# NETMAZE: UM JOGO DE AÇÃO MULTIMÍDIA DISTRIBUÍDO

Charles A. G. Madeira, Alessandro C. M. de Araújo, Hendrik T. Macedo, Rogério de C. Andrade, Dave A. T. Cavalcanti, Carlos A. G. Ferraz, Geber L. Ramalho

Universidade Federal de Pernambuco
Centro de Informática
Caixa Postal 7851, CEP 50732-970
Recife – PE – Brasil

E-mail: {cagm, acma, htm, rogerio, datc, cagf, glr}@cin.ufpe.br

**RESUMO**

Atualmente aplicações multimídias distribuídas vêm sendo bastante difundidas, ao mesmo tempo em que seu desenvolvimento apresenta problemas complexos. Este artigo apresenta um jogo de ação multi-usuário distribuído desenvolvido em linguagem Java sobre a plataforma de distribuição CORBA. Discutimos os problemas relevantes a este tipo de aplicação, tais como consistência da interface gráfica, distribuição, inteligência dos personagens, sincronização e apresentação de mídias, sincronização de eventos e tráfego de rede. Apresentamos as soluções que empregamos nos problemas levantados, assim como uma avaliação das alternativas e resultados alcançados utilizando três diferentes arquiteturas de comunicação para jogos distribuídos.

**Palavras-chave:** aplicações distribuídas, sincronização, jogos.

**ABSTRACT**

Nowadays, we can note a big growth on the development of distributed multimedia applications despite of some complex issues that are intrinsic to this development. This paper presents a distributed multi-user action game in Java on a CORBA compatible platform. We discuss the problems related to this type of application, as graphical interface consistence, distribution, personages intelligence, synchronisation and media presentation, event synchronisation and network traffic. We present our solutions to these problems, and an evaluation of the alternatives and results we got using three different communication architectures to distributed games.
**Keywords:** distributed applications, synchronisation, games.

## 1 INTRODUÇÃO

O desenvolvimento de aplicações distribuídas vem crescendo muito hoje em dia, grande parte impulsionado pela necessidade do compartilhamento de recursos, mas principalmente, pela busca de uma padronização de desenvolvimento que ajude na integração de software e hardware de fabricantes distintos e que muitas vezes são fatalmente incompatíveis entre si.

Acrescentando o item multimídia neste contexto, temos um caso particular de aplicações que vem despertando muito o interesse tanto no meio industrial quanto no

acadêmico. É bem verdade que para o desenvolvimento de boas aplicações multimídia distribuídas, faz-se necessário uma capacidade de transmissão de rede muito boa para lidar com o tráfego de sons e imagens e garantir a consistência dos mesmos. Mas a solução não depende apenas da evolução dos meios de transmissão, mas também de técnicas de programação para lidar com o processo.

Este artigo identifica os principais problemas relacionados ao desenvolvimento de aplicações multimídia distribuídas utilizando uma plataforma CORBA e apresenta propostas para solução dos mesmos. Para avaliar estas propostas de solução, bem como a utilização do padrão CORBA, procuramos desenvolver uma aplicação, como um estudo de caso, a qual apresentasse os problemas de consistência de dados, sincronização, interface de usuário, inteligência artificial, consumo de banda passante, tráfego de rede e interoperabilidade.

Identificamos um jogo de ação multimídia distribuído como uma aplicação crítica em relação aos problemas relacionados acima e apropriada para um estudo de caso. O mesmo foi implementado de três formas diferentes, relativamente à organização dos objetos envolvidos e comunicação entre eles. A engenharia do jogo é baseada em um servidor que controla o estado do jogo e clientes que modificam este estado. Pretendeu-se avaliar qual a melhor maneira de implementação para esse tipo de aplicação de forma que se garantisse uma boa sincronização som/imagem, consistência de interface dos usuários do jogo, e nível satisfatório de consumo de banda passante. São apresentadas soluções para esta sincronização som/imagem, construção de uma interface bastante amigável com o usuário e comunicação entre servidor e clientes.

Na seção 2 é descrito todo o funcionamento do jogo. Os problemas referentes ao desenvolvimento de um jogo de ação distribuído estão relacionados na seção 3. Em seguida, soluções para esses problemas são propostas na seção 4. E, finalmente, algumas conclusões sobre esse trabalho e propostas futuras são apresentadas na seção 5.

