Avaliação de Arquiteturas para Jogos de Ação Distribuídos Desenvolvidos em CORBA

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

Centro de Informática, Universidade Federal de Pernambuco 50732-970, Recife, PE, Brazil

htm@cin.ufpe.br

(Recebido em 23 de junho de 2005; aceito em 27 de julho de 2005)

Este artigo identifica os principais problemas relacionados com o desenvolvimento de jogos multi-usuários distribuídos e apresenta três diferentes arquiteturas para implementação deste tipo de jogo, utilizando a plataforma de distribuição CORBA. As arquiteturas foram implementadas em um jogo de ação visando a avaliação das mesmas com relação ao tráfego de pacotes, consumo de banda passante, perda de eventos e à consistência dos dados no processo de atualização da tela de cada jogador. Como resultado obtido, observamos que para diferentes tipos de jogos o uso de arquiteturas adequadas pode levar a uma considerável economia de processamento, redução de tráfego de rede e ganhos com a qualidade de apresentação ao usuário.

Palavras-chave:

This paper identifies the main problems related to the development of distributed multi-user games and presents three different architectures to implement this kind of games using the CORBA distribution platform. The architectures were implemented in an action game aiming at evaluating them with respect to packets traffic, bandwidth requirements, events lost and data consistency on the redrawing process of each player's screen. As results, we see that to different kind of games the use of appropriate architectures may lead to a great processing economy, network traffic reduction and gains on the user presentation quality.

Keywords:

1. INTRODUÇÃO

O desenvolvimento e processamento de aplicações multimídia distribuídas apresentam problemas em sua maioria causados pelas limitações da taxa de transmissão de dados em rede. No caso de jogos de ação distribuídos, aplicações que estão se tornando cada vez mais populares, surgem novas dificuldades principalmente devido ao comportamento dos usuários, os quais estão constantemente realizando alguma ação de movimento durante a execução do jogo. Em decorrência desse comportamento ativo e também do fato dos jogos possuírem comportamento multi-usuário, surgem problemas relativos à manutenção da consistência de dados, compartilhamento de recursos e desempenho.

A consistência dos dados é um fator que influi diretamente na qualidade do jogo percebida pelos usuários. Para que um jogo de ação seja executado com uma qualidade satisfatória é desejável que cada ação realizada por um jogador seja imediatamente apresentada na tela do próprio jogador, assim como na tela dos demais participantes do jogo. Como vários jogadores executam ações diferentes e possivelmente ao mesmo tempo, a manutenção de consistência do estado atual do jogo torna-se mais complexa, sendo comum a centralização do processamento do jogo, limitando a permissão de alteração nesse estado a um servidor que o controla.

Neste trabalho foi desenvolvido um jogo de ação distribuído e multi-usuário, baseado em um servidor centralizado que processa e controla o estado do jogo e em vários clientes que enviam suas ações para que o servidor as processe e atualize o estado do jogo. São apresentadas três arquiteturas para a comunicação entre clientes e servidor. Avaliamos o comportamento destas

arquiteturas, aplicando-as ao jogo desenvolvido, em relação ao tráfego de pacotes gerados na rede por cada uma, a consistência dos dados percebida na tela de cada jogador e a perda de eventos, ou seja, o número de ações enviadas pelos jogadores que deixam de ser apresentadas em suas telas, sem degradar a qualidade percebida pelo usuário.

Na seção 2 apresentamos os principais problemas encontrados no desenvolvimento de aplicações multimídia distribuídas, destacando em particular os jogos de ação multi-usuários. Na seção 3 descrevemos o jogo desenvolvido como estudo de caso e apresentamos as arquiteturas utilizadas. Na seção 4, descrevemos experimentos de teste realizados para cada implementação diferente e avaliamos os resultados obtidos através da medição do tráfego na rede. Finalmente, na seção 5 apresentamos as conclusões e possíveis trabalhos futuros.

2. APLICAÇÕES MULTIMÍDIA DISTRIBUÍDAS

Atualmente, as aplicações multimídia distribuídas consomem de forma muito elevada o potencial dos computadores, devido ao grande número de gráficos, imagens, vídeos, e outras informações que ocupam muito espaço em memória. Desta forma, torna-se necessário que a capacidade do tráfego de informações entre computadores, através das redes, seja alta o suficiente para que se obtenha um bom grau de satisfação por parte dos usuários.

CORBA [1] representa uma nova geração de facilidades aos ambientes que provêm aplicações para sistemas distribuídos, e seu principal componente é o *Object Request Broker* (ORB), mecanismo através do qual objetos transparentemente emitem requisições e recebem respostas. O ORB provê interoperabilidade entre aplicações em diferentes máquinas e em ambientes heterogêneos, como também interconecta sistemas com múltiplos objetos. CORBA isola os usuários finais e os programadores de aplicações das características distribuídas heterogêneas dos sistemas de informação. Diante disso, é desejável que o desenvolvimento de aplicações multimídia distribuídas possa ser beneficiado com a incorporação deste padrão [7].

