor* estabelece então uma conexão com o *SD* (e_6), fazendo com que o *SD* envie mensagem (e_7) do tipo *DOMAIN*, contendo uma cópia do estado atual do domínio correspondente.

Após a obtenção do estado atual do domínio, o *NovoAtor* então anuncia (e_8) a sua entrada no grupo *multicast*, através de uma mensagem do tipo *JOINED*.

3.2 Protocolo para Entrada na Região de Fronteira

Nesta seção, descreveremos as interações que ocorrem quando um ator (*Ator*) já participante da simulação se aproxima da região de fronteira *RF* entre seu domínio *Atual* e um domínio *Novo*. Participam deste protocolo o servidor de pertinência do atual do domínio (*SP_Atual*) e do novo domínio (*SP_Novo*), e o servidor de domínio (*SD*) do novo domínio. A interação entre estes elementos é mostrada na figura 7.

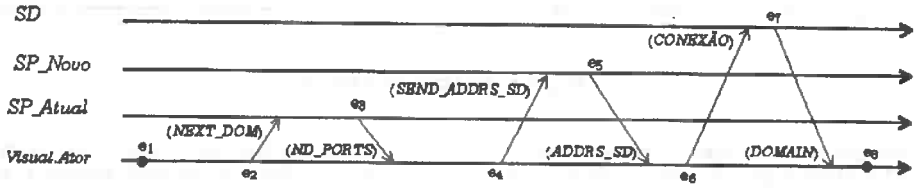

Figura 7: Entrada na Região de Fronteira

Ao detectar (e_1), através de sua aura, que o seu *Ator* está na *RF* entre os domínios controlados pelo *SP_Atual* e pelo *SP_Novo*, o visualizador de *Ator* envia mensagem (e_2) do tipo *NEXT_DOM* para o *SP_Atual*, informando que poderá mudar de domínio. Esta mensagem contém uma referência para a porta (*Mngt*) do visualizador do *Ator*. O *SP_Atual* envia a mensagem (e_3) do tipo *ND_PORTS* para o visualizador do *Ator*, contendo referências para as portas *DomainPort* (do *SP_Novo*) e a porta *multicast GMC*, relativas ao novo domínio. A seguir, o visualizador do *Ator* envia mensagem (e_4) do tipo *SEND_ADDRS_SD* para o *SP_Novo*, para obter o endereço do servidor de domínio (*SD*) do novo domínio. O *SP_Novo* por sua vez envia mensagem (e_5) *ADDRS_SD* com uma referência para a porta *SDomain* do *SD*, através da qual o visualizador do *Ator* receberá o estado atual do novo domínio.

De forma similar ao que ocorre no protocolo de inclusão, caso o novo domínio esteja vazio, o *SP_Novo* assume o papel do *SD*. No evento (e_6), o visualizador do *Ator* estabelece conexão *TCP* com a porta *SDomain* do *SD* e este envia mensagem (e_7) do tipo *DOMAIN*, contendo o estado atualizado do domínio. O visualizador do *Ator* insere (e_8) as informações recebidas do *SD* (cópia do estado do domínio vizinho) em uma lista de domínios vizinhos.

É necessário que o visualizador do *Ator* mantenha a informação sobre domínio vizinhos a uma fronteira, pois o *Ator* pode estar na região de fronteira com vários domínios, e nesta situação é impossível prever para qual domínio o ator de fato migrará. Além disso, a manutenção simultânea das visões de domínios vizinhos é necessária para realizar a migração de forma que o usuário não tenha a sensação de mudança abrupta. As cópias de domínios vizinhos são mantidas atualizadas através da escuta por mensagens *multicast* nas respectivas portas *GMC* (*GMC_Atual*, *GMC_Novo*).

3.3 Protocolo para Mudança de Domínios

A segunda etapa na migração para outro domínio ocorre quando o *Ator* transpõe o limite entre os domínios gerenciados pelo *SP_Atual* e o *SP_Novo*. Nesta transposição, o *Ator* usa também o grupo *multicast* atual (*GMC_Atual*) e o grupo *multicast* do novo domínio (*GMC_Novo*).

O protocolo (mostrado na figura 8) é iniciado quando o *Ator* detecta, através de sua aura (e_1) que está ultrapassando a fronteira entre os domínios. O mesmo então verifica para qual novo domínio está migrando (e_2), e atualiza a informação sobre domínio principal e domínios vizinhos, tais como os respectivos endereços *multicast*, endereços dos servidores de domínio, etc. Ao sair da região de fronteira (de qualquer domínio), o visualizador percorre a lista de domínios vizinhos, descarta os dados correspondentes ao domínio que o *Ator* está deixando para trás e envia mensagem (e_3) do tipo *LEAVE_DOM* para o *GMC_Atual*, fazendo com que ele seja assim excluído do atual grupo *multicast* do domínio antigo.

Os eventos (e_4), (e_5) e (e_6) são similares aos eventos (e_4), (e_5) e (e_8) da seção 3.1. A diferença é que neste caso o visualizador do *Ator* envia a mensagem (e_5) do tipo *SEND_ADDRS_SD* ao *SP_Novo*, com a

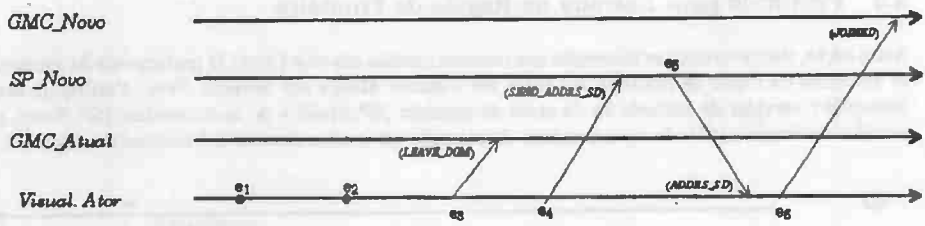

Figura 8: Mudança de domínios

intenção de ser declarado o servidor de domínio do novo domínio, enquanto que no evento (e_5) da seção 3.1, o objetivo era o de obter o endereço do servidor de domínio. Ao passar para o outro domínio, o *Ator* passa a executar o protocolo de entrada na região de fronteira (descrito na seção 3.2).

