

Relatório de Tolerância a Faltas

Projecto Komparator

Grupo T24

Repositório do GitHub:

<https://github.com/tecnico-distsys/T24-Komparator>



Margarida Simões
Nº 84611, LETI



Miguel Gonçalves
Nº 84613, LETI



Pedro Maria
Nº84618, LETI

Operações auxiliares de actualização de estado:

Tal como acontecia anteriormente com o método `imAlive()`, o mediador primário invoca no mediador secundário através de um cliente por ele criado as funções auxiliares de actualização de estado. Estas operações são unidireccionais tal como o método `imAlive()`, ou seja não enviam resposta. É necessária a existência destas funções de actualização para as operações que alterem o estado do servidor, que neste projecto são: “Clear”, “AddToCart” e “BuyCart”. Quando o mediador primário realiza uma destas operações manda actualizar o mediador secundário e só depois disso é que envia a resposta para o cliente. Para se definirem estas novas operações, alterou-se o WSDL, mantendo a compatibilidade com as operações existentes. O mediador secundário ao ser actualizado vai guardar no seu domínio um histórico dos resultados, cada um associado ao Id do respetivo pedido.

Front-end e identificador de pedidos:

O front-end adopta a semântica no-máximo-uma-vez. Ou seja, se o stub cliente não recebe uma resposta num prazo limite significa que o timeout expirou e nesse caso é necessário repetir o pedido. Como o servidor não executa pedidos repetidos, se receber uma resposta o cliente tem a garantia que o pedido foi executado no máximo uma vez. Para o servidor poder detetar se um pedido é repetido, todos os pedidos têm um identificador único. Assim o mediador primário ao receber um pedido faz uma verificação do seu Id. Se for repetido não executa a operação e retorna o resultado obtido da primeira vez que executou esse mesmo pedido.

Tanto o mediador primário como o mediador secundário usam mapas para guardar os pedidos já executados associados a um determinado resultado. O mediador secundário guarda este histórico de respostas tanto nas funções de actualização como nas operações principais (que executa quando substitui o primário). Note-se que a verificação do Id dos pedidos e histórico de respostas é apenas necessário nas operações não idempotentes, ou seja que ao serem executadas mais do que uma vez devolvem como resultado algo diferente do que seria esperado se fossem executadas apenas uma vez. Neste caso temos como não idempotentes as operações “BuyCart”, “AddToCart” e “Clear”.

Modo de envio dos Ids entre cliente e mediador e entre os dois mediadores:

O `mediator-ws-cli` é responsável por criar os identificadores e associá-los a cada pedido garantindo a unicidade dos mesmos. Para transmitir os identificadores entre o cliente e o servidor primário criou-se um novo handler denominado “`MessageIDHandler`”. Se a mensagem é outbound (lado do cliente), durante a execução do handler lê-se o ID do contexto da mensagem que foi previamente colocado aí pelo cliente na chamada remota. De seguida cria-se um novo header e adiciona-se o Id. Quando a mensagem é inbound (lado do servidor) realiza-se o processo contrário. Ou seja, lê-se o Id do header e coloca-se o mesmo no contexto da mensagem para que possa ser acedido pelo servidor. Entre o mediador primário e secundário o identificador de pedido é passado como argumento nas mensagens de actualização.

Conclusão:

Combinando todos os procedimentos descritos neste relatório foi possível criar um sistema tolerante a faltas. Integrando as 4 partes do projeto, obtemos assim um serviço que assegura segurança das mensagens (através dos handlers) e tolerância a faltas do mediador.