Desenho e implementação de um jogo distribuído na Internet

Oliveira Rui, Araujo Paulo, Costa Orlando

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

Resumo The abstract should summarize the contents of the paper and should contain at least 70 and at most 150 words. It should be written using the *abstract* environment.

Keywords: We would like to encourage you to list your keywords within the abstract section

1 Introdução

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

Para a abstração dos Bits, foi criada a classe PDU, que é capaz de criar as PDU's e depois transformas 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. Uma lista porque a mesma PDU pode ter varias campos com o mesmo tipo como é o exemplo de uma questão que têm varias respostas.

A Enumeração PDUType [#(3)] é uma enumeração hierárquica, isto é: existem os valores de origem que representam o tipo da PDU como por exemplo: Register, Login, Replay. Depois existem os valores que representam os campos, como por exemplo: Register_Name ou Register_Nick. A Diferença entre estes

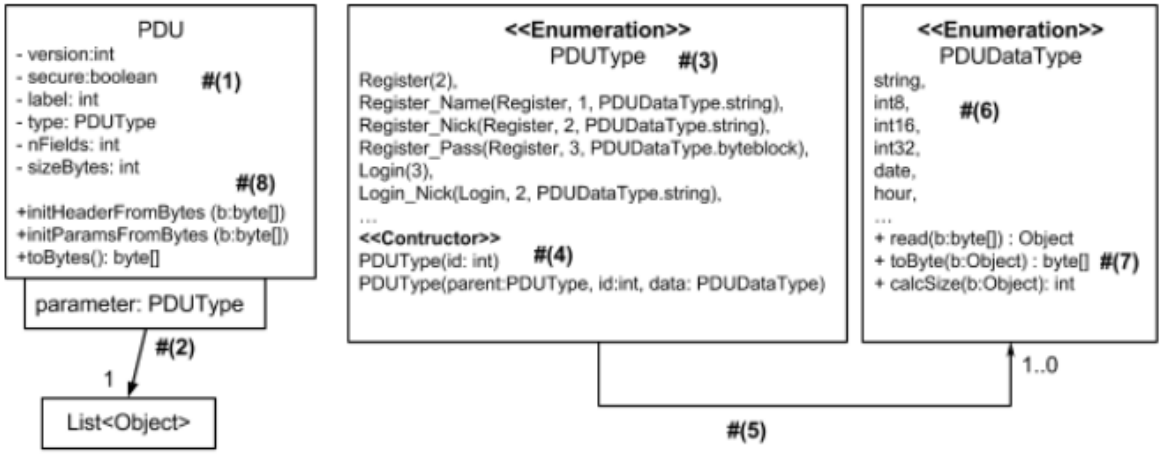

Figura 1. Diagrama UMP da interpretação de PDU

dois está no construtor utilizada [#(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 suposta, 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).

3.2 Cliente

3.3 Servidor

O Servidor é responsável guardar a informação sobre do sistema tais 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, um deles para atender clientes, e outro para atender outros servidores

A arquitetura geral de um servidor posso ser vista no diagrama 3.

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

- Utilizadores identificados por o seu nick
- Sessões, guardam os utilizadores identificados por o seu ip actual
- Utilizadores globais, para guarda o ranking global do sistema.
- Desafios identificados por o seu nome
- Questões para criar novos desafios, identificados por o seu ip

