Sistemas Distribuídos

9 Objetos Distribuídos e Invocação Remota

- ***Protocolo Request-Reply***
- **Modelo de Objeto Remoto**
- **Semânticas de Invocação Remota**
- **Arquitetura de Invocação Remota**

Profª Ana Cristina B. Kochem Vendramin
DAINF / UTFPR

Introdução

- Características de programação encontradas na middleware:
  - Provisão de transparência de localização;
  - Independência dos detalhes dos protocolos de comunicação, sistema operacional e hardware;
  - Uso de diferentes linguagens de programação.

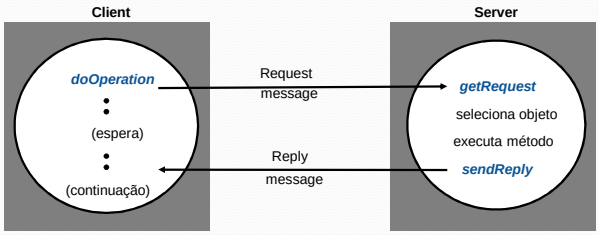
  Obs.: RMI é usado de forma genérica para designar invocação remota de métodos. Não confundir com Java RMI.

Middleware

Protocolo *Request-Reply*

- Normalmente, síncrono e razoavelmente confiável
  - Síncrono - processo cliente bloqueia até que uma resposta chegue do servidor.
  - Confiável:
    - Reply serve como um ack.
    - Novo Request do mesmo cliente = ack do reply anterior.

Primitivas do Protocolo *Request-Reply*



Primitivas do Protocolos *Request-Reply*

- **doOperation**
  - public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)
- **getRequest**
  - Usada pelo servidor para obter a requisição de um cliente.
  - public byte[] getRequest()
- **sendReply**
  - public void sendreply (byte[] reply, InetAddress clientHost, int clientPort)

Estrutura da Mensagem

Tipo_Mensagem	***int (0 = Request, 1= Reply)***
ID_Mensagem	***int***
ReferenciaObjetoRemoto	***Referencia***
ID_Metodo	***int ou Nome_Metodo***
Argumentos	***array de bytes***

[CDK01]

Modelo de Falhas

- doOperation configura temporizador para esperar resposta.
  - Opções após timeout:
    - Indicar ao cliente uma falha no servidor;
    - Retransmitir o pedido antes de caracterizar a falha.
- Descarta Requests duplicados.
- Mensagens Reply perdidas
  - Servidor deve analisar a operação para evitar a reexecução da mesma ao receber request duplicado.
  - A execução da operação é necessária para obter um resultado a menos que este esteja armazenado.

Modelo de Falhas

- **Operação idempotente**:
  - Pode ser executada repetidamente que produzirá sempre o mesmo resultado.
  - Não precisa tomar medidas especiais para evitar dupla execução.
- **Operação não idempotente**.

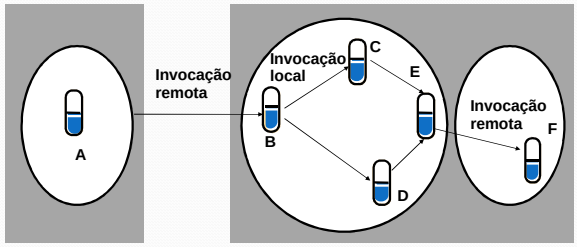
Modelo de Falhas

- Histórico:
  - Utilizado em servidores que precisam retransmitir resultados sem executar novamente as operações.
  - Estrutura que contém registros de mensagens reply transmitidas.
  - Cada entrada contém um requestId, uma mensagem e o ID do cliente para qual a mensagem foi enviada.
  - Problema:
    - Custo de memória.

Modelo de Falhas

- Problema de crescimento do histórico
  - Como clientes só podem processar um pedido de cada vez, servidor pode interpretar cada pedido como um ack do reply anterior.
  - Histórico guarda apenas o último reply enviado para cada cliente.
  - Muitos clientes
    - Quando um processo cliente termina, ele não envia um ack do último reply recebido.
    - Mensagens são descartadas após um limitado período de tempo.

Modelo de Objeto Distribuído



[CDK01]

Conceitos Fundamentais

- **Referência de objetos remotos**
  - Todo objeto pode receber invocações de métodos contato que os invocadores tenham acesso a sua referência.
  - Identificador que referencia um único objeto em todo o SD.