Estes protocolos usados na migração também prevêem o caso em que um ator está se movimentando para uma fronteira com mais de um domínio vizinho. Esta situação é ilustrada na figura 9, onde um ator está na região de fronteira entre os domínios D_3 e D_2, mas onde D_3 também faz fronteira com D_1. Se só considerássemos a posição do ator na *RF*, provavelmente indicaríamos que o ator está migrando para D_2, quando na verdade a sua trajetória indica uma migração para D_1. No entanto, o protocolo de mudança de domínios implementado em nosso protótipo também leva em conta a direção e sentido do vetor velocidade, fazendo com que o novo domínio real detectado seja D_1.

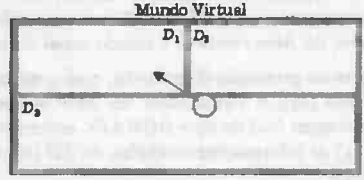

Figura 9: Migração entre domínios

4 O Protótipo do *AVD-PTD*

A fim de avaliar a viabilidade e eficiência dos protocolos usados durante uma migração, incorporamos os mesmos em uma versão distribuída de um jogo simples de asteróides em 2D, para o qual implementamos um particionamento do domínio. Este protótipo é uma extensão de um jogo demonstrativo para Java chamado *Javaroids* [2] e foi implementado em Java 1.2. Em sua versão original, *Javaroids* é um programa 2D mono-usuário, onde o usuário é representado por uma nave que tem a capacidade de alterar a direção de movimentação, acelerar e disparar projéteis contra asteróides e naves inimigas.

A figura 10 mostra uma imagem da interface gráfica do protótipo implementado, onde nota-se uma divisão do mundo virtual em 3 domínios. As linhas em preto representam as regiões de fronteira entre os domínios. No centro da figura podemos observar um ator com sua aura navegando pela região de fronteira dos domínios. Na região de fronteira entre os domínios superiores e o domínio inferior do mundo virtual, observamos três retângulos cinzas menores, que representam a projeção dos atores que estão presentes no domínio inferior. Estas projeções foram incorporadas no protótipo a fim de mostrar que mesmo estando na *RF* de seu atual domínio (superior direito), um ator já acompanha a movimentação dos atores no(s) domínio(s) vizinho(s). Os domínios e regiões de fronteira e as projeções de atores em outros domínios foram representados explicitamente em nosso protótipo apenas para efeito de teste. Em uma implementação real de um *AVD* estes naturalmente não devem estar representados.

Em nosso protótipo o controle de movimentação para cada ator pode ser de dois tipos: trajetórias parametrizadas ou controladas pelo usuário. Os servidores de pertinência são processos executando em diferentes *hosts* de uma rede local. Os visualizadores são processos que vão se conectando e disconectando dos respectivos servidores de pertinência à medida em que os atores (controlados pelo usuário) transpõem fronteiras entre domínios.

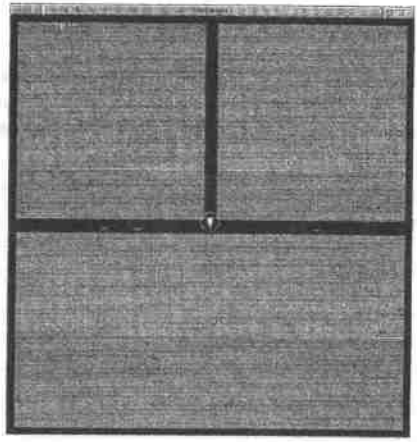

Figura 10: Imagem do *MV* no visualizador do protótipo

5 Testes

Foram feitos diversos testes com o protótipo para avaliar o seu funcionamento e desempenho em diferentes situações de carga, isto é, para um número variável de atores. Um dos objetivos dos testes foi o de obter valores quantitativos que caracterizassem de forma precisa a impressão visual que um usuário do protótipo obtém sobre o tempo de resposta durante uma migração entre domínios.

Um outro objetivo foi o de identificar qual ação executada por um visualizador durante a migração de um ator é a principal responsável pela degradação do desempenho do protótipo. Um terceiro objetivo foi o de medir qual é o impacto que o acesso compartilhado dos servidores (de pertinência e de domínios) pelos visualizadores tem sobre a atualização da visão do mundo virtual em cada visualizador.

Todos os testes foram executados em um conjunto de 23 máquinas executando o sistema operacional *Solaris* (*SunOS 5.7*). O resultado de cada teste foi obtido tomando-se a média de 20 execuções para cada conjunto de robôs, representando os demais atores presentes no mundo virtual. Para efeito de medições, nos dois primeiros testes todos os robôs estavam parados. Isto foi feito para evitar uma influência de outros fatores (como o *multicast* entre os atores) sobre o resultado, uma vez objetivo destes testes foi o de analisar isoladamente o desempenho do protocolo de comunicação para migração entre domínios (seções 3.1 e 3.2). Mediu-se portanto os tempos que um único ator precisa para percorrer a distância durante a passagem pela fronteira, movendo-se em três diferentes velocidades:

- v_1: representa um ator navegando pelo mundo virtual de forma muito lenta. (1 unidade de distância por segundo)
- v_2: velocidade que consideramos comum em jogos deste tipo. (3 unidades/seg.)
- v_3: alta velocidade. Através dela, quisemos analisar o comportamento do protótipo em situações extremas. (8 unidades/seg.)

