Um estudo do uso de SGBDs relacionais em arquiteturas de jogos multi-jogador

Lauro Eduardo Kozovits
Computer Science Department, PUC-Rio
lauro@inf.puc-rio.br

Rubens Nascimento Melo
Computer Science Department, PUC-Rio
rubens@inf.puc-rio.br

Bruno Feijó
Computer Science Department, PUC-Rio
bruno@inf.puc-rio.br

PUC-RioInf.MCC35/03 October, 2003

Abstract: Massively Multiplayer game technology is both a growing and profitable business and a rich field, nowadays, of research due to its multidisciplinary nature. Although being a specialized field in networked virtual environments (Net-VE), multiplayer games have some unique requisites and demand some technologies not common to all Net-VE applications. Therefore, an analysis of the use of RDBMSs in those game architectures, became interesting in order to investigate the possibilities of new uses or improvements. A new technique to obtain game session balance, a feature often desired in those architectures, is proposed based on the use of RDBMSs.

Keywords: Distributed virtual environment, Net-VE, multiplayer game technology, client-server design, peer-to-peer design, RDBMS, game balance, scalability.

Resumo: A tecnologia de jogos com número elevado de jogadores é tanto um negócio crescente e rentável quanto um rico campo de pesquisas devido a sua natureza multidisciplinar. Embora sejam uma especialização dentro da área de ambientes virtuais distribuídos, jogos multi-jogador possuem certos requisitos e demandam certas tecnologias que os tornam bem particulares. Por isso, torna-se interessante uma análise do uso de SGBDs relacionais nestas arquiteturas de jogos, a fim de investigar a possibilidade de novos usos ou melhorias. Uma nova técnica para obtenção de balanceamento de sessões de jogos, uma característica desejável nestas arquiteturas, é proposta baseada no uso de SGBDs relacionais.

Palavras-chave: Ambientes virtuais distribuídos, Net-VE, tecnologia de jogos multi-jogador, arquitetura cliente-servidor, arquitetura *peer-to-peer*, SGBDs relacionais, balanceamento de jogos, escalabilidade.

This work is being sponsored by CNPq and ICAD.

1. Introdução

Jogos com número elevado de jogadores constituem uma área multidisciplinar inserida formalmente (vide figura 1) na grande área ambientes virtuais distribuídos ligados por rede.

Uma busca sem refinamento pela expressão "multiplayer games" em portais científicos da Internet como (ACM, 2003) e (CITESEER, 2003) retorna cerca de duas centenas de trabalhos no primeiro portal citado e uma centena no segundo, sendo que, a maioria deles, à partir do ano 2000. No entanto, a tecnologia de jogos comerciais em rede e o interesse do público em geral pelo tema é algo presente desde o início da década de 90 (vide figura 2).

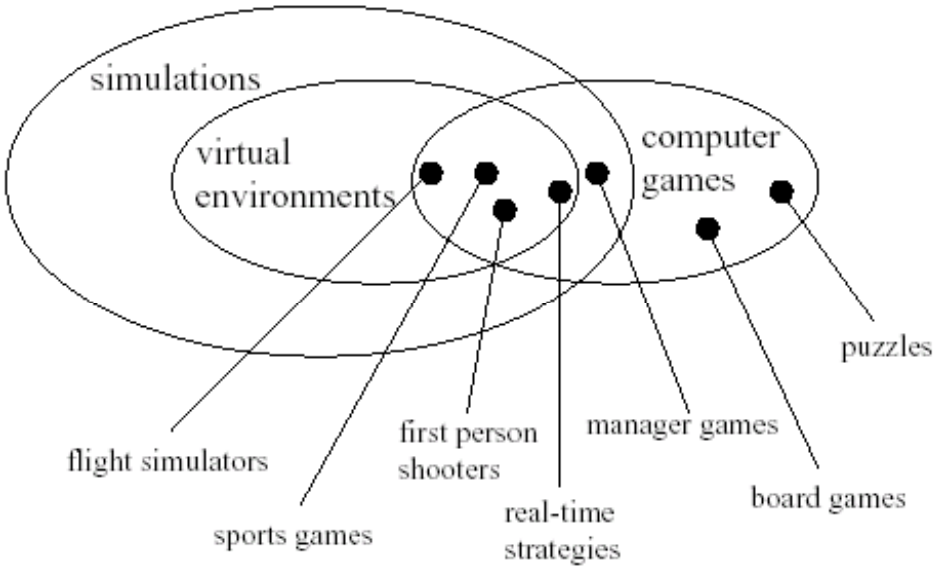

figura 1: classificação parcial de jogos e relacionamento entre as áreas de jogos, simulação e ambientes virtuais segundo (Smed, 2002).

Isto nos leva a pensar que ocorreu um certo distanciamento dos meios acadêmicos em relação a pesquisas nesta área num passado recente. Podemos tentar especular fatores que podem ter influenciado este comportamento do meio acadêmico nos próximos três parágrafos.

Não é fácil justificar um trabalho de pesquisa propondo uma nova técnica, quando uma outra já existente e bem aceita é deixada em segundo plano por ser de custo elevado para a maioria das pessoas ou por não estar ainda disponível em larga escala. Por exemplo, quando falamos em jogos multi-jogador, implicitamente estamos levando em conta jogadores ligados pela Internet em geral utilizando modem, ao contrário de pesquisas de simuladores (sistemas médicos, militares, etc) onde são utilizadas redes de alta velocidade e tecnologias como broadcast (inexistente na Internet) ou multicast (tecnologia ainda não disponível em larga escala na Internet)

Também existe uma certa complexidade na elaboração de um trabalho de pesquisa nesta área. Isto ocorre porque, muitas vezes, uma pesquisa visa obter uma técnica geral, estabelecer um framework ou mesmo formalizar uma base teórica, o que é complexo numa área com tantas categorias e sem uma fronteira distinta. Todos os jogos em rede estão sob a classificação "jogos multi-jogador" (vide figura 1).

Finalmente, o objetivo de um jogo que, ao contrário de uma simulação física ou visual, é obter o entretenimento do usuário, implica no uso de uma métrica de qualidade ainda não formalizada que é a "boa jogabilidade" do sistema. É fácil constatar o sucesso de certos jogos já

lançados no mercado, porém muito difícil prever os resultados de um sistema ainda a ser construído. Assim sendo, há ainda uma grande dose de empirismo no levantamento dos requisitos de projeto deste tipo de sistema.

O que foi dito pode, eventualmente, justificar toda esta antiga reserva sobre o assunto "jogos multi-jogador" como objeto de pesquisa.

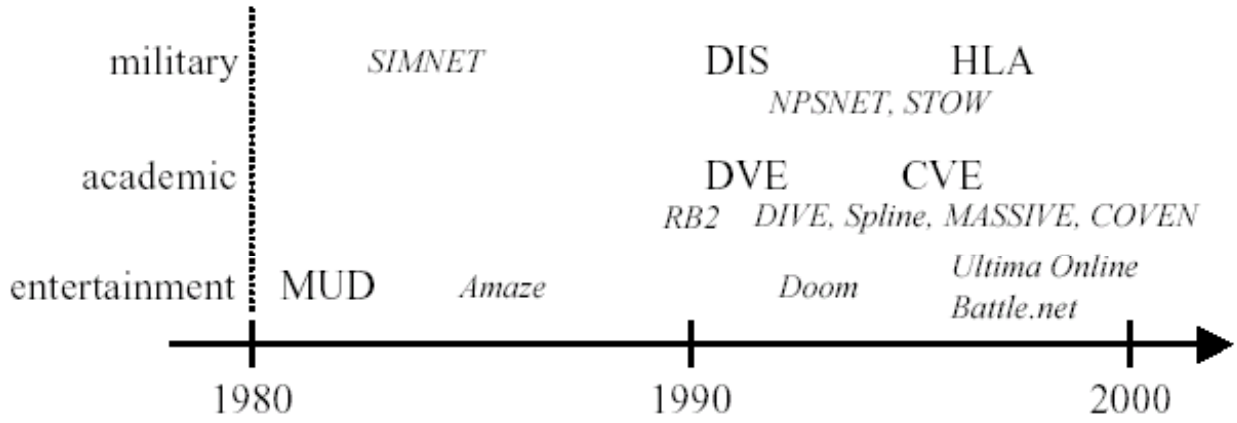

figura 2: comparação histórica de ambientes virtuais distribuídos ligados por rede de acordo com (Smed, 2002).

