Fundamentos de Sistemas Distribuídos
Livraria e Banco - Transações distribuídas e distribuição transparente
MIEI

André Diogo

A75505

5 de Janeiro de 2018

Conteúdo

Capítulo 1

Introdução

Este relatório tratará de oferecer uma visão arquitetural do projeto realizado, além de considerações e justificações sobre a implementação escolhida para os diversos componentes.

A descrição da API do programa estará embebida no código fonte no formato javadoc.

Capítulo 2

Arquitetura

2.1 Visão Geral da Arquitetura

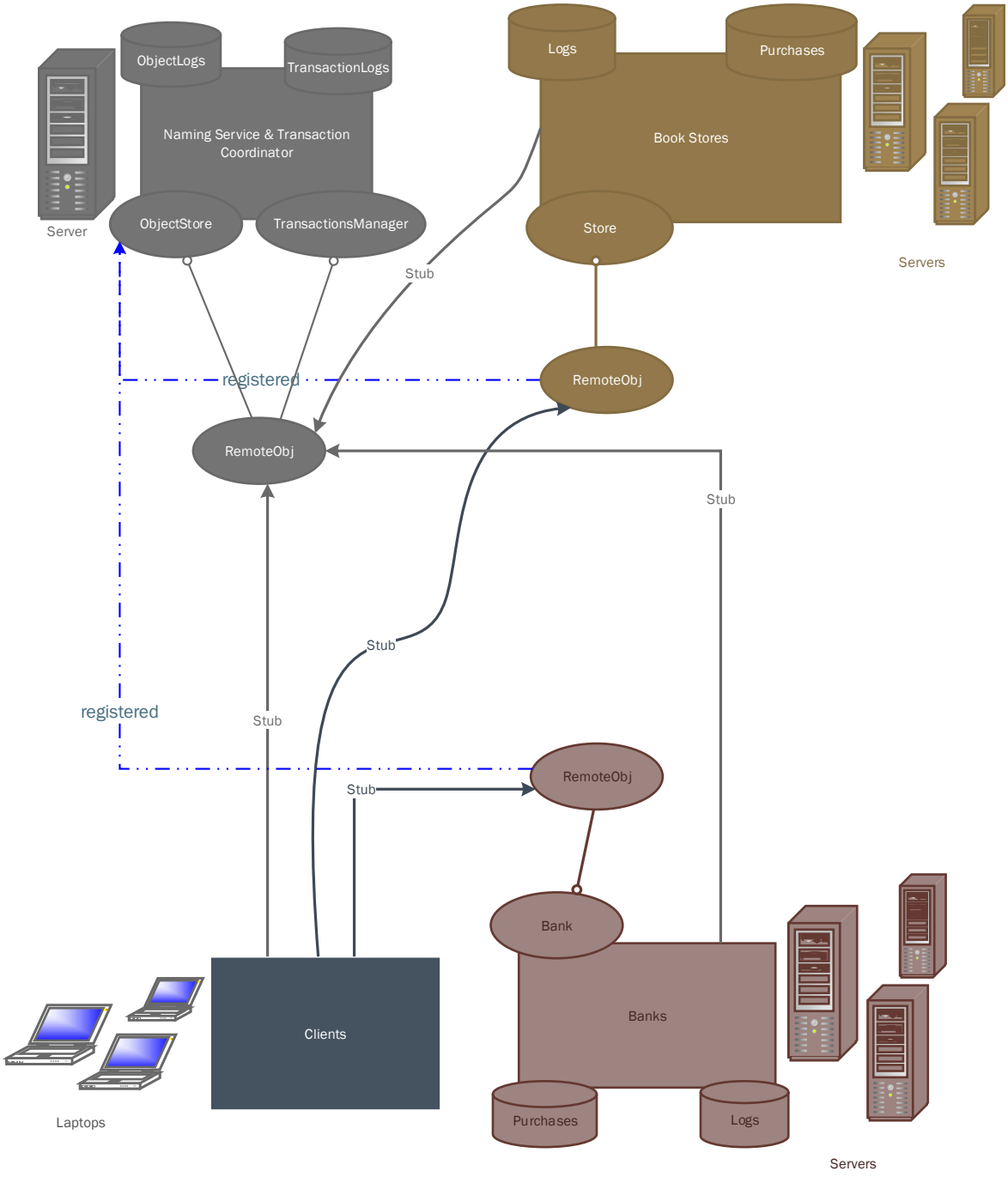

Figura 2.1: Arquitectura dos processos.

