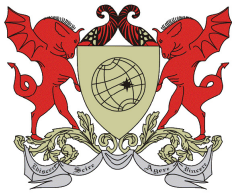

Universidade Federal de Viçosa
Campus Florestal

Trabalho Prático 2 - Sistemas Distribuídos

Bruno Ribeiro Diniz - 2648

Caio Victor Fernandes Silva - 2647

Florestal
Junho de 2019

Sumário

1 Introdução

Neste trabalho, o objetivo foi o desenvolvimento de um sistema distribuído que funciona da seguinte maneira: o cliente manda uma mensagem para o servidor solicitando uma figura geométrica, o servidor escolhe uma figura aleatoriamente(quadrado, retângulo, círculo ou triângulo) e envia para o cliente.Esse processo se repete até que atenda às X requisições do cliente.

Para a comunicação entre cliente/servidor fora utilizadas três estratégias diferentes, sendo elas: comunicação utilizando a API de Socket, comunicação utilizando Java RMI e comunicação utilizando Web Service. Na API de Sockets, temos a comunicação feita entre processos, apenas com o envio de mensagens. Em Java RMI acontece a invocação remota do método que envia figura geométrica do servidor para o cliente. E no Web Service, o envio de uma figura geométrica se torna um serviço, que é oferecido para ser acessado por um cliente remoto.

2 Desenvolvimento

Aqui, serão mostrados os três tipos de comunicação utilizados, descrevendo também, como essas estratégias foram utilizadas juntamente com nosso sistema distribuído.

2.1 API de Sockets

Nesta etapa, optamos pelo protocolo TCP. Para a implementação, utilizamos o exemplo disponível nos slides. Após entendermos cada etapa que estava sendo executada nesse algoritmo, bastou realizar algumas mudanças para que o mesmo se encaixasse em nosso contexto.

Para essa essa comunicação, foram criadas três classes:

-ClienteSocket.java: O objetivo desta classe é exercer o papel do cliente no sistema distribuído.Sua função é basicamente se conectar ao servidor, e através dessa conexão, enviar as requisições para que receba as figuras geométricas. Dentro desta classe é configurado o número de requisições que serão feitas, o mesmo, é escolhido na interface.

-ServidorSocket.java: Esta classe tem a função de iniciar o servidor do sistema distribuído em uma porta e ficar "escutando", para que, quando um novo cliente fizer a requisição de se conectar, essa conexão ocorra e o mesmo seja enviado para a classe Connection.

-Connection.java: A classe Connection implementa thread, devido ao fato de que, um cliente não deve atrapalhar a execução dos demais. Portanto, sempre que um cliente se conecta, é criada uma thread, que ficará aberta enquanto o mesmo estiver conectado. A classe implementa o método run, que é o responsável por fazer o processamento(gerar a figura aleatoriamente), para que a resposta seja enviada para o cliente.

2.2 Java RMI

Para implementar a comunicação utilizando Java RMI, buscamos informações no tutorial presente na especificação do trabalho. No tutorial, os conceitos e utilizações estavam bem especificados, portanto, bastou que entendêssemos cada parte, para que pudéssemos adaptar para o nosso contexto.

Para essa comunicação foram implementadas três classes:

-ClientRMI.java: Utilizando o Java RMI, o nosso cliente passa a ter a função de invocar um método que se encontra presente na interface remota.

-ServerRMI.java: O servidor nesse caso, é responsável por implementar um objeto remoto. Este objeto implementa a interface remota. Ele cria uma instância da implementação do objeto remoto, exporta-o e em seguida, a associa a um nome em um registro.

-InterfaceRemota.java: Esta classe estende a interface java.rmi.Remote e tem como principal objetivo declarar um conjunto dos métodos remotos oferecidos pelo servidor. No nosso contexto, temos apenas uma método disponível na mesma. O método escolheFigura(), que será responsável por retornar uma figura quando o cliente requisitar.

2.3 Web Service

Para a comunicação utilizando Web Service, optamos por SOAP. Na especificação do trabalho, alguns exemplos foram passados, e com base neles, implementamos a comunicação para o nosso contexto. As seguintes classes foram implementadas:

-ClientWS.java: Para que essa classe cliente consiga acessar os serviços do servidor, alguns passos devem ser feitos. Necessitamos passar o endereço do web service para a URL, indicando onde o serviço poderá ser encontrado. Desse modo, o cliente pode acessar o serviço oferecido.

