**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

Engenharia Informática e de Computadores

**Infra-Estruturas de Sistemas Distribuídos**

**4º Trabalho Prático**

**Gestor de Transacções**

**Desenvolvido por:**

Ricardo Neto (#26657)

**Análise do Problema**

O trabalho proposto tem o objectivo de permitir consolidar os conhecimentos relativos ao processamento de transacções distribuídas sendo para isso disponibilizado um cenário em que se pretendem fazer transferências inter-bancárias a partir de uma Applet.

Numa primeira fase cada transacção é feita em série não existindo mais do que um cliente a aceder ao mesmo recurso o que não levantou problemas no que respeita à integridade dos dados.

Após algumas alterações, nomeadamente o aumento do número de transacções a executar em simultâneo, identificaram-se situações em que o resultado final não foi o esperado devido a conflitos no acesso aos recursos pela ausência de mecanismos de sincronização.

No sentido de impedir interferências na execução de uma transacção propõe-se então a especificação de um gestor de acessos aos recuros partilhados (contas bancárias) implementado sob a forma de um serviço JINI e que cumpra com os requisitos do protocolo *two-phase locking*.

**LockManager - Implementação do Protocolo *two-phase locking***

O protocolo 2PC permite a serialização de transacções gerindo a concorrência no acesso a recursos partilhados, protegendo-os na sua forma mais simples com um lock. A obtenção ou não de permissão para aceder aos dados é dada por um LockManager responsável por gerir possíveis conflitos.

A execução de uma transacção, do ponto de vista de obtenção de locks, é dividida em duas fases: a de expansão e a de contração. Na fase de expansão são obtidos os locks para os recursos que se pretendem aceder. A partir do momento em que o um lock é libertado inicia-se a fase de contração e, até serem libertados todos os locks, não pode ser feita mais nenhuma obtenção por parte dessa transacção.

Por forma a gerir a atribuição de locks a objectos, optou-se por utilizar um dicionário em que a chave é o recurso partilhado e o seu correspondente o objecto *LockTableRow*, que implementa os mecanismos associados ao 2PC. Assim, ao tentar obter o lock para um recurso é feito um acesso em exclusão à tabela de objectos na tentativa de obter o seu *LockTableRow* ou de criar uma nova instância, passando a fazer-se o acesso em exclusão à linha e não ao objecto. Desta forma reduz-se a contenção na tabela permitindo outras a transacçoes acederem a diferentes recursos.

A obtenção de um lock leva a que sejam avaliadas condições de conflito por parte de *LockTableRow*. Ao solicitar acesso a um recurso, cada transacção terá que disponibilizar um objecto *TableEntry* que representa o seu pedido através dos campos identificador da transacção, tipo de pedido e se o mesmo já foi aprovado. A avaliação do pedido cumpre a especificação do 2PC na medida em que pedidos de leitura estão em conflito com acessos de escrita e pedidos de escrita estão em conflito com acessos de leitura ou escrita.

Além das regras em cima indicadas é também garantido que uma transacção não pede acesso a um recurso que está na sua posse.

No sentido de permitir a comunicação com transacções distribuidas o *LockManager* foi implementado como um serviço JINI e utiliza um JavaSpace onde são depositados os objectos *LockEntry* das transacções que obtiveram o lock do recurso, tendo o seu campo indicador de aprovação o valor lógico verdadeiro.

**Aplicação Cliente**

Independentemente da tecnologia utilizada para as implementações (JINI ou JEE) a estrutura base dos clientes manteve-se tendo sido adicionado comportamento que permite indicar o número de threads que irão executar transacções.

Por cada execução são obtidos todos os locks dos objectos envolvidos e, só após o commit é feito o unlock desses objectos. Após o pedido de lock a thread cliente bloqueia-se no JavaSpace até obter o seu objecto *LockEntry* com o valor lógico verdadeiro no campo indicador de aprovação.

**Transição JINI - JEE**

A implementação do *LockManager* não sofreu praticamente alterações devido a este continuar a ser um serviço JINI. Todavia, a utilização do *LockManager* por parte de *SessionBeans* revelou-se complicada, tendo mesmo sido entregue esta parte do trabalho sem que alguns EJB conseguissem ser instanciados no servidor Glassfish.