Desenho e implementação de um jogo distribuído na Internet

Oliveira Rui, Araújo Paulo, Costa Orlando

Universidade do Minho, Gualtar, Portugal
http://www.uminho.pt

Resumo O propósito deste trabalho consiste na implementação de um jogo em tempo real que se baseia num serviço de distribuição de conteúdos com posterior interação por parte dos utilizadores e que otimize a utilização da largura de banda. A aplicação encontra-se implementada utilizando a linguagem de programação Java, e aborda paradigmas de distribuição de dados tais como *broadcast* e *unicast*. O jogo consiste num quiz (do estilo quem quer ser milionário), e permite aos utilizadores registarem-se e participar em desafios, assim como criar desafios outros para competir com demais utilizadores. A arquitetura encontra-se dividida em aplicações cliente e servidor, sendo que estas comunicam não apenas entre si (cliente-servidor), mas consigo próprias (servidor-servidor), permitindo trocar informação sobre a massa global de utilizadores que participam no jogo. A troca de informação é efetuada através de protocolos TCP e UDP, estando definido um PDU personalizado com o objetivo de diminuir a utilização de largura de banda. Este documento apresenta o trabalho desenvolvido.

Keywords: PDU, TCP, UDP, Servidores, Clientes, Comunicação por Computadores

1 Introdução

Os servidores possuem a capacidade de processar diversos clientes simultaneamente (através de *threads*), e possuem toda a lógica do jogo, nomeadamente as questões a serem realizadas e o controlo e anúncio dos resultados. Estes possuem também a capacidade de comunicar não apenas com os clientes mas entre si, de forma a homogeneizar o estado da aplicação. Na aplicação cliente reside a interface gráfica que interage com o utilizador, e onde são apresentadas as questões e imagens relacionadas com o jogo, assim como a música. O sistema distribuído implementa comunicações TCP entre servidores e comunicações UDP entre clientes e servidores, e essas comunicações são realizadas através de PDU's criados com o intuito de poupar largura de banda e transferir dados entre unidades computacionais. As diferenças entre os PDU's implementados e os descritos no enunciado encontram-se descritas neste documento, assim como toda a estrutura dos diferentes componentes que constituem a aplicação geral.

2 Diferenças da especificação

O protocolo implementado segue o proposto no enunciado com duas diferenças. Na lista de argumentos de uma PDU tinha o tamanho do parâmetro. Estava previsto que este tamanho apenas tivesse 1 bytes, mas isto não era suficiente por exemplo no caso do bloco de musica. Portanto modificou-se para 2 bytes. Adicionou-se também um tipo de PDU de controlo para os servidores fecharem os sockets em segurança.