Endereço IP	Número Porta	Data e Hora de criação	Número objeto	Interface do objeto

Semânticas de Invocação Remota

- A semântica depende das garantias de entrega:
  - **Retransmissão da mensagem request**
    - Até que a resposta seja recebida ou que seja assumida falha no servidor.
  - **Filtragem de requests duplicados** no servidor.
  - **Retransmissão dos resultados**.
    - Deve-se ou não manter um histórico dos resultados para possível retransmissão.

Semânticas de Invocação Remota

Medidas de Tolerância a Falhas			
Retransmite request	***Filtra mensagens duplicadas***	***Executa método novamente ou retransmite o reply***	***Semântica de Invocação***
Não	**Não se aplica**	**Não se aplica**	***Maybe***
Sim	**Não**	**Executa método novamente**	***At-least-once***
Sim	**Sim**	**Retransmite *reply***	***At-most-once***

Semânticas de Invocação Remota

- **Maybe**
  - Nenhuma medida de tolerância a falhas é aplicada.
  - Sofre de falhas em canais de comunicação e processos.
  - Invocador não é capaz de saber se o método foi ou não executado.
    - O resultado pode ter sido perdido.
    - Crash pode ter ocorrido no processo que abriga o objeto logo após a execução do método.
  - *Timeout* no cliente - não há retransmissão.
  - Útil apenas quando falhas ocasionais são aceitáveis.

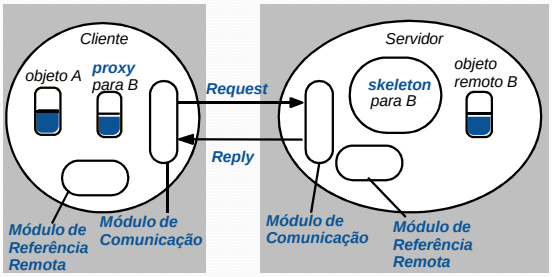
Semânticas de Invocação Remota

- **At Least Once**
  - Invocador recebe o resultado significando que o método foi executado pelo menos uma vez ou recebe uma exceção.
  - Retransmissões de request mascaram falhas de omissão.
  - Sofre de falhas arbitrárias.
    - Quando invocação é retransmitida, método pode ser executado mais de uma vez podendo armazenar ou retornar valores incorretos.
  - Aceitável se objetos em um servidor forem projetados de forma que todos os métodos em suas interfaces remotas sejam idempotentes.
  - Sun RPC fornece chamadas com semântica at-least-once.

Semânticas de Invocação Remota

- **At Most Once**
  - Usa todas as medidas possíveis para tolerância a falhas.
  - Invocador recebe o resultado esperado (método foi executado apenas uma vez) ou, então, recebe uma exceção.
  - Retransmissão de request.
  - Filtragem de *requests* duplicados e retransmissão da resposta garante que o método não idempotente seja executado uma única vez
  - Java RMI e CORBA empregam a semântica at-most-once.
  - CORBA permite que a semântica maybe possa ser aplicada para métodos que não retornam resultados.

Arquitetura de Invocação Remota



Arquitetura de Invocação Remota

- **Serviço de nomes (Binder)**
  - Mantém tabela que mapeia nomes em referências de objetos.
  - Utilizado pelos servidores para registrar seus objetos por nome.
  - Cliente faz leitura (*lookup*) na tabela do *binder* para obter a referência de objeto remoto antes de fazer a invocação.
  - Exemplos: RMIregistry, CORBA Naming Service.

Garbage Collector Distribuído

- O GCD assegura que se não existe mais uma referência local ou remota de um objeto, este será destruído, liberando espaço em memória.
- Em algumas linguagens, como Java, o GCD pode ser realizado automaticamente pelo próprio compilador.
- Em outras linguagens, como o C++, o GCD deve ser implementado pelo programador, explicitamente.
- O GCD trabalha em cooperação com o GC local.

Algoritmo GCD

- Usa semântica de invocação at-most-once.
- Tolera falhas de comunicação em processos cliente
  - Servidores arrendam (***lease***) seus objetos aos clientes por um limitado período de tempo.
  - Para cada *lease*, o servidor mantém um identificador do cliente e o tempo do *lease*.
  - O período do *lease* começa quando o cliente adiciona uma referência
  - Termina quando o tempo do *lease* expirar ou quando o cliente remove a referência.

Referências Bibliográficas

- Coulouris, George; Dollimore, Jean; Kindberg, Tim. Distributed Systems: concepts and design. Third Edition. Addison-Wesley 2001.
- Coulouris, George; Dollimore, Jean; Kindberg, Tim; tradução João Tortello. Sistemas Distribuídos: conceitos e projeto. 4. ed. Bookman 2007.