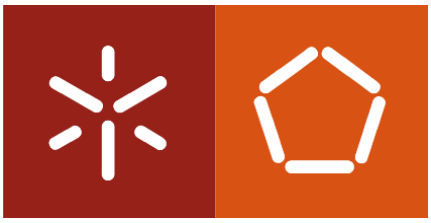

UNIVERSIDADE DO MINHO
Mestrado Integrado em Engenharia Informática

FUNDAMENTOS DE SISTEMAS DISTRIBUIDOS

Livraria - Protótipo de sistema distribuído

Alexandre Silva — a73674
André Cunha Santos — a61778
Renato Rebelo — pg33872

Braga, 18 de Fevereiro de 2018

Resumo

Este trabalho foi feito no âmbito da disciplina de Fundamentos de Sistemas Distribuidos em que implementamos um protótipo de sistema distribuído e gestão de uma livraria usando objetos e transações distribuídos, de modo a tornar transparente a distribuição. Numa primeira fase serão apresentados os conteúdos e estrutura do nosso protótipo e de seguida serão dados alguns exemplos da sua utilização.

Conteúdo

1. Protótipo

Como já foi dito anteriormente, este trabalho consiste na criação de uma aplicação distribuída que ofereça a um programador uma API para poder gerir uma livraria.

Em seguida é apresentado e explicado todos os processos que foram construídos de modo a criarmos uma aplicação distribuída, contudo transparente nesse sentido para o utilizador, de forma modelar e sem gargalos.

1.1 Cliente

Este é o processo que usufrui da API do servidor das livrarias.

Neste processo foi criado um *Stub* que é responsável por fazer os pedidos remotos ao servidor de livrarias. Para o programador poder utilizar a API foram criadas três classes que servem a função de *Stub*:

- **LivrariaC:** que é responsável por procurar livros no servidor, criar novos carrinhos de compras e obter o histórico de uma conta bancária.
- **LivroC:** esta classe é responsável pelos métodos relacionados com os livros como por exemplo, obter um título ou o preço de um livro.
- **CarrinhoC:** por fim foi criado o CarrinhoC que tem as funções de adicionar um livro ao carrinho e finalizar o carrinho, procedendo assim à sua compra.

1.2 Livraria

Este processo é o servidor da livraria. Inicialmente foram criados os objetos que dão as funcionalidades ao sistema como o Livro, Livraria e Carrinho. Estas três classes implementam as três interfaces dos Stubs no processo de cliente e representam o *skeleton* na parte do servidor.

O servidor mantém um *Map* com todos os objetos do servidor e sempre que um pedido novo chega ao servidor, uma *thread* "atende"esse pedido e utiliza estes objetos para responderem ao pedido.

O servidor de livrarias, ao finalizar uma compra, tem que fazer um pagamento a uma conta

bancária num banco. Esse serviço é disponibilizado por outro processo, o servidor de bancos que é explicado em seguida. Contudo, o servidor de livraria atua como um cliente deste outro servidor.

Neste sentido, foi criado um *stub* neste servidor, tal como no processo cliente, para que o servidor de livrarias comunicasse e utilizasse a API do servidor de bancos.

1.3 Banco

O processo banco é o servidor de bancos. Este servidor disponibiliza bancos e contas bancárias bem como uma API para interagir com estes objetos.

Este servidor foi idealizado e construído segundo os mesmos princípios do servidor de livrarias. Foram usados Stubs/Skeletons para a sua comunicação.

1.4 Interfaces utilizadas nos *Stubs/Skeletons*

Existem 5 interfaces:

- **Livraria:**É composta pelos métodos:
  - **novoCarrinho()**, que cria um novo Carrinho.
  - **procurarLivro(String titulo)**, que retorna um livro com um determinado título.
  - **getHistorico()**, que responde com uma lista de todos os livros comprados com uma determinada conta bancaria.
- **Livro:** Esta interface contém apenas as funções *get* do Livro. Todas elas retornam o atributo referido no método.
  - *getTitle()*
  - *getIsbn()*
  - *getPrecos()*
- **Carrinho:** Esta interface permite adicionar livros ao carrinho bem como finalizar a respetiva compra:
  - *adicionaLivro(Livro livro)*, que adiciona um Livro ao carrinho de compras.
  - *finalizarCompra()*, que finaliza a compra de todos os Livros contidos no carrinho.
- **Banco:** Permite retirar informações sobre a conta bancária do utilizador com o seguinte método:
  - *getContaBancaria(int nib)*

- **Conta Bancaria:** Proporciona a interação com a conta bancária do utilizador com as seguintes funções:
  - *getSaldo()*, retorna o saldo do utilizador dessa conta.
  - *registarPagamento(float valor)*, regista o pagamento na conta do utilizador.
  - *historicoPagamentos(int nr)*, retorna uma lista com todos os pagamentos efetuados pela conta.

