Sistemas Distribuídos

Comunicação e sincronização entre processos

Objetos Remotos - 2

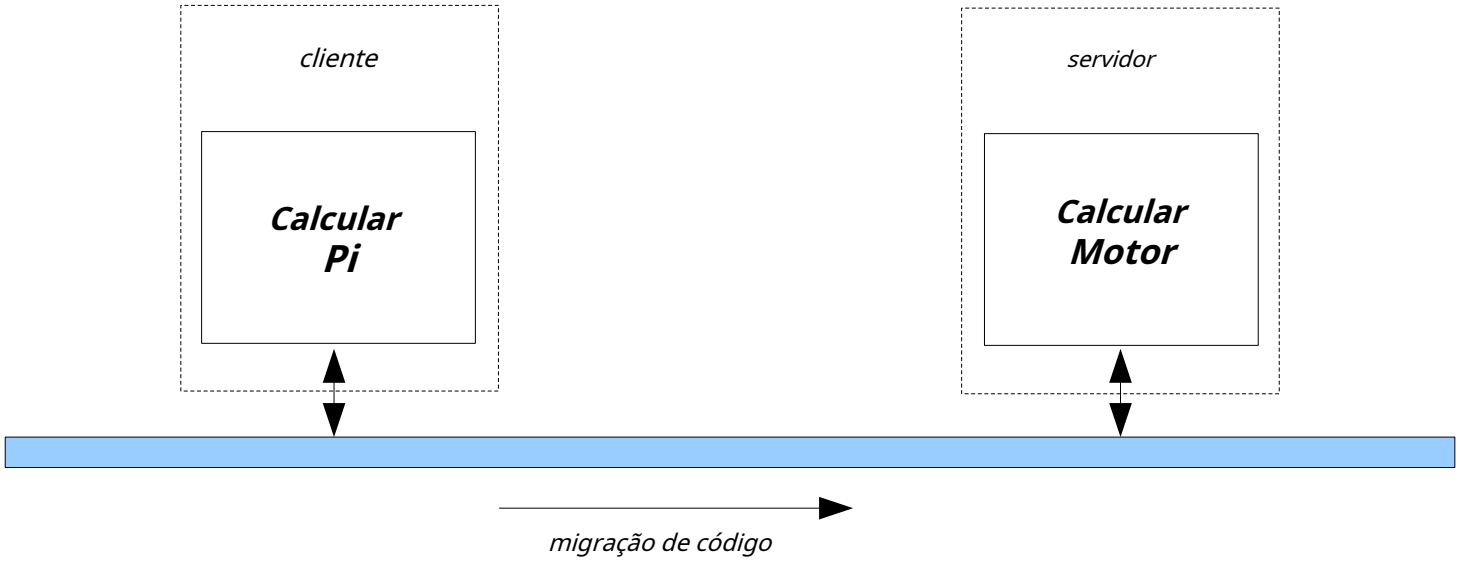
António Rui Borges

Resumo

- *Caracterização do problema*
- *Descrição do código*
- *Diagramas de interação*
- *Organização do pacote BackEngine*
- *Política de segurança local*
- *Construir e implantar*
- *Executando o aplicativo*
- *Leitura sugerida*

Caracterização do problema – 1

- é um exemplo, adaptado do tutorial da Sun sobre *RMI*, para ilustrar como o código pode ser transferido entre máquinas virtuais Java, localizadas na mesma ou em diferentes plataformas de hardware, e ser executado em uma JVM diferente daquela onde foi armazenado anteriormente