Aplicações multimídia distribuídas possuem requisitos bem definidos baseados nas características dos tipos de mídia que devem ser transmitidos. Esses requisitos apresentam exigências no que diz respeito a reserva de recursos computacionais e de rede. A largura de banda, por exemplo, depende da natureza da tarefa a ser executada; o tempo de resposta aceitável deve ser especificado de acordo com a aplicação; e a confiabilidade dos dados, é geralmente um fator crítico de todos esses sistemas [2].

2.1. Consistência dos Dados e Atualização da Interface com o Usuário

Ao se adotar uma aplicação interativa, a resposta a cada ação do usuário deve ser rápida, de forma a gerar uma visualização consistente dessas ações e ao mesmo tempo refletir o estado atual correto da aplicação. A demora na atualização de tela pode degradar a qualidade do serviço percebido pelo usuário, podendo gerar falhas em ações futuras.

A programação para jogos é conduzida para aplicações de tempo real e simulação. O processamento básico de um vídeo game é basicamente constituído por um laço contínuo com execuções lógicas e atualizações de imagens na tela [6]. Os computadores trabalham em velocidades variáveis, o que eleva o nível de complexidade dos jogos. Por exemplo, se existirem 1000 objetos sendo executados em uma tela, a carga de trabalho será bem maior que a existente numa tela onde existam apenas 10 objetos sendo executados. Neste caso, a taxa de quadros apresentados por segundo será variável, o que poderá causar efeitos inaceitáveis para o usuário, comprometendo a qualidade do jogo.

Em jogos multi-usuários para uma única máquina, o laço principal do jogo processa cada jogador envolvido. Isto inclui capturar os dados de entrada, movimentá-los, construir as suas visões, e computar qualquer outra lógica as quais estejam associados. Em jogos multi-usuários em rede, o laço do jogo precisa processar apenas um jogador, ou seja, o jogador da máquina que o jogo está sendo executado. O modelo cliente-servidor é geralmente usado, de maneira que cada máquina transmite seus dados a um servidor, sendo as informações sobre os outros jogadores retornadas pelo servidor. O servidor é o centralizador do jogo e armazena todas as

informações para manter o seu funcionamento. Todos os clientes devem ter informações consistentes através do servidor.

2.2. Tráfego de Rede

A maioria das aplicações multimídia distribuídas com controle centralizado gera um grande tráfego de rede, em virtude da constante troca de grande volume de dados entre o servidor e seus clientes.

No nosso estudo de caso analisamos um jogo onde um servidor trata do processamento e a troca de mensagens com seus clientes se resume a informações de alteração no estado do jogo. Todo o volume de dados multimídia necessários para o jogo fica armazenado nos próprios clientes, evitando que grandes volumes de dados trafeguem pela rede.

Mesmo reduzindo o volume dos dados multimídia, a taxa de transferência de dados na rede faz desta um desafio para implementar jogos de ação multi-usuários. Nesse tipo de aplicação, os clientes devem ter suas telas atualizadas assincronamente em relação uns aos outros mantendo a consistência do estado atual do jogo, e a taxa de quadros exibidos por segundo em cada cliente não deve variar com o tráfego. Neste trabalho, aplicamos as diferentes arquiteturas de comunicação apresentadas na próxima sessão, visando a redução de tráfego na rede e a manutenção da qualidade do serviço apresentado aos usuários.

No nosso modelo, o servidor controla o estado do jogo, processando os movimentos dos jogadores. Os clients devem requisitar a criação, modificação e remoção de objetos diretamente ao servidor, que é responsável pela alteração dos mesmos, gerenciamento de consistência e, em alguns casos, pela comunicação de modificações do estado do jogo aos clientes. Não é permitido que clientes enviem broadcast com modificação de estado para outros clientes. Isto é obrigação do servidor, para garantir a consistência dos dados.

Numa das técnicas aplicadas, transferimos algumas funções, antes executadas nos clientes, para o servidor com o objetivo de diminuir a troca de mensagens entre clientes e servidor necessárias para a execução dessas funções. Nas seções seguintes apresentamos e avaliamos os resultados obtidos para uma arquitetura onde os clientes ficam constantemente enviando requisições de atualização de tela para o servidor e comparamos com uma arquitetura onde essas funções são transferidas para o servidor.

3. ESTUDO DE CASO

A aplicação desenvolvida consiste num jogo multi-usuário distribuído o qual denominamos de NetMaze. Seu funcionamento consiste em vários caçadores que se movimentam dentro de um labirinto tentando pegar uma caça, que por sua vez tem como objetivo manter-se o maior tempo possível sem ser pega. Assim, os usuários ao se conectarem podem aparecer na tela como caça ou caçador dependendo do estado do jogo no momento da conexão e da seqüência de conexão. O estado do jogo é representado pelas posições dos jogadores na tela, pelo formato do labirinto atual (o jogo possui vários labirintos diferentes) e pelos eventos que ocorrem, como colisões entre caçadores, de caçadores com a caça ou dos jogadores com as paredes do labirinto. 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 pega. Nesse momento, os clientes mostram na tela uma imagem comum referente a 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 (figura 1).

