

Uma Plataforma para Jogos Móveis Massivamente Multiusuário

Andrea Menezes¹ Carlos Eduardo Silva¹ Daniel Arraes^{2,3}
Davi Pedrosa¹ Fernando Brayner^{2,3} Fernando Trinta¹
Isabel Wanderley^{2,3} Marcus Machado^{2,3} Rafael Borges^{2,3}
Geber Ramalho^{1,3}

¹Universidade Federal de Pernambuco, Centro de Informática, Brasil

²Centro de Estudos e Sistemas Avançados do Recife, Brasil

³Meantime Mobile Creations, Brasil

Abstract

Mobile Massively Multiplayer Online Games (3MOG) represent an important market trend in games industry, combining massive multiplayer with mobile games. Although some requirements for a 3MOG supporting platform have been enumerated in the literature, there is neither a reference implementation, nor an in-depth study of the criteria, guiding such an implementation. In this paper, we discuss the importance of the platforms in the development of 3MOGs and criteria to evaluate them. Finally, we present a case study that illustrates the implementation and evaluation of two different approaches to this kind of platform.

Keywords: Mobile Massively Multiplayer Online Games, Platform, 3MOG.

Resumo

Jogos móveis massivamente multiusuário (3MOG) representam uma importante tendência na indústria de jogos, unindo os massivamente multiusuário aos móveis. Embora alguns requisitos para uma plataforma que suporte 3MOG já foram enumerados na literatura, não há nem uma implementação de referência nem um estudo aprofundado dos critérios para guiar tal implementação. Neste artigo, nós discutimos a importância das plataformas no desenvolvimento de 3MOGs e os critérios utilizados para avaliá-las. Por fim, apresentamos um estudo de caso que envolve a implementação e a avaliação de duas diferentes abordagens deste tipo de plataforma.

Palavras-chaves: Jogos Móveis Massivamente Multiusuário, Plataforma, 3MOG.

Contato:

¹{afm3, cesps, dvp, famt, glr}@cin.ufpe.br

²{dap, flb, iws, mvlm, rmb}@cesar.org.br

1. Introdução

No modelo de jogos multiusuário distribuídos, os jogadores interagem por meio da troca de mensagens

em uma rede de computadores. Inicialmente, devido a restrições de largura de banda e latência, estes jogos eram limitados a um pequeno número de usuários situados na mesma rede local. Entretanto, com o avanço das tecnologias de comunicação e o sucesso da Internet, foi possível criar jogos nos quais milhares de jogadores interagem simultânea e continuamente em um mundo virtual persistente. Estes jogos são classificados como *massivamente multiusuário* (MMOG – do inglês, *Massively Multiplayer Online Game*).

Em paralelo a este cenário, os jogos móveis vêm crescendo significativamente no mercado. Da convergência entre a computação móvel e os jogos massivamente multiusuário nasceram os jogos móveis massivamente multiusuário, chamados 3MOG (do inglês, *Massively Mobile Multiplayer Online Games*). Estes jogos prometem um grande poder de entretenimento e um forte apelo comercial, embora estes fatores ainda não tenham sido amplamente explorados. Hoje a maioria dos jogos disponíveis para celulares ainda é monousuário e *offline*.

Desenvolver um jogo 3MOG não é uma tarefa simples, pois devem ser superados vários desafios tecnológicos, como alta latência das redes de telefonia móvel e baixa capacidade de processamento dos dispositivos móveis. Para facilitar o trabalho dos desenvolvedores, *publishers* e operadoras, o uso de uma plataforma de apoio ao desenvolvimento e execução destas aplicações é de fundamental importância. Nesse contexto, o conceito de plataforma reflete um conjunto de serviços e a infra-estrutura necessária para executá-los.

Os serviços de 3MOG incluem, por exemplo, manipulação de contas de usuários, pontuação, ranking e notícias. De um modo geral, eles já foram especificados na literatura [Saha et al. 2003; Nokia 2006a]. Contudo, não há disponível nenhuma implementação aberta que sirva de referência para o desenvolvedor de 3MOGs, obrigando-o a implementar toda a infra-estrutura e os respectivos serviços.

Assim, este artigo dá uma contribuição ao apresentar um projeto concreto e de porte industrial, descrevendo o papel que uma plataforma exerce durante o desenvolvimento de um 3MOG e quais critérios devem guiar a sua escolha e/ou construção (Seções 2 e 3). Este artigo também apresenta um estudo de caso de desenvolvimento de duas versões de uma plataforma e de um jogo utilizado para validá-las (Seção 4).

2. Contexto

Embora a idéia de 3MOGs seja bastante interessante, até 2004 não havia nenhum esforço significativo nesta área de desenvolvimento de plataformas. Por isso, com o apoio da FINEP¹ e a parceria de algumas instituições², a Meantime juntamente com o C.E.S.A.R propuseram a criação de uma plataforma de código aberto que oferecesse os principais serviços de suporte ao desenvolvimento e à execução de Jogos Móveis Massivamente Multiusuário.

