Teste-Tipo de Sistemas Distribuídos – RPC, RMI, Web Services
Guia de resolução

Grupo RPC

Considere o seguinte código que ilustra uma componente programática de um sistema de RPC, neste caso do SUN-RPC.

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
   int length;
   char buffer[MAX];
};
struct writeargs {
   FileIdentifier f;
   FilePointer position;
   Data data;
};
struct readargs {
   FileIdentifier f;
   FilePointer position;
   Length length;
};
```

```
program FILEREADWRITE {
   version VERSION {
      void WRITE(writeargs)=1;
      Data READ(readargs)=2;
   }=2;
} = 9999;
```

1. Considere a mensagem de invocação da função `WRITE` do protocolo.
   a. Quais os campos da mensagem necessários para que o servidor possa executar o procedimento remoto? Complete a sua resposta o mais possível, retirando elementos do programa acima.

Campo	Correspondência com os elementos do código acima (caso exista)
Identificador de função	"=1"
Identificador da interface	Composto pelo par (FILEREADWRITE=9999; VERSION=2)
Identificador do pedido (CallID)	(não definido acima)
Argumentos serializados	FileIdentifier f; FilePointer position; Data data (length e buffer).

Nota: Estes são os campos mais importantes que se esperam numa resposta. Se analisado em maior detalhe, existem outros campos menos relevantes.

Ver secção 5.2 do livro e slide "Sun RPC:Serviço de Nomes e encaminhamento de RPCs".

   b. Na representação anterior efetuou uma descrição dos campos da mensagem, mas a mensagem realmente enviada através do protocolo de transporte depende de decisões de arquitetura do protocolo de RPC nomeadamente da sua codificação ser *receptor converte* ou não, e de ser *explícita* ou não.
   O que sabe sobre estas decisões no protocolo SUN-RPC? Justifique

Referir que o SUN RPC usa codificação canónica e estrutura implícita.
Ver slide "Sun XDR (External Data Representation)".

   c. Um cliente que pretenda usar o serviço tem de conhecer a sua identificação e descobri-lo antes de o invocar.
      i. Qual é a identificação do serviço remoto no exemplo acima?

É dada pelo par *(program; version).*
Ver secção 5.3.3 do livro.

ii. Justifique neste contexto o interesse do campo VERSION.

Explicar que permite que o serviço tenha novas interfaces (usando novas *versions*) ao mesmo tempo que mantém compatibilidade com os clientes que usam as interfaces anteriores.
Ver secção 5.3.3 do livro.

iii. Explique a forma como os clientes obtêm o porto do servidor, indicando que componentes da arquitetura são utilizadas.

Referir que é contactando o serviço de nomes (*rpcbind*) num porto bem conhecido, indicando-lhe: PROGRAM, VERSION, protocolo.
Indicar as componentes da arquitetura usadas: serviço de nomes (também chamado *rpcbind* ou *port mapper* no SUN RPC); e biblioteca de *run-time*.
Ver secção 5.3.3 do livro e slide "Arquitetura: O Sistema de Suporte – Run-time system".

iv. Suponha que o cliente quer invocar procedimentos remotos em dois servidores diferentes, a correr em máquinas diferentes mas oferecendo a mesma interface remota. Que conceito existe do lado do cliente para tal ser possível? Explique.

Referir o conceito de *binding explícito*.
Explicar como o *binding* explícito permite ter 2 ou mais *bindings handles* que são usados para especificar, a cada chamada, o servidor pretendido.
Ver slides "Exemplo Binding : Cliente – Sun RPC " e "Outras opções de binding".

d. Considere a função WRITE, que abre o ficheiro *f* e a escrever os bytes passados como argumento no deslocamento indicado por *position*.

i. É idempotente ou não? Justifique.

Sim. Explicar por que é que chamar a função uma ou múltiplas vezes sobre o mesmo estado inicial e com mesmos argumentos produz o mesmo estado final e resultado.
Ver secção 5.2 do livro.

ii. O ideal seria a semântica de execução da função ser exatamente-uma-vez. Exemplifique com base na execução de operação WRITE uma situação de falha que ilustre a dificuldade de oferecer esta semântica no RPC.

Dar exemplo de situação em que o retorno ao cliente é incoerente do que realmente aconteceu no servidor.
Possível exemplo (entre outros): o servidor inicia a execução da função *write* mas falha algures a meio; o cliente recebe "Erro de RPC" mas a função foi executada parcialmente no servidor.
Ver secção 5.3.1 do livro e slide "Semântica exatamente-uma-vez".

2. Considere o stub cliente do SUN RPC correspondente a uma função de soma.