-ServerWS.java: No servidor, utilizamos a classe Endpoint, que nos permite fazer a publicação do serviço. Precisamos passar como parâmetro para ela a porta em que o servidor irá se encontrar e a instância do serviço a ser publicado.

-ServiceWS.java: Esta classe contém a implementação e descrição dos métodos da interface. No caso do nosso contexto, é nesta classe que se encontra a implementação da geração de uma figura aleatória a cada requisição.

-InterfaceServer.java: Nesta classe, temos apenas a implementação da interface responsável por oferecer o serviço. Ressaltando a notação @Webservice, que avisa ao compilador java que o arquivo atual corresponde à definição SEI de um serviço Web. E @WebMethod indica que um determinado método corresponde a uma operação de serviço e assim pode ser invocado por um cliente.

3 Execução e exemplos de saída

Para executar o SD implementado, os seguintes passos devem ser seguidos:
-Inicializar os servidores dos três tipos de comunicação.
-Executar a classe MainScreen.java.
Após isso, a seguinte janela será aberta:

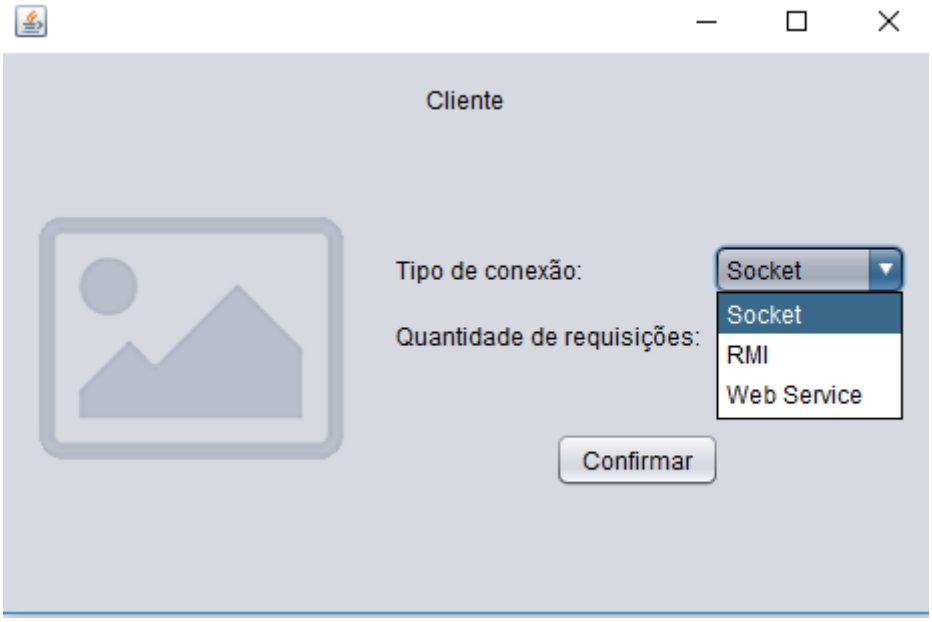

Figura 1: Tela Inicial

Esta é a tela inicial. Nela o cliente pode escolher o tipo de comunicação que será feita, e o número de requisições que o mesmo deseja. Para que mais de um cliente seja criado, basta executar a clase MainScreen.java novamente, e uma nova janela será aberta para este novo cliente. A seguir, serão mostradas algumas imagens que mostram o funcionamento do SD depois que o usuário seleciona as configurações e clica no botão "Confirmar":

Figura 2: Resposta obtida do servidor: "retângulo"

