Paradigmas de Computação Paralela

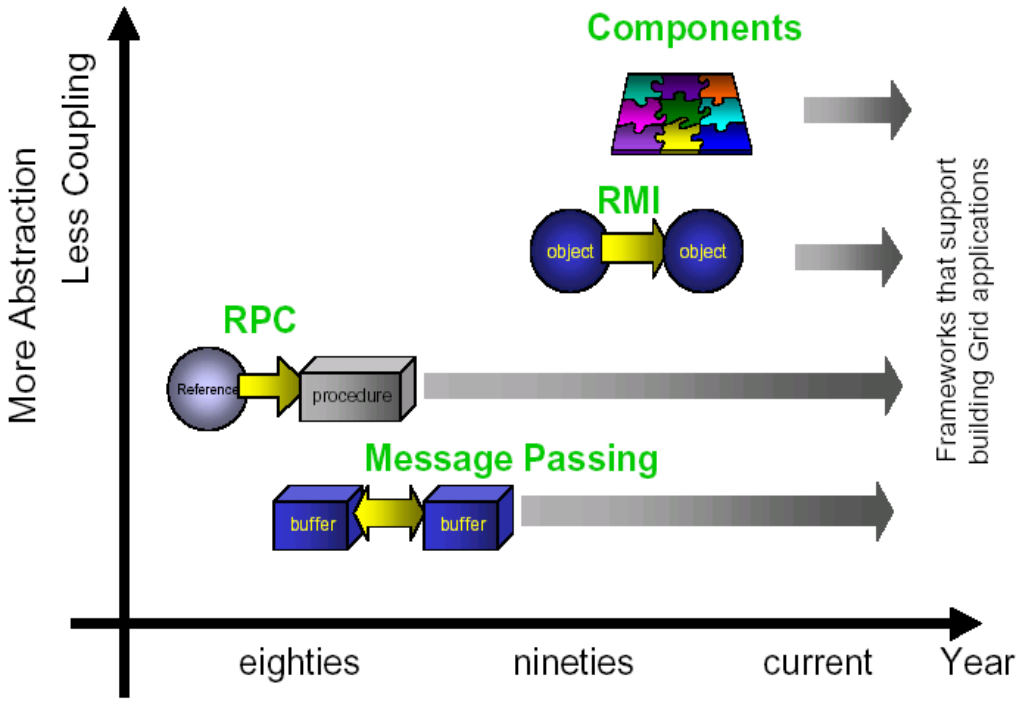
Middleware de Aplicações Paralelas/Distribuídas

João Luís Ferreira Sobral
Departamento do Informática
Universidade do Minho

27 Outubro 2015

Paradigmas de Computação Paralela

Tecnologias para desenvolvimento de aplicações



Middleware para aplicações paralelas/distribuídas

Principais aspectos a gerir pelo *Middleware*

- **Como comunicam as entidades (objectos, componentes ou processos)?**
- **Como são identificadas as entidades?**
- **Qual o ciclo de vida das entidades (como são criadas e destruídas)?**

Comunicação entre entidades

- **Invocação directa de métodos ou de procedimentos,** em aplicações não distribuídas
  - podem ser implementadas através de um salto para o endereço correspondente ao método/ procedimento a realizar, sendo os parâmetros passados na pilha ou em registos do processador.

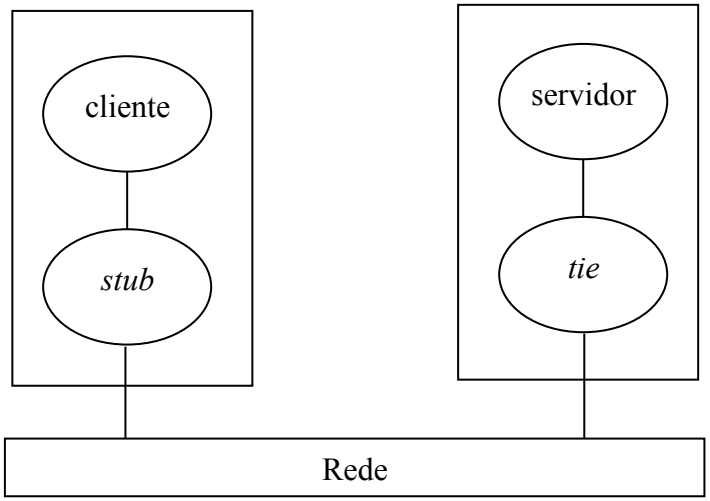
- **Primitivas do tipo *send(host, port, message*) e *recv([host],port*).**

