LEIC/LETI 2016/17, Repescagem do 1º Teste de Sistemas Distribuídos

4 de julho de 2017

Responda no enunciado, usando apenas o espaço fornecido. Identifique todas as folhas.
Uma resposta errada numa escolha múltipla com N opções desconta 1/(N-1) do valor da pergunta.

Duração da prova: 1h30m

Grupo I [7]

Considere o seguinte excerto (incompleto) de um programa cliente em SUN RPC:

```
CLIENT *c;
readargs a;
Data *data;
c = clnt_create(...);

a.f = 10;
a.position = 100;
a.length = 1000;
data = read_2(&a, c);
```

1. Indique que elementos do código acima correspondem a:
   a. [0,6] *Binding handle.*
   b. [0,6] Função *stub.*
   c. [0,6] Argumento que será passado por valor para a função remota.
   d. [0,6] Número de versão (*version*) da interface remota.

2. [1,2] O programa acima não está a tratar a possibilidade de faltas relacionadas com a distribuição (como falhas do canal ou falhas do servidor). Proponha alterações ao programa acima que assegurem que, sempre que uma falta relacionada com a distribuição ocorra, seja impressa a mensagem "`Erro RPC`" no ecrã e o programa termine com `exit(1)`.

3. [0,9] Que argumentos recebe a função `clnt_create`? A sua resposta deve ser clara quanto ao significado de cada argumento.

4. O SUN RPC suporta as semânticas "pelo-menos-1-vez" e "no-máximo-1-vez".

   a. [0,9] De que forma pode o programa cliente especificar a opção que pretende para o seu programa? Na sua resposta, relacione com o código acima.

Especifica no argumento passado à função clnt_create que define qual o protocolo (UDP ou TCP) a usar. Escolher UDP ou TCP leva o SUN RPC a oferecer, respetivamente, as semânticas pelo-menos-1-vez e no-máximo-1-vez.

   b. [0,8] No caso da semântica "pelo-menos-1-vez", caso a invocação de uma função remota devolva um erro de RPC, que garantias pode ter o programa cliente quanto ao número de vezes que a função se executou no servidor? Justifique.

Na semântica pelo-menos-1-vez, um erro de RPC significa que a *runtime library* enviou o pedido múltiplas vezes mas nunca recebeu resposta. Sendo assim, não há nenhuma garantia quanto ao número de vezes que a função remota executou – tanto pode não se ter executado, como pode ter executado 1 ou mais vezes.

   c. [0,8] No caso da semântica "no-máximo-1-vez", caso a invocação de uma função remota devolva um erro de RPC, que garantias pode ter o programa cliente quanto ao número de vezes que a função se executou no servidor? Justifique.

Na semântica no-máximo-1-vez, um erro de RPC significa que a *runtime library* enviou o pedido múltiplas vezes e nunca recebeu resposta; sendo que os pedidos que cheguem duplicados ao servidor não causam execução duplicada. Sendo assim, a função pode ter sido executada no máximo 1 vez.

Grupo II [6]

Considere o sistema de transportes de uma dada cidade. Neste sistema, os utentes adquirem um cartão, que contém um identificador único. Cada cartão pode depois ser carregado com viagens e usado para aceder a cada viagem nos transportes públicos da cidade.

Cada cartão contém apenas uma etiqueta RFID que indica o identificador único desse cartão. O estado associado a esse cartão é mantido num servidor que gere todos os cartões da cidade, suportado por Java RMI. As interfaces remotas oferecidas por este servidor são as seguintes:

```
public interface GestorCartoes extends Remote {
      Cartao novoCartao() throws RemoteException;
      Cartao obtemCartao(int id) throws RemoteException;
      void eliminarCartao(int id) throws RemoteException;
}

public interface Cartao extends Remote {
      int numViagensDisponiveis() throws RemoteException;
      boolean validaViagem() throws RemoteException;
      void carregaViagens(int quantidade) throws RemoteException;
}
```