## 2 JOGOS DE AÇÃO DISTRIBUÍDOS: O PROJETO NETMAZE

O objetivo era desenvolver um jogo multi-usuário distribuído sobre uma plataforma de distribuição CORBA[7] que atendesse bem às características básicas de distribuição[16] e ao mesmo tempo apresentasse uma boa jogabilidade.

Denominamos o jogo de *NetMaze*. Seu funcionamento consiste de personagens, caça e caçadores, que se movimentam dentro de um labirinto em duas dimensões. Os caçadores têm como objetivo procurar a caça no labirinto com a intenção de capturá-la, enquanto que a caça, tem como objetivo manter-se o maior tempo possível sem ser pega. Assim, os usuários ao se conectarem ao jogo podem aparecer na tela como caça ou como caçador dependendo do seu estado no momento da conexão. O estado do jogo é representado pelas posições dos jogadores na tela, pelo formato do labirinto atual e pelos eventos que ocorrem, como colisões entre caçadores, de caçadores com a caça ou de jogadores com as paredes do labirinto. Os jogadores podem se movimentar nas direções horizontal ou vertical.

O jogo possui cinco fases, sendo cada uma representada por um labirinto e por um fundo de tela diferentes. A mudança de fase ocorre quando a caça é capturada. Nesse momento, os clientes mostram na tela uma imagem comum referente à mudança de fase, tocam os sons específicos para a caça ou caçadores e reiniciam o jogo passando para a fase seguinte, onde outro jogador passa a ser a caça. A figura 1 mostra um momento do jogo NetMaze com uma caça e um caçador.

A arquitetura geral do jogo consiste de uma máquina que armazena o programa servidor do jogo. Ou seja, aquele responsável por controlar o estado do jogo: posições de todos os jogadores na tela, evitar sobreposição de imagens quando da ocorrência de colisões, verificar que tipo de colisão ocorreu e com que jogador(es) especificamente, para enviar mensagens de disparo de sons respectivos à colisão em cada máquina cliente envolvida. A figura 2 mostra uma visão geral da arquitetura do jogo.

O NetMaze possui uma funcionalidade onde o jogador pode optar por ter seus movimentos controlados pelo computador. Neste modo, denominado jogador autônomo, todas as ações são solicitadas ao servidor por um agente inteligente especializado [8], desenvolvido com base no motor de inferência JEOPS[3] e com uma base de conhecimento[8] desenvolvida para ações específicas do NetMaze. O comportamento do agente dependerá do papel do jogador – se caça ou caçador - em uma determinada fase do jogo. O quadrado de linhas escuras em torno do caçador na figura 1 representa o raio de visão do mesmo, explicado posteriormente.


**Figura 1:** *Um momento do jogo*

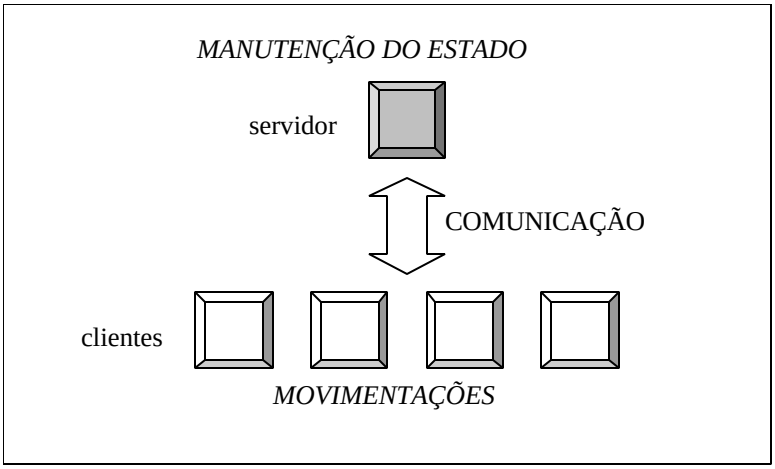



**Figura 2:** *Arquitetura geral do jogo*

## 3 PROBLEMAS NO DESENVOLVIMENTO DE JOGOS DE AÇÃO

Nesta seção descreveremos os principais problemas encontrados no desenvolvimento de jogos de ação distribuídos e multi-usuário.

