LEIC/LETI 2015-2016, 1º Teste de Sistemas Distribuídos, 1 de abril de 2016

Responda no enunciado, usando apenas o espaço fornecido. Identifique todas as folhas.
Uma resposta errada numa escolha múltipla desconta 1/N do valor da pergunta (sendo N o número de opções).
Duração da prova: 1h30m

Grupo I [7,4]

Considere o seguinte extrato de um programa escrito na IDL 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) O protocolo de RPC envia uma mensagem de invocação e obtém uma mensagem de resposta. Alguns dos elementos presentes no cabeçalho da mensagem de invocação derivam diretamente de elementos presentes na especificação de IDL.
   a) [0,6] Considere a função `READ` e preencha os elementos que conseguir a partir da especificação do procedimento remoto.

Id do Pedido	Id do Programa	Id da Operação
Não definido	9999, 2	2

   b) [0,4] Há algum elemento que não preencheu?
      i) Todos os campos estão definidos na IDL
      ii) O campo Programa não é possível preencher
      iii) O ID do Pedido só pode ser definido em tempos de execução
      iv) O ID do Pedido depende do *binding handle*

iii

2) A especificação dos parâmetros da operação remota na IDL do Sun RPC tem uma estrutura não habitual em C. Indique duas razões para a operação remota não poder ser especificada no formato abaixo:

```
void READ(FileIdentifier f; FilePointer position; int *length; char *buffer)
```

   a) [0,5] Primeira razão: Justifique.

Não é possível determinar se os parâmetros *buffer* ou *length* são de entrada ou saída da função remota

b) [0,5] Segunda razão: Justifique.

char* é ambíguo: pode ser um caracter ou uma cadeia de caracteres

3) Considere o seguinte extrato do programa cliente correspondente à ligação ou *binding* ao serviço remoto.

```
cH = clnt_create(serverName, FILEREADWRITE, VERSION, "udp");
if (cH==NULL) {clnt_pcreateerror(serverName)};
```

a) [0,3] Para que esta função possa ser executada no cliente o que teve de fazer previamente o servidor?

Teve que registar o serviço UDP no rpcbind e tem que estar à escuta de datagramas

b) [0,5] Explique qual é a vantagem de se efetuar a ligação ao servidor desta forma em alternativa a este conhecer previamente os portos onde o servidor está à escuta.

Permite utilização dinâmica e flexível dos portos disponíveis na máquina servidora,
e o cliente adapta-se automaticamente.

c) [0,5] É especificado o parâmetro `udp`. Explique todas as componentes do sistema que permitem esta escolha ao cliente.

Por omissão, o RPCGEN quando gera o main() do servidor cria os portos TCP e UDP e
regista ambos no rpcbind. O cliente quando efetua a ligação bind pode optar porque
os stubs são os mesmos e o *run-time* adapta os timers à semântica adequada ao protocolo de transporte

d) [0,4] Considere os parâmetros da função `clnt_create`.

`serverName` o que indica? Escolha a resposta mais adequada.

i) O endereço da máquina onde se executa o servidor de nomes
ii) O IP do servidor
iii) Deve ter 9999 para ser reconhecido pelo servidor de nomes
iv) O nome DNS da máquina onde o serviço está disponibilizado

i (iv também aceite)

e) [0,5] O programa cliente necessita de conhecer a definição dos parâmetros das funções WRITE e READ para poder usar as funções remotas. Qual é a forma como o ambiente de compilação proporcionado pelo RPCGEN simplifica esta tarefa?

O RPCGEN cria o ficheiro de headers (.h) com as estruturas de dados necessárias para os stubs

f) [0,4] O que é o parâmetro `cH`?
   i) É uma referência que o RPC gera para identificar a ligação entre este cliente e o servidor
   ii) É o identificador do *socket* que ficou ligado ao servidor
   iii) É apenas o código de erro da função de *binding*
   iv) É um *file descriptor*

i

g) [0,8] No contexto do programa descrito na IDL, dê um exemplo realista onde mostre a necessidade de utilização deste parâmetro `cH`.