1) Ao passar um cartão físico numa entrada de um transporte público, um leitor obtém o respetivo identificador e executa os seguintes passos para decidir se permite ou não acesso ao utente:
   - Usando esse identificador, o programa a correr no leitor deverá ligar-se a uma instância remota de um objeto do tipo `GestorCartoes` que está registada com o nome:
   `rmi://transportes.pt/gestor`
   - O programa pesquisa então pelo cartão cujo identificador é o que foi lido do cartão físico.
   - Finalmente, caso esse cartão exista, o método `validaViagem` respetivo é invocado, que subtrai uma viagem ao saldo associado ao cartão. Caso a validação tenha sucesso, deve ser dado acesso ao utente.

   a) [1,2] Programe o procedimento descrito atrás. Na sua resposta, omita quaisquer definições relativas a políticas de segurança.

```
// Deve retornar true caso o cartão com o identificador passado como argumento
// permita entrar no transporte público. Retorna false caso contrário.
boolean pedeAcesso(int idCartao) {









}
```

   b) [0,8] No caso de retornar com sucesso, quantas instâncias de *proxies* existem no programa cliente (antes do retorno acontecer)?

2) Considere a hipótese em que a interface `Cartao` não era remota; em alternativa, a classe que implementava essa classe oferecia a interface `Serializable`.

a) [1,0] Quanto às classes carregadas do lado do cliente, o que mudaria?

Já não seria carregada a classe proxy da interface Cartao. Em substituição, seria carregada a classe que implementa o objeto remoto que está a ser passado por valor (e, caso necessário, as respetivas super-classes).

b) [1,0] O que mudaria no comportamento do programa pedido na alínea 1?
Concorda com esta alteração?

Passando a ser uma cópia local, a execução do método validaViagem irá alterar o saldo do objeto local e não do objeto remoto. Ou seja, deixa de haver um controlo distribuído do saldo disponível em cada cartão. Assim sendo, passaria a ser possível o mesmo cartão passar em diferentes leitores (clientes) sem ter saldo suficiente para tal, logo a alteração não é adequada.

3) Considere a seguinte situação:
- O cliente C1 obteve uma referência para o cartão 1234 (que tem saldo positivo), e está prestes a invocar o respetivo método `validaViagem`.
- Antes de C1 chamar `validaViagem`, o cliente C2 chamou o método `eliminarCartao`, que eliminou a referência que o objeto do tipo `GestorCartoes` mantinha para a instância de cartão 1234.

a) [1,0] Quando C1 chamar `validaViagem`, o método será executado com sucesso? Justifique.

Sim. O método eliminarCartao apenas elimina uma referência, não o objeto. Como o objeto manterá pelo menos uma referência para si, manter-se-á alocado em memória, logo acessível através dessa referência.

b) [1,0] Na situação descrita acima, indique os momentos em que o servidor recebeu mensagens `addRef` ou `removeRef`. Justifique.

Grupo III [7]

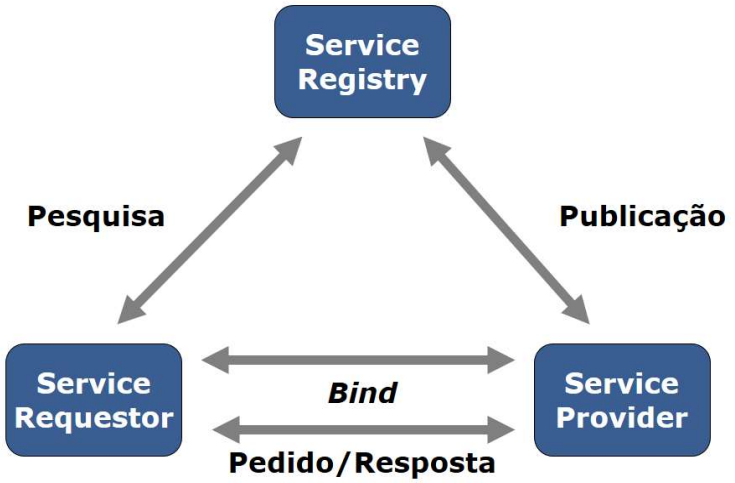
1) [0,5] Qual foi a principal motivação para o desenvolvimento da tecnologia de Web Services?
   A. Obter melhor desempenho na comunicação.
   B. Fazer uma melhor gestão de objetos remotos.
   C. Conseguir interoperabilidade na comunicação entre plataformas.
   D. Otimizar o processo de serialização de dados.