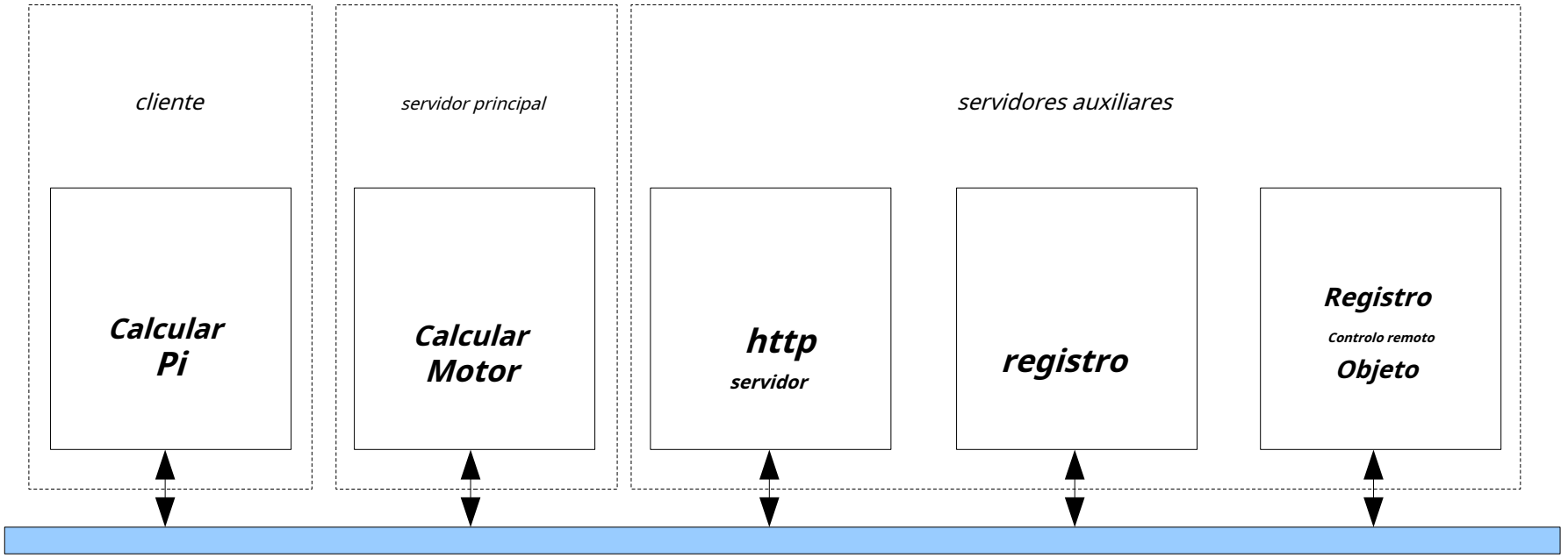


Caracterização do problema – 2

- duas classes principais de entidades são consideradas
  - a *objeto servidor*, do tipo ComputeEngine, que fornece um serviço para execução local de código migrado sob controle remoto
  - diversos *objetos cliente* para lidar com a execução local de código migrado sob controle remoto (interfaces CalculareTarefa); no presente caso, outro objeto, de tipo ComputePi, transfere um objeto do tipo Pi (cálculo dep com um número variável de decimais) para execução remota
- e três entidades auxiliares
  - um serviço de nomenclatura para registrar objetos remotos (registro)
  - a *objeto servidor*, do tipo RegistrarRemoteObject, que fornece suporte para registrar objetos remotos (aqueles localizados em JVMs residentes em plataformas de hardware diferentes daquela em que o serviço de nomenclatura está localizado)
  - a *servidor http* para auxiliar o download dinâmico dos tipos de dados utilizados em chamadas remotas.