Por ser ainda um tema bastante recente, não havia informações claras sobre que abordagem de implementação adotar, embora já houvesse alguma para a definição e modelagem dos serviços [Open Mobile Alliance 2003]. Assim, após estudos e pesquisas para definirmos a maneira que essa plataforma seria implementada de fato, terminamos por desenvolver duas versões da plataforma: (A) uma cuja implementação abrange toda a infra-estrutura e (B) outra que adapta uma plataforma genérica às necessidades de um 3MOG. A existência dessas duas versões nos deu uma condição ímpar para a obtenção um conhecimento mais profundo ao tomar decisões sobre o desenvolvimento e escolha de uma plataforma.

No decorrer do nosso projeto, centros de pesquisa e empresas de *software* conceberam diversas iniciativas para desenvolver esse tipo de plataforma. Atualmente, estas iniciativas oferecem produtos que disponibilizam vários dos principais serviços necessários a tais aplicações [Nokia 2006b; Terraplay 2003; Pellerin et al. 2005]. Contudo, não há plataformas abertas voltadas totalmente para 3MOGs e suas especificidades (rede intermitente e limitações de processamento e memória, por exemplo).

3. Análise da Implementação de Plataformas 3MOG

Com base em nossa experiência no desenvolvimento da plataforma 3MOG, pudemos observar alguns critérios importantes para avaliação da escolha de implementação. Nesta seção, apresentaremos os principais critérios observados: escalabilidade,

distribuição de carga, tolerância a falhas, segurança, gerenciamento de transação e acesso a dados.

Tendo em vista as duas implementações da plataforma, será discutida a importância de cada aspecto assim como seu impacto em relação a duas premissas básicas: o tempo de resposta de uma requisição do usuário e a facilidade de implementação da infra-estrutura necessária para este tipo de jogo. A primeira premissa compreende a percepção de interação do jogador com o jogo, e pode determinar seu fracasso comercial, visto que o jogador pode se entediar ao ter que esperar muito tempo por uma resposta do jogo. Já a segunda valoriza o *time-to-market* da iniciativa e concentra melhor os esforços do desenvolvedor nos aspectos inerentes à lógica do jogo.

3.1 Escalabilidade

A escalabilidade de um sistema indica sua capacidade de processar uma quantidade crescente de requisições de maneira apropriada, afetando minimamente o tempo de resposta ao usuário [Tanenbaum e Van Steen 2002]. Em geral, essa capacidade é conquistada de duas formas: (i) verticalmente, através do incremento de recursos disponíveis em uma máquina (como o poder de processamento ou a quantidade de memória), ou (ii) horizontalmente, que consiste na adição de nós ao conjunto de servidores da aplicação.

O crescente aumento no número de usuários e a grande quantidade de conteúdo de jogo, necessidades inerentes a jogos 3MOG, geram uma maior demanda de mensagens, requisições e interações. Isto faz com que haja um maior consumo de processamento e largura de banda dos servidores, tornando o sistema susceptível a problemas de escalabilidade.

Particularmente, a escalabilidade horizontal transforma a aplicação em um sistema distribuído, abordagem escolhida para a implementação de nossas versões da plataforma 3MOG. Desta forma, o uso da abordagem A requer do desenvolvedor um maior esforço para implementar aspectos inerentes a essa distribuição, tanto na aplicação quando na infra-estrutura, coisa que, na abordagem B, esta infra-estrutura é oferecida transparentemente, cabendo lidar apenas com a aplicação.

3.2 Distribuição de Carga

Este tópico trata da distribuição da carga de processamento das requisições entre diversos servidores, aumentando o poder de processamento como um todo e possibilitando o real paralelismo entre os diversos processos.

Esta distribuição de carga necessita que os processos se utilizem de alguma estratégia para lidar com a situação de concorrência, evitando assim que haja uma degradação do desempenho. Dentre as várias estratégias, podemos citar: compartilhamento de estado

¹ Aprovado pela FINEP, edital número 01040973, assinado em 15/12/2004.

² UFPE, Intel do Brasil e Unicamp

(todas as informações estão concentradas em um único lugar), sincronização entre as instâncias (cada processo tem sua própria cópia, idêntica às demais) e particionamento (cada instância é responsável por uma parte disjunta do estado) [Knutsson et al. 2004].

Neste aspecto, a solução mais utilizada na prática envolve um servidor *Web* que recebe todas as requisições e redireciona-as (empregando um algoritmo de *round-robin*) para as demais máquinas, detectando, ainda, se alguma delas está indisponível no momento [Apache 2006]. Esta solução, por ser madura e se adequar às nossas necessidades, foi utilizada em ambas as implementações de nossa plataforma.

3.3 Tolerância a Falhas

A propriedade de tolerância a falhas se refere a um sistema que continua executando adequadamente, mesmo sob condições de erros [Tanenbaum e Van Steen 2002]. Geralmente, essa propriedade é obtida com redundância, ou seja, múltiplas instâncias do mesmo sistema executando em diversas máquinas; em caso de falha de alguma delas, outra assume o seu lugar. Vale a pena ressaltar, entretanto, que os serviços devem levar em consideração novos fatores, como a possibilidade de inconsistência do estado da aplicação quando ocorrer a falha de uma das instâncias.