Na figura seguinte é possível visualizar a estrutura do nosso protótipo de modo a ter uma melhor compreensão de como foi implementado:

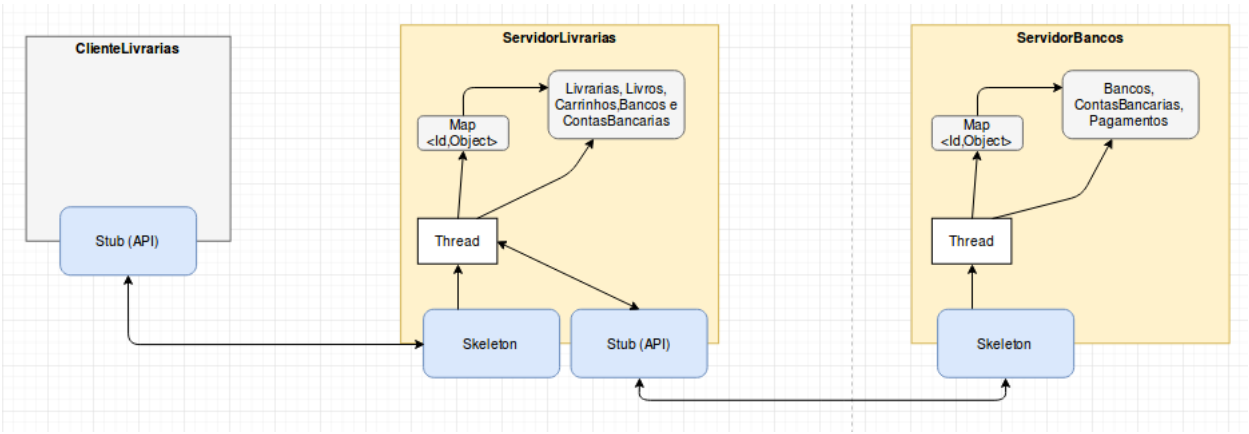

Figura 1.1: Estrutura do protótipo

Na figura em cima podemos visualizar os *Stubs* e os *Skeletons* usados para a comunicação entre Cliente-Servidor, bem como alguns componentes utilizados nos dois servidores, como é o caso do Banco dos *Maps* dos objetos criados por si e que estão a ser utilizados.

2. Exemplos de Utilização

Neste capitulo estão expostas algumas das formas de utilização do nosso protótipo.

2.1 Procura por livros

O excerto de código seguinte exemplifica a criação de uma Livraria remota e procura por um Livro na Livraria:

```
LivrariaC livraria = new LivrariaC();
LivroC livro = livraria.procurarLivro("Titulo 1");
LivroC livro2 = livraria.procurarLivro("Titulo 2");
```

2.2 Criação de Carrinhos e Adição ao Cart

No excerto é possivel visualizar o processo de criação e adição de livros, anteriormente procurados, num Carrinho para posteriormente poder ser efetuada uma compra.

```
CarrinhoC cart = livraria.novoCarrinho();
cart.adicionaLivro(livro);
cart.finalizarCompra(1);

CarrinhoC cart2 = livraria.novoCarrinho();
cart2.adicionaLivro(livro);
cart2.adicionaLivro(livro);
cart2.adicionaLivro(livro2);
cart2.finalizarCompra(1);
```

2.3 Pedido de histórico de compras

O código seguinte demonstra como é feito um pedido do histórico de compras de uma conta bancária utilizando a Livraria remota criada no ponto 2.1.

```
List<String> historico = livraria.getHistorico(1);
```

3. Conclusão

Para concluir este relatório, é importante mencionar que este projeto foi bastante importante para o grupo voltar a rever e implementar a matéria lecionada nesta unidade curricular.

Após análise do problema foi claro que existiam três principais componentes: um processo cliente que iria implementar a API do servidor de livrarias e dois servidores: o servidor de bancos e o servidor de livrarias.

A primeira matéria a colocar em prática foi a utilização de interfaces e as suas implementações que serviram de *Stubs* e *Skeletons* que juntamente com o Catalyst foram usados para a comunicação entre os demais processos.

Na criação do cliente, o grupo realizou quase todo o trabalho que queria ter realizado à exceção de uma coisa: o servidor de livrarias quando é arrancado cria uma livraria que guarda no seu *Map* com o id 1. Este objeto livraria é assumido que existe com esse id pelo cliente.

Caso o servidor de livrarias tenha apenas uma livraria esta solução é aceitável, contudo não é escalável para mais livrarias. Assim sendo, devíamos ter criado outro método que permitia instanciar livrarias pelo servidor.

Outro conceito foi implementado neste processo (e não só) que é a existência de objetos remotos (compostos por três atributos)

Em seguida foram implementados os dois servidores. O servidor de livrarias também funciona como cliente do servidor de bancos. Nestes servidores o controlo de concorrência foi implementado com *Locks* segundo o protocolo aprendido na aula, *Two-Phase Lock*. Este protocolo tem o problema de poderem ocorrer *DeadLocks*, contudo este problema é resolvido com a implementação do protoclo de *Two-Phase Commit* que iria assegurar a atomicidade nas operações. A implementação deste protocolo foi outra das funcionalidades que o grupo não realizou e que é fundamental, tanto para não permitir *Dead Locks* bem como permitir a atomicidade e falhas dos sistemas.

Para finalizar, o grupo atribui um balanço bastante positivo à realização deste trabalho. Voltamos a rever e a estudar todos estes conhecimentos aprendidos nas aulas, apesar de não termos conseguido implementar tudo (má gestão do nosso tempo) estes conceitos foram revistos e consolidados.

Bibliografia

[1] 2 phase commit. https://en.wikipedia.org/wiki/Two-phase_commit_protocol.

[2] Catalyst api reference. http://atomix.io/catalyst/api/latest/.