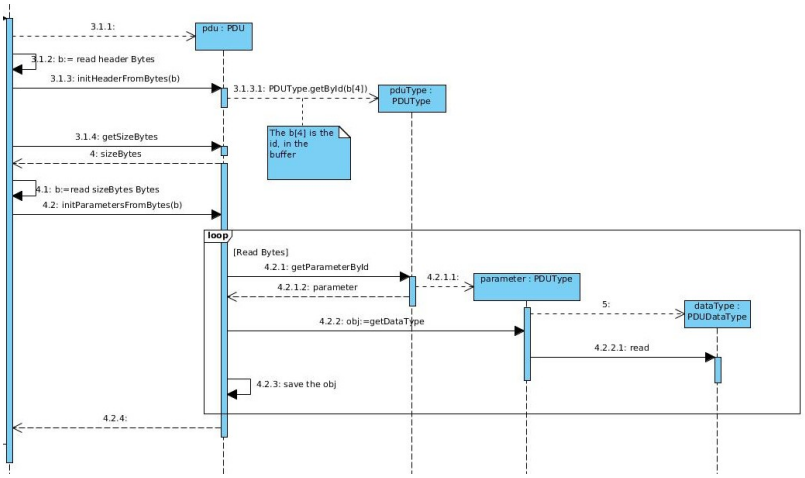

Figura 2. Diagrama UMP da interpretação de PDU

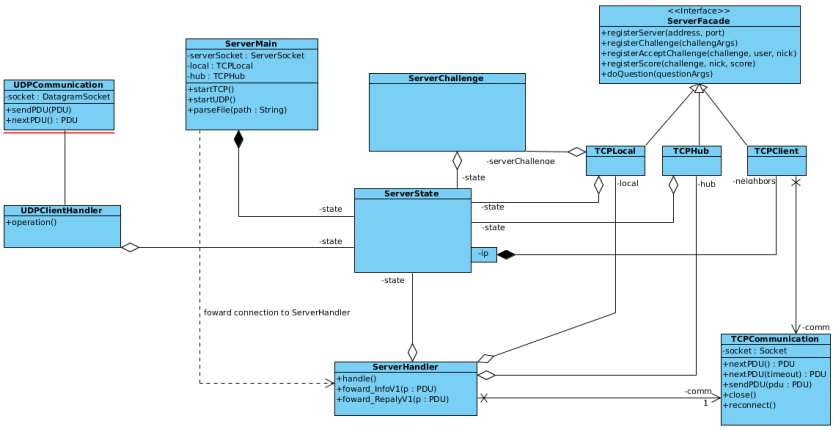

Figura 3. Diagrama UML da arquitetura do servidor

– Servidores vizinhos poder comunicar.

No caso dos desafios 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, aqui mediante o tipo de desafio, é realizada as regras de negocio correspondente e enviada uma resposta para o cliente.

Existem alguns pedidos do cliente que podem gerar que o servidor que esteja a antender tenha que informar os restantes servidores como por exemplo: 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 temporalizador 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 reencainhará para os clientes finais.

Atender Servidores O processo de atender servidores é semelhante ao de atender clientes. Os pedidos são interpretados na class ServerHandler, esta class mediante os parametros existentes irá executar a logica de negocio necessária. O negocio está implementado em 3 classes distintas: TCPLocal, TCPHub e TCPClient. Cada uma destas implementam a interface ServerToServerFacade que especifica todos os métodos existentes no negocio da aplicação (diagrama 4).

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.

Informar Servidores Vizinhos Para comunicar com outros servidores utiliza-se ou TCPHub ou TCPClient mediante as necessidades. O TCPClient que irá fazer é abrir um socket se necessário, e enviar uma PDU do tipo INFO com a informação pretendida.

4 Testes Realizados

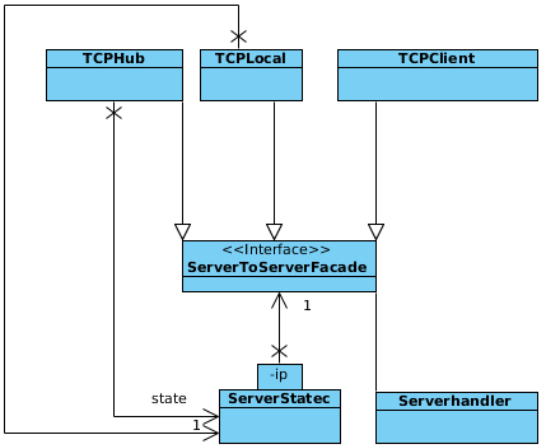

Figura 4. Diagrama UML da arquitetura do servidor