Caracterização do problema – 3

Configuração operacional



Registrar thread base de objeto remoto - 1

```
aula pública ServerRegisterRemoteObject {

    vazio estático público principal(String[] argumentos) {

      /* obtém a localização do serviço de registro */

        String rmiRegHostName;
        interno rmiRegPortNumb;

        GenericIO.writeString ("Nome do nó de processamento onde está localizado o serviço de registro? ");
        rmiRegHostName=GenericIO.readlnString();
        GenericIO.writeString ("Número da porta onde o serviço de registro está escutando? ");
        rmiRegPortNumb=GenericIO.readlnInt();

      /* cria e instala o gerenciador de segurança */

        se(System.getSecurityManager() == nulo)
              System.setSecurityManager (novo SecurityManager ());
        GenericIO.writelnString ("O gerenciador de segurança foi instalado!");

      /* instancia um objeto remoto de registro e gera um stub para ele */

        RegisterRemoteObject regEngine = novo RegisterRemoteObject (rmiRegHostName, rmiRegPortNumb); Registrar
        regEngineStub = null;
        interno listenPort = 22001;                                   /* deve ser definido de acordo em cada caso */

        tentar
        { regEngineStub = (Registrar) UnicastRemoteObject.exportObject (regEngine, listenPort); }

        pegar(RemoteException e)
        { GenericIO.writelnString ("Exceção de geração de stub RegisterRemoteObject: " + e.getMessage ());
            Sistema.exit (1);
        }
        GenericIO.writelnString ("Stub foi gerado!");
```

Registrar thread base de objeto remoto - 2

```
        /* registre-o no serviço de registro local */

          String nomeEntry = "RegisterHandler"; Registro
          de registro = null;

          tentar
          { registro = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb); }

          pegar(RemoteException e)
          { GenericIO.writelnString ("Exceção de criação de registro RMI: " + e.getMessage ());
              Sistema.exit (1);
          }
          GenericIO.writelnString ("O registro RMI foi criado!");

          tentar
          {registro.rebind (nameEntry, regEngineStub); }

          pegar(RemoteException e)
          { GenericIO.writelnString ("RegisterRemoteObject exceção remota no registro: " +
                                                  e.getMessage());
              Sistema.exit (1);
          }
          GenericIO.writelnString ("Objeto RegisterRemoteObject foi registrado!");
      }
}
```

Registrar Objeto Remoto e interfaces relacionadas - 1

interface pública Registro**estende**Controlo remoto {

vazio públicobind (nome da string, referência remota)**lança**RemoteException, JáBoundException;

vazio públicodesvincular (nome da string)**lança**RemoteException, NotBoundException;

vazio públicoreligar (nome da string, referência remota)**lança**RemoteException;
}

Registrar Objeto Remoto e interfaces relacionadas - 2

```
aula pública RegistrarRemoteObject implementos Registro {

    privado String rmiRegHostName; privado
    interno rmiRegPortNumb = 1099;

    público RegisterRemoteObject (String rmiRegHostName, interno rmiRegPortNumb) {

        se ((rmiRegHostName == nulo) || ("".igual a (rmiRegHostName)))
            jogue novo NullPointerException ("RegisterRemoteObject: parâmetro de ponteiro nulo na instanciação!"); this.rmiRegHostName =
            rmiRegHostName;
        se ((rmiRegPortNumb >= 4000) && (rmiRegPortNumb <= 65535))
            this.rmiRegPortNumb = rmiRegPortNumb;
    }

    vazio público bind (nome da string, referência remota) lança RemoteException, JáBoundException {

        Registro de registro;

        se ((nome == nulo) || (referência == nulo))
            jogue novo NullPointerException ("RegisterRemoteObject: parâmetro(s) de ponteiro nulo ativado(s) na ligação!"); registro =
        LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
        registro.bind (nome, referência);
    }
```

Registrar Objeto Remoto e interfaces relacionadas - 3

```
    vazio público desvincular (nome da string) lança RemoteException,NotBoundException{

        Registro de registro;

        se((nome == nulo))
            jogue novo NullPointerException ("RegisterRemoteObject: parâmetro(s) de ponteiro nulo ao desvincular!"); registro =
        LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
        registro.unbind (nome);
    }
    vazio público religar (nome da string, referência remota) lança RemoteException {

        Registro de registro;

        se((nome == nulo) || (referência == nulo))
            jogue novo NullPointerException ("RegisterRemoteObject: parâmetro(s) de ponteiro nulo na religação!"); registro =
        LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
        registro.rebind (nome, ref);
    }
}
```