```
calc_result *
sum_2(calc_args *argp, CLIENT *clnt)
{
        static calc_result clnt_res;
        if (clnt_call(clnt, SUM,
                    xdr_calc_args, argp,
                    xdr_calc_result, &clnt_res,
                    TIMEOUT) != RPC_SUCCESS) {
         return (NULL);
        }
        return (&clnt_res);
}
```

1. O cliente que utiliza esta função tem de obter previamente a localização do servidor.

a) Descreva a forma como a localização do servidor é obtida.

Descrever os passos que ocorrem durante o estabelecimento da sessão, em que o cliente contacta o serviço de nomes para descobrir o porto do servidor.
Ver secção 5.3.3 do livro.

b) Depois de conhecer a localização, tem de ser estabelecido um canal de comunicação e este tem de ser utilizado pelo procedimento *sum_2.* Explique detalhadamente como é que esta associação do canal à função é efetuada e identifique no stub se existe alguma estrutura de dados que o represente.

Referir que a associação do canal à função é feita através do *binding handle* passado como segundo argumento da função sum_2.
Ver slide "Exemplo Binding : Cliente – Sun RPC".

c) A função tem a designação sum_2. Se existir uma função _3, o que potencialmente pode mudar nesta função? A sua resposta deve ser concreta utilizando os elementos do programa.

Referir que a assinatura da função remota *sum* pode mudar.
Ver secção 5.3.3 do livro.

2) A função clnt-call chama duas funções xdr_calc_args e xdr_calc_result, cuja função é efetuar o *marshalling* dos parâmetros. Explique concretamente para a função xdr_calc_result, qual é o input e o respetivo formato.

Explicar que o input é a secção da mensagem de resposta recebida que contém o retorno serializado; e que o formato é binário, implícito e traduzido num formato canónico.
Ver slides "Sun XDR (External Data Representation)" e "Funções de conversão via XDR".

Grupo RMI

Considere as seguintes interfaces Java:

```
public class TTT extends Remote {
public String currentBoard() throws RemoteException;
public boolean play(int row, int column, int player) throws RemoteException;
public int checkWinner() throws RemoteException;
}

public class TTTGameManager extends Remote {
public void addGame(TTT game, String name) throws RemoteException;
public TTT getGame(String name) throws RemoteException;
public void removeGame(String name) throws RemoteException;
}
```

Assuma que:

- A interface TTT permite interagir com uma instância de um jogo do Galo, para consultar o estado do tabuleiro (currentBoard), efectuar uma jogada (play) e verificar se já existe um vencedor do jogo (checkWinner).
- A interface TTTGameManager permite gerir um conjunto de instâncias de TTT, cada uma indexada por um nome único; mais precisamente, a interface permite registar uma nova instância de TTT com um dado nome (addGame), obter uma instância pelo seu nome (getGame) e remover uma instância (removeGame).
- As classes TTTServant e TTTGameManagerServant implementam a TTT e TTTGameManager, respectivamente.
- Um servidor S1 instanciou um objecto remoto da classe TTTGameManagerServant e registou esse objecto no *RMI registry* com o nome "//sd.ist.utl.pt/gestorTTT".

1. Apresente o método main de um outro servidor S2 com a seguinte especificação:
   i. Criar uma instância da classe TTTServant (classe que implementa a interface TTT, com construtor vazio);
   ii. Adicionar esse objecto remoto ao gestor registado em "//sd.ist.utl.pt/gestorTTT" (chamando o método addGame), associando o nome "meuJogo" ao novo jogo;
   iii. Finalmente, entrar em ciclo infinito.

   Nota: por simplicidade, omita a preparação do *SecurityManager* da sua resposta.