O jogo 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 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.

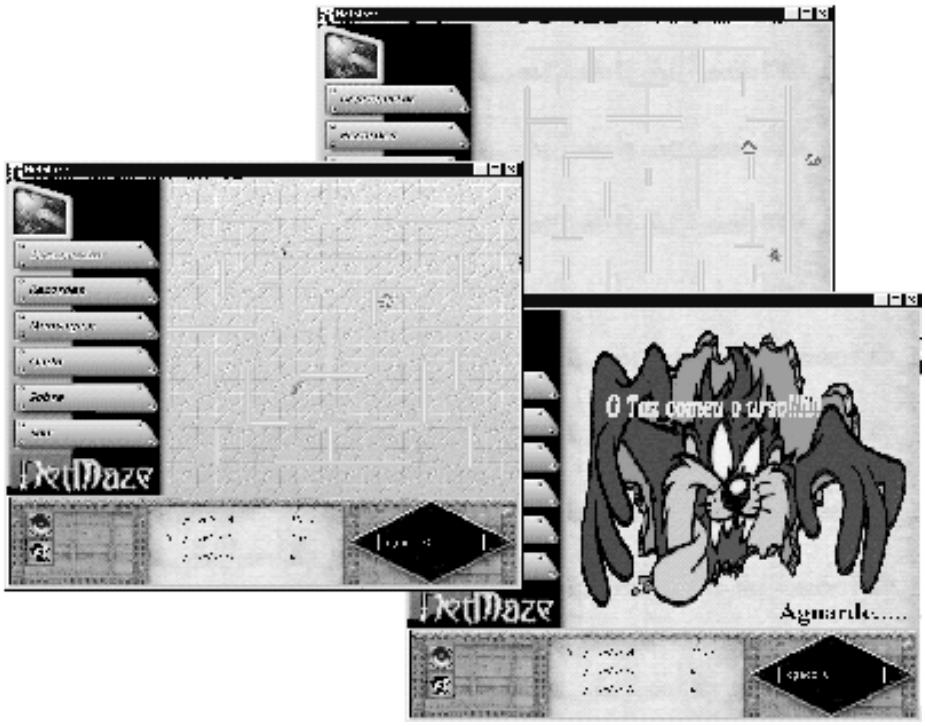

Figura 1 - Telas do jogo NetMaze.

O jogo foi implementado utilizando a plataforma de distribuição CORBA e a linguagem de programação Java [4], [7]. O processamento do jogo é centralizado em um objeto que atua como servidor. A localização desse objeto servidor é efetuada pelos clientes através de requisições ao serviço de nomes da plataforma.

Como descrito anteriormente, durante o processamento não há trafego multimídia na rede, uma vez que as imagens e sons são armazenados nas máquinas de cada um dos clientes. Os jogadores podem se movimentar na horizontal, na vertical ou ficar parados e essas requisições de movimentação, que representam a ação de uma tecla pressionada ou liberada, são enviadas para o servidor onde são processadas e o estado atual do jogo é atualizado.

A sincronização é um aspecto muito importante para uma aplicação interativa e multi-usuário e tem grande influência na qualidade final da aplicação percebida, uma vez que os movimentos de cada jogador vistos na tela devem corresponder exatamente aos pedidos de movimentação enviados. Outro fato que torna esse processo de sincronização mais complexo é o de que além de visualizar na tela as suas movimentações, cada jogador tem que visualizar ao mesmo tempo os movimentos dos outros jogadores. As informações necessárias para a atualização das telas dos usuários são enviadas pelo objeto servidor, correspondendo ao estado atualizado do jogo.

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, 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, ou seja, apenas um cliente tem o acesso à esses métodos por vez. Esse tipo de acesso é importante para que as modificações no estado atual do jogo sejam controladas pelo servidor. Assim, apenas um jogador pode alterar o estado do jogo, por vez, e o servidor testa se essa alteração corresponde a um movimento permitido, impedindo a ocorrência de situações de inconsistência.

Outro fato importante que foi constatado é a ocorrência de perda de eventos durante a execução do jogo, ou seja, nem todas as ações enviadas pelos clientes são processadas pelo objeto servidor. 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. Isso pode ser verificado nos resultados obtidos (seção 4), onde relacionamos a perda de eventos e o nível de qualidade percebido pelo usuário.

Todo esse ciclo que engloba o envio de ações dos jogadores para o objeto servidor, processamento e envio de atualizações de telas do servidor para os jogadores foi projetado utilizando-se três arquiteturas diferentes. Das três arquiteturas, duas foram implementadas, e o objetivo principal deste trabalho é a análise e avaliação dessas implementações, procurando verificar qual delas apresenta melhor desempenho em termos de tráfego em rede, tempo de resposta percebido pelos jogadores em relação a uma ação executada e a perda de eventos em cada arquitetura. A partir dos resultados obtidos, procuramos também melhorar o desempenho do jogo com a implementação, em andamento, da terceira arquitetura proposta.