- Em aplicações distribuídas clássicas, baseadas em processos, as entidades podem comunicar através da passagem de mensagens, por exemplo, utilizando primitivas do tipo *send(host, port, message*) e *recv([host],port*).
- A primitiva *send* pode ser implementada através do envio de um pacote de informação ao *host* indicado, contendo a identificação do destinatário *port* e a mensagem.
- No receptor é necessário extrair a mensagem do pacote de informação e activar o processo receptor à espera da mensagem.

Middleware para aplicações paralelas/distribuídas

Principais aspectos a gerir pelo *Middleware*

- **Comunicação entre entidades (cont.)**
  - **Invocação remota de métodos**
    - Geralmente utilizado em aplicações distribuídas desenvolvidas através de componentes ou objectos
    - Pode ser implementado "colando" um procurador (*stub*) junto do cliente, procurador que implementa a mesma interface que o objecto ou componente remoto e que o substitui de forma transparente. Este *stub* converte cada invocação de método numa mensagem enviada ao objecto remoto.
    - Junto ao objecto remoto existe um *tie* que descodifica a mensagem e invoca o método correspondente no objecto local:



Middleware para aplicações paralelas/distribuídas

Principais aspectos a gerir pelo *Middleware*

- **Identificação das entidades**
  - **Em aplicações não distribuídas, uma entidade pode ser identificada directamente pelo seu endereço de memória (referência).**
  - **Em aplicações distribuídas**
    - **um par (máquina, endereço) ou (máquina, porta)**
    - **um servidor de nomes que permite a utilização de nomes abstractos para identificar as entidades. Cada entidade pode-se registar nesse servidor de nomes, o qual é utilizado pelos clientes para obter referências para entidades registadas.**
  - **O esquema de identificação das entidades dever ser universal, de forma a ser possível passar os identificadores entre máquinas**

Middleware para aplicações paralelas/distribuídas

Principais aspectos a gerir pelo *Middleware*

- **Ciclo de vida das entidades**
  - **Para que uma entidade (objecto, componente ou processo) seja criado numa máquina é necessário que o executável correspondente se encontre nessa máquina ou que seja descarregado juntamente com o pedido de criação (processo conhecido por *deployment*).**
  - **A criação de entidades pode ser efectuada de forma manual (correndo o executável na máquina remota), ou utilizando uma fábrica de entidades. Neste último caso, é necessário activar a fábrica.**
  - **Podem existir várias fábricas de objectos (nomeadamente, uma por classe em cada nodo), sendo necessário introduzir um serviço para localizar as várias fábricas.**
  - **A destruição automática dos objectos em aplicações distribuídas é mais complexa, uma vez que é necessária comunicação entre os diversos nodos para efectuar a contagem das referências existentes.**

Passagem de mensagens *versus* Invocação remota de métodos (RMI)

	Passagem de Mensagens	Invocação Remota de Métodos
Dados a transmitir	Empacotamento de dados	Lista de parâmetros
Envio de pedidos/ informação	Envio explícito de mensagens etiquetadas	Invocação de um método específico
Recepção de pedidos/ informação	Recepção explícita de mensagens	Execução implícita do método invocado
Reacção do receptor	A acção a executar é determinada pela etiqueta (tag) da mensagem	A acção a executar é determinada pelo método invocado
Identificação do receptor	Canal, nome ou anónima	Apontador para objecto remoto (proxy)

Middleware para aplicações paralelas/distribuídas

Outros serviços a gerir pelo *Middleware (não abordados na disciplina)*

- **Transacções distribuídas**
- **Autenticação/Autorização**
- **Persistência**

Middleware para aplicações paralelas/distribuídas

Exemplos de *Middleware*

- **PVM (*Parallel Virtual Machine*) e MPI (*Message Passing Interface*) –**
  - **o envio e a recepção de mensagens entre processos (na mesma máquina ou em máquinas diferentes).**
  - **primitivas para o envio e recepção de mensagens (síncronas/assíncronas e bloqueantes e não bloqueantes).**
  - **identificação é realizada através de portas, existindo primitivas para dinamicamente lançar um processo remoto (PVM).**

