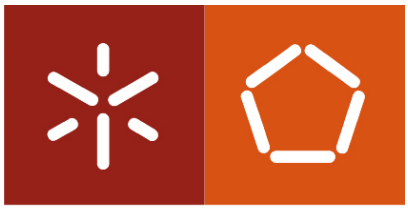

Escola de Engenharia
Universidade do Minho

Departamento de Informática

Mestrado Integrado em Engenharia Informática

Sistemas Distribuídos

Gestor de Deslocações

Implementação de uma aplicação distribuída para gestão de um serviço de taxis

Grupo 88

Bruno Pereira
Aluno nº 72628

Daniel Malhadas
Aluno nº 72293

Alexandre Silva
Aluno nº 72502

Braga, 3 de Janeiro de 2016

Resumo

Projeto para a disciplina de Sistemas Distribuídos 2015/2016. Consiste num serviço de Táxis informatizado onde se inscrevem passageiros e taxistas para, de diferentes formas, interagirem entre si.

Conteúdo

Introdução

No presente ano letivo (2015/2016) foi proposto aos alunos da unidade curricular de Sistemas Distribuídos que desenvolvessem uma aplicação, com recurso à linguagem de programação Java, que emulasse o funcionamento de um serviço de táxis recorrendo a sockets TCP/IP e a programação concorrente.

O nosso grupo (Grupo 88) tomou a iniciativa de tentar fazer uma aplicação estruturada e bem pensada de modo a que apenas olhando para o código se consiga facilmente entender o que se pretendeu com cada método e cada estratégia. Tivemos também em conta a dificuldade e a, por vezes pouco intuitiva, natureza da programação concorrente, tudo isso levou-nos a querer esquematizar bem o projeto antes de realmente o começarmos a programar, porque entendemos que não seria eficiente começar de imediato a "escrever qualquer coisa"apenas para no futuro apagar e fazer de novo. Para isso desenvolvemos este relatório simples onde definimos e explicamos a estratégia para realizar cada um dos objetivos a que nos propusemos.

1. Aplicação de gestão de serviços de *taxi*

1.1 Análise do Problema

Este projeto deverá ser capaz de emular um serviço de táxis de acordo com as seguintes funcionalidades:

registar utilizadores (fazer log in/sign in) que poderão ser passageiros ou taxistas. Um taxista será capaz de escolher usar o serviço como passageiro ou taxista enquanto que um passageiro somente terá direito a usufruir das funcionalidades permitidas a um passageiro. Para além disso os passageiros deverão ser capazes de solicitar um taxista ficando em espera até que haja pelo menos um taxista livre para realizar o seu pedido. No caso de existirem vários taxistas livres aquele que se encontrar mais perto será o taxi indicado para realizar o pedido em questão.

Para além destas funcionalidades obrigatórias implementamos também uma funcionalidade extra que consiste num chat indiscriminado entre utilizadores. Dizemos indiscriminado pois é permitido tanto aos passageiros como aos taxistas a sua utilização. A única restrição que se impõe é que enquanto um utilizador está a usar o chat não pode usar nenhuma das outras funcionalidades do sistema: pedir por um taxista/esperar por um passageiro.

Todas estas funcionalidades deverão ser capazes de funcionar concorrentemente, várias threads serão lançadas, uma por cada utilizador ativo e em situação alguma deverão ocorrer deadlocks, starvation, ou corrupção dos dados por acesso indevido a uma zona crítica.

1.2 Implementação

Para se implementar este serviço de acordo com a análise do problema mencionada anteriormente, decidimos criar um classe Client que iria comunicar diretamente com o utilizador. Com recurso a janelas "PopUp"o utilizador é informado do que está a acontecer e da mesma forma são-lhe pedidas informações pertinentes.

Aquando do início da aplicação o utilizador tem duas opções: Log in e Sign in.

Estas opções baseiam-se em trabalhar com uma classe de nome Dados onde são guardados todos os dados relevantes: utilizadores inscritos, taxistas livres, escritores de utilizadores no chat, etc... Para gerir as threads concorrentes nos métodos relativos a Sign in e a Log in, usa-se um simples bloco de synchronized onde se bloqueia o objecto que contém todos os utilizadores inscritos no sistema.

Depois de ter feito Log in com sucesso depara-se com mais três opções: entrar no chat, solicitar um taxista/solicitar um passageiro ou sair.

Qualquer utilizador poderá entrar no chat. este chat baseia-se numa janela com uma área para escrever mensagens (enviadas com o premir do botão Enter) e de uma área de apresentação das mensagens. Todos os utilizadores no chat serão capazes de ler todas as mensagens enviadas por qualquer utilizador, mas apenas desde o momento em que entraram no chat. As mensagens

são apresentadas de acordo com a seguinte terminologia: -> Taxista Anand diz:"se for um taxista de nome Anand ou -> Passageiro Karpov diz:"se for um passageiro de nome Karpov.