Como mostra a figura, o sistema envisionado está dependente de um processo que atua como serviço de nomes e gestor de transações conhecido por todos os outros processos.

Para garantir a consistência do sistema, caso este serviço falhe, ele mantém logs que refletem o seu estado atual, para poder ser reiniciado num estado são.

Temos assim ao nossa dispor a possiblidade de correr múltiplas lojas, bancos e clientes, sendo que estes se ligam apenas quando necessário recorrendo ao serviço de nomes para obterem a localização na rede do interveniente que requerem.

2.2 Serviço de Nomes e Gestor de Transações

Para permitir uma configuração de rede razoavelmente dinâmica torna-se necessário ter um serviço de nomes capaz de registar servidores (quer de bancos quer de lojas) ativos, prontos para servir os clientes.

Como é necessário também ter um coordenador de transações para implementar o *Two-phase commit*, o serviço de nomes tratará também de o fazer no mesmo processo, para simplificar.

2.3 Modularidade e classes auxiliares

Para permitir uma implementação bastante modular e genérica do código envolvido na manutenção de objetos distribuídos e conexão de rede criaram-se classes auxiliares dedicadas à gestão de referências distribuídas (DistObjManager), encapsuladora de funções comuns de *leasing* de objetos e *garbage collection*, assim como importação de *Stubs* e exportação de *Servants* (usou-se o termo Skeleton para estes na base de código); e de gestão de conexões (Server e Stub), encapsuladoras de funções comuns de gestão de ligações, registo de *handlers* e registo de *callbacks*.

Estas classes abstratas tentam deferir os detalhes mais específicos (classes concretas envolvidas) para implementações mais concretas ligadas à lógica de negócio dedicada às livrarias e bancos. Tentam ser o mais *Plug 'n Play* possível, seguindo uma abordagem de inversão de controlo.

2.4 Handlers e Pedidos

Na abordagem orientada ao evento proporcionada pelo framework *Catalyst*, existem adicionalmente dois recursos principais envolvidos na distribuição transparente.

Os handlers permitem atualizar o estado de acordo com pedidos que cheguem ao servidor em questão (quer seja loja, banco, serviço de nomes ou gestor de transações) multiplexando muitas ligações e pedidos em poucos recursos físicos (CPUs, threads). Em seguida podem ou não responder a estes pedidos com respostas, que outros serviços interpretarão como pedidos.

Fazem então estes dois componentes uma parte principal deste projeto.

Cada um dos quatro serviços acima referidos contém uma quantidade considerável destes dois recursos disponíveis para poder responder às interfaces genéricas expostas e a eles associadas.

Assim, um cliente que não possui um destes serviços embebido na sua máquina ou no seu processo, vê uma interface simples e transparente à distribuição, estando esta escondida nestes handlers.

2.5 Gargalos

Para simplificar este projeto e algumas das suas principais interfaces, optou-se por adoptar um comportamento síncrono no que toca à troca de mensagens para invocação remota, o que diminui bastante a performance potencial do sistema, pelo que a solução a este problema passaria por um sistema bem mais complexo de *caching* de futuras respostas com recurso a técnicas de tratamento de operações asíncronas, mas que complicaria consideravelmente estas interfaces.

Como os serviços oferecidos são bastante modulares, os gargalos são relativamente mitigados aumentando simplesmente a distribuição, assumindo que as mensagens são pequenas em tamanho, e a rede relativamente robusta.

Assim, as duas principais simplificações, utilização de contextos de execução de uma única thread e interfaces síncronas no que diz respeito a mensagens *Stub<->Skeleton*, figuram-se aceitáveis face aos ganhos em clareza e transparência.

2.6 Por fazer

O cliente destes serviços ficou inteiramente por fazer, assim como diversos *handlers* e alguns *stubs* como o da loja por falta de tempo.

Visto esta arquitetura ser um pouco complexa, alguns pedidos e respostas não são também tratados com todo o rigor e tratamento de exceções.

Alguns outros pedidos, relacionados com a compra também ficaram por fazer.

Uma implementação do carrinho, com recurso ao padrão da fábrica, ficou também por fazer.

Este projeto carece também de exemplos de utilização da API, se bem que esta se encontra devidamente documentada.