Se o cliente pretender contactar dois ou mais servidores diferentes ao mesmo tempo, deve usar um *binding handle* distinto para cada servidor

4) Considere o programa descrito e o *binding* efetuado.
   a) [0,5] Qual a semântica de invocação do Sun RPC que este exemplo utiliza?

Pelo-menos-uma-vez

   b) Acha correta esta semântica?
      i) [0,5] Para a função `Read`. Justifique.

Sim, porque é função idempotente, ou seja, se for repetida tem o mesmo efeito prático, ou seja, devolve o mesmo resultado e deixa as variáveis do servidor no mesmo estado.

      ii) [0,5] Para a função `Write`. Justifique.

Sim, a semântica é adequada uma vez que WRITE tem como parâmetro a posição do cursor FilePointer e portanto a sua repetição escreve sempre o mesmo, na mesma posição do ficheiro tornando a função idempotente

   c) [0,5] No caso do exemplo concreto do serviço de leitura e escrita de ficheiros descrito, suponha que o sistema retransmite do lado cliente até ter resposta ou erro e que o servidor também retransmite até ter ACK do lado cliente. Para que a semântica fosse exatamente-uma-vez o que seria ainda necessário no servidor? Justifique.

Eliminar as repetições das mensagens através de uma tabela que para cada cliente regista o ID da mensagem e respetiva resposta. Um mecanismo transacional que em situação de erro no servidor ou ausência de *acknowledge* do cliente faz *rollback* da execução do procedimento no servidor.

Grupo II [5,4]

Considere as seguintes definições em Java.

```
import java.rmi.*;

public enum Suite {
    HEARTS, CLUBS, DIAMONDS, SPADES
}
public enum Figure {
    A,K,Q,D,10,9,8,7,6,5,4,32
}

public class Card implements Serializable {
    private Suite suite;
    private Figure figure;

    public Card(Suite s, Figure f) {
        this.suite = s;
        this.figure = f;
    }
};

public interface DeckofCards extends Remote {
    void addNewPlayer(Player p) throws RemoteException;
    void addNewCard(Player p, Card c) throws RemoteException;
...
}
public interface Player extends Remote {
    DeckofCards getPlayerDeck() throws RemoteException;
...
}
```

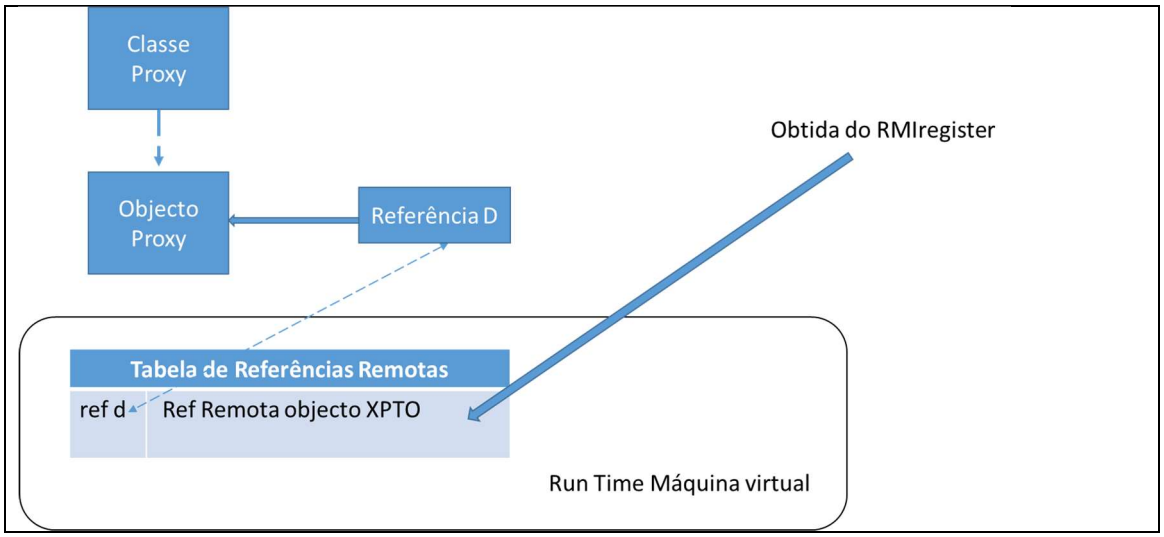
Considere agora que no cliente se codifica o seguinte extrato (corresponde a um jogo de cartas distribuído totalmente imaginário...)

```
1. DeckofCards d = (DeckofCards) Naming.lookup("//gameserver.net/xpto");

2. Player p = new PlayerServant(Francisco, frj@foo.com);
3. Card c = new Card(SPADES, 9);

4. d.addNewPlayer(p);
5. d.addNewCard(p,c);
```

1) Na Linha 1 explique os aspetos abaixo relacionados com essa instrução.
   a) [0,4] Qual a função do parâmetro `//gameserver.net/xpto`?