### 3.1 CONSISTÊNCIA DE DADOS

Em uma aplicação interativa, como no caso do NetMaze, a resposta a cada ação do usuário deve gerar uma visualização consistente das mesmas refletindo o estado atual da aplicação. A demora na apresentação destes dados pode degradar a qualidade do serviço percebido pelo usuário, gerando falhas em ações futuras. Para que uma aplicação seja consistente devemos garantir, então, que o conteúdo e o tempo de apresentação dos dados sejam coerentes com as ações executadas pelos usuários.

### 3.2 SINCRONIZAÇÃO

A sincronização é um fator muito importante para aplicações interativas e multi-usuários e tem grande influência na qualidade final da aplicação. O processo de sincronização constitui um problema no desenvolvimento de jogos de ação, pois para que estes sejam executados com uma qualidade satisfatória é desejável que cada ação realizada por um jogador seja imediatamente percebida pelo mesmo, assim como pelos demais participantes do jogo. Como vários jogadores executam ações diferentes e possivelmente ao mesmo tempo, a sincronização entre os participantes torna-se ainda mais complexa, sendo comum a centralização do processamento, limitando-se, assim, a permissão de alteração no estado do jogo a um servidor controlador.

### 3.3 INTERFACE COM O USUÁRIO

A interface com o usuário exerce grande influência na aceitação e sucesso comercial de jogos. Uma boa interface deve ser amigável e possuir uma estética agradável para que possa atrair a atenção do jogador.

O desenvolvimento de interfaces gráficas não é uma atividade essencialmente técnica, ela envolve a participação de várias outras disciplinas, como psicologia, *design* gráfico,

cognição, etnografia, etc. de modo que é preciso saber quais e que tipos de variáveis estão envolvidas e o que significam, por exemplo, formas e gráficos que estejam dispostos no sistema[15]. As atividades de projeto e implementação de interfaces consomem cerca de 50% do tempo de desenvolvimento de uma aplicação[14], o que demostra sua complexidade e importância.

A interface de jogos utiliza gráficos, imagens e figuras, e está continuamente recebendo modificações do ambiente, devendo, assim, apresentar um cenário bem definido para que estas modificações sejam percebidas continuamente e sem alterações bruscas.

### 3.4 INTELIGÊNCIA

Em determinados tipos de jogos é aconselhável o uso de recursos de computação inteligente para prover autonomia aos jogadores. No caso de um único jogador desejar disputar com o próprio computador ou mesmo quando se quer desenvolver formas automáticas para suas próprias ações, deve-se utilizar recursos de inteligência. Neste caso, enfrentamos novas dificuldades para alcançar uma boa percepção do ambiente, tomada de decisão e ação no jogo. Um agente especialista deve ser definido de acordo com a estratégia do jogo e sua eficácia será melhor quanto melhor for a eficiência das regras escritas para o mesmo.

### 3.5 TRÁFEGO DE REDE E CONSUMO DE BANDA

Uma das principais características de uma aplicação distribuída é o tráfego de rede gerado pela troca de mensagens entre clientes e servidores. Em jogos de ação distribuídos esse problema se intensifica devido à troca de informações de grande volume como por exemplo, imagens e sons, gerando um alto consumo de banda passante. Mesmo reduzindo o volume de dados multimídia, a taxa de transferência na rede faz a implementação de jogos de ação multi-usuário um desafio, dado que em certas ocasiões existe uma limitação de recursos da rede.

### 3.6 INTEROPERABILIDADE

As estações de trabalho que os usuários de computadores utilizam atualmente, diferem-se enormemente umas das outras. Existem diferentes sistemas operacionais, diferentes estilos de interfaces de usuários, diferentes padrões de gráficos e diferentes tempos de resposta dos sistemas. Esta grande heterogeneidade acarreta grandes desafios para o desenvolvimento de aplicações distribuídas multi-plataforma. É preciso uma abordagem técnica para permitir que uma mesma aplicação funcione corretamente nesses diferentes ambientes[15].

O ORB, *Object Request Broker*, núcleo central da especificação CORBA[1,7] provê interoperabilidade entre aplicações em diferentes máquinas, entre ambientes heterogêneos, e também interconecta os múltiplos objetos de um sistema distribuído. CORBA isola os usuários finais e os programadores de aplicações das características distribuídas heterogêneas dos sistemas de informação.