Thread base do Compute Engine: 1

```
aula pública ServidorComputeEngine {

    vazio estático público principal(String[] argumentos) {

      /* obtém a localização do serviço de registro */

        String rmiRegHostName;
        interno rmiRegPortNumb;

        GenericIO.writeString ("Nome do nó de processamento onde está localizado o serviço de registro? ");
        rmiRegHostName=GenericIO.readlnString();
        GenericIO.writeString ("Número da porta onde o serviço de registro está escutando? ");
        rmiRegPortNumb=GenericIO.readlnInt();

      /* cria e instala o gerenciador de segurança */

        se(System.getSecurityManager() ==nulo)
              System.setSecurityManager (novo SecurityManager ());
        GenericIO.writelnString ("O gerenciador de segurança foi instalado!");

      /* instancia um objeto remoto que executa código móvel e gera um stub para ele */

        Motor ComputeEngine =novo ComputeEngine();
        Mecanismo de computaçãoStub =nulo; interno listenPort =
        22002;                                                    /*deve ser definido de acordo em cada caso*/

        tentar
        {engineStub = (Cálculo) UnicastRemoteObject.exportObject (mecanismo, listenPort); }

        pegar(RemoteException e)
        { GenericIO.writelnString ("Exceção de geração de stub ComputeEngine:" + e.getMessage ());
            e.printStackTrace();
            Sistema.exit (1);
        }
        GenericIO.writelnString ("Stub foi gerado!")
```

Thread base do Compute Engine: 2

```
/* registre-o no serviço de registro geral */

  String nameEntryBase = "RegisterHandler"; String
  nameEntryObject = "Cálculo"; Registro de registro =
  nulo; Registro de registro =nulo;


  tentar
  { registro = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb); }

  pegar(RemoteException e)
  { GenericIO.writelnString ("Exceção de criação de registro RMI: " + e.getMessage ());
      e.printStackTrace();
      Sistema.exit (1);
  }
  GenericIO.writelnString ("O registro RMI foi criado!");

  tentar
  { reg = (Registro) registro.lookup (nameEntryBase); }

  pegar(RemoteException e)
  { GenericIO.writelnString ("exceção de pesquisa RegisterRemoteObject:" + e.getMessage ());
      e.printStackTrace();
      Sistema.exit (1);
  }
  pegar(NotBoundException e)
  { GenericIO.writelnString ("RegisterRemoteObject não vinculado exceção:" + e.getMessage ());
      e.printStackTrace();
      Sistema.exit (1);
  }
```

Thread base do Compute Engine: 3

```
        tentar
        { reg.bind (nomeEntryObject, engineStub); }

        pegar(RemoteException e)
        { GenericIO.writelnString ("Exceção de registro do ComputeEngine: " + e.getMessage ());
            e.printStackTrace();
            Sistema.exit (1);
        }
        pegar(JáBoundException e)
        { GenericIO.writelnString ("ComputeEngine já vinculado exceção:" + e.getMessage ());
            e.printStackTrace();
            Sistema.exit (1);
        }
        GenericIO.writelnString ("Objeto ComputeEngine foi registrado!");
    }
}
```

Objeto remoto do Compute Engine e interfaces relacionadas

```
interface pública Calcular estende Controlo remoto {
    Objeto executeTask (Tarefa t) lança RemoteException; }

interface pública Tarefa estende Serializável {
    público estático final longo serialVersionUID = 2021L;
    Objeto executar();
}
aula pública ComputeEngine implementos Calcular {
    objeto público executeTask (Tarefa t) {
        retornar t.execute();
    }
}
```

Base de thread do cálculo Pi - 1

```
aula pública ComputePi
{
    vazio estático público principal(String args[]) {

      /* obtém a localização do serviço de registro genérico */

        String rmiRegHostName;
        interno rmiRegPortNumb;

        GenericIO.writeString ("Nome do nó de processamento onde está localizado o serviço de registro? ");
        rmiRegHostName=GenericIO.readlnString();
        GenericIO.writeString ("Número da porta onde o serviço de registro está escutando? ");
        rmiRegPortNumb=GenericIO.readlnInt();

      /* procura o objeto remoto pelo nome no registro do host remoto */

        String nameEntry = "Cálculo"; Calcular
        cálculo = nulo; Registro de registro =
        nulo;

        tentar
        { registro = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb); }

        pegar(RemoteException e)
        { GenericIO.writelnString ("Exceção de criação de registro RMI: " + e.getMessage ());
            e.printStackTrace();
            Sistema.exit (1);
        }
```