3 Implementação

3.1 Protocolo

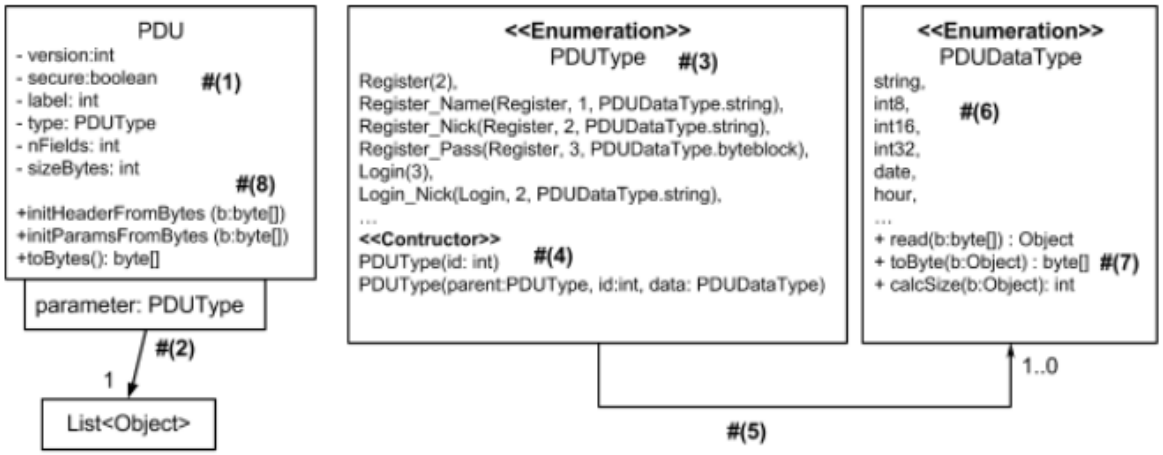

Figura 1. Diagrama UMP da interpretação de PDU

Para a abstração dos Bits, foi criada a classe PDU, que é capaz de criar as PDU's e depois transforma-as em Bytes e também inicializar-se através de array de Bytes. Internamente existem os parâmetros do cabeçalho da PDU [#(1)] (versão, segurança, label, tipo da PDU, número de campos e tamanho tamanho total dos campos em Bytes), toda esta informação pode ser consultada com os seus respetivos métodos. Para guardar os campos da PDU, utiliza-se um Map [#(2)] onde a chave é o tipo do campo e o valor é uma lista de valores. Trata-se de uma lista devido a que a mesma PDU pode ter vários campos com o mesmo tipo como p.e. uma questão que tem varias respostas.

A Enumeração PDUType [#(3)] é uma enumeração hierárquica: existem os valores de origem que representam o tipo da PDU como por exemplo: Register, Login, Reply. Existem também os valores que representam os campos, como

por exemplo: Register_Name ou Register_Nick. A diferença entre estes dois está no construtor utilizado [#(4)]. No caso do construtor de um campo podemos reparar que recebe um tipo de dados (PDUDataType).

A enumeração PDUDataType [#(6)] tem todos os tipos de dados que o PDU suporta, e aqui ficam implementadas as funções de leitura e conversão para arrays de bytes. Desta forma é fácil a adição de novos tipos de dados.

O diagrama 2 UML explica o funcionamento da interpretação do campos do PDU (função initParamsFromBytes).

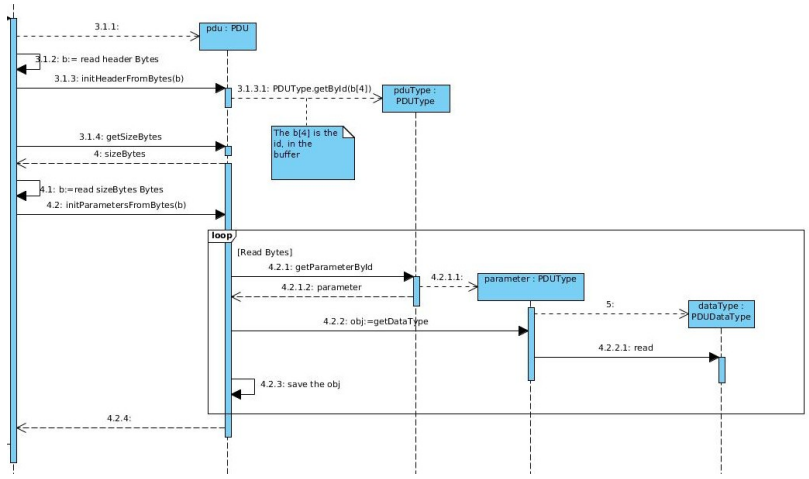

Figura 2. Diagrama UML da interpretação de PDU

3.2 Cliente

A aplicação cliente encontra-se implementada com o auxílio da classe UDPClient. Este classe contém todos os métodos que formam os pacotes de dados a serem transmitidos através de protocolos UDP, sendo que a lógica desta comunicação encontra-se definida na classe UDPCommunication. A interface gráfica possui assim uma instância de UDPClient, com a qual efetua todos os desafios e troca de informação com os servidores. Esta classe constrói-se através dos parâmetros referente ao IP e porta local, assim como o IP e porta do servidor com o qual comunica nesse momento. Esta instância é criada assim que o utilizador inicia a bash através da qual introduz os comandos de comunicação com o servidor. Quando o servidor se encontra conectado ao cliente e este efetua login, o servidor é capaz de manter o estado do cliente dado o seu IP.

Interface gráfica A interface com a qual a aplicação interage com o utilizador encontra-se desenvolvida em Java Swing, e apresenta de forma simples o

conteúdo recebido na aplicação cliente referente às questões de um dado desafio. Assim, quando o utilizador aceita entrar num desafio, é iniciada a interface gráfica, bloqueando o utilizador de participar em mais desafios simultaneamente. Quando é alcançada a hora marcada para efetuar o desafio, a 1ª questão é apresentada, sendo possível escutar a música e observar a imagem associadas a esta. O utilizador responde à pergunta apresentada clicando nos botões disponíveis para responder, sendo que esta ação pára a música atualmente a tocar, e bloqueia os restantes botões de forma a não permitir comportamentos indesejados na aplicação. Após a resposta à pergunta, o datagrama de resposta é enviado através da aplicação cliente, e o resultado é apresentado ao utilizador. Este processo repete-se até ao fim do desafio, altura na qual é apresentado ao utilizador a sua pontuação final. 3