O uso de CORBA soluciona parte do problema associado à interoperabilidade, mas alguns problemas, como por exemplo, qual esquema de comunicação será usado, quais serviços de CORBA serão utilizados, etc., têm de ser analisados de acordo com as necessidades da aplicação em questão. Abordagens diversas podem beneficiar diferentes tipos de aplicações de acordo com suas características.

## 4 SOLUÇÕES PROPOSTAS

Apresentamos nesta seção as soluções propostas para os problemas identificados na seção 3 os quais foram considerados durante o desenvolvimento do jogo NetMaze. Estas soluções podem ser aplicadas a outros jogos com características semelhantes.

### 4.1 CONSISTÊNCIA DE DADOS, SINCRONIZAÇÃO E INTERFACE COM O USUÁRIO

Para que a apresentação das informações aos usuários seja realizada de forma consistente e sincronizada, utilizamos um objeto servidor que centraliza todo o processamento e é responsável por realizar as alterações no estado do jogo, bem como pelo envio das informações atualizadas para todos os jogadores. Assim, cada cliente transmite seus dados a um servidor que os processa e envia aos outros clientes. O servidor armazena todas as informações necessárias para manter o funcionamento adequado do jogo e todos os clientes devem obter informações consistentes através deste.

As movimentações requisitadas pelos usuários também devem ser testadas de forma que apenas movimentos permitidos sejam vistos na tela. Como exemplo, temos o caso em que dois jogadores tentam se movimentar para a mesma posição. Neste caso, o servidor deve testar a validade do movimento para impedir a sobreposição de imagens. Para conseguir uma sincronização satisfatória, os métodos do objeto servidor que processam o jogo são acessados pelos clientes de forma seqüencial. Esse tipo de acesso é importante para que as modificações no estado atual do jogo sejam controladas pelo servidor. Assim, apenas um jogador poderá alterar o estado do jogo por vez e o servidor testará se essa alteração corresponde a um movimento permitido, impedindo a ocorrência de situações de inconsistência.

O descarte de eventos é um fato bastante importante que ocorre durante a execução de um jogo. No caso do NetMaze observamos que nem todas as ações enviadas pelos clientes, apesar de serem processadas pelo objeto servidor, são retornadas para a apresentação aos jogadores. Pela característica da aplicação, um jogo de ação, é comum que os usuários mantenham as teclas de movimentação constantemente pressionadas, o que gera um grande número de pedidos de movimentação. No entanto, devido a própria limitação da visão humana, que não consegue perceber movimentos a uma velocidade muito alta, não é necessário que o objeto servidor envie uma atualização de tela para cada movimento solicitado pelos clientes, e sim, envie atualizações numa proporção em que a perda não seja percebida pelos usuários. Alguns eventos, no entanto, devem ser enviados a todos os clientes. O evento de caça capturada, por exemplo, não pode ser perdido pois causaria inconsistência no jogo.

A interface principal do jogo é composta por três quadros: um que corresponde ao menu, outro que apresenta o cenário do labirinto, e um outro quadro que apresenta a pontuação dos jogadores. Existem algumas opções as quais os usuários podem configurar, como por exemplo, ligar e desligar o som de fundo, e habilitar ou desabilitar a autonomia do jogador. Quando a opção de autonomia é habilitada para um jogador, esta instancia o agente inteligente para obter as percepções do estado do ambiente e realizar suas tarefas.

As figuras correspondentes a interface do NetMaze foram todas criadas e recriadas de maneira que os seus desenhos correspondessem às características do jogo. Uma sucinta apresentação dos personagens é realizada, enquanto é reproduzida a música de abertura.

A tela principal apresenta quadros, como descrito anteriormente. No canto superior esquerdo, o do menu, no canto superior direito, o do labirinto, e no canto inferior, o quadro da

pontuação dos jogadores (veja figura 1). O quadro do menu é composto por diversas figuras, que são montadas para a formação de uma única imagem. A técnica *de Double Buffering*[4] foi utilizada para corrigir os efeitos da sobreposição. Para facilitar a montagem, utilizamos a classe *MediaTracker* do Java, que tem o objetivo de armazenar as figuras capturadas, e criar índices para cada uma[4].