O primeiro teste (seção 5.1) foi executado para um ator com as três velocidades. Nos outros dois testes (seções 5.2 e 5.3), achamos suficiente efetuar os testes somente com a velocidade v_2, dado que esta reflete uma situação mais comum de uso do jogo. No terceiro teste (seção 5.3), programamos os robôs para que todos eles se movimentassem horizontalmente e verticalmente pelo mundo virtual com a velocidade v_2.

5.1 Efeito visível da migração para o usuário

Este teste foi realizado em duas etapas. Inicialmente, medimos a quantidade de tempo que um ator necessita para percorrer uma distância igual a duas vezes a largura da região de fronteira mais o diâmetro de sua aura. Esta é a distância que um ator, passando perpendicularmente pela fronteira, percorre durante o processo de migração entre domínios.

Depois, tomamos o tempo gasto por um ator que migra de um domínio para o outro, desde o instante em que ele detecta a entrada na região de fronteira do seu domínio atual até que ele deixa a região de fronteira do domínio vizinho. O gráfico na figura 11 apresenta a relação entre os tempos obtidos nas duas etapas para as três velocidades testadas.

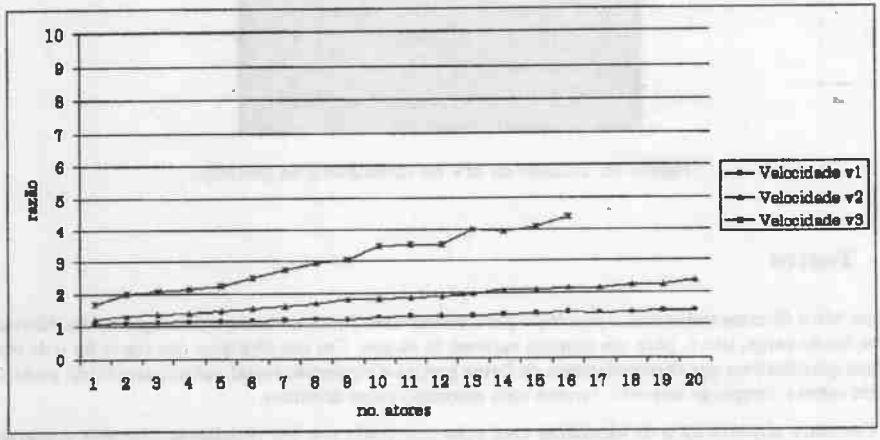

Figura 11: Efeito visual da migração para o usuário

No gráfico podemos observar que, quanto maior a velocidade, maior é a razão entre os tempos (passagem pela fronteira/movimentação dentro do domínio), ou seja, maior é o efeito visual de descontinuidade de movimento notado pelo usuário. Por ser a velocidade v_2 uma velocidade normal neste tipo de jogos, consideramos que o atraso de um pouco mais que 100% é razoavelmente pequeno tendo em vista todas as ações envolvidas na migração entre domínios. Visualmente, para o usuário isto representa uma pequena parada na animação da nave por ele controlada.

5.2 Tempo gasto por ações executadas durante a migração

O objetivo deste teste foi o de identificar quais as ações executadas por um visualizador têm maior contribuição para o atraso na atualização da visão do usuário. Por estarem diretamente relacionados aos protocolos usados na migração as seguintes ações são as mais relevantes:

1. **Pedido ao servidor de pertinência (*SP*)**: tempo para que o visualizador do ator envie uma mensagem ao *SP*, pedindo a lista de domínios vizinhos relativos à região de fronteira no qual o ator entrou.

2. **Resposta do *SP***: tempo de espera pela resposta do *SP* ao pedido feito no item 1.

3. **Recebimento da lista de vizinhos**: tempo de estabelecimento de uma conexão *TCP* com *SP*, e recebimento dos dados relativos aos servidores de domínios (*SDs*), correspondentes aos domínios vizinhos.

4. **Conexão com servidor de domínio (*SD*)**: tempo de estabelecimento de uma conexão *TCP* com *SD*, por onde será enviado o estado corrente (por exemplo, objetos inanimados) do respectivo domínio.

5. **Recebimento do domínio**: tempo gasto para o recebimento completo do estado do domínio.

6. ***Join Group***: tempo de entrada em um novo grupo *multicast*.

7. ***Leave Group***: tempo necessário para sair dos grupos *multicast* (ao sair da região de fronteira).

Através do gráfico na figura 12, pode-se verificar que a ação que mais contribui para o atraso na atualização da visão do usuário é o recebimento dos objetos contidos no domínio para o qual o ator está migrando. Isto se deve ao fato de que nesta versão do protótipo transferimos todo o conjunto de objetos, sem qualquer preocupação em restringir a quantidade de dados trocada entre o servidor de domínio e o visualizador do ator.