Nos últimos anos, esta forma de entretenimento deixou de ser um nicho, ficando cada vez menos restrita a uma faixa etária ou classe social. Além disso, os ótimos resultados financeiros obtidos por gigantes dos setores de informática ou entretenimento como Disney (vide figura 3), Microsoft, Sony, Electronic Arts, etc, pode ser sugerido como uma explicação para o estímulo, seguido de rápido crescimento do número de trabalhos científicos nos últimos três anos.

figura 3: imagens do jogo *massive multiplayer* ToonTown (2003) da Disney

Um dos primeiros problemas a serem abordados e, ainda, freqüentemente considerado é como contornar as limitações da largura de banda da rede e minimizar o envio freqüente de mensagens de atualização entre os participantes de um jogo ou simulação. Estes trabalhos estão relacionados a tecnologia de *dead-reckoning*, termo original de técnicas de navegação e introduzido nesta área no simulador militar SIMNET (Pope, 1989). Uma análise detalhada desta linha de

pesquisa, bem como uma proposta de generalização de *dead-reckoning* pode ser vista em (Szwarcman, 2001).

No presente trabalho, explorando o aspecto multidisciplinar desta área, são abordados os usos de SGBDs relacionais na arquitetura de jogos multi-jogador. Ao final, é proposta uma nova técnica de balanceamento colaborativa, numa arquitetura em três camadas, destinada à obtenção e manutenção da ocupação ótima de sessões *peer-to-peer* utilizando SGBDs relacionais.

2. Uso de orientação a dados em jogos multi-jogador

Neste item, é analisado o uso da técnica de orientação a dados no projeto de jogos multi-jogador. Em seguida, é detalhado o uso de SGBDs relacionais nestes ambientes.

A técnica de desenvolvimento de sistemas orientados a dados (*data-driven systems*) (Rabin, 2000) está se tornando um padrão dentro do acervo de ferramentas para criação de jogos multi-jogador. Jogos de todos os gêneros, desde RPG (*role-playing games*) a jogos de estratégia, assim como FPS (*first-person shooters*), têm usado sistemas orientados a dados possibilitando flexibilidade e extensibilidade durante o seu desenvolvimento. Esta técnica permite ainda, que os próprios usuários façam alterações ou "*mods*" num jogo já pronto.

Ao tratar de jogos com número elevado de jogadores (*massive multiplayer* ou *MMP*), temos que considerar estes sistemas como sistemas "vivos", ou seja, em permanente evolução. Considerando esta característica, os desenvolvedores devem, com maior interesse ainda, se beneficiar da flexibilidade promovida pelo desenvolvimento de sistemas utilizando orientação a dados.

2.1 As idéias componentes de projeto orientado a dados

As principais idéias de projeto orientado a dados descritas em (Rabin, 2000) são:

1. Permitir que o sistema obtenha seus dados a partir de repositórios de dados facilmente acessíveis a usuários comuns como arquivos texto, xml, bancos de dados relacionais, etc.

2. Não embutir constantes nem dados de objetos em código alterável apenas por programadores.

3. Evitar "amarrar" ao código, decisões de projeto que possam vir a ser modificadas posteriormente. Por exemplo, se for definido que um jogo terá apenas duas janelas com vistas do ambiente virtual, sugere-se fazer o projeto de tal forma genérico, que o número de vistas possa ser especificado em um repositório de dados. Isto, além de facilitar a experimentação de novas alternativas, possibilita que eventuais mudanças no curso do projeto impliquem em menor retrabalho.

4. Usar linguagens de *scripts* simples para permitir que leigos em programação possam fazer alguma experimentação num ambiente bem acessível.

5. Em oposição a idéia anterior, não permitir o uso de *scripts* tão complexos que levem os *designers* a se tornarem programadores.

6. Seguir as normas de engenharia de software, evitando a criação de códigos específicos semelhantes entre si e buscar a criação de funções gerais. Naturalmente, um bom projeto de classes e o uso adequado do mecanismo de herança podem atender bem a este requisito.

7. Criar ferramentas para manipular os dados ou utilizar ferramentas já disponíveis. Apesar do trabalho extra inicial, a compatibilidade com ferramentas de modelagem existentes ou a criação de um editor de níveis ou editor de *scripts* de um jogo, permitirão que mais integrantes do time de desenvolvimento possam se envolver no projeto, já na fase inicial.

2.2 Benefícios de sistemas orientados a dados em jogos MMP

Do ponto de vista técnico, uma característica de jogos MMP é que eles requerem uma enorme quantidade de código com uma longa vida de duração, tanto na etapa de "pré-lançamento", quanto após este evento. Além disso, é requerida uma quantidade de conteúdo ainda mais expressiva. Estas características criam certos desafios que só podem receber tratamento adequado pelo uso de orientação a dados. A seguir, são apresentados os três principais benefícios da técnica de desenvolvimento de sistemas orientados a dados.

Menor custo de desenvolvimento

São três os principais benefícios que o uso desta técnica permite para a redução do custo de desenvolvimento:

- aumento da capacidade produtiva de um maior número de membros do time de desenvolvimento;
- experimentação imediata das idéias por parte dos "*game designers*" e
- redução da quantidade de código a ser desenvolvida

Sistemas orientados a dados dão maior poder aos não programadores que podem obter um maior *feedback* de suas idéias. Este *feedback* é possível pela interação direta com o modelo de dados, sem ter que esperar a criação das primeiras versões por parte dos programadores.

Isto permite um envolvimento maior de todo o time de criação para experimentação de suas idéias e para identificar as partes boas e ruins do projeto. Isto evita retrabalho até mesmo de programação. Este envolvimento pode, com o uso desta técnica, ocorrer já nos primeiros estágios do processo de criação do jogo.

O uso do conceito de *refactoring* (Fowler, 1999) em seções de código repetitivas, possibilita o desenvolvimento de menos código. Menos código a ser desenvolvido significa também, obviamente, menos tempo de desenvolvimento e menor custo do sistema.

Menor custo de manutenção

Uma extensão do que foi dito no ítem anterior é o fato de que menos código também significa menor custo de manutenção.

Particularmente em jogos MMP, teremos milhares de usuários *online* submetendo a *stress* as diversas situações e níveis do jogo (e as correspondentes linhas de código). Assim sendo, fazer mudanças apenas em *scripts* ou em repositórios de dados tende a ser algo mais confiável, requerer menos esforço na operação e nos testes e, portanto, implicar também num custo menor de manutenção.

Criação de conteúdo mais rápida

Jogos MMP são ricos em conteúdo e sua criação deve ser permanente para estimular o jogador a continuar cliente do empreendimento.

A separação bem definida entre conteúdo (dados) e lógica (programação) estabelecida pelo projeto orientado a dados, permite que a criação de conteúdo se dê, antes mesmo do início da programação do sistema propriamente dito.

Durante o processo de desenvolvimento, as duas etapas podem ocorrer em paralelo, permitindo um prazo menor de desenvolvimento do produto.

Também após o lançamento do produto, esta técnica de projeto possibilita que a criação de conteúdo adicional possa ser feita, em geral, sem depender muito de modificações no código, geralmente demoradas e arriscadas.

2.3 Tipos de repositórios de dados

Arquivos texto

Arquivos texto representam a solução mais simples para o desenvolvimento de sistemas orientados a dados e apresentam uma considerável facilidade de uso por parte da equipe que deseja experimentar modificações no sistema. Contudo, a ausência de estrutura e a dependência de um *parsing* específico diminuem seu uso generalizado. Este problema dá-se pelos seguintes complicadores:

- dependência de plataforma; em alguns sistemas, a mudança de linha é dada pelos bytes "CRLF" em outros apenas "LF".
- dependência de formato; não é especificado, num arquivo texto comum, se o texto é ASCII, UNICODE ou mesmo algo para permitir ao programa leitor suportar um ou outro formato.
- delimitadores "fracos" geralmente são usados para separar os dados como, por exemplo, tabuladores, aspas ou vírgulas, o que torna o documento muito vulnerável a erros de "sintaxe" introduzidos por quem manipula diretamente este arquivo texto

XML

O uso de XML (*eXtensible Markup Language*) (XML, 2003) é um padrão para definição de dados bastante flexível, pois permite uma maior estruturação destes dados e tem o *parsing* facilitado pela existência de muitas bibliotecas de software de apoio. Contudo, não é tão legível para nós humanos quanto os arquivos texto e sua apresentação, extremamente verbosa, conduz a geração de arquivos muito grandes.

Linguagens de *Script*

Linguagens como Python (2003), TCL (2003) ou Lua (2003), por sua natureza interpretada, além de recursos específicos que possibilitam seu uso ótimo como linguagem de *script* permitem também sua utilização como repositórios de dados.

Por serem linguagens de programação, a informação nelas contida dispensa o uso de *parsing* permitindo fácil integração com a linguagem de codificação do servidor de jogos.

Também por serem linguagens de programação, é possível um controle maior da informação que venha a ser nelas embutida. Por outro lado, isto também implica na aquisição de conhecimento mais técnico por parte do *game-designer* ou de algum outro integrante da equipe de criação não programador.