No momento em que um usuário passar com o mouse sobre um determinado item do menu, este item modifica a sua cor. Isso acontece porque, um evento de mouse é disparado na sua movimentação, e neste evento realizamos a checagem de coordenadas correspondentes a cada figura. Se é detectado que o mouse está sobre uma determinada figura, é realizada uma nova montagem do menu, inserindo outra figura correspondente a um determinado índice no *MediaTracker*. Então é apresentada uma nova formação do menu para o usuário, como mostrado na figura 3.

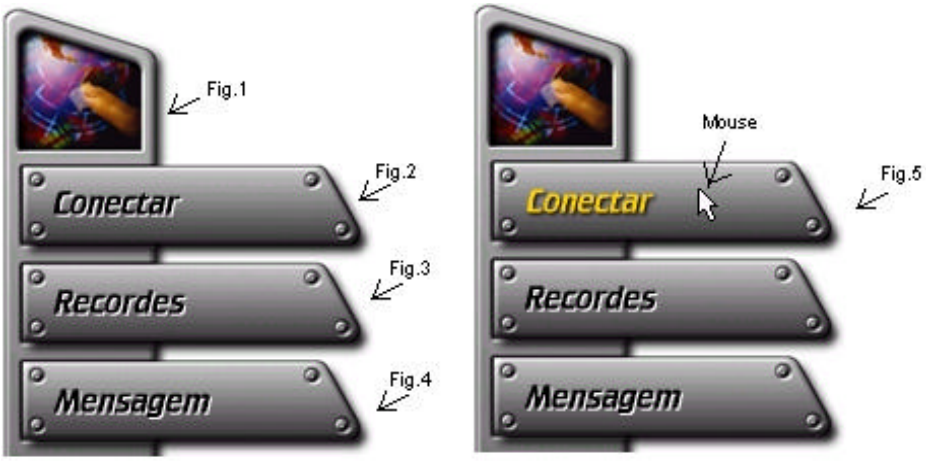



**Figura 3:** *Esquerda: Montagem do Menu*
*Direita: Montagem com o mouse sobre a figura 2*

### 4.2 INTELIGÊNCIA

A necessidade de inserir autonomia aos personagens do jogo, nos levou a criar agentes inteligentes para assumir os papeis destes personagens. Uma boa implementação de um agente inteligente é aquela em que, o ambiente de atuação, as funções de percepção e de ação estão muito bem definidas. A definição dos fatores para a elaboração dos agentes deste projeto, é apresentada na figura abaixo. Apesar da simplicidade do jogo, o ambiente no qual os agentes estão inseridos não é dos mais simples, visto que, segundo os parâmetros clássicos, ele pode ser caracterizado como parcialmente acessível., dinâmico, não determinista e contínuo [8].

| Tipo | Ambiente | Objetivos | Percepções | Ações |
|---|---|---|---|---|
| Agente caça | Labirinto | Esconder-se, Fugir dos caçadores | Choque c/ a parede, caçador próximo | Caminhar na direção atual, girar p/ qualquer direção |
| Agente caçador | Labirinto | Procurar, Perseguir a caça | Choque c/ a parede, choque c/ outro caçador, caça próxima, pegou a caça | Caminhar na direção atual, girar p/ qualquer direção, comer caça |

**Figura 4:** *Caracterização das tarefas dos agentes*

Os agentes do jogo são agentes reativos [8], e quando habilitados, realizam um trabalho de verificação e percepção do estado atual do ambiente através de seus sensores. A partir das informações coletadas e de seus objetivos, os agentes decidem que ação executar e a executam em seguida, atualizando a representação interna do ambiente. Os agentes caminham sem objetivo pelo labirinto enquanto não estiverem enxergando inimigos. Caso contrário, eles passam a ter como objetivo a perseguição ou a fuga.

Cada agente do jogo possui um campo de visão que funciona como um radar, e este radar só capta informações de uma determinada porção do ambiente (labirinto). Portanto inimigos serão percebidos apenas quando estiverem no campo de visão de captação do radar. A figura abaixo demonstra o campo de visão dos agentes utilizado para determinar as suas tarefas.

