Universidade do Minho
Escola de Engenharia
Departamento de Informática

Mestrado integrado em Engenharia Informática
Fundamentos de Sistemas Distribuídos

Relatório do Trabalho Prático

5 janeiro 2018

Realizado por:

Carlos Silva – A75107
Catarina Cardoso – A75037
José Silva – A74601

Índice

Introdução

Neste trabalho prático foi proposta a implementação de um modelo de um sistema distribuído que fizesse a administração de uma livraria, encobrindo a existência de distribuição. Para isso, e tal como nas aulas práticas, foram utilizados objetos e transações distribuídas.

De modo a garantir o bom funcionamento do protótipo distribuído foi também necessário implementar o protocolo Two-Phase Commit e o método de controlo de concorrência Two-Phase Locking.

Decisões e Implementação

- **Remote Procedure Call**

A comunicação cliente-livraria e livraria-banco é feita através da metodologia RPC. Quando o cliente se inicia, conecta-se ao servidor (livraria). Através dos métodos implementados no servidor remoto, consegue enviar e receber pedidos da livraria, obtendo assim os dados que necessita do servidor. Este, para obter as respostas, utiliza procedimentos localmente.

- **Two-Phase Locking**

O método 2PL foi utilizado para evitar problemas relacionados com o controlo de concorrência. Quando se inicia uma transação, são efetuados lock requests dos objetos necessários para que se garanta a exclusividade de acesso durante a execução desta. Assim, quando é efetuado um pedido, este verifica se o lock requerido está disponível. Caso esteja disponível, adquire o lock e quando termina de computar nessa região critica notifica a classe gestora do 2PL, dando disponibilidade daquele recurso a outra transação. Se o lock não está disponivel no momento do request é colocado em espera numa queue.

- **Two-Phase Commit**

As operações que confirmam a aquisição de livros (atualização o stock na livraria e debito do preço da compra na conta do cliente) têm de ser atómicas, de modo a evitar erros como a repetição de débito ou a ausência desta ação. Para garantir que as operações realizadas fossem atómicas foi implementado o protocolo 2PC em que intervêm a livraria, o banco e um coordenador.

- **Garbage Collection**

De modo a eliminar os objetos que já não serão utilizados existe uma thread que partilha com o servidor o seu mapa de objetos e percorre esta coleção uma vez a cada segundo. Cada vez que há um acesso a esses objetos, é atualizada uma variável que avalia há quanto tempo estão inutilizados. Dependendo do tipo do objeto, quando é ultrapassado um certo tempo de permanência no mapa, o elemento é removido e, como é eliminada a sua referência, é libertada a memória ocupada por este.

Interface de Programação

- **Clientes**

DistributedObjects

Tipo	Método e Descrição
`DistributedObjects`	`DistributedObjects()` Construtor da classe
`Object`	`importObj(ObjRef o)` Retorna um objecto remoto ao qual se refere o.
`int`	`beginTransaction()` Inicia a conexão com o coordenador. Retorna o id da transação, -1 em caso de erro.
`boolean`	`commitTransaction(int txid)` Inicia o mecanismo de confirmação da transação (2PC) Retorna true caso o coordenador confirme a transação com os outros intervenientes, false caso contrário.

Exemplos

```
DistributedObjects distObj = new DistributedObjects();
```

```
ObjRef o = new ObjRef(address, 1, “store”);
Store store = (Store) distObj.importObj(o);
```

```
int txid = distObj.beginTransaction();
```

```
boolean result = distObj.commitTransaction(txid);
```

Store

Tipo	Método e Descrição
`Book`	`search(String title)` Retorna o objeto Book a que corresponder title, null caso não exista.
`Cart`	`newCart()` Retorna um novo objeto Cart.

Exemplos

```
Book book = store.search("one");
```

```
Cart cart = store.newCart();
```

Cart

Tipo	Método e Descrição
`void`	`add(Book b)` Adiciona o livro b ao carrinho.
`boolean`	`buy(int txid, String iban)` Retorna true se for possível concluir a compra, false caso contrário.

Exemplos

```
cart.add(book);
```

```
cart.buy(txid, "PT12345");
```

Book

Tipo	Método e Descrição
`int`	`getIsbn()` Retorna o identificador do livro.
`String`	`getTitle()` Retorna o título do livro.
`String`	`getAuthor()` Retorna o autor do livro.

Exemplos

```
int isbn = book.getIsbn();
```

```
String title = book.getTitle();
```

```
String author = book.getAuthor();
```

- **Servidores**

DistributedObjects

Tipo	Método e Descrição
`DistributedObjects`	`DistributedObjects()` Construtor da classe.
`void`	`ObjRef exportObj(Object o)` Incrementa o id e insere o objeto o no mapa. Retorna uma referência para o objecto o.
`void`	`initialize()` Inicia a conexão com os outros servidores. Regista e fica alerta dos requests que receberá do client.
`void`	`Initialize_bank()` Inicia a conexão com os outros servidores. Regista e fica alerta dos requests que receberá da loja.
`void`	`initializeCoordinator()` Inicia a conexão com os outros servidores.

StoreImpl

Tipo	Método e Descrição
`StoreImpl`	`StoreImpl()` Construtor da classe StoreImpl.

BankImpl

Tipo	Método e Descrição
`BankImpl`	`BankImpl()` Construtor da classe BankImpl.

Exemplo – Servidor Loja

```
DistributedObjects distObj = new DistributedObjects();
StoreImpl store = new StoreImpl();
distObj.exportObj(store);
distObj.initialize();
```

Exemplo – Servidor Banco

```
DistributedObjects distObj = new DistributedObjects();
BankImpl store = new BankImpl();
distObj.exportObj(store);
distObj.initialize_bank();
```

Exemplo – Coordenador

```
DistributedObjects distObj = new DistributedObjects();
distObj.initializeCoordinator();
```

Conclusões

O primeiro passo no desenvolvimento deste sistema distribuído foi completar o package bookstore iniciado nas aulas práticas, tanto a camada de negócio como a distribuição de servidores, mantendo esta última transparente. Seguiu-se a implementação do servidor bancário, que acompanhou a mesma lógica (RPC) utilizada na livraria.

Como próximo passo veio o desenvolvimento do 2PC e do 2PL, seguindo-se a implementação da sua interação com as entidades previamente criadas (livraria e loja). Para que houvesse libertação da memória ocupada por objetos que permaneciam inutilizados, criou-se um sistema de monotorização que apaga as suas referências.

No final, obteve-se um sistema que não tem problemas de concorrência no que toca ao acesso a áreas críticas, certifica-se que as operações de compra são atómicas e liberta da memória objetos não utilizados.

É importante também referir que caso o pedido de compra seja malsucedido, o cliente será informado, mas não saberá a razão. Como trabalho futuro poder-se-ia modificar as replies de forma a enviar as exceções (que implementam CatalystSerializable) relativas aos erros encontrados durante a realização da transação. Assim, o cliente receberia uma mensagem indicando qual a razão da impossibilidade de concluir a compra (algum livro sem stock, saldo bancário insuficiente, reinicio de algum servidor).