

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 N° 84613, LETI



Pedro Maria N°84618, LETI

Sistemas Distribuídos, 2ªsemestre 2016/2017

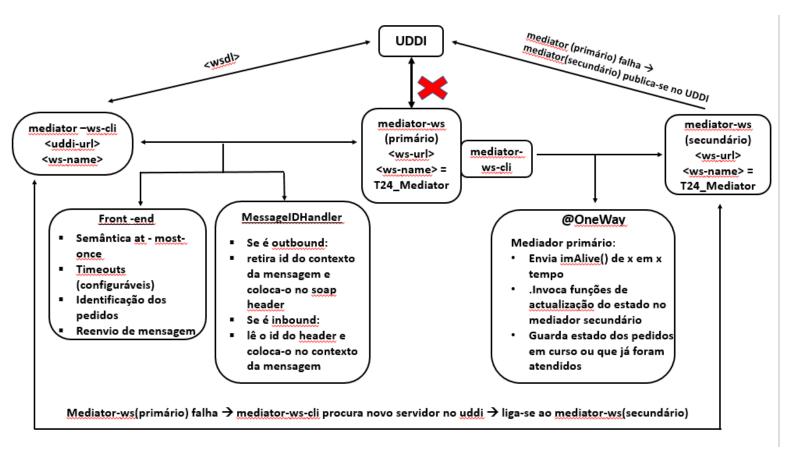


Figura 1: Esquema da solução de tolerância a faltas implementada

Objetivo:

O objetivo desta parte do projeto é garantir a tolerância a faltas, neste caso essa falta é a paragem do processo que implementa o mediador. Para assegurar a tolerância do mediador, criou-se uma réplica do mesmo, que funciona como servidor secundário alternativo. O mediador primário é responsável por garantir que o estado do mediador secundário é actualizado de forma síncrona sempre que este realiza uma operação que não seja idempotente. O mediador secundário é responsável por detetar a falha do mediador primário e substituí-lo.

Ligação cliente-servidor através do UDDI:

O UDDI permite o registo e descoberta de Web Services de forma dinâmica. O mediator-ws inicialmente publica-se no UDDI com o <ws-name> de "T24_Mediator" e o seu <ws-url>. Deste modo o cliente do mediador pode pesquisar pelo nome do web service no UDDI e obter o respectivo <ws-url> de modo a poder estabelecer uma ligação com o servidor. Além do mediador primário existe um mediador secundário que serve como um" backup". Assim quando o primário falhar este irá substituí-lo publicando-se nesse momento no UDDI.

Classe LifeProof e envio de provas de vida:

Para o mediador secundário poder detetar que o primeiro falhou criou-se uma classe "LifeProof" que faz o lançamento de provas de vida do mediador primário para o secundário. Esta tarefa executa-se de forma independente da tarefa principal e repete-se ao fim de um intervalo de tempo configurável. O mediador primário chama o método imAlive() no mediador secundário através de um cliente criado (mediator-ws-cli). Este recebe as provas de vida do mediador primário, e verifica se o tempo que decorreu desde o último envio está dentro do limite configurado. Se não estiver, assume que ocorreu uma falha no mediador primário e por isso publica-se de imediato no UDDI substituindo o servidor antigo. O cliente ao detetar que o servidor foi abaixo, procura de novo no UDDI pelo <ws-name> de modo a poder obter o novo <ws-url> e assim poder estabelecer uma nova ligação e continuar a efetuar pedidos.

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 atualização de estado. Estas operações são unidirecionais tal como o método imAlive(), ou seja não enviam resposta. É necessária a existência destas funções de atualização 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.