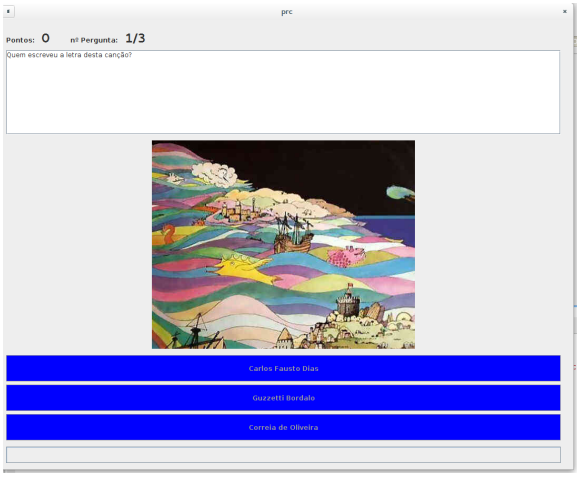

Figura 3. Imagem da interface gráfica do cliente

3.3 Servidor

O servidor é responsável por guardar a informação sobre o sistema tal como os desafios e utilizadores existentes. Por outro lado o servidor também é responsável por partilhar o seu conhecimento com os outros servidores. No arranque o servidor cria 2 threads, uma deles para atender clientes, e outra para atender outros servidores.

A arquitetura geral de um servidor pode ser vista no diagrama 4. Também em anexado a este relatório existe um diagrama com a arquitetura de 3 servidores ligados entre si.

Dados Guardados Toda a informação relativa ao atual estado de um servidor são guardados na classe ServerState:

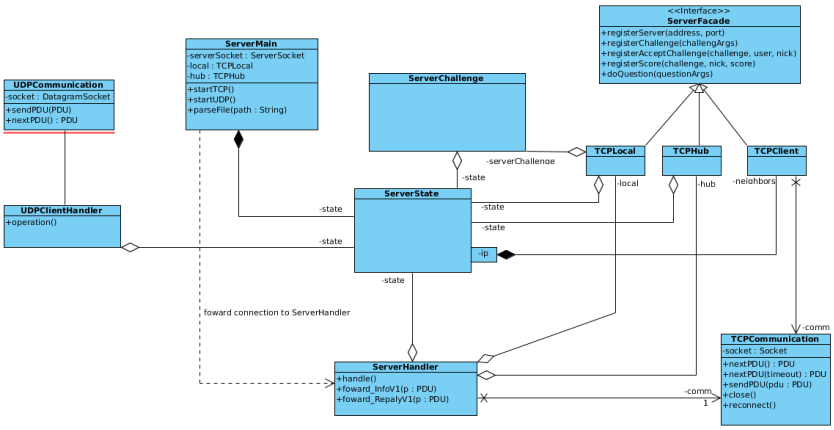

Figura 4. Diagrama UML da arquitetura do servidor

- Utilizadores, identificados pelo seu nick.
- Sessões, guardam os utilizadores identificados pelo seu ip actual.
- Utilizadores globais, para guardar o ranking global do sistema.
- Desafios, identificados pelo seu nome.
- Questões para criar novos desafios, identificados pelo seu ip.
- Servidores vizinhos, com os quais comunicar.

Cada desafio tem o ranking atual dos utilizadores inscritos nesse desafio, os utilizadores e servidores subscritos e as questões selecionadas.

Atender Clientes Um pedido vindo de um cliente é reencaminhado para a class UDPClientHandler. Mediante o tipo de desafio, são realizadas as regras de negócio correspondentes e é enviada uma resposta para o cliente.

Existem alguns pedidos do cliente que podem obrigar o servidor a informar os restantes servidores, como p.e., makeChallenge, que inicia um desafio e irá ser explicado de seguida.

Iniciar desafio Quando um makeChallenge e gerado, o sistema cria uma nova thread que terá um temporizador para correr às horas pretendidas. Quando esta thread inicia o seu processo confirma que existem pessoas suficientes no desafio, caso não existam cancela o desafio. Caso esteja tudo bem continua e envia a primeira pergunta.