   C

2) [0,5] O protocolo HTTP pode ser considerado uma chamada remota de procedimento?
   A. Sim, porque permite o envio e receção de informação.
   B. Não, porque tem um conjunto estático de procedimentos.
   C. Sim, porque permite o retorno de informação de erro.
   D. Não, porque utiliza o protocolo TCP/IP.

   B

3) O diagrama seguinte representa, de forma resumida, a arquitetura dos Web Services.



   a) [0,5] Qual a ordem cronológica habitual para as etapas apresentadas nas setas?
   Responda colocando um número de ordem - 1, 2, 3, 4 - junto ao texto de cada seta na figura acima.

   b) [1,0] Qual é a norma (*standard*) de Web Services mais importante em cada etapa?
   Descreva sucintamente o papel que a norma desempenha na etapa respetiva.

1
2
3
4

4) Considere agora o método principal do programa do Web Service TTT (Tic-Tac-Toe - Jogo do Galo) implementado com JAX-WS (Java API for Web Services):

```
 1.   public static void main(String[] args) throws Exception {
 2.       String uddiURL = "http://uddi.sd.rnl.tecnico.ulisboa.pt:9090";
 3.       String name = "TTT";
 4.       String url = "http://myhost.pt/endpoint";
 5.
 6.       Endpoint endpoint = Endpoint.create(new TTTImpl());
 7.       endpoint.publish(url);
 8.
 9.       uddiNaming = new UDDINaming(uddiURL);
10.       uddiNaming.rebind(name, url);
11.
12.       // await requests
13.       // ...
14.    }
```

a) [0,5] O que faz a instrução da linha 7 ?

Disponibiliza o *endpoint* (extremidade) do serviço para que este possa ser invocado no URL indicado. Para tal o *JAX-WS run-time* inicia um pequeno servidor HTTP, que abre um *socket* para esperar pedidos.

b) [0,5] E o que faz a instrução da linha 10 ?

c) [1,2] Escreva o código Java de um cliente que consiga localizar dinamicamente e depois invocar a operação `boolean play(int row, int column, int player)` do serviço TTT.

```
public class TTTClientApp {
      public static void main(String[] args) throws Exception {

            String uddiURL = "http://uddi.sd.rnl.tecnico.ulisboa.pt:9090";
            String name = "TTT";

            // obter URL do serviço
            UDDINaming uddiNaming = new UDDINaming(uddiURL);
            String endpointAddress = uddiNaming.lookup(name);

            // criar stub
            TTTImplService service = new TTTImplService();
            TTT port = service.getTTTImplPort();

            // invocar operação remota
            port.play(2, 2, 0);
      }
}
```

d) Considere agora a operação `boolean play(int row, int column, int player)`
Pretende-se que os argumentos sejam enviados de forma confidencial.

i) [0,6] Uma alternativa seria ter o cliente a agrupar e cifrar os 3 argumentos inteiros, que enviaria como argumento único do tipo *byte array*; o método no servidor trataria depois de decifrar esse argumento e obter os argumentos originais em claro.
Que limitações encontra nesta abordagem?

Caso o cliente e o servidor sejam heterogéneos na representação de inteiros, a conversão de dados não irá ser corretamente feita pelos *stubs* o que força o programador a ter que fazer o *marshalling*. Adicionalmente, o contrato WSDL do serviço deixa de representar os argumentos reais da operação.