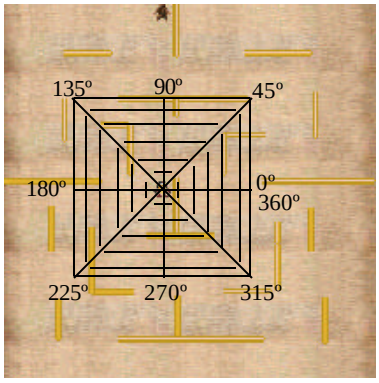



**Figura 5:** *Campo de visão dos agentes*

A escolha das ações, bem como a dedução de outras propriedades do ambiente, é feita por intermédio de uma base de conhecimento que é manipulada por um motor de inferência de primeira ordem com encadeamento para frente [3]. Abaixo, encontram-se dois exemplos, de um total de 29 regras, da base de conhecimento do NetMaze. A primeira regra é uma das que servem para modelar o ambiente, indicando as direções que estão atualmente livres, enquanto que a segunda tem um impacto direto na tomada de decisão.

1. ∀a,j Agente(a) Λ Jogo(j) Λ ( NumeroJogadores(j) > 1 ) Λ ExisteAdversarioEntreAngulos_45_90(a) => DirecaoNaoEstaLivre(a,0)
2. ∀a,j Agente(a) Λ Jogo(j) Λ ( NumeroJogadores(j) > 1 ) Λ ¬SouCacador(a) Λ DirecaoEstaLivre(a,DirecaoAtual(a)) Λ ¬ExisteParedeNestaDirecao(a,DirecaoAtual(a)) => AndarNestaDirecao(a,DirecaoAtual(a))

### 4.3 TRÁFEGO DE REDE, CONSUMO DE BANDA PASSANTE E INTEROPERABILIDADE

Com o objetivo de solucionar o problema do elevado tráfego na rede, consumo de banda passante e interoperabilidade, foi proposto em [13] a aplicação de três arquiteturas distintas de comunicação, as quais são apresentadas na Figura 4.

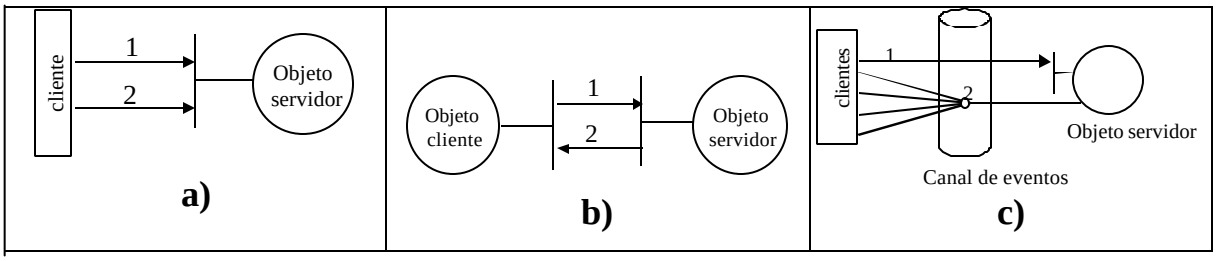



**Figura 6:** *a) : CA, onde 1 é o pedido de movimentação e 2 é a requisição do estado corrente. B) CB, onde 1 é o pedido de movimentação e 2 é referente ao estado atualizado enviado pelo servidor. c) CE, onde 1 é o pedido de movimentação e 2 refere-se ao multicast do evento.*

Na arquitetura Cliente Ativo (CA), apresentada na figura 6(a), os jogadores enviam ao servidor seus pedidos de movimentação para serem processados e requisitam o estado atual do jogo. O objeto servidor possui uma interface com métodos responsáveis pelo envio de labirintos, posições dos jogadores e outros eventos que representam o estado atual do jogo. Esses serviços são requisitados pela aplicação cliente, através da interface do servidor.