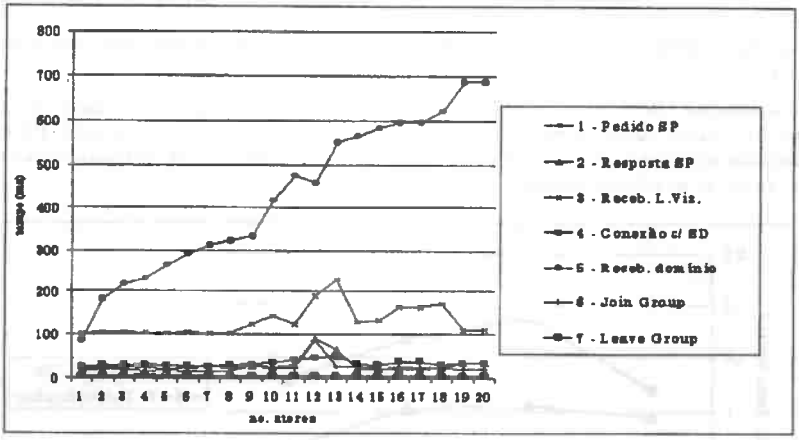

Figura 12: Tempo gasto por ações durante migração

5.3 Influência do compartilhamento dos servidores pelos visualizadores

O objetivo deste teste foi o de avaliar o impacto que o compartilhamento dos servidores de pertinência e de domínios pelos visualizadores têm sobre a atualização da visão do mundo virtual pelo usuário. Neste teste fizemos medições similares às do teste anterior, mas agora para uma situação mais realista, onde todos os atores estão em movimento pelo mundo virtual. Basicamente, fizemos medições similares às do segundo teste para 2, 5, 8 e 12 atores no mundo virtual, cada um com velocidade v_2 e com os robôs movimentando-se horizontalmente e verticalmente pelo mundo virtual. Neste teste percebe-se que o recebimento de domínios e da lista de vizinhos tem uma contribuição ainda maior sobre o atraso causado durante a migração.

Na figura 14 apresentamos a razão entre o tempo gasto pelas ações durante o processo de migração descritas no segundo e terceiro testes. Comparando-se os valores obtidos no teste 5.2 e 5.3, percebe-se que o compartilhamento dos servidores de pertinência e de domínios pelos demais atores praticamente

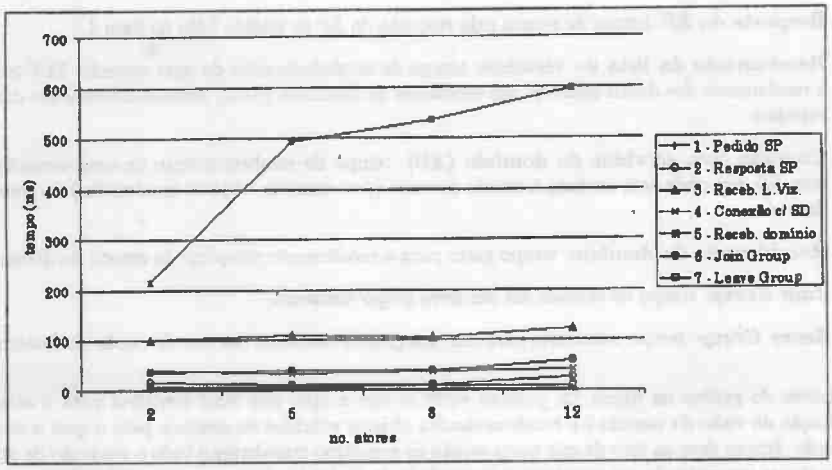

Figura 13: Tempo gasto por ações com robôs em movimento

só tem influência sobre o atraso no recebimento dos objetos do domínio, e que nestes itens a razão está entre 1 e 2.

Deve-se observar que testes envolvendo robôs em movimento estão sujeitos a grandes variações que dependem da situação particular em que os atores (robôs) se encontram em cada momento. Por exemplo, se a maioria dos robôs por acaso estiver migrando para um mesmo domínio, pode-se esperar uma contenção maior no acesso ao servidor de domínio.

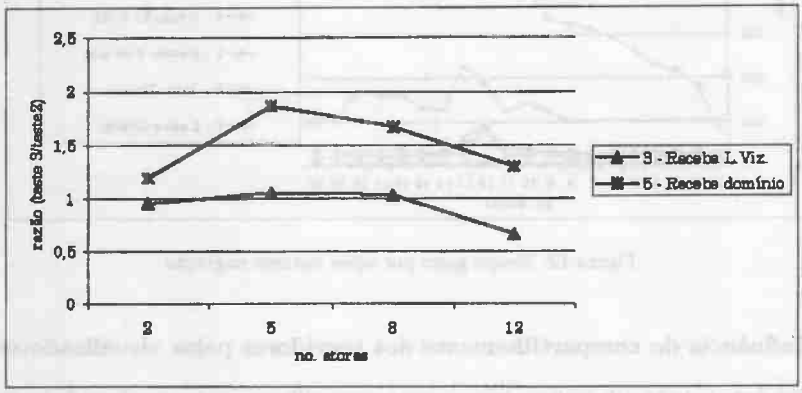

Figura 14: Razão entre tempo gasto pelas ações no segundo e terceiro testes

Com este conjunto de testes pudemos verificar que, de um modo geral, os protocolos apresentados na seção 3 não causam uma degradação muito perceptível na visão do usuário. Concluímos ainda que o fator que mais contribui para o atraso na atualização da visão do usuário é o tempo de espera pelo envio de domínios e que este cresce rapidamente quando se tem muitos atores participando de uma simulação.

Uma forma de diminuir o *overhead* pelo recebimento de um domínio seria reduzir a quantidade de informações enviadas. Por exemplo, cada visualizador poderia armazenar localmente todas as informações e características estáticas relativas a todos os domínios no mundo virtual (tais como, topografia, "objetos"

virtuais nos domínios, etc.) e ao invés de enviar todos os atributos associados aos atores no domínio corrente, poderia-se enviar somente uma lista com suas respectivas posições e velocidades, uma vez que as demais informações já estariam previamente armazenadas em cada visualizador.