Nas subseções seguintes faremos uma descrição mais detalhada das três arquiteturas desenvolvidas e aplicadas no estudo de caso.

3.1. Arquitetura Cliente Ativo

Nesta arquitetura, os jogadores enviam ao servidor seus pedidos de movimentação para serem processados e são os responsáveis por requisitar 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 services são requisitados pela aplicação do jogador, através da interface do servidor, cabendo a este o retorno das informações solicitadas (figura 2).

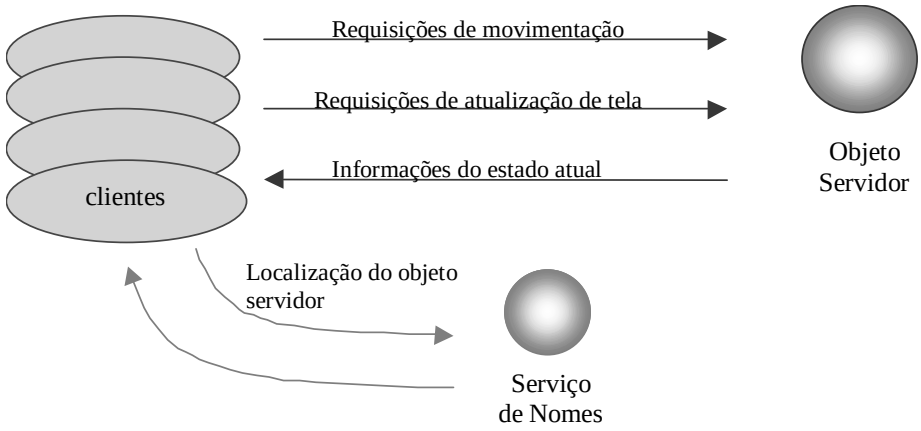

Figura 2 - Arquitetura Cliente Ativo

As funções de apresentação de imagens, sons e texto são processadas no cliente com base nas informações de estado do jogo recebidas. Neste modelo, o cliente fica constantemente requisitando e apresentando na tela as atualizações no estado do jogo. Mesmo quando não há alterações no estado atual, os jogadores têm suas telas atualizadas, gerando assim, um tráfico desnecessário na rede, bem como processamento local desnecessário para atualizar a tela.

3.2. Arquitetura Call-Back

Na arquitetura Call-Back, o objeto servidor é o responsável por passar aos jogadores o estado atual do jogo. Essa informação só é passada quando há alteração nesse estado. Assim, quando um jogador qualquer se movimenta, após o processamento no objeto servidor, o estado do jogo é enviado pelo mesmo para todos os jogadores conectados.

Neste modelo uma nova interface (*callback*) é criada no cliente para que o servidor possa se comunicar com este. 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 (figura 3).

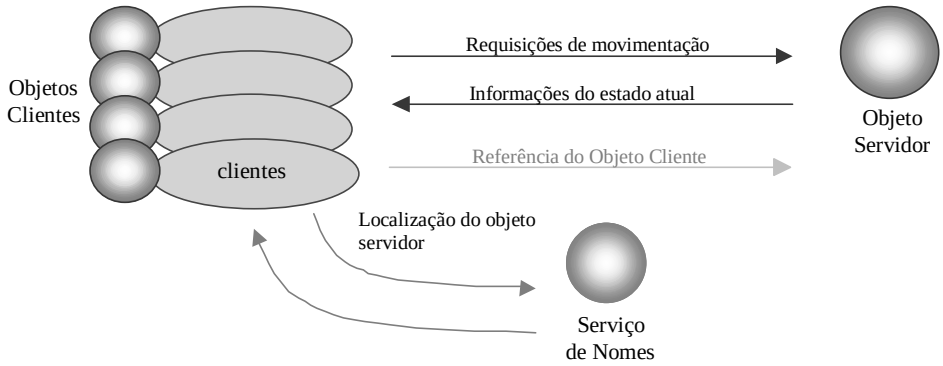

Figura 3 - Arquitetura Call-Back

No servidor é criado um novo *thread* para cada jogador conectado. Este *thread* é responsável por enviar o estado atual do jogo para seu respectivo cliente, sempre que há alguma alteração.

De forma diferente da arquitetura Cliente Ativo, no modelo Call-Back não há troca de mensagens entre o servidor e seus clientes se não houver mudança no estado do jogo. Assim, não haverá tráfego desnecessário de rede quando não houver movimentação dos jogadores, nem haverá atualização desnecessária de tela.

As requisições de movimentação dos jogadores continuam sendo feitas à partir da aplicação cliente da mesma forma que na arquitetura anterior.

3.3. Arquitetura com Serviço de Eventos

Nesta arquitetura, o objeto servidor também é responsável por passar aos jogadores o estado atual do jogo, quando houver alteração no mesmo. O Serviço de Eventos CORBA [12] é utilizado e um Canal de Eventos permanece aberto durante o jogo no qual os clientes se conectam para receber as informações atualizadas do jogo. Quando ocorre uma mudança de estado no jogo, o servidor gera um evento que será enviado a todos os jogadores através de um *multicast* feito através do canal de eventos (figura 4).