Bancos de dados relacionais

Bancos de dados relacionais são repositórios de dados por natureza e têm um conjunto único de vantagens e desvantagens se comparados com técnicas mais simples. A maneira de obtenção dos dados é padrão e permite a interface com praticamente qualquer linguagem de programação de alto nível.

Além disso, a estrutura que pode ser conferida aos dados, a rica possibilidade de consultas otimizadas, bem como as características de preservação das propriedades ACID em transações torna, em muitos sistemas, principalmente em jogos MMP, uma escolha inevitável.

Há porém, que se mencionar desvantagens que são, principalmente, a complexidade de administração e os custos elevados de licenciamento e manutenção quando comparados aos outros repositórios de dados citados anteriormente. Além disso, é necessária a presença de algum tipo de visualizador ou editor para tornar os dados armazenados facilmente manipuláveis por parte da equipe não técnica de produção de um jogo.

3 Uso de bancos de dados relacionais em jogos MMP com arquitetura cliente-servidor

Apesar da existência de muitas propostas na literatura propondo o uso de ambientes virtuais *peer-to-peer* em conjunto com técnicas de *broadcast* (utilizada em redes locais) e *multicast*, muitos jogos comerciais MMP são projetados utilizando-se a arquitetura cliente-servidor (Bogojevic, 2003) (vide figura 4).

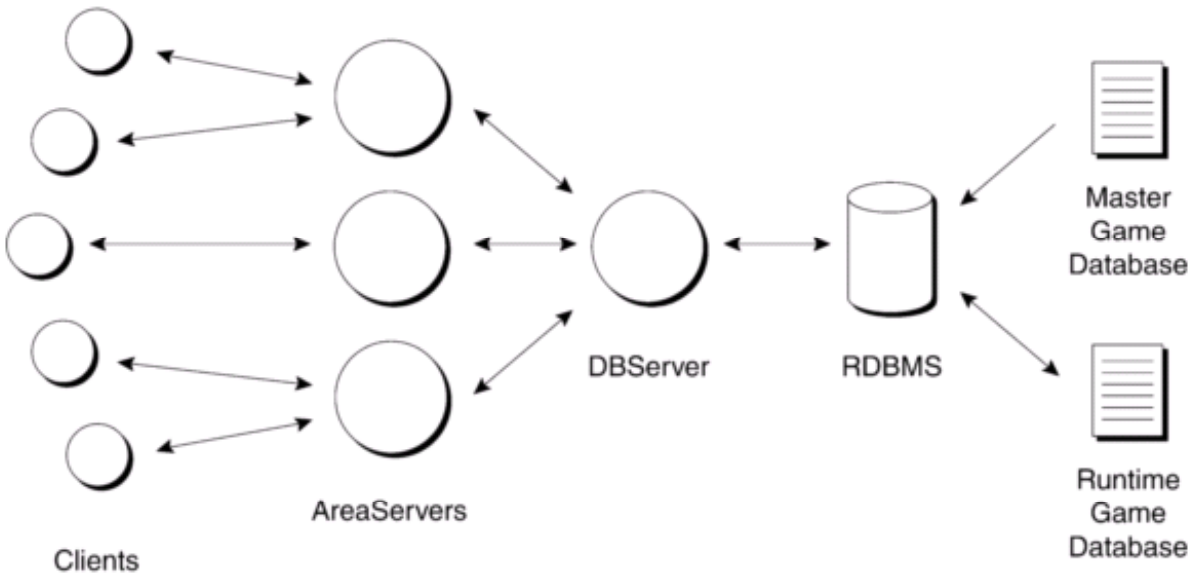

figura 4: arquitetura típica de um jogo *massive multiplayer* ilustrando clientes conectados a servidores de áreas de interesse (ou segmentos) do ambiente virtual (Bogojevic, 2003)

Sistemas de bancos de dados relacionais estão presentes no mundo de aplicações de negócios, como um sólido elemento de sustentação destas aplicações, há mais de duas décadas. Por isso, também é natural que os sistemas MMP usem SGBDs relacionais no apoio a esta arquitetura cliente-servidor.

Isto dá-se por duas razões principais:

- necessidade de sistema de cobrança (*billing*) e validação de usuários como em qualquer outro negócio;
- necessidade de preservar com segurança os estados correntes de um jogador e do mundo virtual modelado.

Após décadas de aprimoramento de software e hardware, os SGBDs relacionais têm mostrado números de desempenho expressivos como a capacidade de execução de muitos milhares de transações por segundo. Isto, junto à flexibilidade da linguagem SQL, permite a implementação trivial, pelo menos em teoria, de um servidor de jogos para armazenar e recuperar informações de estado do mundo virtual e de seus jogadores.

As características e requisitos de um MMP, entretanto, não são as mesmas de uma aplicação comercial convencional.

Um primeiro empecilho é de ordem não técnica: o custo do SGBD relacional a ser empregado. Os custos de desenvolvimento de certos tipos de MMP já são notoriamente altos. Para permitir um desempenho razoável, há um custo também expressivo em servidores multi-processados de alto desempenho. Ocorre que muitas licenças de SGBDs relacionais têm um custo proporcional, em muitos casos, ao desempenho e quantidade de processadores disponíveis. Isto pode inviabilizar o uso comercial de uma certa tecnologia de jogos ou de um certo modelo de negócios.

Outro problema é que prever o desempenho de um certo jogo é algo extremamente complicado e arriscado: os resultados só podem ser vislumbrados quando muito já foi codificado e a base de dados já está em plena atividade. Obviamente, é extremamente indesejável apresentar para o investidor um indicador de insucesso do empreendimento, apenas no ponto em que boa parte dos investimentos já foi realizada.

Finalmente, se num sistema convencional é razoável que, de vez em quando, apareça um cursor na tela "solicitando" ao usuário que aguarde a finalização de uma determinada consulta, nem sempre uma espera de alguns segundos é razoável durante uma simulação de combate em tempo real.

Descobrir e eliminar as causas de comportamento inadequado à dinâmica do jogo por parte de um SGBD relacional é algo extremamente complexo, por não ser fácil reproduzir situações intermitentes ou eventuais. Isto pode, inclusive, comprometer a aceitação do ambiente por parte dos seus jogadores ou mesmo dos investidores. Ou seja, atualmente, as métricas de performance fornecidas pelos fabricantes de bancos de dados são adequadas apenas para estimativa de desempenho de aplicações convencionais.

Assim sendo, é razoável que certos cuidados sejam tomados no uso de bancos de dados com jogos MMP.

Sugere-se, portanto, num projeto de um jogo MMP, uma etapa de validação e justificativa da necessidade de uso de um SGBD relacional para este projeto o que já é algo quase implícito numa aplicação convencional. Também é aconselhável buscar arquiteturas alternativas que possibilitem o licenciamento de um SGBD relacional de menor custo. Iremos, nos próximos tópicos, discutir técnicas para aliviar a carga de transações sobre um banco de dados e mecanismos visando sua otimização no contexto de sistemas MMP.

3.1 Uso de cache de dados

A maneira mais rápida de acessar dados é já dispor deles em memória no momento de seu uso. A implicação da adoção desta proposição é fazer uma pré-carga de toda a informação necessária para se evitar o impacto do acesso demasiado à base de dados.

Por exemplo, se os dados de um determinado ítem estão distribuídos em algumas tabelas e se, este ítem for utilizado neste ambiente, então é possível carregar toda esta informação no momento de inicialização do sistema e mantê-la em memória. Isto é particularmente útil e viável, se esta informação for imutável. Estas estruturas de dados mantidas em memória e correspondentes a tabelas dos bancos de dados são conhecidas como *lookups* (Lee, 2003).

Uma boa estratégia para dinamicamente refletir na forma de código programado as alterações feitas em tabelas da base de dados é o uso de linguagens interpretadas como Python (2003), por exemplo. Este tipo de linguagem permite o acréscimo de módulos durante a execução do sistema, para que dados obtidos de tabelas do banco apareçam como constantes ou estruturas de dados do sistema.

Um outro tipo de técnica de cache é a utilização da abordagem "just-in-time", ou seja, aguardar a ocorrência de algum evento que gere a demanda de um determinado objeto para que ele possa, à partir deste momento, ser movido para a memória. Um exemplo clássico é a informação sobre um determinado jogador que, segundo esta abordagem, deverá residir em memória apenas no momento de sua conexão com o ambiente, sendo, evidentemente, removida com a desconexão.

É fácil entender a motivação para este uso, se considerarmos uma tabela com milhões de possíveis jogadores e seus diversos atributos(energia, pontuação, dados pessoais, armamento, posição, etc) que pode ser objeto de variadas consultas combinadas.