Finalmente é importante ressaltar que sabemos que os testes realizados só representam um passo inicial na avaliação dos protocolos e não permitem tirar conclusões de caráter mais geral. Na verdade, o principal objetivo dos mesmos foi o de determinar os principais custos causados pelos protocolos e identificar possíveis otimizações nos mesmos.

6 Trabalhos Relacionados

Vários *AVDs* têm sido propostos e implementados nos últimos anos, tais como *MASSIVE* [4], *DIVE* [5], *AGORA* [6], *Diamont Park and Spline* [10], *PARADISE* [8], etc. Apresentamos a seguir, três destes trabalhos, *MASSIVE*, *DIVE* e *AGORA*, devido a algumas semelhanças com o nosso trabalho.

6.1 MASSIVE

MASSIVE é um sistema experimental de Realidade Virtual Distribuída [4] desenvolvido por Chris Greenhalgh e Steve Benford no Departamento de Ciência da Computação da Universidade de Nottingham. *MASSIVE* ("Modelo, Arquitetura e Sistema para Interação Espacial em Ambientes Virtuais") dá uma ênfase particular a ambientes virtuais multi-usuários de grande escala.

Em *MASSIVE*, cada objeto em um mundo virtual possui uma aura específica de interação com outros objetos, que depende da mídia utilizada (por exemplo, gráficos, textos, áudio, etc.). Esta aura define uma vizinhança espacial na qual é possível a interação com outros objetos e assim determina quais outros objetos um determinado objeto poderá perceber visualmente, auditivamente, por meio de colisão, etc. Define-se que a interação entre dois objetos é possível somente quando existe uma intersecção de suas auras ou um objeto se encontrar dentro da aura do outro (colisão de auras).

Segundo os autores, o emprego de auras facilita a implementação de mundos virtuais com muitos usuários, dado que o número de interações (entre objetos) a serem tratadas é bem menor, pois só é necessária para os objetos próximos uns dos outros. A interação entre objetos neste ambiente é implementada através de *negociação espacial*, uma combinação da técnica de detecção de colisões com o conceito de serviço de nomes baseado em atributos (*trader*) [7].

A passagem de objetos de um mundo para outro é feita através de portais (*gateways*), que são pontos de passagem entre diferentes mundos, ou diferentes localizações dentro do mesmo mundo. Portais implementam uma interface especial com o usuário, através da qual o mesmo pode escolher se deseja passar para o outro mundo. Em *MASSIVE*, o problema da escalabilidade é aliviado através da distribuição da responsabilidade pela detecção de colisão de auras para diferentes *Gerenciadores de Auras*. No entanto, caso um número excessivo de objetos estiver em um mesmo mundo, *MASSIVE* poderá apresentar problemas de contenção no Gerenciador de Auras do respectivo mundo.

6.2 DIVE

DIVE (Ambiente Virtual Distribuído Interativo) [5] é uma plataforma experimental para o desenvolvimento de ambientes virtuais em três dimensões, interfaces com o usuário e aplicações baseadas em ambientes compartilhados. *DIVE* é voltado principalmente para aplicações multi-usuário, onde vários participantes interagem entre si através da rede *Internet*. *DIVE* foi desenvolvido por Olot Hagsand e Christer Carlsson do Instituto de Ciência da Computação da Suécia.

DIVE é baseado em uma arquitetura de comunicação direta entre pares de clientes (*peer-to-peer*) sem servidor centralizado, onde os pares se comunicam através de *multicast* confiável e não confiável (baseado em *IP-multicast*). Todo visualizador *DIVE* armazena localmente uma cópia completa do mundo virtual com seu estado atual. O controle de concorrência e consistência de estado (na forma de objetos comuns)

é feito através de protocolos de comunicação de grupos que implementam Replicação Ativa [11], onde a réplica é mantida consistente com as demais por meio de um protocolo de *multicast* atômico e ordenação total de mensagens. Através deste protocolo todas as réplicas executam exatamente a mesma seqüência de atualizações.

6.3 AGORA

AGORA [6] é um projeto dos Laboratórios Fujitsu do Japão. O objetivo do projeto foi desenvolver um sistema de comunicação flexível e generalizado combinando o acesso a *sites* na *World Wide Web* (*WWW*), e a comunicação em tempo real, como por exemplo salas de bate-papo. *AGORA* apresenta uma arquitetura para um ambiente virtual tridimensional, que emprega uma técnica de divisão e gerenciamento do espaço virtual, um modelo de controle do fluxo de mensagens de atualização da visão, e uma técnica de compartilhamento do espaço virtual entre clientes (*space-sharing*).Uma característica importante do modelo de mensagens do *AGORA* é que a troca de mensagens só ocorre entre clientes localizados em um mesmo espaço comum, denominado região.

AGORA utiliza a técnica de *space-sharing* (*SSh*) para manter cópias locais em clientes consistentes. Através do *SSh* é possível fazer uma notificação imediata de qualquer evento em um espaço virtual para todos os clientes. *AGORA* não usa o conceito de aura, como nos sistemas *MASSIVE* e *DIVE*, por assumir que em uma comunidade virtual, ao contrário de um *AV*, o número de clientes participantes é pequeno e a chance de colisão entre os mesmos é bastante remota. Em vez disso, o *AGORA* aplica a técnica de particionamento em regiões, que são áreas de tamanho fixo, ao invés da técnica dinâmica de aura, por considerar que é muito difícil estabelecer o tamanho ideal de auras de forma a não gerar uma sobrecarga excessiva na detecção de colisões.