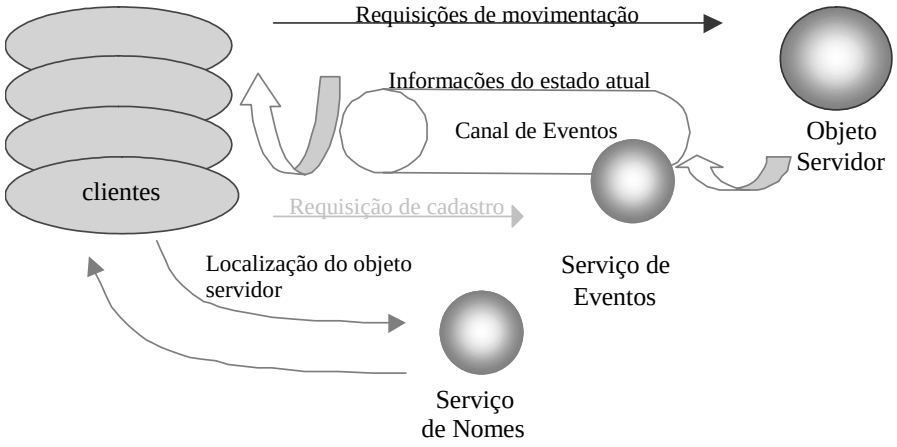

Figura 4 - Arquitetura com Serviço de Eventos.

A principal diferença entre o modelo com Serviço 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*.

4. EXPERIMENTOS

Para avaliar o comportamento de cada arquitetura proposta, executamos o jogo e efetuamos várias medições de tráfego de rede em um laboratório montado de acordo com a topologia mostrada na figura 5.

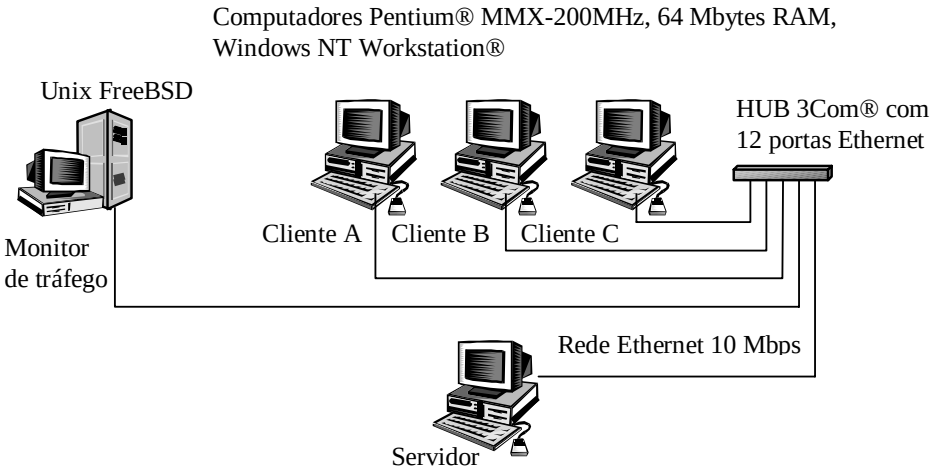

Figura 5 - Topologia da rede

Em um microcomputador servidor executamos o ORB Visibroker v. 3.4 e seu Serviço de Nomes. Outros três computadores da rede, com Windows NT-Workstation v. 4.0, executaram o módulo cliente do jogo *NetMaze*. Em um quinto computador com sistema operacional Unix - FreeBSD v. 3.2 rodamos o utilitário TCP-Dump v. 3.4 [10] para efetuar as medições de tráfego na rede.

O modo jogador autônomo foi utilizado, de forma a gerar movimentos mais uniformes entre os diferentes jogadores. Esta uniformidade se deve à característica do agente inteligente conseguir movimentos mais constantes que um jogador externo. Com o jogo em andamento, foram executadas várias medições com duração de dois minutos cada.

Nas medições, levamos em consideração a quantidade de pacotes monitorados na rede, o número de atualizações de tela enviadas pelo servidor em função do número de requisições de movimentação efetuadas pelos clientes e o número de eventos perdidos, ou seja, a quantidade de mudanças do estado do jogo que não foram recebidas pelos clientes, para apresentação na tela.

A rede utilizada foi isolada de qualquer tráfego externo e pacotes não relacionados com as conexões da aplicação analisada (como tráfego NetBEUI, ICMP e ARP) foram desconsiderados.

As arquiteturas Cliente Ativo e Call-Back apresentaram aproximadamente o mesmo nível de desempenho, qualidade de apresentação e sincronismo de movimentos verificados a olho nu na tela de todos os jogadores.