Para a opção de solicitar taxista/passageiro recorremos à seguinte estratégia. Um taxista ao solicitar um passageiro anuncia que está livre (faz um "put") e assinala essa mesma informação a um só passageiro que estava à espera que isso acontecesse. Quando existem táxis livres um passageiro pode então solicitar um taxi (faz um "get") e é-lhe atribuído um taxista. Quando isto acontece todos os taxistas livres em espera de passageiro são assinalados para irem verificar se são eles o táxi atribuído, se não forem voltam a estar em estado de espera até um novo cliente os assinalar de novo. Podemos então perceber que para gerir a concorrência se uma um ReentrantLock e duas variáveis de condição. Umas das variáveis irá servir para fazer esperar ou notificar os passageiros e a outra para fazer esperar ou notificar os taxistas. Desta maneira podemos assinalar um só passageiro para deixar de esperar quando temos taxistas livres pois sabemos de certeza que estamos a assinalar um passageiro e nunca um taxista e que os passageiros serão assinalados de uma maneira minimamente ordenada de modo a evitar starvation dos mesmos (a natureza do ReentrantLock garante-nos isto). Com os taxistas não podemos assinalar apenas um pois não temos como saber que é o taxista certo, por isso assinalamos todos e os errados irão voltar a esperar. Isto é mau apenas em termos de eficiência, pois não tem mal haver starvation dos taxistas visto que o objetivo é servir o passageiro e não o taxista. A posição dos taxistas será sempre guardada no seu próprio objeto (começando em (0,0) quando este se inscreve no sistema, assumiremos que esta posição é relativa à central de táxis) e é atualizada aquando de cada viagem, este conhecimento pelo sistema da posição simula a existência de um gps que indica a posição dos táxis. O passageiro não terá uma posição guardada no sistema, pois por razões óbvios nenhum passageiro ia querer que o sistema de táxis soubesse a sua posição a todos os momentos. Notas: Estipulamos que demora um segundo a passar de uma posição consecutiva para outra e que esse segundo de viagem custa 0.1 euros. Apenas se começa a cobrar a partir do momento que o taxista e o passageiro se encontram.

Caso o utilizador opte por sair o seu escritor, o seu leitor e a sua ligação ao servidor são fechados e a aplicação termina.

1.3 Testes

Para testar a nossa aplicação fizemos uma script de testes para assim podermos fazer uma verificação exaustiva, para além das nossas capacidades humanas, do sistema. O intuito principal era detetar problemas pouco intuitivos como deadlocks ou starvation ou até corrupção dos dados por controle de concorrência mal programado, com um grande número de threads em execução ao mesmo tempo.

Conclusão

Terminada a fase de desenvolvimento do projeto, depois de feita uma minuciosa análise de requisitos e avaliação do problema, chegamos a conclusões (apresentadas ao longo do corpo deste relatório) que nos deixaram satisfeitos. Pensamos ter sido capazes de apresentar explicações detalhadas no sentido em que toda a informação essencial para entender implementação da aplicação em questão e o seu funcionamento se encontra presente sem nenhuma exceção, mas ao mesmo tempo bastante simples na maneira como será possível para alguém com pouco conhecimento de o que é Java, a linguagem usada neste projeto, consiga ainda assim acompanhar e entender o que está a ser dito.

Tendo isto em conta temos também a noção que nunca nenhum projeto se pode dar por realmente terminado ou por realmente perfeito, há sempre a possibilidade de melhorar algo e esse fato não deve nunca ser ignorado, por mais experiência que venhamos a ter, existe sempre um limite para quão boas as nossas primeiras impressões poderão ser. Desta forma estariamos preparados, caso tivessemos sido contratados por uma empresa, para no futuro a pedido da mesma modificar algo que aqui poderá ter sido afirmado e/ou concluído. Isto é, o nosso projeto foi elaborado com a ideia de criar algo que permita uma fácil expansão tanto das funcionalidades já existentes como de possíveis outras totalmente novas. Isto é essencial em qualquer projeto, pois sabemos que no mundo real o empregador nem sempre sabe de raiz tudo aquilo que realmente quer implementar por isso torna-se natural que, num momento mais avançado da implementação, liste novas capacidades que quer ver presentes. Um projeto mal pensado para expansões e modificações poderia ter de ser refeito de raiz para não dar problemas ou graus baixos de eficiência, no entanto tivemos a ideia presente de tentar criar algo que podesse minimizar todas essas possíveis perdas que poderiam acontecer se este fosse um projeto no mundo real e não um trabalho académico protegido pela redoma que acaba sempre por ser uma universidade.

Esta facilidade de expansão verifica-se, por exemplo, na maneira como se interpretam as mensagens recebidas pela thread de cada utilizador. Na Classe Cliente vemos que adicionar novas mensagens para interpretar se pode fazer apenas com duas linhas sem ter que apagar nada.

Da mesma maneira podemos observar que o código está estruturado e organizado em packages de forma a reduzir ao máximo repetição de código. Métodos estão também divididos em vários de modo a serem o mais pequeno e concisos possível, desta forma será fácil para alguém que não quem programou perceba como a aplicação funciona.