6.4 Comparação entre os Sistemas

Nesta seção apresentamos uma comparação entre os sistemas *MASSIVE*, *DIVE*, *AGORA* e nossa abordagem (*AVD-PTD*), com relação a algumas de suas características relativas à percepção mútua entre atores, sua migração, a forma de implementação e a sincronização entre visualizadores (figura 15).

	MASSIVE	DIVE	AGORA	*AVD-PTD*
Diversos meios de interação	Sim	Sim	Não	Não
Detecção de colisão	Sim	Sim	Não	Sim
Particionamento/gerenciamento de mundos	Sim	Não	Sim	Sim
Migração transparente entre regiões	Não	-	Não	Sim
Estado do mundo virtual	Distribuído	Distribuído	Servidor *I-VRML*	Distribuído
Percepção mútua entre atores	Colisão de Auras	Colisão de Auras	Localização	Localização
Sincronização de visão	-	*Multicast* para todos	*Multicast* Seletivo	*IP-multicast*

Figura 15: Comparação entre *MASSIVE*, *DIVE*, *AGORA* e *AVD-PTD*

Enquanto nos sistemas *MASSIVE* e *AGORA* visualizadores podem utilizar diversos meios de comunicação (tais como áudio, texto, etc.) para estabelecer uma comunicação entre atores, em nossa abordagem (e no sistema *AGORA*) tal facilidade não existe.

O *AGORA* implementa o mundo virtual através de um servidor *I-VRML* centralizado e o estado do mundo virtual fica distribuído nos clientes (usuários). No caso do *MASSIVE*, *DIVE* e em nossa abordagem o estado do mundo virtual é distribuído.

No sistema *MASSIVE* e em nossa abordagem o objetivo do particionamento e gerenciamento descentralizado de mundos é garantir a escalabilidade dos mesmos. No *AGORA*, apesar da presença de um servidor centralizado (*I-VRML*), também existe um particionamento do mundo virtual em regiões como forma de garantir a escalabilidade do sistema. Em *DIVE* não há particionamento do mundo virtual e por isto o sistema é pouco escalonável.

No *MASSIVE*, a passagem de um mundo para outro é explicita, pois a presença de um portal de transferência para outro mundo é representado explicitamente e o usuário tem a possibilidade de escolher se deseja ou não usar o portal para se transferir para outro mundo. No *AGORA*, apesar de não requerer uma ação do usuário, a migração também não é transparente uma vez que neste ambiente não há uma continuidade entre as visões nova e antiga.

A percepção mútua entre atores em *MASSIVE* e *DIVE* ocorre somente quando há colisão de auras específicas para cada meio de interação. No sistema *AGORA* e em nossa abordagem, a percepção mútua só é possível entre atores presentes em uma mesma região. Em nossa abordagem, no entanto, quando um ator se encontra na região de fronteira entre dois domínios, este "percebe" a presença dos outros objetos em ambos os domínios que compartilham a fronteira. Desta forma, é possível realizar uma passagem "suave" de um domínio para outro.

A sincronização do estado do mundo virtual nos visualizadores em *DIVE* ocorre através de mensagens *multicast* para todos os atores presentes no mundo. No *AGORA* a atualização da visão só ocorre entre clientes dentro de uma mesma região. Em nossa abordagem, todos os atores dentro de um mesmo domínio mantêm suas visões sincronizadas, através da técnica de *Dead Reckoning* e do envio de mensagens *multicast* de atualização.

7 Conclusão

Neste trabalho, abordamos técnicas para a construção de *AVDs* compartilhados por muitos usuários, buscando novas formas de otimização da comunicação e redução do volume e da freqüência das atualizações entre os elementos distribuídos.

A principal contribuição deste trabalho foi o desenvolvimento de protocolos distribuídos em uma arquitetura de servidores descentralizados que permitem um particionamento de um mundo virtual em regiões (domínios) regulares, e de tal forma que o usuário não perceba a migração de um ator de um domínio para outro. O particionamento teve como principal motivação a redução do número de atores que precisam ser informados mutuamente sobre as movimentações de atores dentro do mundo virtual, reduzindo assim o tráfego de mensagens de atualização no sistema.

A fim de mostrar a viabilidade de nossa abordagem, implementamos um protótipo, que é extensão de um jogo tipo Asteroids, chamado *Javaroids* [2]. A partir dos resultados de vários testes e medições de tempo usando este protótipo, pudemos comprovar que, apesar da complexidade dos protocolos envolvidos em uma migração, na maioria dos casos o efeito visual para o usuário é somente uma pequena retração na animação dos movimentos dos atores. Além disso, pudemos verificar que a maior parcela de atraso é devido à transmissão do estado do novo domínio para o visualizador correspondente.

Consideramos este trabalho como um primeiro passo no estudo de formas de particionamentos transparente de Ambientes Virtuais Distribuídos. Naturalmente, existiam muitos outros problemas e questões interessantes a serem abordadas em futuros trabalhos.

Em particular, sabemos que uma divisão do *MV* em regiões fixas (domínios) não garante que a todo instante o conjunto de atores estará distribuído de forma balanceada nos domínios. Pode muito bem acontecer que por acaso, ou devido a algum "evento virtual" ocorra uma migração em massa para um determinado domínio. Nesta situação, o ideal seria que fosse possível fazer um reparticionamento dinâmico de um *MV* em domínios.