É o nome do objeto que o identifica no RMIRegistry e que permite obter a sua referência remota.

b) [0,8] Faça um diagrama em que represente o resultado da execução desta instrução no cliente. Deve explicitamente referir os seguintes elementos:
   i) Referência **d**
   ii) Proxy
   iii) Classe *proxy*
   iv) Referência remota



2) [0,4] Referência Remota
   i) Só pode ser obtida a partir do servidor de nomes
   ii) É criada sempre que um objeto entra ou sai como parâmetro de uma invocação remota
   iii) É criada para um objeto que implementa a interface Remote quando é referenciado como parâmetro de um método remoto
   iv) É criado quando se faz *new* da classe do objeto remoto

iii

3) [0,4] Considere a linha 2.
   a) Construtor do objeto
      i) É executado na máquina remota onde se executa o servidor
      ii) É executado na máquina cliente
      iii) Como implementa Remote estes objetos não podem ser criados no cliente
      iv) Os objetos que implementam a interface Remote não têm construtor

ii

4) [0,8] Na chamada remota da linha 5 tem dois parâmetros `p` e `c` de entrada. São passados da mesma forma? Justifique detalhadamente.

P é passado por referência (implementa Remote).
C por valor (implementa Serializable).

5) No final da execução da linha 5.
   a) [0,5] Quantos *proxies* existem no cliente? Justifique.

Apenas 1 corresponde ao objeto XPTO obtido no *lookup* e cujo *proxy* é referenciado por d.
Os outros objetos são criados localmente

   b) [0,5] Resultante da atividade deste cliente quantos *proxies* existem no servidor? Justifique.

1 correspondente ao PlayerServant

6) No RMI existe um mecanismo de *garbage collector* distribuído. Admita que o método usado é de contagem de referências.
   a) [0,5] No servidor onde está em execução o objeto de nome `//gameserver.net/xpto` e se não existir mais nenhuma invocação desse objeto, que valor deverá ter o contador de referências no final da linha 5? Justifique.

O contador tem o valor 2 correspondentes a duas exportações do objeto.
O seu registo no RMIRegistry e a entrada da referência remota no cliente na linha 1.
Em ambas foi chamado addRef()

   b) Considere que o `main()` do programa cliente termina.
      i) [0,6] O que sucede ao objeto cuja referência é `p`? Justifique.

O objeto continua ativo porque tem uma referência no cliente pelo que o valor do contador é 1
não podendo ser eliminado pelo *garbage collector* distribuído

      ii) [0,7] E ao objeto cuja referência é `c`? Responda considerando o caso da VM onde corre o cliente e da VM onde corre o servidor. Justifique.

Este objeto é local e transmitido por cópia portanto é *garbage collected* localmente
e a cópia poderá ou não ficar no servidor dependendo das referências que lhe sejam feitas

Grupo III [7]

Uma **fábrica** de montagem de automóveis decidiu usar a tecnologia de Web Services para interligar os seus sistemas com os sistemas informáticos dos seus **fornecedores** de peças.

Os fornecedores são dezenas e estão geograficamente dispersos, em diferentes países. Os sistemas informáticos estão assentes em plataformas muito diversas (Java, .Net, C++, etc.).

1) [0,6] Concorda com a decisão de se usar a tecnologia de Web Services? Fundamente a resposta.

Sim, porque os Web Services suportam diversos protocolos para comunicação entre máquinas remotas e podem ser implementados em plataformas computacionais heterogéneas.

