Trabalho 2 - Sistemas Distribuídos

Dupla: Antonio Herik Como Martins (516098) e Luigy Gabriel de Oliveira Ferreira (542161)

O Trabalho 2 da disciplina de Sistemas Distribuídos teve como objetivo implementar a comunicação entre cliente e servidor utilizando RMI (Remote Method Invocation). A aplicação segue a mesma proposta do Trabalho 1: um sistema de venda de livros (sebo), agora com a comunicação feita por requisição-resposta.

A base da comunicação foi a interface *RemoteInterface*, que estende a interface Remote do Java RMI. Nela, foi definido o método *doOperation*, que é o responsável por receber as requisições do cliente. Os parâmetros são: uma *RemoteObjectRef* (que identifica o objeto remoto), o *methodId* (nome do método que deve ser executado) e os *arguments* (os dados que o método precisa). Além disso, o método também recebe informações do cliente, como o IP e a porta de origem da chamada. A implementação dessa interface ficou na classe *RemoteInterfaceImpl*, onde estão definidos os métodos que podem ser chamados remotamente.

Essa implementação usa duas classes de serviço: *ProdutoService* e *VendasService*, que encapsulam as regras de negócio da aplicação. Quando uma requisição chega, ela é desserializada e o método solicitado é identificado e executado. Se o método precisar de parâmetros, eles são extraídos da requisição. Depois da execução, o resultado (seja sucesso ou erro) é serializado de volta para o cliente.

Para estruturar essa troca de dados entre cliente e servidor, foram criadas as classes *RequestMessage* e *ReplyMessage*, seguindo o formato pedido no enunciado. Elas têm os seguintes campos:

- *messageType*: 0 para requisição e 1 para resposta
- *requestId*: um número que identifica a requisição
- *objectReference*: nome do serviço remoto a ser acessado
- *methodId*: nome do método que será executado
- *arguments*: os parâmetros (em JSON)
- *result*: o retorno do método, presente só na resposta

Para serializar e desserializar os dados, foi utilizado o formato *JSON* com a biblioteca Jackson. A classe *JsonMarshaller* foi criada para facilitar esse processo, com métodos que transformam objetos em *JSON* e vice-versa.

Do lado do servidor, a classe *RMIServer* inicializa o sistema, adiciona alguns livros de exemplo, cria os serviços, instancia o objeto remoto e registra tudo no *RMIRegistry*, usando uma porta e um nome de serviço. Com isso, o servidor fica pronto para receber chamadas dos clientes.

Já o cliente é representado pela classe *RMIClient*, que se conecta ao servidor, monta as mensagens com o nome do serviço, o método e os parâmetros, e envia tudo através do *doOperation*. O cliente também gera um ID único para cada requisição usando

AtomicInteger, e cuida de empacotar os dados em *JSON* e interpretar as respostas recebidas do servidor.

Por fim, foi criada uma classe *TestRMI* para testar as chamads utilizando a classe de cliente do RMI.

A implementação atendeu aos seguintes requisitos exigidos:

- 4 classes entidade: Produto, Livro, Apostila, CD, DVD, Ebook e Loja
- 2 composições "tem-um": Loja possui uma lista de produtos
- 2 composições "é-um": Livro, Apostila, CD, DVD e Ebook estendem Produto
- 4 métodos remotos: getProductDetails, sellProduct, searchProductsByTitle, getAllBooks
- Passagem por referência: Referência RMI obtida via RMIRegistry
- Passagem por valor: Objetos serializados em JSON entre cliente e servidor

As principais dificuldades ao implementar o trabalho foi no momento de serializar e desserializar os objetos no momento das requisições, onde aconteciam erros de inconsistência de tipo de dado ou dados incorretos.