```
public static void main(String args[]){

try{

  //Passo i. da especificação acima
  TTTServant s = new TTTServant();

  //Passo ii. da especificação acima
  TTTGameManager m = Naming.lookup("//sd.ist.utl.pt/gestorTTT");

  //Passo iii. da especificação acima
  m.addGame(s, "meuJogo");

} catch (RemoteException e) {…};

while (true) ; //ciclo infinito
}
```

Ver secção 5.5.1 do livro e aula de laboratório de RMI.

2. Na resposta anterior, onde é executado o método *addGame*: S1 ou S2? Justifique.
   (Responda mesmo que não tenha respondido à alínea anterior.)

Reponder P1. Explicar que o método é invocado sobre um objecto remoto alojado em P1.
Ver secção 5.5 do livro e aula de laboratório de RMI.

3. Programe um cliente que, através do mesmo gestor de jogos do galo ("//sd.ist.utl.pt/gestorTTT"), obtém uma referência para o jogo "meuJogo" e imprime o respetivo tabuleiro no ecrã (obtido pelo método currentBoard).
   Nota: por simplicidade, omita a preparação do *SecurityManager* da sua resposta.

```
public static void main(String args[]){

try{

  //Obtém referência para o gestorTTT
  TTTGameManager m = Naming.lookup("//sd.ist.utl.pt/gestorTTT");

  //Obtém uma referência para o jogo "meuJogo"
  TTT g = s.getGame("meuJogo");

  //Imprime o respectivo tabuleiro no ecrã
  System.out.println(g.currentBoard());

} catch (RemoteException e) {...};