Naturalmente, apesar da possibilidade de existência de diversos índices, a busca restrita apenas aquele conjunto online e em memória tende a ser mais eficiente e a poupar recursos do banco de dados.

É importante, é claro, considerar a existência de mecanismos que possibilitem o carregamento assíncrono de um ítem para uma estrutura de dados em memória como, por exemplo, a existência de uma *thread* no servidor de jogos para não gerar um impacto negativo na fluidez da simulação.

Um caso a ser considerado, quando se utiliza cache, é a possibilidade de dados no banco de dados ficarem desatualizados em relação ao que ocorre em memória. Isto é crítico, pois há sempre um risco de falha de software ou hardware com a conseqüente perda das informações em cache.

Além disso, todas as propriedades ACID preservadas em transações do SGBD podem ser perdidas ao se manipular dados em cache. Pode-se considerar retrabalho, uma implementação destes mecanismos de controle de transações que já estão disponíveis no banco de dados.

Obviamente a idéia da existência do cache, sob certas circustâncias, é válida e visa evitar acessos freqüentes ao banco de dados. Uma solução, então, é priorizar certas atualizações e postergar outras.

Exemplificando:

- se um certo jogador é promovido de nível ou conquista um objeto de valor real (Ward, 2003), é fundamental que este usuário possa ter a segurança de que, independentemente da implementação adotada, ele possa contar com o fato de que aquele nível ou objeto obtido esteja a ele associado daquele momento em diante. Nestas circunstâncias, é conveniente adotar-se à imediata atualização do banco de dados referente as informações presentes no cache;

- se for adotada uma política de atualização infreqüente para certos dados, por exemplo, atualizar apenas a cada cinco minutos a informação sobre a posição de um certo jogador, então o sistema poderá ficar sujeito a um fenômeno conhecido como "*warping*". *Warping* é o efeito de teletransporte de um jogador da posição atual para uma posição anterior, eventualmente correspondente a minutos atrás que foram recuperadas do banco de dados como resultado da perda do cache. É algo negativo para o ambiente, mas é uma

estratégia que elimina atualizações freqüentes sem um risco tão indesejável quanto no primeiro exemplo.

Outras alterações de dados podem provocar a invalidação de um cache que não um erro de software ou falha de hardware. Por exemplo, se alterações no poder de uma arma ou personagens, foram promovidas pelos administradores do ambiente visando um maior balanceamento do jogo *online* quanto ao equilíbrio de forças (Hanson, 2003), então pode ocorrer que os dados em cache sejam invalidados intencionalmente, para que o ambiente já se torne mais balanceado imediatamente.

3.2 Outros mecanismos para otimizar o uso do servidor de jogos e o acesso a dados

Um servidor de jogos realiza, dentro de um laço de programação contínuo, um número grande de tarefas referente ao tratamento de, eventualmente, milhares de jogadores simultâneos. Por exemplo, tratamento de colisão, registro de pontuação ou danos, modificações geométricas do ambiente, cálculos de *dead-reckoning* envolvendo posição e velocidade de jogadores etc.

Além disso, o servidor recebe e envia milhares de mensagens para manter os jogadores atualizados. Pode não ser razoável, portanto, interromper estas tarefas para esperar o resultado de uma consulta a base de dados de forma síncrona.

Acesso assíncrono, contudo, não é algo embutido em SGBDs relacionais. É possível, como já mencionado, utilizar *threads* para que o servidor de jogos não fique esperando a resposta de um comando SQL. É fundamental, neste caso, o uso de uma biblioteca *multi-thread* de acesso ao banco de dados, além, é claro, de estabelecer um mecanismo robusto para comunicação entre as *threads* de acesso ao banco de dados e a *thread* servidora de jogos.

Uma alternativa importante ao uso de *threads* para promover o isolamento do servidor de jogos do envio e tratamento de comandos SQL, é incluir na arquitetura, um servidor dedicado a acessar o banco de dados e realizar a comunicação entre os demais servidores por meio de envio de mensagens. Quando o servidor de jogos (vide figura 4) desejar realizar uma consulta ou uma atualização de dados, ele envia uma mensagem para este servidor de acesso a dados e continua realizando outras atividades.

Ao receber uma mensagem como resposta à sua solicitação, o servidor irá tratá-la de modo semelhante ao tratamento dado a mensagens vindas dos clientes do jogo, possivelmente numa maior prioridade. Esta estratégia de implementação nada mais é que uma arquitetura multi-camadas.

Esta arquitetura é mais especializada, conceitualmente mais simples, pelo menos em termos de projeto e pode, eventualmente, possibilitar vantagens no custo do licenciamento do SGBD relacional, por permitir a utilização de um servidores de dados com processadores menos poderosos.

Finalmente, devemos levar em conta também numa arquitetura de jogos, todo o conhecimento obtido no desenvolvimento de aplicações convencionais utilizando SGBDs relacionais nas últimas décadas. Por exemplo, à semelhança de aplicações comerciais comuns, deve-se utilizar *stored procedures* ao se identificar consultas freqüentes e parametrizáveis.

A modelagem de dados deve respeitar o modelo entidades-relacionamentos, um assunto de domínio comum e bastante discutido (Date, 1999) . Também importantes são as tarefas de "*database-tuning*" como a criação de índices quando conveniente, tendo-se o cuidado que índices que otimizam consultas podem porém, atrasar o processamento de operações de remoção.

Uma das regras mais importantes para tornar os bancos relacionais mais robustos às mudanças dos dados feitas pelas aplicações é o uso de normalização. A normalização minimiza o armazenamento redundante e torna mais elegante o manuseio dos dados: um "*schema*" aderente a um certo conjunto de regras de normalização é dito normalizado (Codd, 1972).

Apesar do que foi dito no parágrafo anterior é raro, devido a considerações de performance, encontrar aplicações "reais" cujo *schema* esteja inteiramente normalizado. Um administrador de dados experiente irá, a partir de uma visão lógica do *schema*, obter uma implementação "física" destinada a uma performance máxima para contemplar consultas freqüentes. Há portanto, um balanceamento entre a performance desejada e um eventual trabalho extra para manter o *schema* não normalizado.

3.3 **Características das propriedades ACID de transações de SGBDs**

Um dos mais antigos e mais importantes conceitos da teoria de bancos de dados é a coleção das chamadas propriedades ACID. Seu nome vem do acrônimo Atomicity, Consistency, Isolation e Durability e define quatro grandes objetivos que um SGBD deve alcançar.

Qualquer banco de dados que falha na apresentação destas quatro características não pode ser considerado confiável.

Vamos examinar cada uma destas características em detalhe:

Atomicidade implica que as modificações na base de dados devam seguir a regra do "tudo ou nada". Cada transação é "atômica". Se uma parte da transação falhar, a transação inteira irá falhar. É crítico que cada sistema gerenciador de banco de dados mantenha a natureza atômica das transações a despeito de qualquer falha do SGBD, sistema operacional ou hardware.

Consistência implica que somente dados válidos sejam escritos na base de dados. Se, por alguma razão, dentro de uma transação for executada alguma operação violando as regras de consistência, a transação inteira será desfeita (*rolled back*) e a base de dados será restaurada a um estado consistente e de acordo com estas regras. De outra maneira, se uma transação for bem sucedida, a base de dados irá para um novo estado, também consistente com as regras.

Isolamento requer que múltiplas transações ocorrendo ao mesmo tempo não causem impacto negativo em outras em execução. Exemplificando com uma situação de jogo, se um jogador A estiver obtendo a posição "x1,y1,z1" de um jogador B e, "ao mesmo tempo" o jogador B estiver fazendo a atualização de sua posição para "x2, y2, z2", não haverá a possibilidade de A recuperar algo inconsistente como, por exemplo "x1, y2, z1".

Portanto, ou a posição recuperada será a anterior ou será a nova e não algo intermediário, parcial ou inconsistente. Observe com este mesmo exemplo, que isolamento não implica em quem irá executar primeiro e sim numa não interferência entre as duas ações.

Durabilidade assegura que qualquer transação efetuada com sucesso ("committed") não irá ser perdida. Durabilidade é assegurada através do uso de *backups* da base de dados e *logs* de transações que facilitam a restauração de transações bem sucedidas a despeito de qualquer falha de software ou de hardware.

As operações acima constituem os fundamentos de qualquer modelo de transações. Em geral, quando falamos de *On Line Transaction Processing* - OLTP é fundamental a presença das características ACID numa base de dados para suportar a computação completa de uma transação de negócios em tempo real.

Em particular, a presença das características ACID é útil na sincronização e na operação correta de muitos sistemas distribuídos. Jogos são apenas um exemplo.

A técnica que proposta faz uso, ao utilizar comandos SQL, do fato de que as propriedades ACID são preservadas nas transações realizadas no SGBD relacional.