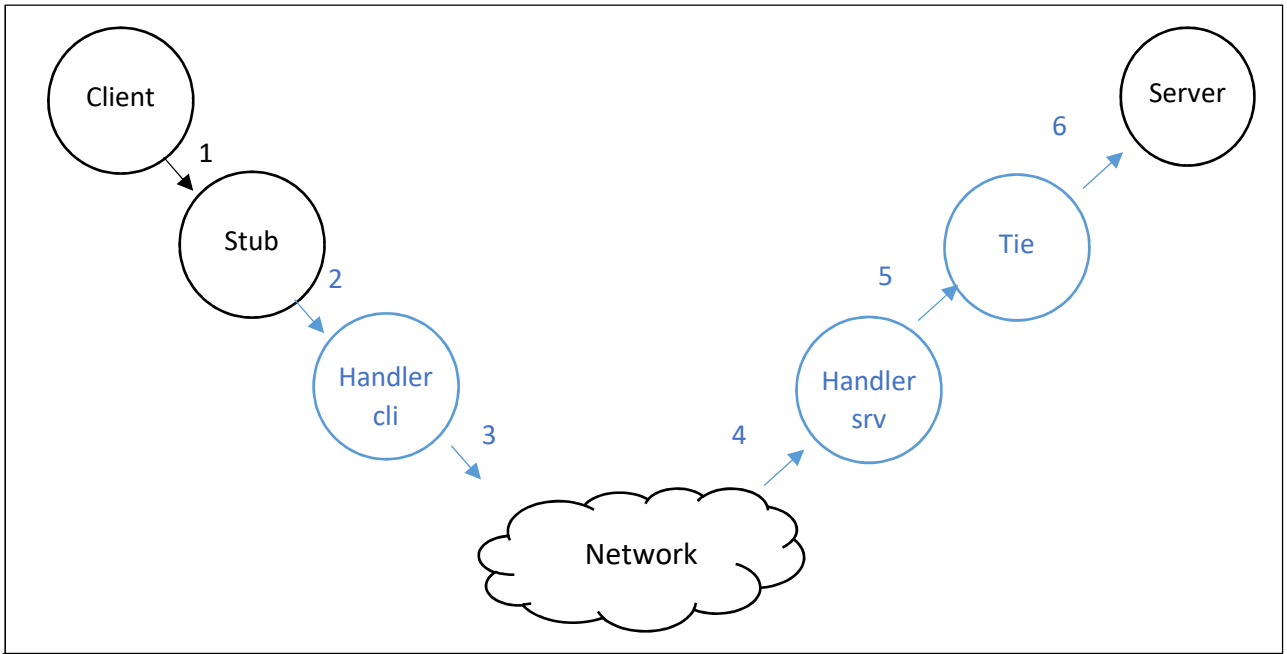
ii) [0,5] Que alternativa sugere tendo em conta os mecanismos disponibilizados pelo JAX-WS?

Sugiro que a cifra seja realizada diretamente na mensagem SOAP, onde os dados já estão serializados. Para aceder e modificar as mensagens SOAP criadas pelos *stubs* devem utilizar-se o *handlers*.

iii) [1,2] Complete o diagrama seguinte para ilustrar o funcionamento da alternativa proposta na alínea anterior na comunicação entre o cliente e o servidor (apenas o **pedido**).

Identifique todos os intervenientes. Represente objetos como círculos, e use setas para simbolizar o envio de dados. Assuma que o cliente já conhece o endereço do servidor.

Numere os passos e faça uma legenda com um breve que resumo o que acontece em cada um.



Legenda:
1 – o cliente passa os argumentos para o stub; 2 – o *stub* serializa os dados em XML;
3 – o *handler* cliente interceta a mensagem, obtém o *body*, cifra os dados e cria um *body* cifrado. Depois deixa a mensagem seguir pela rede (a mensagem é enviada pelo *JAX-WS run-time* do cliente);
4 – o JAX-WS run-time do servidor recebe a mensagem SOAP ainda cifrada;
5 – o *handler* servidor decifra a mensagem e repõe o *body* original.
6- o *tie* recebe a mensagem XML e converte os dados para o formato do servidor, que recebe o pedido