}
```

Ver secção 5.5.1 do livro e aula de laboratório de RMI.

4. No momento em que o programa anterior chama o método currentBoard:
   (Responda mesmo que não tenha respondido à alínea anterior.)
   a. Quantas referências remotas tem o processo chamador? Justifique.

Referir 2 referências remotas: uma para o objeto gestor "gestorTTT", outra para o jogo "meuJogo".
Ver secções 5.4.1 e 5.5 do livro.

   b. [0,8v] Quantas classes proxy existem carregadas no cliente, e quais?

Referir as classes Proxy da interface TTT e proxy da interface TTTGameManager.
Ver secção 5.4.2 do livro.

   c. [0,8v] Quem gerou essas classes?

Explicar que foi o compilador de interfaces do RMI, de forma automática.
Ver secção 5.4.2 do livro.

5. O Java RMI usa o método de contagem de referências para assegurar a recolha automática de memória (*garbage collection*) dos objetos remotos.
   (a) Em que situação é invocada a operação *addRef*? Ilustre com o exemplo das alíneas anteriores, esclarecendo qual ou quais as máquinas que invocam addRef.

Explicar que acontece sempre que surge uma nova referência remota num programa, é invocado addRef sobre a máquina onde reside o objecto remoto referenciado.
Ilustrar com a(s) referência(s) remota(s) que são criadas no programa das alíneas anteriores.
Ver secção 5.4.3 do livro.

   ii) Considere que a implementação da operação *addRef* que não é idempotente. Neste caso, a invocação desta operação deve ser feita usando um protocolo de pedido-resposta que garanta a semântica no-máximo-1-vez. Explique uma situação incorreta que poderia ocorrer se a semântica fosse pelo-menos-1-vez.

Referir o risco de uma invocação de addRef resultar em múltiplas execuções sobre a máquina alvo, consequência da semântica pelomenos-1-vez.
Explicar a que, com consequência, do garbage collector alvo registará mais referências remotas que aquelas que existem.

Grupo Web Services

1. Adaptação à heterogeneidade dos dados dos sistemas: como é resolvida em Web Services?

Referir o uso de XML. Explicar como o XML permite troca de informação entre interlocutores heterogéneos.
Ver secção 4.3.3 do livro.

2. Adaptação ao protocolo de transporte
   a) Porque é mais genérico que o Sun-RPC?

Referir a possibilidade de usar SOAP sobre qualquer protocolo de transporte; ao contrário do Sun RPC, limitado a TCP/IP ou UDP/IP.
Ver secção 9.2.1 do livro.

   b) Onde é que se especifica a capacidade de utilização de um protocolo de transporte no WSDL?

Secção binding do WSDL
Ver secção 9.3 do livro.

3. Os RPC iniciais baseiam-se em protocolo de pedido-resposta. No caso dos web services não é este o único modo em que podem ser utilizados. Explique porquê.

Justificar enumerando os diferentes modelos de interação previstos no SOAP.
Ver slide "Interações previstas no SOAP".

1. Numa abordagem **implementation-first** para criação de um web service:
   a. [0,8v] Escreva a interface Java que corresponderia ao mesmo serviço do programa do grupo RPC, oferecido no endereço "http://sd.tecnico.ulisboa.pt/testeWS".

Apresentar interface Java com os métodos write() e read(), cada qual recebendo e retornando parâmetros equivalentes aos definidos no grupo do RPC.

Antes do cabeçalho da interface, deve surgir a anotação:
@WebService
Consultar aula de laboratório "Web Services I – implementation first".

   b. [0,6v] Admitindo que os clientes remotos são também desenvolvidos em Java, é dispensável a existência do documento WSDL? Justifique.

Não. Explicar que o WSDL é necessário para a geração dos *stubs* (também chamados *proxies*).
Ver secção 9.3 do livro.

2. Considere que o protocolo de transporte é HTTP.
   a. [0,7v] Quais os campos que constituiriam o corpo (body) da mensagem SOAP de invocação da operação WRITE?

Campo	Correspondência com os elementos da interface
Várias respostas possíveis, dependendo das opções de binding usadas. Consultar o slide "Exemplo de opções no binding" para exemplos.	

   b. A codificação da mensagem é neste caso XML. Quando comparado com SUN-RPC, indique:
      i. Uma vantagem.

Várias respostas possíveis - *ver secção 4.3.3 do livro.*

      ii. Uma desvantagem.

Várias respostas possíveis - *ver secção 4.3.3 do livro.*

   c. *Read* e *write* são palavras em inglês com numerosos significados. Explique como é que o recetor da mensagem pode ter a certeza do contexto em que estas designações são utilizadas quando surgem numa mensagem SOAP. Seja objetivo na sua resposta relacionando com a IDL que especifica o serviço.

Referir o uso de um namespace, definido no WSDL e usado no documento XML onde as palavras surgem.
Ver secção 4.3.3 do livro.

d. Suponha que, em vez de utilizar para protocolo de transporte o HTTP, pretende usar mensagens SMTP. Em que sítio concreto do WSDL estaria isso especificado?

Referir a secção *binding*.
Ver secção 9.3 do livro.

3. Um cliente que pretende usar o serviço tem de conhecer a identificação do serviço e descobri-lo antes de o invocar.
   a. Considere que não existe UDDI. É possível? Justifique como.

Explicar que o elemento Port no WSDL especifica o endereço onde o serviço está disponível.
Ver secção 9.3 do livro.

b. [0,6] Se existir UDDI qual é a principal vantagem para a gestão do sistema?

Referir a possibilidade de mudar de endereço do servidor dinamicamente, permitindo que os clientes descubram o novo endereço.
Ver secção 9.3 do livro.

(Este guia foi elaborado pelo Professor João Pedro Barreto e foi baseado em questões de testes e exames de 2013/14.
O guia foi revisto em 2016 pelo Professor Miguel Pardal)