Usamos uma arquitetura em três camadas, onde a camada de apresentação (*first-tier*), que interage com o jogador, invoca a camada de controle (*middle-tier*) para operar transformações atômicas, consistentes, isoladas e duráveis que são, por sua vez, intrínsecas da base de dados componente da última camada (*third-tier*)

4. Técnica proposta

Em muitos jogos multi-jogador é necessário cuidar da persistência e permitir a evolução consistente do ambiente. Torna-se, portanto, necessária a presença de SGBDs relacionais de acordo com o que está apresentado neste trabalho.

Além disso, bancos de dados, têm um papel importante na validação de usuários em ambientes multi-jogador cliente-servidor e também na função de *masterserver* em ambientes *peer-to-peer*, ou seja, na função de identificação de novas sessões abertas a novos jogadores.

Na arquitetura híbrida cliente-servidor/*peer-to-peer* proposta em (Kozovits, 2003), torna-se necessário o uso de uma técnica para obter a ocupação ótima e balanceada das diversas sessões *peer-to-pe*er que compõem o ambiente (vide figura 5). Esta técnica irá tirar proveito da presença de um SGBD relacional cuja existência nestas arquiteturas já é justificável como demonstrado em tópicos anteriores.

Como não há um programa servidor central de jogos, um problema com o uso de uma técnica distribuída e colaborativa para este balanceamento é o fato de que cada *peer* poderia tentar executar uma ação já iniciada por outro. Porém, se for utilizado um banco de dados relacional, pode-se aproveitar o fato de que os comandos SQL garantem a manutenção das propriedades ACID das transações efetuadas, o que permite obter a sincronização e o controle das ações iniciadas independentemente por cada *peer* responsável (*host-peer*) por uma determinada sessão.

A técnica

Considerando que o volume de mensagens de atualização em arquiteturas peer-to-peer ou simplesmente P2P, é proporcional ao quadrado do número de jogadores em uma sessão, torna-se imperativo restringir o número de jogadores em cada uma destas sessões ou salas.

Pode-se avaliar, para um determinado jogo, o número máximo de jogadores, tomando-se como base o número médio de mensagens a serem trocadas, sua freqüência e tamanho, assim como a largura de banda da conexão.

Um valor típico para este número máximo por sessão P2P de jogos de ação pela Internet conectados por modem de 56kbps é de 8 a 16 jogadores, um número discreto, portanto.

Como a arquitetura híbrida *peer-to-peer* e cliente-servidor proposta em (Kozovits, 2003) prevê a criação de jogos de estratégia (cliente-servidor) integrados a diversas salas (*peer-to-peer*) onde jogadores executam jogos de ação, é importante acrescentar a este trabalho um mecanismo para definir a formação automática destas salas e permitir seu uso em condições ótimas (balanceadas).

Alguns critérios devem ser atendidos visando atingir esta situação ótima:

- evitar que salas fiquem vazias ou com poucos usuários, buscando-se, permanentemente, a formação de salas com o número máximo de usuários o que minimiza o tráfego de mensagens entre jogadores *host-peer* (veremos a seguir a definição) e o servidor SQL.

- possibilitar que um usuário, ao se conectar, tenha sempre outros jogadores para interagir, o que também é fundamental para a jogabilidade do ambiente

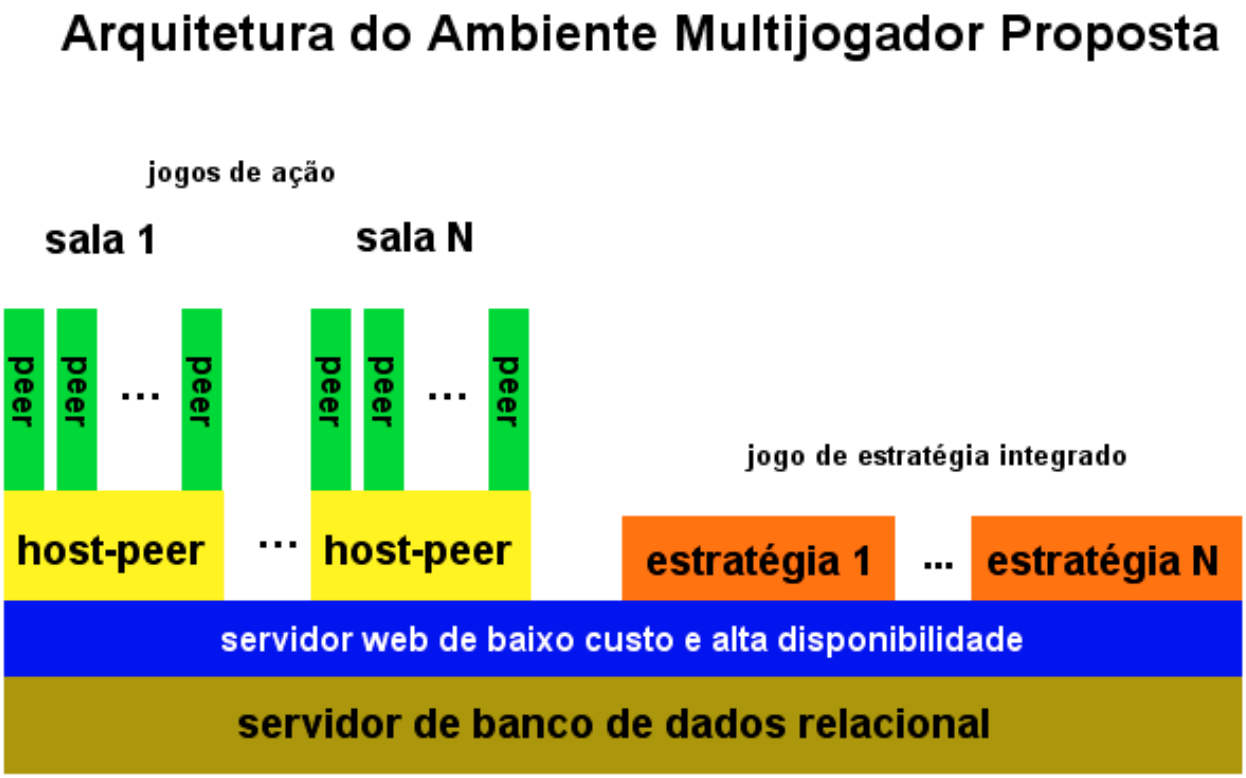

figura 5: diagrama de blocos ilustrando a arquitetura híbrida e integrada proposta em (Kozovits, 2003)

Uma detalhe de implementação, já relevante neste momento, é que, apesar de os jogos de ação estarem organizados em salas *peer-to-peer*, o ambiente estabelece que, em cada sala haja um usuário exercendo um papel de controle. Este usuário, na verdade, o programa utilizado pelo jogador e, de forma invisível para este usuário, irá atuar como um *host-peer* ou um mini-servidor local.

Na implementação realizada, usa-se a tecnologia DirectPlay da Microsoft que já contempla a existência de um jogador atuando como *host-peer*, bem como um mecanismo de eleição para promover um outro jogador *peer* para *host-peer* caso o primeiro encerre sua participação ou perca, por qualquer motivo, sua conexão. Naturalmente, caso o ambiente venha a ser portado para Linux, por exemplo, existem alternativas *open-source* e multiplataforma a esta tecnologia (Hawksoft, 2003).

A principal tarefa de um *host-peer*, além do eventual uso como árbitro no controle de pontuação, é estabelecer a comunicação com o servidor de banco de dados e mantê-lo atualizado em relação a informação geral sobre a sala e aplicar também os critérios de balanceamento que descreveremos a seguir.

O primeiro critério para ocupação ótima das salas citado acima, leva em conta o fato de que, o dinamismo de jogos pode fazer com que uma sala cheia, além da natural e freqüente troca de usuários, possa vir a se esvaziar com o tempo, o que é indesejável principalmente para os jogadores remanescentes.

O segundo critério leva em conta o fato de que um jogador, ao entrar num sistema dito multi-jogador possa, de fato, encontrar outros jogadores com quem possa interagir. Se, num dado momento, todas as salas estiverem com sua ocupação máxima, este jogador será o hospedeiro de uma sala (*host-peer*) com apenas um jogador (ele próprio) o que, apesar de ser a única alternativa inicial, é contudo, algo indesejável de se manter a longo prazo.

As situações descritas anteriormente, apontam para a implementação de um algoritmo guloso distribuído, que visa, primariamente, buscar a formação e manutenção do menor número de salas com a maior ocupação possível.