4.1. Tráfego de Pacotes

Inicialmente analisamos o tráfego de rede em relação ao número de pacotes enviados e recebidos pelos clientes, sem levar em conta o tamanho do campo de dados de cada pacote. Durante a execução do jogo não há tráfego entre clientes, apenas entre clientes e servidor. Os pacotes enviados pelos clientes podem ser divididos em dois tipos: requisições de movimentação e solicitações de atualização de tela. Já os pacotes recebidos pelos clientes correspondem as atualizações de tela e contêm informações sobre o estado atual do jogo.

Os gráficos da figura 6 resumem os resultados comparativos entre as duas arquiteturas avaliadas.

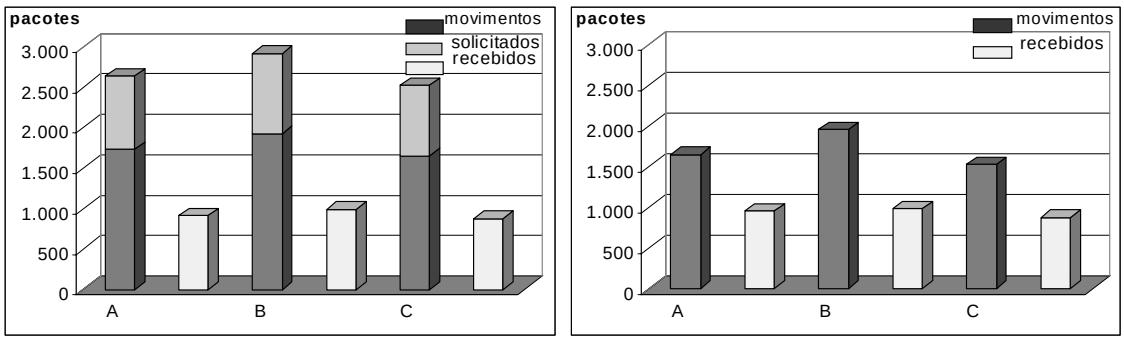

Figura 6 - Esquerda: Cliente Ativo; Direita: Call-Back.

Na arquitetura Cliente Ativo há um tráfego adicional no sentido cliente-servidor representado pelas solicitações de atualização de tela. Já na arquitetura Call-Back, houve uma redução de cerca de 34% no total de pacotes enviados no mesmo sentido uma vez que o processo de atualização de tela passa a ser controlado pelo servidor sem a necessidade de solicitações por parte dos clientes. O número de atualizações do estado do jogo enviadas do servidor para os clientes não apresentou diferença para as duas arquiteturas.

Em ambas arquiteturas foi verificada uma mesma quantidade média de pedidos de movimentação enviados por cada cliente, demonstrando que em todos os testes a atividade dos jogadores foi semelhante.

4.2. Perda de Eventos

Na figura 7 apresentamos o somatório do número de solicitações de movimentação dos três clientes e o número de atualizações de tela enviadas pelo servidor para cada cliente.

Considerando que cada movimento de jogador representa uma mudança no estado do jogo, poderíamos pensar que o número de respostas do servidor deveria corresponder ao número total de pedidos de movimentação dos jogadores. No entanto, como em um jogo de ação ocorrem freqüentes pedidos de movimentação de vários clientes em um mesmo instante, as mudanças no estado do jogo referentes a essas movimentações podem ser repassadas para os clientes em um único pacote.

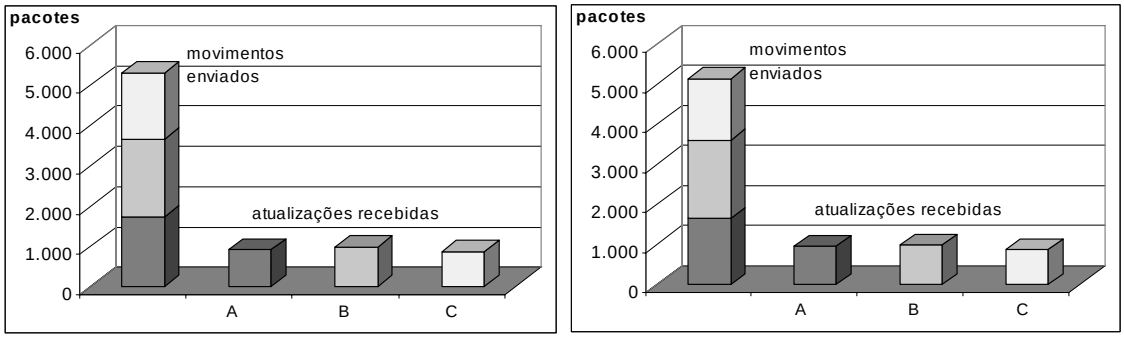

Figura 7 - Esquerda: Cliente Ativo; Direita: Call-Back

Com base nisso e considerando que os três jogadores estão enviando movimentações simultaneamente, para avaliar a perda de eventos, podemos considerar o pior caso, o qual é representado pelo maior número de pedidos de movimentações enviados (cliente B) e pelo menor número de atualizações de tela recebidas (cliente C). A perda de eventos, então, foi da ordem de 45%, na pior situação. Ainda assim, para o jogo utilizado, tal perda não é significativa, já que não afeta a consistência de atualização de tela do usuário e este valor foi alcançado após medições e ajustes na aplicação. Uma maior taxa de recebimento de atualizações de tela não afeta a qualidade de apresentação para o usuário, em virtude da não percepção visual desses movimentos adicionais.