As perguntas ao longo do desafio são enviadas para todos os clientes subscritos e para todos os servidores subscritos, depois cada um destes servidores reencaminhará para os clientes finais.

Atender Servidores O processo de atender servidores é semelhante ao de atender clientes. Os pedidos são interpretados na class ServerHandler, que mediante os parâmetros existentes irá executar a lógica necessária. A lógica está implementado em 3 classes distintas: TCPLocal, TCPHub e TCPClient. Cada uma destas implementa a interface ServerToServerFacade que especifica todos os métodos existentes no negócio da aplicação (diagrama 5).

Cada uma das implementações implementa o negocio de forma diferente:

- TCPLocal aplica as regras de negocio na propria maquina.
- TCPClient envia um pedido a um servidor para aplicar aquele método.
- TCPHub aplica as regras de negocio a todos os servidores chamando o TCPClient de cada servidor.

O socket que permite a comunicação entre servidores é terminado com um timeout parameterizavel.

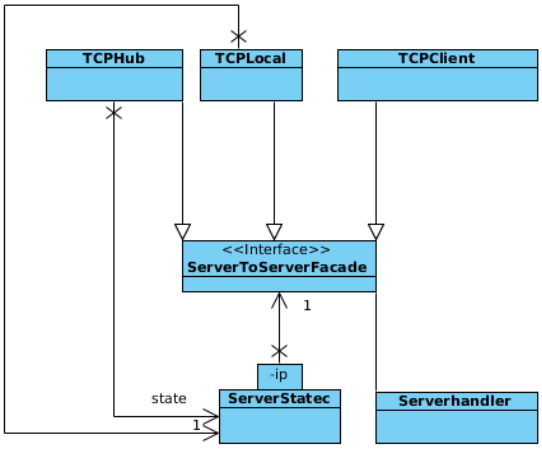

Figura 5. Diagrama UML da arquitetura do servidor

Informar Servidores Vizinhos Para comunicar com outros servidores utiliza-se TCPHub ou TCPClient mediante as necessidades. Existe um TCPClient para cada servidor vizinho, e quando invocamos um método desta classe começa por abrir um socket caso seja necessário, e envia uma PDU do tipo INFO com a informação pretendida.

4 Testes Realizados

De forma a validar o trabalho realizado, foram efetuados testes sobre a plataforma. Os testes foram realizados com a utilização de um router, sendo que

foi necessária a sua configuração de modo a que, a partir do MAC Address dos pc's conectados, fossem atribuídos IP's escolhidos por nós. Através da criação de várias instâncias de clientes e servidores nos computadores conectados, é possível criar uma simulação de um ambiente de comunicação real no qual existem servidores dedicados que recebem os pedidos de ligação dos respetivos clientes.

4.1 Registo de um Utilizador

- IP Servidor: 127.0.0.2
- IP Cliente: 127.0.0.75

Cliente: REGISTER joao rodrigues 123
I'm[/127.0.0.75] sending to [/127.0.0.2] PDU: PDU,parameters:{REGISTER_PASS=[[B@1d56ce6a], REGISTER_NAME=[joao], REGISTER_NICK=[rodrigues]}
I'm[/127.0.0.2] sending to [/127.0.0.75] PDU: PDU,parameters:{REPLY_OK=[0]}
Servidor: OK

4.2 Participação em Desafio

- IP Servidor: 127.0.0.1
- IP Cliente1: 127.0.0.66
- IP Cliente2: 127.0.0.69

Cliente1: MAKE_CHALLENGE Circo 2015-05-02 15:00
I'm[/127.0.0.66] sending to [/127.0.0.1] PDU: PDU,parameters:{MAKE_CHALLENGE_CHALLENGE=
MAKE_CHALLENGE_HOUR=[15:00], MAKE_CHALLENGE_DATE=[2015-05-
02]}
I'm[/127.0.0.1] sending to [/127.0.0.66] PDU: PDU,parameters:{REPLY_CHALLE=[Circo], REPLY_DATE=[2015-05-02], REPLY_HOUR=[15:00]}
Servidor: Desafio: Circo Data: 2015-05-02 Hora: 15:00
Cliente2: ACCEPT_CHALLENGE Circo
I'm[/127.0.0.69] sending to [/127.0.0.1] PDU: PDU,parameters:{ACCEPT_CHALLENG
I'm[/127.0.0.1] sending to [/127.0.0.69] PDU: PDU,parameters:{REPLY_OK=[0]}
Servidor: OK