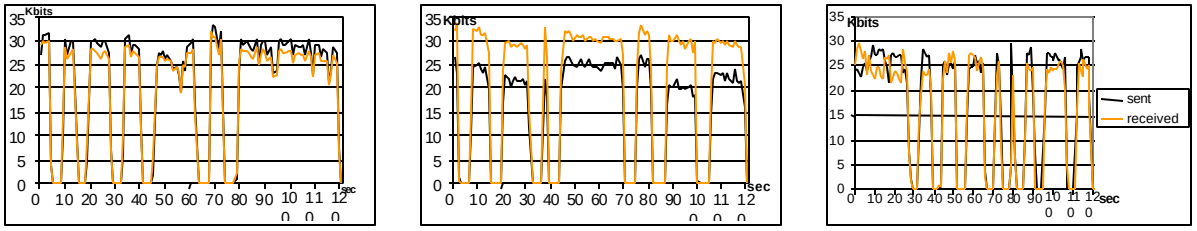
Na arquitetura Call-Back (CB), apresentada na figura 6(b), o objeto servidor é o responsável por passar aos jogadores o estado atual do jogo. Essa informação só é passada quando há alteração do mesmo, não havendo, assim, trafego desnecessário de rede. Assim, quando um jogador qualquer se movimenta, o objeto servidor recebe essa movimentação, a processa e envia-a aos demais jogadores.

Neste modelo uma nova interface (*callback*) é criada no cliente para que o servidor possa se comunicar com o mesmo. No início da conexão a referência do objeto cliente é enviada para o servidor a fim de que a comunicação seja estabelecida nos dois sentidos. No servidor é criado um novo *processo (thread)* para cada jogador conectado. Este *processo (thread)* é responsável por enviar o estado atual do jogo para seu respectivo cliente, sempre que há alguma alteração.

Na arquitetura Canal de Eventos (CE), apresentada na figura 6(c), o objeto servidor também é responsável por passar aos jogadores o estado atual do jogo, quando há alteração do mesmo. O Serviço de Eventos de CORBA[12] é utilizado e um Canal de Eventos permanece aberto durante o jogo para que os clientes possam a ele se conectar e receber as informações atualizadas do jogo. Quando ocorre uma mudança de estado no jogo, o servidor gera um evento que é enviado a todos os demais jogadores através de um *multicast* feito pelo canal de eventos.

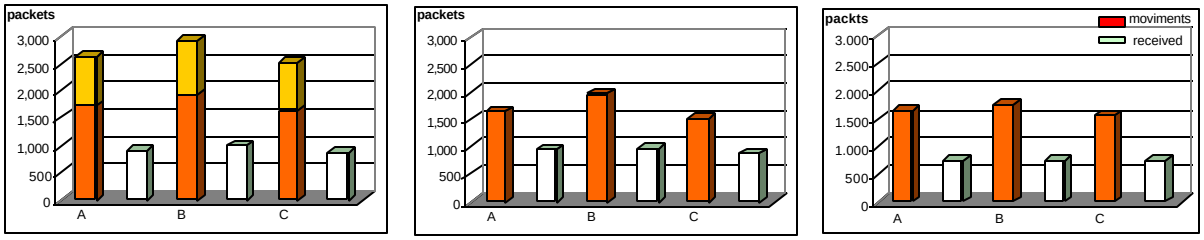
A diferença principal entre o modelo com Canal de Eventos e o modelo Call-Back está na forma como as atualizações são enviadas para os vários jogadores. No primeiro caso é realizado um *multicast*, enquanto no segundo a comunicação se dá por conexões *unicast*.

Com o jogo funcionando com cada uma destas arquiteturas, verificamos que o desempenho conseguido com a arquitetura Cliente-Ativo, Call-Back e Canal de Eventos é similar em termos de consumo de banda passante, conforme apresentado na figura 6. Por outro lado, uma diferença considerável foi observada em relação ao número de pacotes da aplicação enviados dos clientes para o servidor, conforme apresentado na figura 7. A redução na troca de pacotes apresentada na arquitetura Call-Back e Canal de Eventos é decorrente da transferência para o servidor do processo de atualização da tela dos clientes e o número de mensagens que deixam de ser enviadas é referente às requisições de atualização feitas desnecessariamente pelos clientes na arquitetura Cliente-Ativo.




*a)* ) Arquitetura AC *b)* Arquitetura CB c) Arquitetura EV

**Figura 6**




*a)* ) Arquitetura AC *b)* Arquitetura CB c) Arquitetura EV

**Figura 7**

O requisito de interoperabilidade foi alcançado em todas as arquiteturas propostas. O serviço de nomes, utilizado nas três arquiteturas, tornou possivel a transparência de localização do objeto servidor e total independência de ambiente operacional entre os clientes. Devido às características das diversas arquiteturas, o uso de cada uma pode ser mais adequado de acordo com os requisitos de cada aplicação. Em um jogo de ação, que possui maiores exigências com relação a tempo de resposta, tanto o modelo Cliente-ativo quanto o Call-Back mostrou-se adequado. Já em aplicações com menor interação entre clientes e servidor, onde o consumo de recursos de rede podem ser poupados, o modelo Call-Back e Canal de Eventos tornam-se mais eficazes, devido a redução de troca de mensagens neste tipo de aplicação. Maiores detalhes dos resultados alcançados podem ser vistos em [13].