A utilização de serviços cujo estado não é armazenado em cada instância, e sim compartilhado em um banco de dados (instâncias *stateless*), aliada à solução de distribuição de carga já citada (servidor *Web* com balanceamento de carga), mostrou-se simples e eficiente. Como em nossas implementações utilizamos esse tipo de instância, a tolerância a falhas foi contemplada utilizando esta solução. A utilização desta na abordagem B é ainda mais simples, dado que seu modelo de serviços implementa a distribuição transparentemente.

Caso seja necessária a sincronização de estados, é importante considerar que esta sincronização pode oferecer dificuldades para a tolerância a falhas e escalabilidade, devido ao aumento do tráfego interno de rede. Já o particionamento requer cuidados para que aquela porção do estado não fique inativa quando houver a falha de uma máquina.

3.4 Segurança

Segurança é um tema muito amplo, que contempla desde a autenticação e autorização de usuários até a confidencialidade e integridade dos dados. É preciso garantir que a plataforma saiba quem é o usuário e quais permissões ele possui, e que os dados trocados entre o cliente e o servidor sejam íntegros. Qualquer falha de segurança que possa ser utilizada para prejudicar outros jogadores é inaceitável.

Durante o desenvolvimento do projeto, percebemos que deveríamos dar um tratamento simplificado à

questão da segurança. O fato de o jogador utilizar um dispositivo móvel o impede de muitas trapagens aplicáveis a jogos multiusuário convencionais. Como exemplos disso, podemos citar a interceptação e alteração de pacotes de rede trocados com o servidor, a manipulação direta de dados do jogo na memória principal do dispositivo ou ataques de negação de serviço [Baughman e Levine 2001]. Por isso, a segurança não possui um grande impacto na escolha da plataforma, dado que pode ser solucionada de forma simples com o uso de bibliotecas de terceiros [Sun 2006c; Acegi 2006; Codehaus 2006].

3.5 Gerenciamento de Transação

Este critério é um fator importante em sistemas de um modo geral, pois permite que várias ações sejam executadas de forma atômica [Elmasri e Navathe 2004]. Para que isto seja realizado é necessário que recursos sejam retidos, tornando-os indisponíveis para as outras ações.

Existem várias formas de incorporar o gerenciamento de transação a uma aplicação. Numa primeira abordagem, o desenvolvedor deve controlar explicitamente a contenção de um recurso (através de *locks* e *unlocks*), permitindo que mais ações sejam executadas simultaneamente. Por outro lado, esta abordagem é muito suscetível a erros quando, por exemplo, uma ação erradamente não desbloqueia um recurso, tornando-o inacessível a outras ações.

O auxílio de uma biblioteca simplifica bastante o gerenciamento de transações: o desenvolvedor programaticamente inicia uma transação, associa objetos a ela e a finaliza, tornando persistentes as alterações. Indo mais além, ele pode apenas declarar quais operações executam sob um contexto transacional, cabendo à plataforma executá-la. Porém, estas abordagens podem causar a retenção desnecessária de recursos e conseqüente degradação do desempenho. É importante destacar ainda que, em sistemas onde cada ação retém recursos pouco compartilhados, pode ser vantajoso o gerenciamento manual de transações.

Na abordagem A, utilizamos a demarcação programática de transações, embora desejássemos que elas fossem gerenciadas automaticamente pela plataforma. Essa forma de demarcação trouxe problemas durante o encadeamento de operações, quando era necessário controlar se uma transação já estava ou não em execução. Já na abordagem B, como desejado, a plataforma oferecia transparentemente a demarcação de transação de forma declarativa.

3.6 Acesso a Dados

Acesso a dados em servidores de 3MOGs é extremamente importante, uma vez que ocorre muito freqüentemente e demanda grandes quantidades de dados, tomando uma fatia importante do tempo de

processamento de uma requisição. Esta grande quantidade de dados obriga o desenvolvedor a guardá-los em armazenamentos secundários.

Os armazenamentos secundários, por sua vez, possuem tempos elevados de acesso, exigindo o emprego de uma hierarquia de acesso: manter na memória de trabalho os dados estritamente essenciais para uma determinada tarefa, facilitar o acesso aos dados que serão utilizados com mais frequência e disponibilizar uma estrutura de acesso para os dados armazenados em meio secundário.

No topo da escala hierárquica, a memória de trabalho faz uso de estruturas de dados com tempo de acesso constante (por exemplo, *hash tables*). Em seguida, o uso de uma *cache* para os dados acessados recentemente contribui para o aumento do desempenho do sistema. Por fim, no terceiro nível, a utilização de SGBDs³ como mecanismo de gerenciamento e recuperação de dados em armazenamento secundário pareceu-nos a alternativa mais promissora: além de cuidar da estrutura dos dados, o SGBD provê acesso eficiente através de índices e realiza o controle transacional sobre as alterações dos dados. Uma outra opção seria o uso de arquivos especialmente estruturados (em geral, *binarizados*), que possuem um tempo de acesso melhor em relação a um SGBD, porém sem as facilidades inerentes deste.