- **CORBA (*Common Object Request Broker Architecture*) –**
  - **arquitectura que permite a inter-operabilidade de componentes**
  - **baseada na definição de interfaces de componentes (IDL – *Interface Definition Language*) e num serviço de nomes que permite localizar componentes e invocar métodos (ORB – *Object Request Broker*).**
  - **A inter-operabilidade é assegurada através da utilização de um protocolo de comunicação baseado em TCP/IP (IIOP – *Internet Inter-ORB Protocol*), que define a forma como a informação que é trocada entre ORBs.**

Middleware para aplicações paralelas/distribuídas

Exemplos de *Middleware*

- **Java RMI e Java RMI-IIOP – Permite a invocação de métodos entre objectos Java distribuídos. O serviço de nomes pode ser fornecido pelo RMI *Registry* ou por JNDI (*Java Naming and Directory Interface*). RMI-IIOP utiliza protocolos compatíveis com CORBA.**

- **J2EE (*Java 2 Enterprise Edition*) – Especificação para o desenvolvimento de aplicações empresariais baseadas em componentes (*Enterprise Java Beans*), distribuídos e com várias camadas. Suporta o desenvolvimento do componentes para a apresentação (*JSP* e *Servlets*), para a camada de computação (*Session Beans*) e para a camada de BD (*Entity Beans*). Suporta ainda transacções, segurança com autenticação, e passagem de mensagens.**

- **.NET Remoting – Alternativa da Microsoft ao desenvolvimento de aplicações empresariais distribuídas. Incluída na plataforma .NET, suporta a invocação remota de métodos e a gestão do ciclo de vida de objectos distribuídos.**

Middleware para aplicações paralelas/distribuídas

Java RMI/IIOP

- ***Middleware*** **utilizado para efetuar invocações de métodos entre objetos em diferentes JVM ou em componentes compatíveis com CORBA.**
- **Cada objecto servidor deve exportar um ou vários interfaces, existindo um utilitário *rmic* para explicitamente gerar o *stub* e o *tie*.**
- **O objecto servidor regista-se no serviço de nomes (JNDI) com um nome abstracto, sendo esse serviço utilizado pelos clientes para obterem uma referência ao objecto remoto.**
- **O *deployment* é manual, existindo um protocolo que permite que os objetos servidores sejam descarregados e ativados automaticamente (derivando da classe *Activable*)**
- **Na invocação remota de métodos, os objetos declarados como remotos são passados por referência, enquanto os outros objetos são passados SEMPRE por valor:**

Tipo de parâmetro	Método local	Método Remoto
Primitivo	por valor	por valor
Objecto	por referência	por valor (cópia em profundidade)
Objecto Remoto	por referência	por referência

Middleware para aplicações paralelas/distribuídas

Java RMI/IIOP (exemplo – componente que divide dois números)

1. **Definir a interface do componente remoto, estendendo a classe *Remote*. Cada método deve ser declarado como gerando uma *RemoteException***

```
import java.rmi.*;
public interface DivideServer extends Remote {
          double divide(double d1, double d2) throws RemoteException;
}
```

2. **Codificar o objecto servidor que implementa a interface, estendendo a classe *RemoteObject***

```
import java.rmi.*;
import java.rmi.server.*;
public class DivideServerImpl extends RemoteObject implements DivideServer {
          public double divide(double d1, double d2) throws RemoteException {
                    return d1 / d2;
          }
}
```

Java RMI/IIOP (exemplo – cont.)

3. Codificar o programa que cria o servidor e o regista no serviço de nomes.

```
import java.io.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class DivideServerApp {
     public static void main(String args[]) {
        try {
             DivideServerImpl dsi = new DivideServerImpl();
             PortableRemoteObject.exportObject(dsi);
               Context ctx = new InitialContext();
               ctx.rebind("DivideServer",dsi);
                  ...
               ctx.unbind("DivideServer");
               PortableRemoteObject.unexportObject(dsi);
          } catch(Exception e) { e.printStackTrace(); }
     }
}
```

Java RMI/IIOP (exemplo – cont.)