As situações que devem ser tratadas são as seguintes:

1. se um determinado jogador deseja entrar no ambiente, a este jogador será oferecida a participação na sala com menor ocupação segundo a consulta feita na tabela dinâmica de ocupação de salas;

2. se as duas primeiras salas, segundo ordenação crescente por ocupação tiverem um número K1 e K2 de jogadores menor que a metade do número máximo permitido (K1 < MAX_PLAYERS_SALA/2 E K2 < MAX_PLAYERS_SALA/2) então o host-peer da segunda sala deverá determinar a desconexão total de seus usuários, inclusive ele próprio e estabelecer que estes jogadores entrem no estado de busca de uma nova conexão; pelo dinamismo do ambiente não se pode dizer em que salas estes jogadores irão entrar mas, provavelmente, esta conexão irá se estabelecer com a primeira sala;

3. se um determinado jogador deseja se conectar ao ambiente e não houver nenhuma sala livre disponível, então este jogador será conectado como *host-peer* de uma nova sala;

4. se ocorrer de a primeira sala, pelo critério de ocupação, ter apenas um jogador (*host-peer*) e a segunda um número menor que o máximo permitido então a primeira sala (de um jogador) será notificada de que deve se desconectar e tentar nova conexão como peer apenas;

5. se uma determinada sala permanecer vazia (um jogador apenas) durante um certo tempo e as demais completamente cheias, então a próxima sala cheia será notificada de que deve desconectar alguns de seus jogadores até que o seu número de jogadores fique igual a metade do número máximo permitido mais um, ou (MAX_PLAYERS_SALA/2+1); por este critério, dá-se o mecanismo de balanceamento da ocupação de salas inclusive para jogadores que estejam se conectando a um ambiente já num estado de ocupação ótima (balanceado).

Ou seja, pelo apresentado, pode-se inferir que o ambiente estará, em geral, numa permanente busca do equilíbrio que, apenas eventualmente, é mantido estável por um longo tempo. Além disso, o balanceamento ocorre sempre de forma colaborativa e entre as primeiras salas de uma consulta baseada no critério ocupação.

Visando tornar esta lógica de desconexão e re-conexão do jogador um evento inerente ao ambiente e, ao mesmo tempo aceitável, o jogo implementado usando a arquitetura híbrida descrita acima, deverá prover um efeito visual integrado ao momento da desconexão do jogador, para que este usuário não se surpreenda negativamente com a ocorrência destes eventos eventualmente freqüentes.

Um exemplo disto será a implementação num certo jogo específico (uma instância do ambiente proposto) da ocorrência de um evento catastrófico como um terremoto ou explosão ocorrida na sala. Isto torna o jogo compatível com os critérios de consistência visual de ambientes virtuais distribuídos. Esta possibilidade de implementação é, ao mesmo tempo, possível também pelo fato de que, como já mencionamos, o principal requisito de um jogo ser a jogabilidade e não o realismo de simulações em geral.

O algoritmo apresentado acima irá, portanto, monitorar as situações indesejáveis e sinalizar aos programas componentes do jogo (os "clientes") que deixem uma determinada sessão e voltem a se conectar em outra.

Por ser um algoritmo distribuído e cooperativo entre os clientes componentes da simulação, é necessário um mecanismo de sincronização entre estes programas. O "monitor" adotado, baseia-se em atributos intrínsecos de servidores de bancos de dados relacionais como o que foi adotado:

- cada tratamento para conduzir ao balanceamento só deverá ocorrer se um outro tratamento não estiver ainda em andamento. Assim, um comando SQL "UPDATE" para atualizar um registro de um *host-peer*, marcando-o para uma tarefa de balanceamento terá uma cláusula "WHERE" que só tornará o comando bem sucedido se nenhum outro *host-peer* estiver com marcação semelhante.

Também é importante ter em mente que os registros das sessões ativas devem ter um tempo de validade determinado (TICK_TIMEOUT). Isto dá-se pelo fato de que uma conexão pode cair durante uma tarefa de balanceamento em andamento, interrompendo esta tarefa ainda incompleta.

Pelo fato de o balanceamento ocorrer de modo a envolver no máximo duas salas e, sempre as duas primeiras numa consulta indexada quanto ao campo ocupação, não há grande complexidade na implementação. O único cuidado que se deve ter é quanto ao fato de um determinado *host-peer* ou candidato a *peer*, neste algoritmo colaborativo, "não querer colaborar".

Isto pode ocorrer quando, por exemplo, um candidato não conseguir estabelecer conexão com uma determinada sala ativa e ainda não completamente cheia. Algum problema, por exemplo a existência de um firewall no equipamento do candidato bloqueando certas portas pode estar impedindo esta conexão. Isto determina que o algoritmo deva tratar estes casos para que o sistema não fique em permanentes tentativas frustradas de conexão o que irá impedir que outras salas entrem no estado de execução de tarefas necessárias ao balanceamento.

A solução proposta é contar o número de tentativas frustradas e suspender temporariamente tarefas de balanceamento entre o candidato e a sala contactada após um certo número de tentativas. Três possibilidades para apontar o problema devem ser identificadas:

1. o candidato não consegue se candidatar a nenhuma outra sala

2. a sala não consegue (mais) aceitar nenhum novo candidato

3. o problema é temporário e a conexão pode ser estabelecida num momento futuro

No primeiro caso, o programa candidato é impedido de se conectar ao sistema por um certo tempo e o jogador é notificado de que deve tomar alguma providência, por exemplo, desligar o firewall ou habilitar certas portas.

No segundo caso, se algum outro jogador ou um certo número de jogadores não conseguir a conexão com o *host-peer* responsável pela sala, então este *host-peer* é marcado para desconexão e a sala irá eleger um novo responsável.

No último caso, nada precisa ser feito, pois a própria dinâmica do ambiente irá tratar de encaixar o jogador com problemas de conexão em alguma sala num futuro breve.

5. Implementação

O sistema base empregado é o mesmo descrito em (Kozovits, 2003). Foi utilizado Microsoft Visual C/C++ 6.0 para implementar o ambiente gráfico utilizando bibliotecas OpenGL-glut para fazer o *rendering* dos objetos de cada sala. Os modelos utilizados são compatíveis com o formato MD2 facilmente encontráveis e disponíveis na Internet, assim como bibliotecas para sua carga e *rendering*. Criou-se uma biblioteca integrada ao glut para encapsular as complexidades de uso do Microsoft DirectPlay responsável pelas conexões peer-to-peer entre os jogadores.

Uma visão do ambiente implementado durante a execução é ilustrada pela figura 6.

figura 6: uma imagem de uma sessão peer-to-peer ilustrando o sistema implementado

5.1 Tabelas e modelagem do servidor de banco de dados relacional

A primeira tabela contém as informações de cadastro dos jogadores, necessária para admissão de um jogador no ambiente.

PLAYER		
PlayerID	**BIGINT**	número único, gerado automaticamente, chave para identificação de um jogador no sistema
ComputerID	**CHAR 96**	seqüência única para identificação do equipamento do jogador, útil para eventual verificação de fraude
Nome	**CHAR 96**	Nome do jogador
email	**CHAR 96**	
senha	**CHAR 8**	
...		outras informações relevantes para um determinado jogo a ser implementado

A tabela acima é preenchida no momento da inscrição do jogador no ambiente e é dependente do tipo de informações que se deseja manter sobre um determinado jogador. Do ponto de vista do ambiente criado, apenas é relevante o fato de que cada jogador terá um número único que será confirmado no momento de sua conexão.

A segunda tabela, contém as informações das sessões ativas. É uma tabela com informações dinâmicas e, apesar de suas informações serem temporárias, é a tabela mais importante para o controle e o funcionamento balanceado do ambiente.

ACTIVESESSION		
NomeJogo	**CHAR 30**	Nome do jogo, caso mais de um jogo esteja sendo controlado
Sala	**CHAR 30**	Nome da sala ou sessão P2P
IsHost	**CHAR 1**	Controla se o jogador desta entrada é host-peer valor 'T' ou não valor 'F'
IPstr	**CHAR 127**	endereço Internet do jogador naquele momento
PlayerID	**BIGINT**	chave estrangeira, identificador do jogador
NumPlayers	**INTEGER**	número de jogadores na sala
FirstTick	**TIMESTAMP**	momento da conexão
LastTick	**TIMESTAMP**	última atualização
mustLeave	**CHAR 1**	flag para determinar se um jogador host-peer deve esvaziar sua sala total 'T', parcialmente 'P' ou nada a fazer 'F'
failures	**INTEGER**	número de tentativas de conexão sem sucesso
LastFailure	**TIMESTAMP**	momento da última tentativa de conexão sem sucesso
LastComplainantID	**BIGINT**	último PlayerID a não conseguir estabelecer conexão
x	**REAL**	última posição informada (x, y, z)
y	**REAL**	
z	**REAL**	
vx	**REAL**	última velocidade informada (x, y, z)
vy	**REAL**	
vz	**REAL**	
upx	**REAL**	última orientação espacial informada (x, y, z)
upy	**REAL**	
upz	**REAL**	
...		campos dependentes do jogo a ser implementado

Esta tabela será atualizada por mensagens SITE_TICK que cada jogador usa para enviar a atualização de seu estado com uma freqüência TICK_TIMEOUT / 10. Ao deixar uma sessão, o jogador irá enviar a mensagem SITE_LEAVE que irá remover a entrada correspondente a PlayerID desta tabela.