As plataformas que podem ser utilizadas na abordagem B geralmente oferecem soluções de acesso a dados integradas e padronizadas [Hibernate 2006a; Oracle 2006; Sun 2006d], cabendo ao desenvolvedor configurar algumas propriedades conforme suas necessidades. Em contrapartida, na abordagem A, será exigido do desenvolvedor a escolha e integração de alguma solução já existente ou a implementação de todos os aspectos relacionados ao acesso a dados como, por exemplo, controle de concorrência.

4. Estudo de Caso

Em nosso estudo de caso, conforme citado anteriormente, foram desenvolvidas duas versões de plataforma para 3MOGs. Juntamente com estas, foi implementado um jogo, o Cellmons, para que estas plataformas pudessem ser validadas. Nele, procuramos contemplar os principais desafios inerentes aos 3MOGs.

O Cellmons (Figura 1) é um jogo de batalha em turnos. Os jogadores assumem papéis de treinadores que devem evoluir suas mascotes (os Cellmons) com cuidados e treinamentos, preparando-os para as batalhas. Os treinadores devem desafiar uns aos outros em duelos para ganhar pontos e experiência para a mascote. Aquele que possuir mais pontos torna-se, então, o campeão da Liga Mundial dos Cellmons.



Figura 1: Jogo Cellmons.

Dentre os serviços implementados, podemos destacar dois grupos: serviços gerais, inerentes a qualquer 3MOG, e específicos, próprios do jogo Cellmons. Do primeiro grupo, podemos citar o gerenciamento de contas de usuário e jogadores, pontuação e classificação (*ranking*), serviços de mensagens entre jogadores e notícias; do segundo, podemos destacar o serviço de lógica de batalhas e treinamentos. Estes serviços foram implementados nas duas abordagens da plataforma: (A) contemplando a implementação de toda a infra-estrutura e (B) adaptando uma outra plataforma às nossas necessidades. Um modelo geral às duas implementações foi definido com base em alguns critérios, especificados a seguir.

Devido a uma restrição inerente ao cliente (o suporte apenas obrigatório a HTTP), a nossa arquitetura deve ser do tipo *Cliente/Servidor*. Além disso, existem três requisitos, dentre os discutidos na seção anterior, que são contemplados pelo modelo arquitetural: escalabilidade, tolerância a falhas e distribuição de carga. A implementação destes três requisitos leva, portanto, ao *cluster de alta disponibilidade com balanceamento de carga*. Este cluster melhora a disponibilidade e o desempenho dos serviços através da distribuição da carga entre máquinas replicadas. Estas aumentam a escalabilidade horizontal e provêem serviços em substituição às que falharem (eliminação de pontos únicos de falha) [Tanenbaum e Van Steen 2002; Marcus e Stern 2002].

A Figura 2 representa o modelo de cluster escolhido para o desenvolvimento de nossa plataforma. Ele é constituído por quatro partes:

- A. Um conjunto de clientes, que acessam remotamente a nossa plataforma;
- B. O balanceador de cargas, responsável pela escolha de uma das instâncias do servidor para executar a requisição do cliente;
- C. Conjunto de instâncias do servidor, que processam as requisições e, se necessário, acessam o repositório de dados;
- D. O repositório de dados, que armazena e recupera as informações do 3MOG.

³ Sistemas de Gerenciamento de Bancos de Dados.

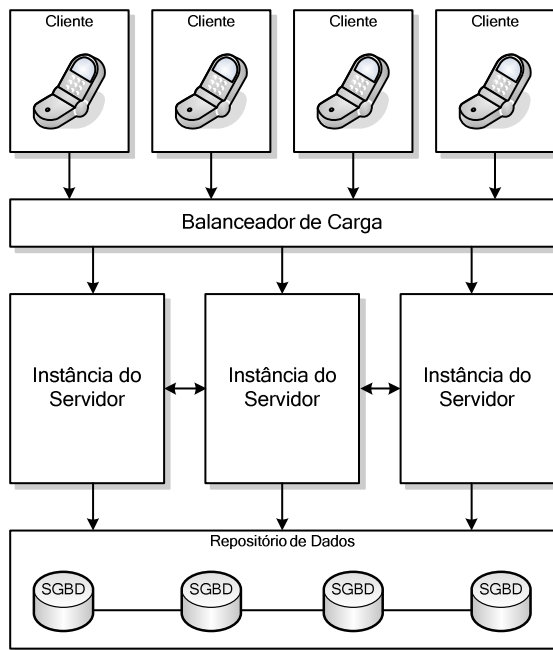


Figura 2: Modelo arquitetural da Plataforma 3MOG.

Dentre estas partes, o nosso objetivo é a construção do conjunto de instâncias do servidor, uma vez que elas contêm os serviços necessários ao jogo e contempla os demais requisitos da seção anterior (a saber: segurança, transação e acesso a dados). Nossa proposta para instâncias do servidor é apresentada na Figura 3.