O outro grande problema diz respeito à confiabilidade da arquitetura. Uma vez que cada domínio do *MV* é gerenciado por um servidor de pertinência (*SP*), uma eventual falha de um dos servidores representa um grande problema para um *AVD* utilizando esta técnica de particionamento. No entanto, a

falha de um servidor não impede que os visualizadores cujos atores já se encontram no domínio continuem interagindo, através da porta *multicast*. O problema maior está na passagem de atores de um domínio para outro, já que o *SP* com falha será a) incapaz de indicar qual deverá ser o novo *SP* a ser contactado pelo visualizador de um ator que está saindo de sua região e b) incapaz de fornecer a porta *multicast* usada em seu domínio, para que o visualizador de um novo ator ingressando no domínio, possa interagir com os demais visualizadores dos atores presentes no domínio. Ou seja, a falha de um *SP* deixaria os atores presentes no domínio correspondente isolados dos demais atores no *MV*.

Referências

[1] Marcos Alves. Particionamento Transparente de Ambientes Virtuais Distribuídos. Master's thesis, IME, University of São Paulo, R. do Matão, 1010. CEP:05508-900. São Paulo, Brazil, apr 2000.

[2] J. Fan and C. Tenitchi E. Ries. *Black Art of JAVA Game Programming*, chapter Building the JAVAroids Game. Waite Group Press, 1996.

[3] R. Gossweiler, M. L. Keller R. J. Laferriere, and R. Pausch. An Introductory Tutorial for Developing Multi-User Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 3(4):255–264, dec 1994.

[4] Chris Greenhalgh and Steve Benford. MASSIVE: a Distributed Virtual Reality System Incoroporating Spatial Trading. In *Proc. IEEE 15th Int. Conference on Distributed Computing Systems*, 1995.

[5] O. Hagsand. Interactive MultiUser VEs in the DIVE System. *IEEE Multimedia*, 3(1), 1996. http://www.sics.se/dce/dive/dive.html.

[6] H. Harada, A. Matsui N. Kawagushi, and T. Ohno. Space-sharing Architecture for three-dimensional virtual community. *IOP/IEE Distributed Systems Engineering*, 5:101–106, 1998.

[7] Castle Hill. *The Advanced Network Systems Architecture (ANSA) Reference Manual*, chapter Architecture Project Management. Cambridge, England, 1989.

[8] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. *Proc. SIGCOMM'95*, 25(4):328–341, oct 1995. Published as Computer Communications Review.

[9] M. R. Macedonia and D. P. Brutzman. MBone provides audio and video across the Internet. *IEEE Computer*, apr 1994.

[10] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S.McKeown, T. Nitta, I.Sterns, & W. Yerazunis. Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction and Runtime Modifiability. Technical report, A Mitsubishi Eletric Research Laboratory (MERL), nov 1996. TR-96-02a.

[11] Fred Shneider. Replication management using the state-machine approach. In Sape Mullender, editor, *Distributed Systems*, chapter 7. Addison-Wesley, 2nd. edition, 1993.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1997 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail (mac@ime.usp.br).

Carlos Eduardo Ferreira, C. C. de Souza e Yoshiko Wakabayashi
REARRANGEMENT OF DNA FRAGMENTS: A BRANCH-AND-CUT ALGORITHM
RT-MAC-9701, janeiro de 1997, 24 pp.

Marcelo Finger
NOTES ON THE LOGICAL RECONSTRUCTION OF TEMPORAL DATABASES
RT-MAC-9702, março de 1997, 36 pp.

Flávio S. Corrêa da Silva, Wamberto W. Vasconcelos e David Robertson
COOPERATION BETWEEN KNOWLEDGE BASED SYSTEMS
RT-MAC-9703, abril de 1997, 18 pp.

Junior Barrera, Gerald Jean Francis Banon, Roberto de Alencar Lotufo, Roberto Hirata Junior
MMACH: A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9704, maio de 1997, 67 pp.

Julio Michael Stern e Cibele Dunder
PORTFÓLIOS EFICIENTES INCLUINDO OPÇÕES
RT-MAC-9705, maio de 1997, 29 pp.

Junior Barrera e Ronaldo Fumio Hashimoto
COMPACT REPRESENTATION OF W-OPERATORS
RT-MAC-9706, julho de 1997, 13 pp.

Dilma M. Silva e Markus Endler
CONFIGURAÇÃO DINÂMICA DE SISTEMAS
RT-MAC-9707, agosto de 1997, 35 pp

Kenji Koyama e Routo Terada
AN AUGMENTED FAMILY OF CRYPTOGRAPHIC PARITY CIRCUITS
RT-MAC-9708, setembro de 1997, 15 pp

Routo Terada e Jorge Nakahara Jr.
LINEAR AND DIFFERENTIAL CRYPTANALYSIS OF FEAL-N WITH SWAPPING
RT-MAC-9709, setembro de 1997, 16 pp

Flávio S. Corrêa da Silva e Yara M. Michelacci
MAKING OF AN INTELLIGENT TUTORING SYSTEM (OR METHODOLOGICAL ISSUES OF ARTIFICIAL INTELLIGENCE RESEARCH BY EXAMPLE)
RT-MAC-9710, outubro de 1997, 16 pp.

Marcelo Finger
COMPUTING LIST COMBINATOR SOLUTIONS FOR STRUCTURAL EQUATIONS
RT-MAC-9711, outubro de 1997, 22 pp.

Maria Angela Gurgel and E.M.Rodrigues
THE F-FACTOR PROBLEM
RT-MAC-9712, dezembro de 1997, 22 pp.

Perry R. James, Markus Endler, Marie-Claude Gaudel
DEVELOPMENT OF AN ATOMIC-BROADCAST PROTOCOL USING LOTOS
RT-MAC-9713, dezembro de 1997, 27 pp.