A cada SITE_TICK , ou a cada "N" SITE_TICKs, ocorre uma limpeza da tabela: - isto permite a remoção de um jogador que saiu por queda de conexão, sem notificação de saída. Ou seja, esta limpeza é a implementação do mecanismo de *timeout:* se o tempo atual for maior ou igual a última atualização mais a constante TICK_TIMEOUT.

5.2 Formato de Mensagens para conexão e balanceamento de jogadores utilizando a técnica proposta

A parte do sistema híbrido citado como "cliente-servidor" é na verdade, composto de três camadas:

- na primeira temos o jogo escrito em VisualC++ responsável pela visualização e interface com o usuário. Este programa além da comunicação com outros programas idênticos

situados em outras máquinas, possui uma *thread* para se comunicar com um servidor web enviando URLs de conexão com aparência mostrada a seguir.

- na camada do meio, escrita em .NET (páginas aspx e componentes em Csharp), faz-se o processamento das mensagens enviadas pela camada de apresentação.

- a última camada refere-se ao servidor de banco de dados relacional.

O aspecto real (com criptografia) de uma mensagem é:

http://hostname/3dg/systemtray.aspx?cmd=12A45F78G2fasdjfhaf87adsyf066f8af6d8a97f6

Observe que o parâmetro *cmd* é constituído de uma mensagem criptografada para que se evite ou minimize acessos indevidos de usuários ao sistema conforme citado em (Kozovits, 2003).

No detalhamento do protocolo de troca de mensagens iremos, evidentemente, usar o formato e protocolo da fase de testes, em mais alto nível.

Durante a fase de depuração, para facilitar o *parsing* e a correção de erros, um formato de *TAGs* , sem compressão ou criptografia, é utilizado na resposta do comando. O comando enviado ao servidor web, após o sinal de interrogação, usa uma cadeia de caracteres no seguinte formato geral:

cmd=SITE_CMD$var1=valor1$var2=valor2$ ... $varN=valorN$

Onde

var1, var2, ..., varN são os nomes parâmetros passados
valor1, valor2, ..., valorN são os valores destes parâmetros
e SITE_CMD é o nome do comando a ser enviado
OBS: usa-se o caracter $ como separador de parâmetros

A resposta ao comando será sempre da forma

<OK>resposta dada entre tags</OK>

O diagrama simplificado, correspondente a transição dos estados descritos a seguir, pode ser visto na figura 7. Vamos enumerar a seqüência de mensagens utilizadas para o funcionamento do ambiente, numa seqüência lógica, a mesma usada por um programa cliente ao tentar se conectar:

- **SITE_CHAVE**

retorna a chave para criptografia das mensagens entre a primeira e a segunda camadas:
<ok> chave para criptografia </ok>

Após obter a chave de criptografia o sistema irá se identificar junto ao servidor e pedir uma conexão ao ambiente do jogo por meio da mensagem

- **SITE_CONECTAR**

exemplo: . . . *systemtray.aspx?cmd=SITE_CONECTAR*,

OBS: é possível, também no comando acima, especificar uma sessão indesejada `$NG=endereçoIP$`

<ok>H</ok>

mensagem recebida quando não há jogadores ou todas as salas estão em sua ocupação máxima, o software cliente recebe a instrução H determinando sua conexão como *host-peer* de uma sala ainda vazia.

<ok>P endereçoIP N numJogadores </ok>

mensagem recebida quando há uma sala com " numJogadores" cujo host-peer está no endereço "endereçoIP". Esta informação é o suficiente para a tentativa de conexão com *este host-peer*, já que a implementação utiliza o mecanismo de *port-scan* do DirectPlay da Microsoft, o que dispensa um número de porta.

<ok>ERROR</ok>

mensagem recebida quando ocorrer um erro não especificado, por exemplo, erro na criptografia da mensagem

Temos agora duas possibilidades: ou o jogador irá abrir uma nova sala como *host-peer* ou irá se conectar a uma sala pré-existente como *peer* simplesmente. As duas mensagens de notificação do servidor web, caso as operações P2P sejam bem sucedidas são:

cmd=SITE_HOST_STARTED$NP=1$ID=0$SL=nomeSala$etc...

ou

cmd=SITE_PEER_STARTED$NP=2$ID=123$SL=nomeSala$etc...

- **SITE_HOST_STARTED**

É criada uma nova entrada na tabela ACTIVESESSION com informações como número de jogadores (no caso H), endereço IP, IsHost=T, PlayerID=numeroDoJogador, nome da sala, posição e velocidade do jogador etc

A resposta, no caso de sucesso, deverá ser

<ok>HINS1</ok>

Caso a resposta acima não ocorra, haverá uma segunda tentativa que, em caso de falha levará a desconexão do jogador e o retorno ao estado inicial de SITE_CHAVE recomeçando todo o processo descrito até aqui.

- **SITE_PEER_STARTED**

É criada uma nova entrada na tabela ACTIVESESSION com informações como número de jogadores (no caso P), endereço IP deste jogador, IsHost=F, PlayerID=numeroDoJogador, nome da sala, posição e velocidade do jogador etc

A resposta, no caso de sucesso, deverá ser

<ok>PINS1</ok>

Caso a resposta acima não ocorra, haverá uma segunda tentativa que, em caso de falha levará a desconexão do jogador e o retorno ao estado inicial de SITE_CHAVE recomeçando todo o processo descrito até aqui.

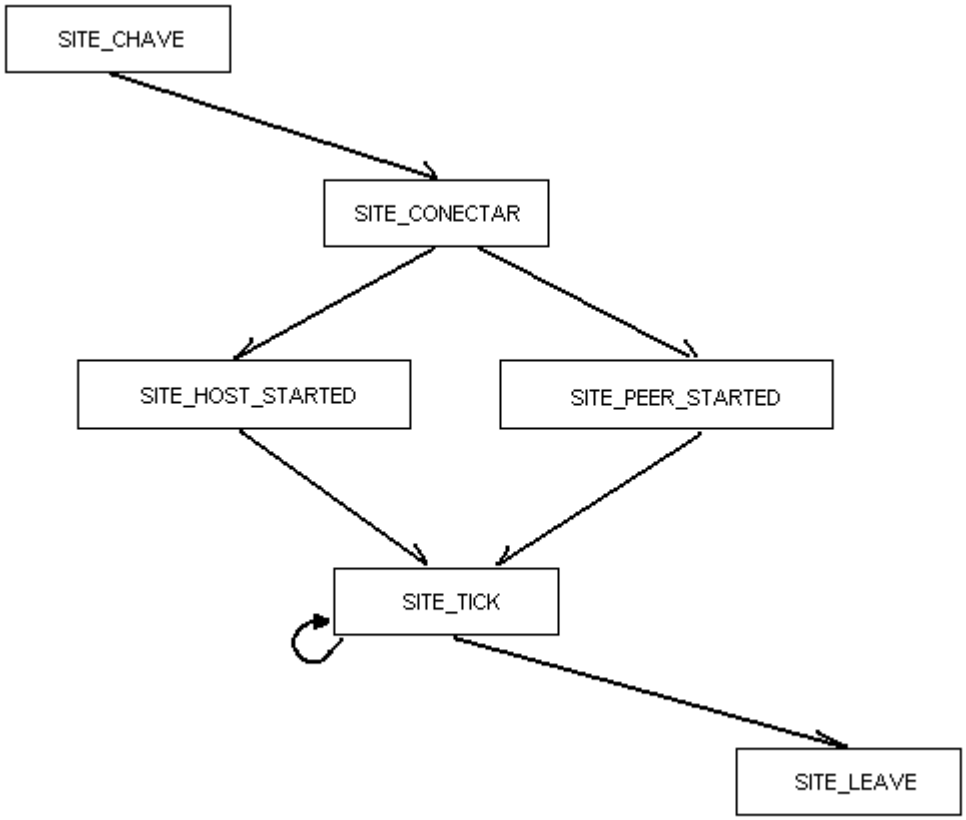

figura 7: diagrama de estados simplificado, ilustrando as principais transições que fazem parte da técnica de conexão balanceada proposta.