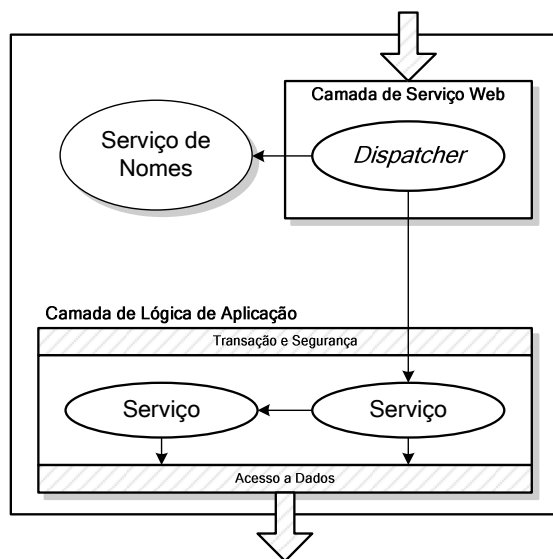


Figura 3: Modelo arquitetural da instância do servidor.

Em nossa proposta, a instância do servidor possui os seguintes componentes:

- A. O *Dispatcher*, responsável por receber a solicitação do balanceador de cargas, decodificar os parâmetros da mensagem,

localizar (através do serviço de nomes) e invocar o serviço a ela atrelado e, por fim, codificar a resposta para o cliente;

- B. O Serviço de Nomes, que contém referências para os serviços disponíveis na plataforma;
- C. Camada de Transação e Segurança, que intercepta chamadas aos serviços, provendo a elas contexto transacional e de segurança;
- D. Os Serviços, que contêm a lógica de negócio do 3MOG. Vale salientar que estes devem estar preparados para lidar com esta situação de distribuição;
- E. Camada de Acesso a Dados, responsável por prover abstrações para os serviços que simplificam o acesso ao repositório de dados.

Este modelo serviu de base para a implementação das duas versões da plataforma, que serão discutidas com detalhes nas subseções seguintes.

4.1 Abordagem A – Implementação de infra-estrutura

A primeira fase do projeto consistiu na implementação de serviços 3MOG e da infra-estrutura para executá-los. Foi desenvolvida uma plataforma utilizando a linguagem Java e bibliotecas de código aberto. Sua estrutura pode ser observada na Figura 4, onde temos uma divisão entre a camada de serviço Web, o contêiner de serviços e o Serviço de Nomes. Em particular, a seta superior representa a entrada proveniente do balanceador de carga, enquanto a inferior o acesso ao repositório de dados.

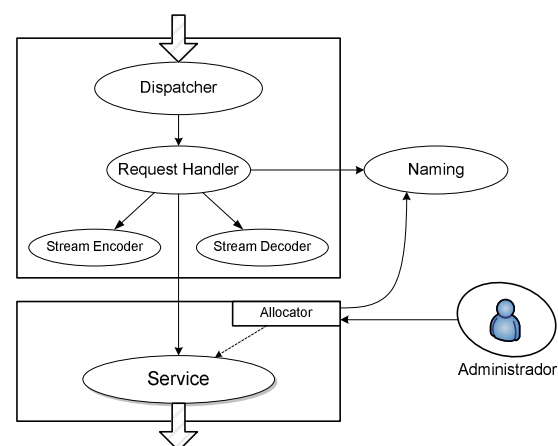


Figura 4: Modelo arquitetural da instância do servidor da abordagem A.

DISPATCHER – As funcionalidades do *Dispatcher* (apresentado na Figura 3) foram implementadas por um conjunto de classes responsáveis (Figura 4) por: (a) decodificar as informações da requisição, (b) recuperar a instância do serviço desejado no serviço de nomes, (c) invocar, via Java RMI (*Remote Method Invocation*) [Sun 2006b], o método desejado e, finalmente, (d) codificar a resposta.

Em especial, estes passos são configurados através de um arquivo XML (Figura 5). A escolha deste formato, na verdade, refletiu uma possibilidade futura: a geração automática de código para o tratamento do protocolo de comunicação tanto para o cliente quanto para o servidor.

```
<operation id="1" method="login" service="account">
  <input-list>
    <input type="string" />
    <input type="string" />
  </input-list>

  <output-list>
    <output type="int" />
  </output-list>
</operation>
```

Figura 5: Trecho do arquivo XML de configuração do *Dispatcher*.

SERVIÇOS – Os serviços da nossa plataforma possuem um ciclo de vida bem definido. Este ciclo compreende o início e o término (métodos *start* e *stop*) da execução de uma instância do serviço, e cabe ao administrador, através do *Allocator*, gerenciá-lo. O *Allocator* provê as funcionalidades de alocação e desalocação dos serviços (Figura 4).

ACESSO A DADOS – A camada de acesso a dados foi implementada utilizando o Hibernate 3 com Annotations [Hibernate 2006b], que oferece uma facilidade de mapeamento objeto-relacional, tornando o acesso a dados mais intuitivo para o desenvolvedor orientado a objetos. O banco de dados utilizado na plataforma foi o PostgreSQL 8.1 [PostgreSQL 2006].