Vale acrescentar que as medições apresentadas nos gráficos acima foram do jogo em sua capacidade máxima de atividade, ou seja, todos os clientes enviando pedidos constantes de movimentação. Em uma outra situação, onde os jogadores se movimentam mais lentamente, a taxa de perda de eventos será significativamente menor, podendo chegar a ser nula.

4.3. Consumo de Banda Passante

Para efeito de avaliação do consumo de banda exigido pelo jogo analisado, utilizando as mesmas medições anteriores, computamos o tamanho, em bits, de todos os pacotes referentes à aplicação.

A figura 8 mostra os resultados obtidos, representados em Kbits por segundo (Kbps), para os dados enviados ou recebidos pelos clientes durante o tempo de medição.

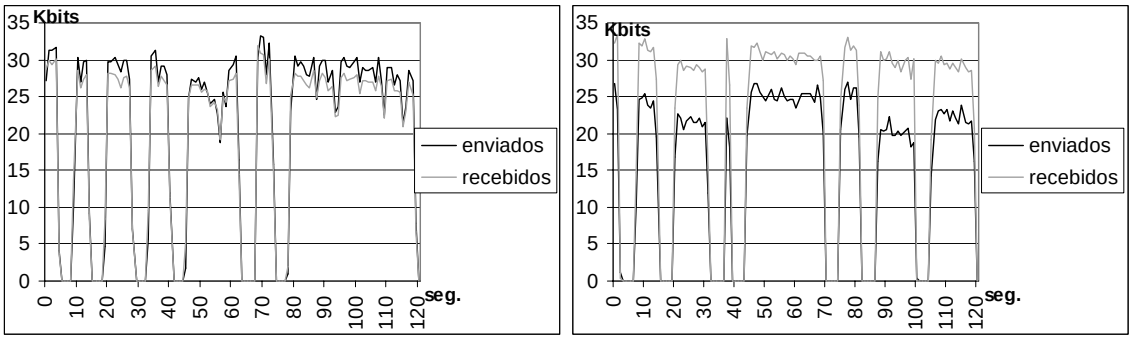

Figura 8 - Esquerda: Cliente Ativo; Direita: Call-Back

Podemos notar que, com relação ao consumo de banda, as duas arquiteturas apresentaram, em média, o mesmo desempenho, apesar da redução significativa do número de mensagens enviadas no sentido do cliente para o servidor na arquitetura Call-Back, como mostrado anteriormente. Isto se dá em virtude do aumento de mensagens do protocolo GIOP utilizado pelo CORBA. Como na arquitetura Call-Back é utilizado um maior número de objetos distribuídos que na Cliente Ativo, essas mensagens chegam a ocupar a rede quase na mesma proporção - em termos de consumo de banda - que os pacotes que deixaram de ser enviados pelo cliente nesta arquitetura.

Lembrando mais uma vez, esta situação apresentada vale apenas para a pior situação de consumo de banda, ou seja, com todos os clientes efetuando solicitações constantes de movimentação ao servidor, que por sua vez retorna atualizações de tela a uma taxa máxima aceitável para apresentação aos jogadores.

Os momentos em que o tráfego cai para zero representa as paradas do jogo no instante em que a caça é pega, o que se mantém por aproximadamente cinco segundos para que este evento seja percebido por todos os jogadores.

Levando-se em consideração apenas o tráfego no sentido do cliente para o servidor, a diminuição no consumo de banda na arquitetura Call-Back é claramente notada.

As medições realizadas mostraram que a taxa de transmissão máxima requerida pelo jogo *NetMaze* é da ordem de, aproximadamente, 33 Kbps. Isto demonstra a possibilidade dessa aplicação ser executada na Internet, inclusive através de uma conexão discada.

Para avaliação das arquiteturas propostas em uma situação diferente de comportamento dos jogadores, efetuamos as mesmas medições de tráfego executando o jogo com menor número de eventos enviados pelos clientes. Durante os dois minutos de teste, efetuamos breves movimentações dos jogadores A, B e C por volta dos instantes 30, 50 e 70 segundos, respectivamente, para que fosse medido o tráfego gerado na rede.

O intuito deste teste foi determinar o comportamento das arquiteturas Cliente Ativo e Call-Back em jogos com características de pouca ação. A figura 9 mostra os resultados obtidos.

Esta situação é típica em jogos de estratégia, como por exemplo o jogo de xadrez, onde não há pedidos constantes de movimentação e nem necessidade de atualizações de tela a cada

instante. Por outro lado, uma mudança do estado do jogo deve ser informada imediatamente a cada jogador.