Carlos Eduardo Ferreira and Marko Loparic
A BRANCH-AND-CUT ALGORITHM FOR A VEHICLE ROUTING PROBLEM WITH CAPACITY AND TIME CONSTRAINTS
RT-MAC-9714, dezembro de 1997, 20 pp

Nami Kobayashi
A HIERARCHY FOR THE RECOGNIZABLE M-SUBSETS
RT-MAC-9715, dezembro de 1997, 47 pp.

Flávio Soares Corrêa da Silva e Daniela Vasconcelos Carbogim
A TWO-SORTED INTERPRETATION FOR ANNOTATED LOGIC
RT-MAC-9801, fevereiro de 1998, 17 pp.

Flávio Soares Corrêa da Silva, Wamberto Weber Vasconcelos, Jaume Agustí, David Robertson e Ana Cristina V. de Melo.
WHY ONTOLOGIES ARE NOT ENOUGH FOR KNOWLEDGE SHARING
RT-MAC-9802, outubro de 1998, 15 pp.

J. C.de Pina e J. Soares
ON THE INTEGER CONE OF THE BASES OF A MATROID
RT-MAC-9803, novembro de 1998, 16 pp.

K. Okuda and S.W.Song
REVISITING HAMILTONIAM DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9804, dezembro 1998, 17pp.

Markus Endler
AGENTES MÓVEIS: UM TUTORIAL
RT-MAC-9805, dezembro 1998, 19pp.

Carlos Alberto de Bragança Pereira, Fabio Nakano e Julio Michael Stern
A DYNAMIC SOFTWARE CERTIFICATION AN VERIFICATION PROCEDURE
RT-MAC-9901, março 1999, 21pp.

Carlos E. Ferreira e Dilma M. Silva
BCC DA USP: UM NOVO CURSO PARA OS DESAFIOS DO NOVO MILÊNIO
RT-MAC-9902, abril 1999, 12pp.

Ronaldo Fumio Hashimoto and Junior Barrera
A SIMPLE ALGORITHM FOR DECOMPOSING CONVEX STRUCTURING ELEMENTS
RT-MAC-9903, abril 1999, 24 pp.

Jorge Euler, Maria do Carmo Noronha e Dilma Menezes da Silva
ESTUDO DE CASO: DESEMPENHO DEFICIENTE DO SISTEMA OPERACIONAL LINUX PARA CARGA MISTA DE APLICAÇÕES.
RT-MAC-9904, maio 1999, 27 pp.

Carlos Humes Junior e Paulo José da Silva e Silva
AN INEXACT CLASSICAL PROXIMAL POINT ALGORITHM VIEWED AS DESCENT METHOD IN THE OPTIMIZATION CASE
RT-MAC-9905, maio 1999, pp.

Carlos Humes Junior and Paulo José da Silva e Silva
STRICT CONVEX REGULARIZATIONS, PROXIMAL POINTS AND AUGMENTED LAGRANGIANS
RT-MAC-9906, maio 1999, 21 pp.

Ronaldo Fumio Hashimoto, Junior Barrera, Carlos Eduardo Ferreira
A COMBINATORIAL OPTIMIZATION TECHNIQUE FOR THE SEQUENTIAL DECOMPOSITION OF EROSIONS AND DILATIONS
RT-MAC-9907, maio 1999, 30 pp.

Carlos Humes Junior and Marcelo Queiroz
ON THE PROJECTED PAIRWISE MULTICOMMODITY FLOW POLYHEDRON
RT-MAC-9908, maio 1999, 18 pp.

Carlos Humes Junior and Marcelo Queiroz
TWO HEURISTICS FOR THE CONTINUOUS CAPACITY AND FLOW ASSIGNMENT GLOBAL OPTIMIZATION
RT-MAC-9909, maio 1999, 32 pp.

Carlos Humes Junior and Paulo José da Silva e Silva
AN INEXACT CLASSICAL PROXIMAL POINT ALGORITHM VIEWED AS A DESCENT METHOD IN THE OPTIMIZATION CASE
RT-MAC-9910, julho 1999, 13 pp.

Markus Endler and Dilma M. Silva and Kunio Okuda
A RELIABLE CONNECTIONLESS PROTOCOL FOR MOBILE CLIENTS
RT-MAC-9911, setembro 1999, 17 pp.

David Robertson, Fávio S. Corrêa da Silva, Jaume Agustí and Wamberto W. Vasconcelos
A LIGHTWEIGHT CAPABILITY COMMUNICATION MECHANISM
RT-MAC-9912, novembro 1999, 14 pp.

Flávio S. Corrêa da Silva, Jaume Agustí, Roberto Cássio de Araújo and Ana Cristina V. de Melo
KNOWLEDGE SHARING BETWEEN A PROBABILISTIC LOGIC AND BAYESIAN BELIEF NETWORKS
RT-MAC-9913, novembro 1999, 13 pp.

Ronaldo F. Hashimoto, Junior Barrera and Edward R. Dougherty
FINDING SOLUTIONS FOR THE DILATION FACTORIZATION EQUATION
RT-MAC-9914, novembro 1999, 20 pp.

Marcelo Finger and WanbertoVasconcelos
SHARING RESOURCE-SENSITIVE KNOWLEDGE USING COMBINATOR LOGICS
RT- MAC-2000-01, março 2000, 13pp.

Marcos Alves e Markus Endler
PARTICIONAMENTO TRANSPARENTE DE AMBIENTES VIRTUAIS DISTRIBUÍDOS
RT- MAC-2000-02, abril 2000, 21pp.