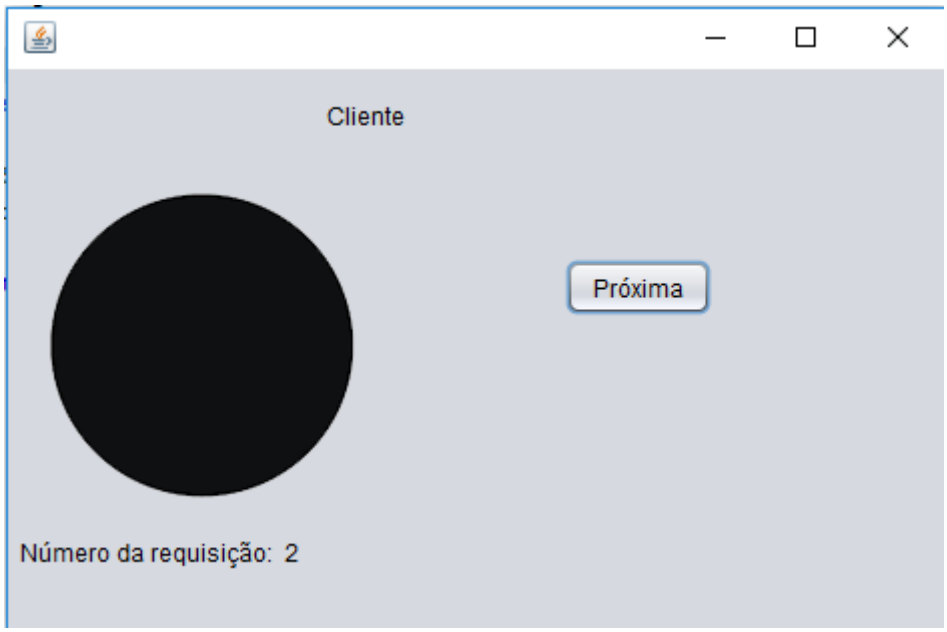

Figura 3: Resposta obtida do servidor: "circulo"

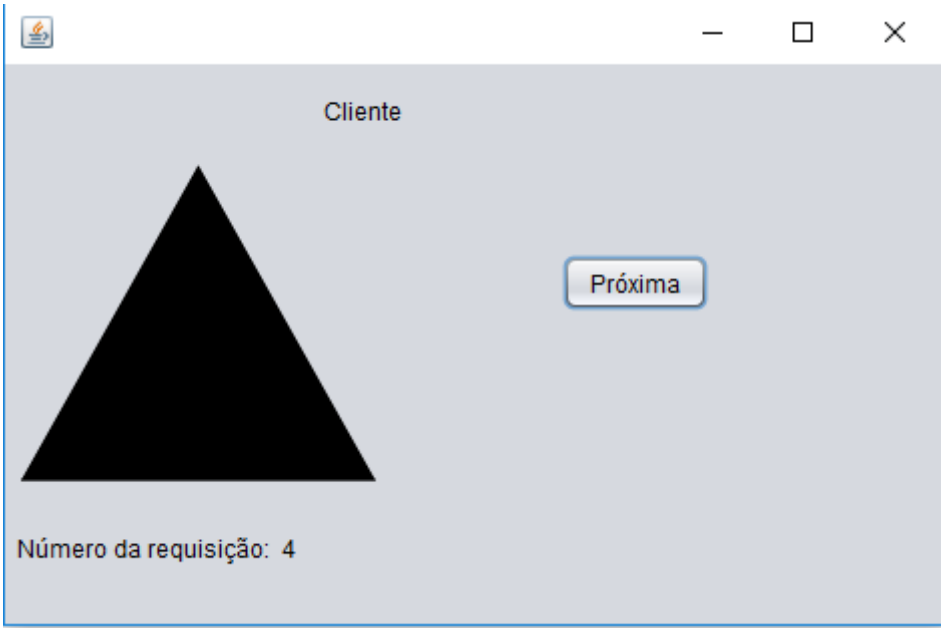

Figura 4: Resposta obtida do servidor: "triangulo"

4 Conclusão

Com este trabalho, tivemos a oportunidade de utilizar na prática as comunicações para sistemas distribuídos vistas em sala de aula. Deste modo, os conceitos e características de cada uma delas ficou bem mais claro para os integrantes.

A principal observação durante o desenvolvimento deste trabalho, foi o nível de abstração que é bem diferente para cada um dos tipos de comunicação. O javaRMI e o Web Service abstraem muitas atividades do programador, desde modo, ele pode programar um sistema distribuído mesmo sem ter um conhecimento avançado em como a comunicação é feita em baixo nível.

5 Referências

https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/index.html

https://www.devmedia.com.br/desenvolvendo-e-usando-web-services-em-java/37261/