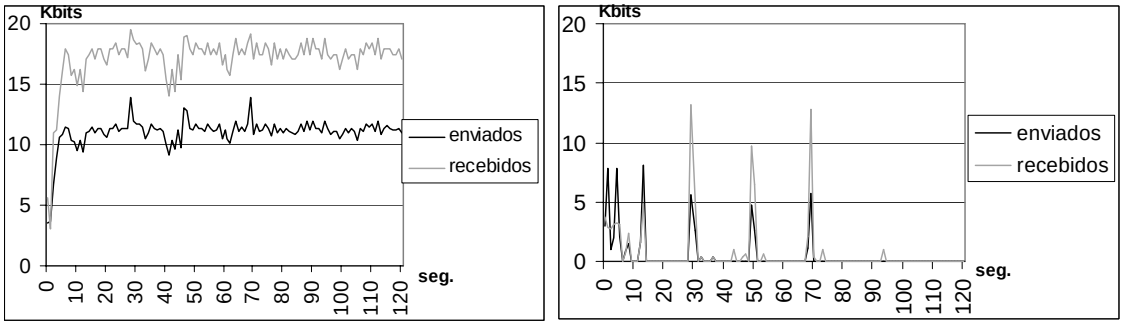

Figura 9 - Esquerda: Cliente Ativo; Direita: Call-Back

Neste caso, podemos notar claramente que a arquitetura Call-Back mostra-se mais indicada por não apresentar tráfego desnecessário na rede quando não há movimento dos jogadores. Os picos de tráfego observados na arquitetura Call-Back no início das medições representam as mensagens CORBA de pedidos de localização e conexão entre objetos remotos na inicialização do jogo.

5. CONCLUSÃO

Para jogos de ação, foi verificado que o desempenho conseguido com a implementação das arquiteturas Cliente-Ativo e Call-Back é similar em termos de consumo de banda passante. Por outro lado, uma diferença considerável foi observada em relação ao número de pacotes da aplicação enviados dos clientes para o servidor. A redução na troca de pacotes apresentada na arquitetura Call-Back 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.

As características de um jogo de ação, onde um grande número de eventos são enviados pelos clientes para processamento no servidor, dificultam a manutenção da consistência dos dados na tela de cada cliente, implicando assim, na necessidade de constantes atualizações de tela. No entanto, pudemos verificar que não se faz necessária uma atualização para cada movimento efetuado pelos jogadores. No jogo desenvolvido foi constatado um nível de perda de eventos da ordem de 45%, no pior caso, para ambas as arquiteturas avaliadas. No entanto, a qualidade da aplicação apresentou-se satisfatória apesar da perda. Assim, podemos concluir que a perda controlada de eventos pode ser aceita no desenvolvimento de jogos de ação, sendo necessário apenas ajustá-la de acordo com a qualidade final desejada.

O jogo *NetMaze* apresentou-se como uma aplicação eficiente para a avaliação das arquiteturas. Apesar de possuir características típicas de um jogo de ação, foi possível a utilização do mesmo para simular um jogo com características diferentes, onde os jogadores apresentam um comportamento menos ativo. Assim, pudemos avaliar as arquiteturas de forma mais ampla e verificamos o melhor desempenho em termos de consumo de banda da arquitetura Call-Back para jogos de pouca movimentação.

Como continuação desse trabalho, temos como objetivo a aplicação da arquitetura com Serviço de Eventos ao jogo desenvolvido e subseqüente avaliação de resultados e comparação com as arquiteturas já implementadas, contribuindo para incrementar os estudos sobre arquiteturas para jogos distribuídos.

6. AGRADECIMENTOS

Agradecemos à CAPES, ao CNPq e à EMBRAPA pelo apoio financeiro.

RON, B. N. *CORBA on the Web*, McGraw-Hill, United States, 1998.
HABERMEIER, B. *Internet Based Client/Server Network Traffic Reduction*. http://www.bolt-action.com
FIGUEIRA, C. *JEOPS – The Java Embedded Object Production System*, UFPE, 1999
HORSTMANN, C., CORNELL, G. *Core Java 2 Fundamentals - Volume I*, Sun Microsystems Inc., California, 1999
JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*, John wiley & Sons, Inc, 1991
LAMOTHE, A. *Tricks of the Windows Game Programming Gurus - Fundamentals of 2D and 3D Game Programming*, Sams, United States, 1999
ORFALI, R., HARKEY, D. *Client/Server Programming with Java and CORBA*, John Wiley & Sons Inc., United States, 1998.
RUSSEL, S. J., NORVIG, P. *Artificial Intelligence: A Modern Approach*, New Jersey, Prentice-Hall Inc., 1995.
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
TCPDUMP - Lawrence Berkeley National Laboratory, Network Research Group - ftp://ftp.ee.lbl.gov/tcpdump.tar.Z
TIRAKIS, A. ET AL *Distributed Multimedia Architectures - State-of-the-Art Report* http://viswiz.gmd.de/DVP/Public/deliv/deliv.222/act222.htm
VisiBroker – Programmer's Guide, Inprise Corporation, Inc., 1999 – http://www.borland.com/techpubs/books/vbj/vbj40