2) [0,4] Indique quais são as normas (*standards*) dos Web Services que concretizam os seguintes aspetos:

Registo e pesquisa de serviços: UDDI

Especificação de tipos de dados: XSD

Especificação da interface dos serviços: WSDL

Formato das mensagens: SOAP

3) [0,6] Neste cenário, qual a abordagem de desenvolvimento que recomenda para construir o Web Service de cada fornecedor: *Contract-first* ou *implementation-first*? Justifique a recomendação.

Contract-first, para garantir que todos os fornecedores vão implementar exatamente a mesma interface, o que vai permitir ao fabricante comunicar de forma eficaz com a grande quantidade de fornecedores

4) Tendo em conta as diferenças que existem entre os fornecedores.

a) [0,4] Qual é a dificuldade de interpretação que existe ao receber os seguintes valores?

```
<dataEncomenda>01-02-16</dataEncomenda>
<valorEncomenda>1,227</valorEncomenda>
```

Dependendo do contexto nacional, não se sabe exatamente qual é a data:

1 de fevereiro, ou 2 de janeiro ? Não se sabe se a vírgula é separador de décimas ou de milhares.

b) [0,6] Qual é a solução para este problema nos Web Services? Concretize a sua resposta.

São utilizados os tipos de dados do XSD como formato canónico que especificam o detalhe dos campos de dados, nomeadamente o formato das datas e dos números decimais.

5) Considere o seguinte endereço de extremidade (*endpoint*) de um serviço:

```
http://autoparts.com:8000/part.service
```

a) [0,3] Trata-se de um URL ou um URN? Justifique.

URL porque contém informação de localização do serviço.

b) [0,3] Consegue identificar a linguagem de programação utilizada para concretizar o serviço?

Não porque a extremidade não revela os detalhes de implementação do serviço

c) [0,8] De que forma se podem ter Web Services em que o cliente se pode ligar ao servidor mesmo que este mude de endereço? Detalhe o que é necessário fazer no servidor e no cliente.

Através de registo e consulta no servidor de nomes UDDI.
O servidor deve publicar o seu endereço e nome no UDDI.
O cliente deve pesquisar no UDDI o nome do serviço para obter o respetivo endereço.

6) Considere a seguinte operação abstrata, descrita em sintaxe Java, correspondente à consulta de disponibilidade de uma peça de automóvel para encomenda:

```
int checkPart(String part) throws UnknownPart;
```

a) [0,5] No documento WSDL vão existir dois elementos XML com o mesmo nome: '`part`' que são o nome do argumento da função `checkPart` e a definição de uma parte de mensagem do próprio WSDL. Que mecanismo XML se utiliza para evitar estes conflitos de nomes?

São utilizados os espaços de nomes (*namespaces*) que permitem adicionar um prefixo a cada etiqueta
que permite identificar a proveniência do nome utilizado.

b) [0,5] A fábrica pretende comunicar com os fornecedores usando mensagens SOAP sobre HTTP. Em que seção do WSDL é que esta opção é comunicada aos fornecedores?

binding

c) [0,6] Construa uma mensagem SOAP de pedido da função acima em Web Services, considerando que se quer saber o número de unidades disponíveis da peça z0612. (pode simplificar a sintaxe do XML, desde que a estrutura seja apresentada de forma clara)

```
<soap:envelope>
        <soap:body>
                <checkPart>
                        <part>z0612</part>
                </checkPart>
        </soap:body>
</soap:envelope>
```

d) [0,6] Construa uma mensagem SOAP de resposta à invocação remota da função acima em Web Services considerando que a peça pretendida existe, e tem 22 unidades.

```
<soap:envelope>
        <soap:body>
                <checkPartResponse>
                        <result>22</result>
                </checkPartResponse>
        </soap:body>
</soap:envelope>
```

e) [0,8] Construa uma mensagem SOAP com uma resposta quando a peça é desconhecida, sendo que neste caso o servidor atira a exceção `UnknownPart`.

```
<soap:envelope>
        <soap:body>
                <soap:fault>
                        <faultCode>soap:Server</faultCode>
                        <faultString>The requested part is unknown!</faultString>
                        <detail>
                                <UnknownPart>
                                        <part>xpto</part>
                                </UnknownPart>
                        </detail>
                </soap:fault>
        </soap:body>
</soap:envelope>
```