SERVIÇO DE NOMES – A API JNDI (*Java Naming and Directory Interface*) foi utilizada como base para o nosso serviço de nomes. Ela é padronizada e já vem integrada ao ambiente de execução Java. Sua implementação oferece acesso a diversos serviços de diretórios, como LDAP (*Lightweight Directory Access Protocol*) e DNS (*Domain Name System*), e cabe ao desenvolvedor apenas a escolha da implementação desejada [Sun 2006a].

Inicialmente, utilizamos o próprio serviço de nomes de RMI (o *RMI Registry*); contudo, este não permitia que instâncias localizadas em outras máquinas pudessem se cadastrar no registro local. Decidimos então utilizar LDAP [Carter 2003] (através do servidor OpenLDAP [Openldap 2006]), uma vez que JNDI já oferecia suporte e executávamos nossa plataforma em ambientes Linux. Além disso, LDAP disponibiliza acesso a um serviço de diretórios que utilizamos para armazenar não só as referências para os serviços, mas também dados de monitoramento (como o consumo de processamento e memória).

Embora existissem soluções LDAP tolerantes a falhas, consideramos mais promissor o uso de *IP Multicast*, mecanismo através do qual uma mensagem pode ser enviada simultaneamente para vários computadores [Tanenbaum 2002]. A implementação

escolhida foi o *JGroups*, um *toolkit* que permite a troca confiável de mensagens multicast [JGroups 2006]; contudo, sua configuração e suas operações eram demasiadamente complexas. Para obter um maior poder de abstração, resolvemos utilizar o *JBoss Cache*, que utiliza as primitivas do *JGroups* para oferecer uma estrutura de dados replicada automaticamente entre as diversas máquinas [JBoss 2006]. Toda essa complexidade para a implementação de um serviço de nomes tolerante a falhas foi um dos fatores que nos levou a buscar outra abordagem de plataforma.

SEGURANÇA E TRANSAÇÃO – Nesta abordagem, não conseguimos contemplar o conceito de uma camada de interceptação para o controle de transação e segurança. O controle de transação era realizado manualmente pelos serviços, através do Hibernate. Contudo, encadear operações (e conseqüentemente aninhar transações) adicionava mais complexidade aos serviços, dado que é necessário controlar se o contexto transacional já existe e deve-se continuá-lo ou abortá-lo.

Já os aspectos de segurança abordados foram o de autenticação do cliente (via login e senha) e autorização, ambos implementados diretamente no serviço de contas. A dificuldade para implementar esta camada de interceptação foi a outra razão que nos levou a procurar uma abordagem diferente.

4.2. Abordagem B – Adaptação de plataforma

Dadas as dificuldades de integração de um serviço de nomes e da implementação da camada interceptadora de transação e segurança, observamos que estas funcionalidades já eram providas em servidores de aplicação mais maduros. Portanto, decidimos adaptar uma plataforma já existente e que provesse as funcionalidades de infra-estrutura desejadas às nossas necessidades, focando mais na implementação das funcionalidades relativas aos serviços gerais de 3MOGs e do Cellmons.

Escolhemos, então utilizar a J2EE (*Java 2 Platform, Enterprise Edition*), uma plataforma de programação para o desenvolvimento e a execução de aplicações distribuídas multicamadas em Java. A J2EE é definida através de especificação e inclui diversas APIs⁴ tais como conectividade (JDBC), serviço de nomes (JNDI), componentes distribuídos transacionais (EJB), componentes Web (Servlet), dentre outras [Sun 2006f].

A J2EE é realizada através de um contêiner que implementa o contrato estabelecido na especificação, provendo, ainda, gerenciamento de ciclo-de-vida e de

⁴ *Application Programming Interface*, ou Interface de Programação de Aplicativos, é um conjunto de rotinas e padrões estabelecidos por um *software* para utilização de suas funcionalidades.

recursos, segurança, concorrência, transação e implantação aos componentes. Desta forma, o desenvolvedor pode concentrar seu foco nas tarefas inerentes à lógica do seu negócio.

A escolha de J2EE como nossa plataforma-base foi graças à sua semelhança com o modelo proposto na Figura 2 e à quantidade de requisitos (segurança e transação, por exemplo) já implementados pelos contêineres. Em particular, sua instância do servidor pode ser vista na Figura 6.

Para a nossa plataforma, optamos pela escolha do *JBoss Application Server*, dado que ele é *open source* e bastante maduro, oferecendo, ainda, *clustering* transparente [JBoss 2006].

SERVIÇOS – Os serviços da plataforma 3MOG foram implementados como *Stateless Session Beans*, que já provêm escalabilidade, distribuição, transação e segurança. *Stateless Session Beans* são um tipo de *EJB*⁵ que oferecem funcionalidades a um cliente sem manter qualquer estado relativo a este [Sun 2006d].