- **SITE_HOST_MIGRATION**

Esta mensagem é disparada quando um *peer* de uma sala é eleito como *host-peer* da sessão em andamento devido a saída ou queda de conexão do host-peer anterior.
Ex:
cmd=SITE_HOST_MIGRATION$ID=123$ ou
cmd=SITE_HOST_MIGRATION$ID=123$ML=T$ (quando a sala está em processo de desconexão, o parâmetro ML reflete o estado do flag "**mustLeave"** que é de conhecimento de todos na sala)
Mensagens de retorno:

<ok>ERROR</ok>
indica erros em geral, por exemplo o registro correspondente já foi eliminado por *timeout*

<ok>HUPD1</ok>
indica que tudo está certo, o registro foi atualizado

<ok>DISCONECT</ok>
indica a ocorrência de alguma situação determinando que este cliente se desconecte

- **SITE_LEAVE**

Esta mensagem é enviada quando um jogador se desconecta da sala ou eventualmente fecha o programa.
Mensagens de retorno:

<ok>ERROR</ok>
indica erros em geral, por exemplo o registro correspondente já foi eliminado por timeout

<ok>DEL1</ok>
indica que tudo está certo, o registro foi removido

- **SITE_UPDATE_PLAYERS**

Esta mensagem é enviada quando ocorre alteração no número de jogadores de uma sessão
Mensagens de retorno:

<ok>ERROR</ok>
indica erros em geral, por exemplo o registro correspondente já foi eliminado por timeout

<ok>UPD1</ok>
indica que tudo está certo, o registro foi atualizado

<ok>DISCONECT</ok>
indica a ocorrência de alguma situação determinando que este cliente se desconecte

- **SITE_FAILURE**

Esta mensagem é enviada quando o jogador tenta se conectar a um *host-peer* e a conexão não é bem sucedida. O endereço IP do *host-peer* tentado é enviado e ocorre incremento no número de tentativas frustradas de conexão. OBS: cada jogador diferente (IP diferente) só faz uma "reclamação" de um host-peer (outro IP diferente) para evitarmos uma contagem indevida de reclamantes. Em outras palavras, considerando programas e sessões ativas num dado momento o par de IPs (IP reclamante, IP citado) é único.

Mensagens de retorno:

<ok>ERROR</ok>
indica erros em geral, por exemplo o registro correspondente já foi eliminado por *timeout*

<ok>UPD1</ok>
indica que tudo está certo, o registro foi atualizado

- **SITE_TICK**

Esta é a mensagem mais freqüente, por ser enviada por todos os clientes numa freqüência TICK_TIMEOUT/10. Além de promover a atualização do estado de cada jogador, ainda permite a aplicação do algoritmo de balanceamento colaborativo proposto, uma vez que o servidor web (páginas aspx) possui uma atitude passiva, ou seja, apenas responde se for invocado.

Exemplo 1:
cmd=SITE_TICK$ID=3$NP=6$

Mensagens de retorno:

<ok>ERROR</ok>
indica erros em geral, por exemplo o registro correspondente já foi eliminado por timeout

<ok>DISCONECT</ok>
indica a ocorrência de alguma situação determinando que este cliente se desconecte

<ok>UPD1</ok>
é a resposta mais freqüente indicando a atualização do registro correspondente

<ok>LEAVET</ok>
indica que este cliente(se host-peer) deverá promover a remoção total dos jogadores da sala inclusive ele mesmo

<ok>LEAVEP</ok>
indica que este cliente(se host-peer) deverá promover a remoção parcial dos jogadores da sala de modo que a sala fique com MAX_PLAYERS_SALA/2+1.

Exemplo 2:
cmd=SITE_TICK$ID=3$NP=3$OK=1$

quando um host-peer termina suas tarefas de balanceamento, ele deve, por meio da mensagem de atualização SITE_TICK, notificar o fim da sua tarefa(por exemplo a remoção parcial de jogadores da sala)
Mensagens de retorno(além das anteriores):

<ok>OKUPD1</ok>
indica que a tarefa de balanceamento teve êxito e o registro foi atualizado para permitir a ocorrência de outras tarefas de balanceamento envolvendo outras salas e cliente.

6. Encerramento e trabalhos futuros

O uso da técnica descrita acima é original e possibilita resolver dois problemas básicos de ambientes virtuais:

- encontrar pessoas com quem interagir e
- permitir que esta interação se mantenha ótima durante a experiência virtual.

Além disso, permite integrar dois tipos de jogos (ação e estratégia) num mesmo ambiente como proposto em (Kozovits, 2003).

Todavia, esta técnica não precisa ficar restrita a utilização em ambientes de jogos. O mesmo problema de balanceamento quanto a ocupação de participantes ocorre em salas de bate-papo (ou *chat*) na Internet, um tipo de entretenimento muito procurado.

Na maioria dos principais portais brasileiros UOL (2003), Globo.com (2003) e Terra (2003), são oferecidas salas de chat sem nenhum critério de balanceamento automatizado. Desta forma, ocorre o fenômeno de algumas salas ficarem com lotação esgotada e outras, por se apresentarem

vazias, não se estabelecem. Isto ocorre pelo fato de que ninguém, em geral, desejar ficar numa sala sem ter com quem conversar, apenas aguardando o eventual aparecimento de interlocutores.

Uma redistribuição ou agrupamento de salas vazias, se realizado de forma automática, pode permitir o aumento do número de clientes para este tipo de entretenimento.

Outro fato importante, não tratado neste trabalho, é o uso de outros critérios de balanceamento de salas ou sessões. O nível de habilidade de um jogador, o tipo de conexão utilizado (modem comum ou banda larga), a faixa etária, entre muitos outros atributos podem ser utilizados como critério de balanceamento primário (critério outro que não o grau de ocupação) ou secundário (em adição ao critério principal de ocupação das salas).

7. Referências

The ACM Digital Library (2003) [online], Available from: http://portal.acm.org/ [Accessed 01 September 2003]

Bogojevic, Sladjan, (2003), The Architecture of Massive Multiplayer Online Games, Master Thesis, Department of Computer Science, Lund Institute of Technology, Lund University, Sweden [online], Available from: http://w1.401.telia.com/~u40134115/som.pdf [Accessed 27 September 2003]

CITESEER, Scientific Literature Digital Library, (2003) [online], Available from: http://citeseer.nj.nec.com/cs/ [Accessed 01 September 2003]

Codd, E. F., (1972) Relational Completeness of Data Base Sublanguages. In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California

Date, C. J., (1999), An Introduction to Database Systems, 7th Ed. Addison-Wesley.

Fowler, Martin, (1999), Refactoring: Improving the Design of existing Code. Addison-Wesley.

Globo.com, chat service (2003) [online]. Available from: http://psiu.globo.com/ [Accessed 15 August 2003]

Hanson, Ben, (2003) Game Balance for Massively Multiplayer Games, Sony Online Entertainment, in Thor Alexander, ed., Massively Multiplayer Game Development, Charles River Media.

Hawksoft, Hawk Network Library, (2003) [online]. Available from: http://www.hawksoft.com/hawknl [Accessed 05 August 2003]

Kozovits, Lauro E. (2003), Arquiteturas para Jogos *Massive Multiplayer*. Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 2003 (em preparação).

Lee, Jay, (2003) Leveraging Relational Database Management Systems to Data Drive MMP Gameplay, NCsoft Corporation, in Thor Alexander, ed., Massively Multiplayer Game Development, Charles River Media

Lua Language website, (2003) [online]. Available from: http://www.lua.org [Accessed 05 August 2003]

Pope, A. (1989), The SIMNET network and protocols. Technical Report 7102, BBN Systems and Technologies, Cambridge, MA

Python Language website, (2003) [online], Available from: http://www.python.org [Accessed 05 August 2003]

Rabin, Steve, (2000), The Magic of data-driven Design. In Mark DeLoura, ed., Game Programming Gems, Charles River Media.

Smed, J., Kaukoranta, T., Hakonen, H. (2002), A Review on Networking and Multiplayer Computer Games, University of Turku, Department of Computer Science, Turku Centre for Computer Science, TUCS Technical Report No 454, April 2002, ISBN 952-12-0983-6

Szwarcman, Dilza de Mattos (2001), Dead Reckoning Orientado a Metas para Personagens Reativos, Tese de Doutorado, Departamento de Informática, PUC-Rio

TCL website, (2003) [online], Available from: http://tcl.sourceforge.net [Accessed 05 August 2003]

Terra Networks, chat service (2003) [online]. Available from: http://chat.terra.com.br:9781/ [Accessed 15 August 2003]

ToonTown (2003) [online]. Available from: http://play.toontown.com/webHome.php [Accessed 27 September 2003]

UOL, chat service (2003) [online]. Available from: http://batepapo.uol.com.br/ [Accessed 15 August 2003]

Ward, Mark, (2003) Making money from virtually nothing, [online], BBC News. Available from: http://news.bbc.co.uk/1/hi/technology/3135247.stm [Accessed 11 August 2003]

XML website (2003)[online], Available from: http://www.w3.org/XML/ [Accessed 07 September 2003]