4. Codificar o programa cliente.

```
public class DivideClient {
   public static void main(String args[]) {
       try {
          Context ctx = new InitialContext();
          DivideServer ds = (DivideServer)
               PortableRemoteObject.narrow(
               ctx.lookup("DivideServer"), DivideServer.class);
          double d1 = Double.valueOf(args[0]).doubleValue();
          double d2 = Double.valueOf(args[1]).doubleValue();
          // Invoke remote method and display result
         double result = ds.divide(d1, d2);
          System.out.println("The result is: " + result);
       }  catch(Exception ex) { ex.printStackTrace();     }
     }
 }
```

Middleware para aplicações paralelas/distribuídas

Java RMI/IIOP (exemplo – cont.)

5. **Compilar os ficheiros e gerar o *stub* e o *tie* com**:

```
javac *.java
rmic –iiop DivideServerImpl
```

6. **Iniciar o servidor de nomes, correr a aplicação do servidor e correr o cliente**:

```
tnameserv
java  -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
     -Djava.naming.provider.url=iiop://localhost DivideServerApp
java  -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
      -Djava.naming.provider.url=iiop://localhost DivideClient
```

7. **Para executar o servidor numa máquina remota é necessário copiar os ficheiros para essa máquina, iniciar servidor de nomes e o servidor nessa máquina. Ao cliente deve ser indicado que o servidor de nome é remoto com:**

```
java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
      -Djava.naming.provider.url=iiop://<hostserver> DivideClient
```

Middleware para aplicações paralelas/distribuídas

.Net Remoting

- ***Middleware*** **introduzido pela Microsoft para suportar invocações remotas entre objectos/componentes, baseado em mecanismos semelhantes aos do Java-RMI**
- **Suporta dois tipos diferentes de mecanismos de transporte para as invocações remotas de métodos: HTTP e TCP**
- **Melhora a facilidade de implementação não sendo necessário:**
  - **definir uma interface para os componentes remotos**
  - **activar explicitamente o servidor de nomes**
  - **gerar/compilar manualmente os proxy/stubs**
- **Proporciona serviços adicionais para gestão do ciclo de vida dos objectos e a invocação assíncrona de métodos**
- **.NET remoting é uma arquitectura extensível, sendo possível modificar o comportamento de vários componentes do sistema**

Middleware para aplicações paralelas/distribuídas

.Net Remoting (cont)

- **Suportar *assemblies* partilhados, podendo os objectos remotos ser implementados como um serviço Windows ou no IIS**
- **Possibilita a extracção de informação (interface) sobre executáveis, mesmo que estes se encontrem numa máquina remota**
- **Alternativas para a activação do objecto servidor remoto:**
  - **Singleton – existe uma só instância remota, criada na primeira chamada e partilhada por todos os clientes**
  - **SingleCall – é criada uma instância do classe para cada chamada**
  - **publicação de objectos já criados (semelhante ao processo utilizado em JavaRMI)**
  - **criação de objectos activada pelo cliente (criação remota através do operador *new* como numa aplicação normal)**
- **Gestão da vida dos objectos servidores remotos**
  - **Nas aplicações distribuídas o sistema de contagem de referências não é indicado porque as falhas nos clientes são frequentes**
  - **Em remoting, os objectos servidores possuem um tempo de vida (5 min) que é incrementado em cada invocação de um método**
  - **No fim do tempo de vida um patrocinador do objecto decide se este deve ser destruído ou não.**

Middleware para aplicações paralelas/distribuídas

.Net Remoting (exemplo)

1. **Definir a interface do componente remoto**

```
public interface DivideServer {
   double divide(double d1, double d2);
}
```

2. **Codificar o objecto servidor que implementa o serviço, derivando da classe *MarshalByRefObject***

```
public class DivideServerImpl : MarshalByRefObject, DivideServer {
          public double divide(double d1, double d2) {
                    return d1 / d2;
}
```

.Net Remoting (exemplo – cont.)

3. **Codificar o programa que cria o servidor e o regista no serviço de nomes.**

   (pode ser feito acrescentando um main na classe anterior).

```
public static int Main (string [] args) {
    HttpChannel cn = new HttpChannel (1000);
    ChannelServices.RegisterChannel(cn);
    RemotingConfiguration.RegisterWellKnownServiceType(
              typeof(DivideServerImpl),     "DivideServer",
                                  WellKnownObjectMode.Singleton);
    Console.Write("Type ENTER to shutdown ");
    Console.ReadLine();
    return 0;
}
```

.Net Remoting (exemplo – cont.)