SERVIÇO DE NOMES – O serviço de nomes utilizado foi o da própria plataforma, o JNDI. Através dele é possível acessar qualquer componente (serviços) ou recurso gerenciado (conexões a SGBDs) disponível em todo o *cluster* de forma transparente. É importante frisar que a API já vem integrada ao contêiner, sem qualquer intervenção do desenvolvedor.

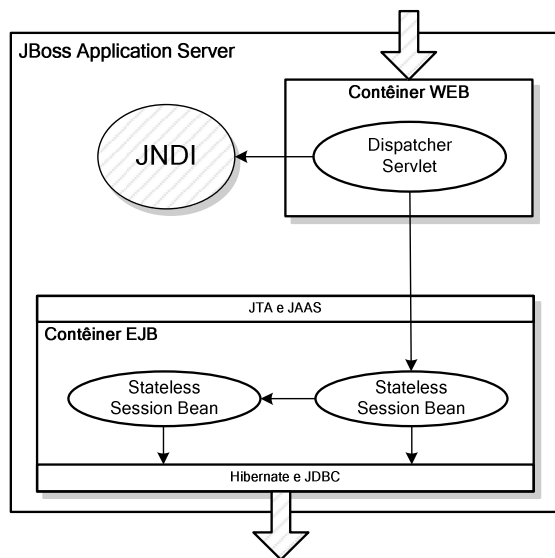


Figura 6: Plataforma 3MOG adaptada à arquitetura J2EE.

DISPATCHER – O *Dispatcher* desta segunda abordagem é idêntico em funcionalidades ao da anterior. A única mudança perceptível é que o serviço recuperado no Serviço de Nomes é um *EJB*.

⁵ *Enterprise JavaBeans*, modelo de componentes da plataforma J2EE

SEGURANÇA E TRANSAÇÃO – A camada de interceptação de transação e segurança já é oferecida pela J2EE através dos modelos de segurança e transação.

O modelo de segurança (JAAS) permite a configuração dos componentes de tal forma que os recursos do sistema só estejam disponíveis para os usuários autorizados.

Já o modelo de transações (JTA) permite a especificação de relacionamentos entre os métodos que constituem uma única transação de tal modo que eles sejam tratados como uma única tarefa.

ACESSO A DADOS – Tal como na Abordagem A, a camada de acesso a dados foi implementada utilizando o Hibernate. Descartamos o uso de *Entity Beans* (componentes EJB específicos para persistência) pela dificuldade de implementação, por algumas restrições desnecessárias [Sun 2006e] e problemas de desempenho em contêineres de código aberto [Tulachan 2002].

4.3. Validação

Para validar estas implementações, fizemos testes de funcionalidade com clientes do jogo em celular e testes de carga contemplando os principais cenários (do login à batalha). A primeira abordagem apresentou melhor tempo de resposta, enquanto a segunda diminuiu drasticamente o esforço de desenvolvimento sem grandes penalidades no desempenho.

5. Conclusão

Diante de todo o exposto, pudemos observar o quão atraente é a idéia de jogos móveis massivamente multiusuário. Entretanto, seu *time-to-market* fica comprometido quando não há uma direção para guiar o desenvolvimento, o que infelizmente não vem sendo tratado na literatura.

Partindo das premissas de que o tempo de resposta ao usuário é extremamente crítico e a facilidade de implementação acelera o *time-to-market*, estudamos como diversos aspectos, tais como escalabilidade, distribuição de carga, tolerância a falhas, segurança, gerenciamento de transações e acesso a dados, fundamentais ao desenvolvimento de 3MOGs, impactam nestas.

Ao empregar duas abordagens distintas, implementando toda a infra-estrutura e serviços e adaptando uma plataforma já existente às nossas necessidades, conseguimos capturar quais são os pontos positivos de cada uma e auxiliar o desenvolvedor na tarefa da escolha de qual rumo tomar. A primeira abordagem oferece benefícios como melhor desempenho por facilitar o acoplamento da plataforma às especificidades do jogo; já a segunda

promete mais facilidade de desenvolvimento, especialmente quando o desejo é apenas criar um protótipo.

Num futuro próximo, pretendemos desenvolver os seguintes trabalhos de melhoria na nossa plataforma:

- Eliminar a restrição de clientes HTTP, permitindo a utilização de *sockets* e outras tecnologias afins;
- Utilizar o protocolo especificado no *Dispatcher* (vide Figura 5) como uma IDL (*Interface Definition Language*, ou linguagem para definição de interfaces), automatizando a geração de código de chamada remota de procedimento (atualmente indisponível nos dispositivos móveis);
- Expandir a plataforma para contemplar outros modelos de jogos, como multiusuário baseado em turnos.

Agradecimentos

Os autores desejam agradecer aos *designers* e *game designers* Alexandre Damasceno, Bruno Palermo, Daniel Benevides, Felipe Borba, Artur Caldas e Arthur Ferreira, da Meantime, por ajudar no desenvolvimento do *game design* e dos gráficos do jogo Cellmons.