## 5 CONCLUSÕES E TRABALHOS FUTUROS

O desenvolvimento de qualquer aplicação para ambientes multimídia demanda além de um projeto cuidadoso e bem documentado, tempo e paciência dos seus desenvolvedores. Em compensação os resultados obtidos são sempre gratificantes e estimulantes, tanto para os usuários da aplicação, no que diz respeito ao ganho obtido com a utilização dos recursos visuais fornecendo uma melhor praticidade; quanto para os desenvolvedores com o uso de novas técnicas e ferramentas introduzidas na computação.

O *NetMaze* apresentou-se adequado, como estudo de caso, para avaliação das alternativas propostas como solução dos principais problemas encontrados no desenvolvimento de uma aplicação multimídia distribuída sobre plataforma CORBA. Verificamos que é possível desenvolver jogos distribuídos com boa qualidade utilizando o padrão CORBA.

Em trabalhos futuros, projetaremos um ambiente tridimensional para o *NetMaze*, tornado o campo de visão dos agentes o mais próximo do mundo real. E utilizaremos o protocolo IIOP do CORBA para realizar conexão das aplicações clientes através da Internet.

# 6 REFERÊNCIAS BIBLIOGRÁFICAS


[1] Ben-Natan Ron, CORBA on the Web, McGraw-Hill, United States, 1998.

[2] Bernt Habermeier – Internet Based Client/Server Network Traffic Reduction. http://www.bolt-action.com

[3] Figueira, C.- *JEOPS – The Java Embedded Object Production System*, UFPE, 1999 – http://www.di.ufpe.br/~csff/jeops

[4] Horstmann, C., Cornell, G.- *Core Java 2 Fundamentals - Volume I*, Sun Microsystems Inc., California, 1999.

[5] Jain, R.- *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*, John wiley & Sons, Inc. 1991.

[6] Lamothe, A.- *Tricks of the Windows Game Programming Gurus - Fundamentals of 2D and 3D Game Programming*, Sams, United States, 1999.

[7] Orfali, R., Harkey, D.- *Client/Server Programming with Java and CORBA*, John Wiley & Sons Inc., United States, 1998.

[8] Russel, Stuart J., Norvig, Peter, Artificial Intelligence: A Modern Approach, New Jersey, Prentice-Hall Inc., 1995.

[9] Siqueira, F.- *A Framework for Distributed Multimedia Applications based on CORBA and Integrated Services Networks.* http://www.cs.tcd.ie/Frank.Siqueira/PhD-Project/index.html

[10] TCPDUMP – Lawrence Berkeley National Laboratory, Network Research Group - ftp://ftp.ee.lbl.gov/tcpdump.tar.Z

[11] Tirakis, A. et al- *Distributed Multimedia Architectures - State-of-the-Art Report* http://viswiz.gmd.de/DVP/Public/deliv/deliv.222/act222.htm

[12] *VisiBroker – Programmer's Guide*, Inprise Corporation, Inc., 1999 – http://www.borland.com/techpubs/books/vbj/vbj40

[13] Macedo, H., Araujo, A., Cavalcanti, D., Andrade, R., Madeira, C., Ferraz, C. - Exploiting Multi User Distributed Action Games' Architectures (Impact) on CORBA Platform, to appear in proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Monte Carlo Resort, Las Vegas, Nevada, USA, June 26 - 29, 2000

[14] Myers, B. A., Rosson, M. B., Survey on User Interface Programming, IBM Research Report, RC-17624, 1992

[15] Leite, J. C., Souza C. S., Visão Geral do Projeto de Interfaces de Usuário, UFRN, 1997

[16] Coulouris, G. et. al., Distributed Systems – Concepts and Design, Second Edition, Addison-Wesley Publishers Ltd, 1995.