4. Codificar o programa cliente.

```
public class DivideClient {
          public static int Main (string [] args) {
          // Obtain a reference to that remote object
          HttpChannel cn = new HttpChannel();
          ChannelServices.RegisterChannel(cn);
          DivideServer ds = (DivideServer)

Activator.GetObject( typeof(DivideServer),
                    "http://localhost:1000/DivideServer");
          double d1 = Convert.ToDouble(args[0]);
          double d2 = Convert.ToDouble(args[1]);

          // Invoke remote method
          double result = ds.divide(d1, d2);
}
```

Middleware para aplicações paralelas/distribuídas

.Net Remoting (exemplo – cont.)

5. **Compilar os ficheiros (o *stub* e o *tie* são gerados automaticamente)**

   - `csc /t:library divideserver.cs`
   - `csc /r:divideserver.dll divideserverimpl.cs`
   - `csc /r:divideserver.dll divideclient.cs`

6. **Correr ao servidor e o cliente (não é necessário iniciar o servidor de nomes)** :

   - `start divideserverimpl`
   - `divideclient.exe 12 14`

7. **Para executar o servidor numa máquina remota é necessário copiar os ficheiros para essa máquina, iniciar o servidor nessa máquina. Ao cliente deve ser indicado que o servidor é remoto com:**

```
DivideServer ds = (DivideServer) Activator.GetObject(
          typeof(DivideServer),
          "http://<<hostname>>:1000/DivideServer");
```

Middleware para aplicações paralelas/distribuídas

.Net Remoting vs Java RMI (s/IIOP)

```
public interface IDServer extends Remote {
  double divide(double d1, double d2) throws RemoteException;
}
public class DServer extends UnicastRemoteObject implements IDServer {
  public double divide(double d1, double d2) throws RemoteException {
    return d1 / d2;
  }
  public static void main(String args[]) {
    try {
      DServer dsi = new DServer();
      Naming.rebind("rmi://host:1050/DivideServer",dsi);
    } catch(Exception e) { e.printStackTrace(); }
  }
}
public class DivideClient {
  public static void main(String args[]) {
    try {
      IDServer ds; // Obtains a reference to the remote object
      ds = (IDServer) Naming.lookup("rmi://host:1050/DivideServer");
...
      double result = ds.divide(d1, d2);
```

```
public interface IDServer {
  double divide(double d1, double d2);
}
public class DServer : MarshalByRefObject, IDServer {
  public double divide(double d1, double d2) {
    return d1 / d2;
  }
  public static int Main (string [] args) {
    TcpChannel cn = new TcpChannel (1050);
    ChannelServices.RegisterChannel(cn);
    RemotingConfiguration.RegisterWellKnownServiceType( typeof(DServer),
      "DivideServer", WellKnownObjectMode.Singleton);
  }
}
public class DivideClient {
  public static int Main (string [] args) {
    TcpChannel cn = new TcpChannel();
    ChannelServices.RegisterChannel(cn);
    IDServer ds = (IDServer) Activator.GetObject( typeof(DivideServer),
      "tcp://localhost:1050/DivideServer");
    ...
      double result = ds.divide(d1, d2);
```

Middleware para aplicações paralelas/distribuídas

Exercícios

- Execute o programa "Divide" apresentado nas aulas
- Utilize a opção *–keep* do *rmic* para não remover o código fonte do *stub* e do *tie*. Observe e tente compreender código gerado pelo *rmic* para estes dois componentes (ficheiros *_*_Stub.java* e *_*_Tie.java*).
- Desenvolva um componente em que efectue a conversão de Euros em Escudos e vice-versa. Experimente executar esse componente na máquina local e numa máquina remota.
- Altere o programa anterior por forma a que sejam medido o tempo necessário para obter uma referência para o objecto e o tempo necessário para invocar um método do objecto. Meça os tempos quando o componente é executado local ou remotamente. Compare com os tempos que são obtidos quando não é utilizado o processo de invocação remota de métodos.

  Nota: para medir o tempo com precisão utilize o método *getTime()* de *java.util.Date*, o qual devolve o tempo actual, em milisegundos.
- Desenvolva um programa que meça a quantidade de dados que consegue ser efectivamente transmitida entre componentes remotos. Baseia-se num componente que possui um método com um parâmetro com uma quantidade elevada de dados e que retorne também a mesma quantidade de dados. Teste este programa para diferentes quantidades de dados