Referências

- ACEGI SECURITY SYSTEM, 2006 [online]. Disponível em: <http://www.acegisecurity.org/> [Acessado em 2006-08-22].
- APACHE SOFTWARE FOUNDATION, 2006. *Working with mod_jk* [online]. Disponível em: tomcat.apache.org/mod_3.3-doc/mod_jk-howto.html [Acessado em 2006-08-22].
- BAUGHMAN, N.E., E LEVINE, B.N. *Cheat-proof payout for centralized and distributed online game*. In Proc. of IEEE Infocom 2001, April 2001.
- CARTER, G. *LDAP System Administration*, 4th ed. Addison O'Reilly, 2003.
- CODEHAUS, 2006. *Gabriel Security Framework* [online]. Disponível em: <http://gabriel.codehaus.org/> [Acessado em 2006-08-22].
- ELMASRI, R., E NAVATHE, S. B. *Fundamentals of Database Systems*, 4th ed. Addison Wesley, 2004.
- HIBERNATE, 2006a. *Hibernate Technology*. Disponível em: <http://www.hibernate.org/5.html> [Acessado em 2006-08-22].
- HIBERNATE, 2006b. *Hibernate Annotation* [online]. Disponível em: <http://www.hibernate.org/247.html> [Acessado em 2006-08-22].
- JBoss, 2006a. *JBoss Cache* [online]. Disponível em: <http://www.jboss.org/products/jboss-cache> [Acessado em 2006-08-22].
- JBoss, 2006b. *Official Website* [online]. Disponível em: <http://labs.jboss.com/portal/> [Acessado em 2006-08-22].
- JGROUPS, 2006. *Official Website* [online]. Disponível em: <http://www.jgroups.org/javagroupsnew/docs/index.html> [Acessado em 2006-08-22].
- KNUTSSON B., LU, H., XU, W. E HOPKINS, B. Peer-to-peer support for massively multiplayer games. In The 23rd Conference of the IEEE Communications Society (INFOCOM'04), Hong Kong, China, march 2004.
- MARCUS, E., E STERN, H. *Blueprints for High Availability*, 2nd ed, Wiley, USA, 2002.
- NOKIA CORPORATION, 2006a. SNAP Mobile: Technical Overview. Forum Nokia. Version 1.1. 27 April 2006.
- NOKIA CORPORATION, 2006b. SNAP Mobile: White Paper. Forum Nokia. Version 1.0. 9 May 2006.
- OPEN MOBILE ALLIANCE, 2003. Gaming Platform Version 1.0. Candidate Version. 12 June 2003.
- OPENLDAP, 2006. *Official Website* [online]. Disponível em: <http://www.openldap.org/> [Acessado em 2006-08-22].
- ORACLE, 2006. *Oracle TopLink* [online]. Disponível em: <http://www.oracle.com/technology/products/ias/toplink/index.html> [Acessado em 2006-08-22].
- PELLERIN, R., DELPIANO, F., DUCLOS, F., GRESSION-SOUDAN, E. E SIMATIC, M., 2005. GASP: an open source gaming service middleware dedicated to multiplayer games for J2ME based mobile phones. In: 7th Conference on Computer Games CGAMES'05, 28-30 November 2005. White Paper.
- POSTGRESQL, 2006. *Official Website* [online]. Disponível em: <http://www.postgresql.org> [Acessado em 2006-08-22].
- SAHA, D., SAHU, S. E SHAIKH, A., 2003. A platform for on-line games. In: *Proceedings of the 2nd workshop on Network and system support for games*, 22-23 May 2003 Redwood City. ACM Press, pages 180–184.
- SUN MICROSYSTEMS, 2006a. *Sun SE Guide to JNDI* [online]. Disponível em: <http://java.sun.com/j2se/1.5.0/docs/guide/jndi/index.html> [Acessado em 2006-08-22].
- SUN MICROSYSTEMS, 2006b. *Java RMI Specification* [online]. Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html> [Acessado em 2006-08-22].
- SUN MICROSYSTEMS, 2006c. *Java Authorization and Authentication Service* [online]. Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html> [Acessado em 2006-08-22].
- SUN MICROSYSTEMS, 2006d. *Enterprise JavaBeans Technology*. Disponível em: <http://java.sun.com/products/ejb/> [Acessado em 2006-08-22].

SUN MICROSYSTEMS, 2006e. *Article: Seven rules for optimizing Entity Beans*. Disponível em: <http://java.sun.com/developer/technicalArticles/ebeans/sevenrules/> [Acessado em 2006-08-22].

SUN MICROSYSTEMS, 2006f. *Java Enterprise Edition [online]*. Disponível em: <http://java.sun.com/javase/> [Acessado em 2006-08-22].

TANENBAUM, A., E VAN STEEN, M. *Distributed Systems: Principles and Paradigms*, Prentice Hall, Pearson Education, USA, 2002.

TANENBAUM, A. *Computer Networks*, 4th ed, Prentice Hall, Pearson Education, USA, 2002.

TERRAPLAY, 2003. Terraplay Move, Mobile Multiplayer Gaming Service. *White Paper*.

TULACHAN, P. V., 2002. *Developing EJB 2.0 Components*. Prentice Hall, USA, 2002.