Base de threads do Compute Pi - 2

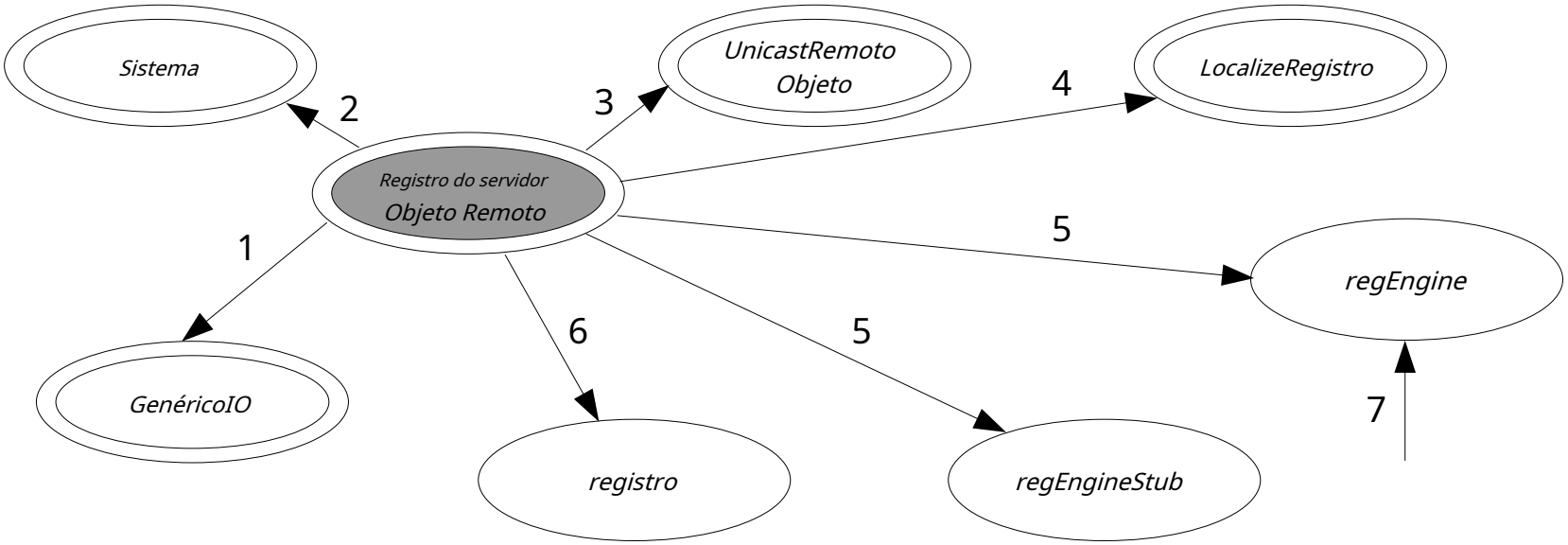
```
tentar
{ comp = (Cálculo) registro.lookup (nameEntry); }

pegar(RemoteException e)
{ GenericIO.writelnString ("Exceção de pesquisa do ComputePi: " + e.getMessage ());
    e.printStackTrace();
    Sistema.exit (1);
}
pegar (NotBoundException e)
{ GenericIO.writelnString ("Exceção ComputePi não vinculada: " + e.getMessage ());
    e.printStackTrace();
    Sistema.exit (1);
}

/* instancia o objeto de código móvel a ser executado remotamente */

Tarefa Pi =nulo;
GrandeDecimal pi =nulo;

tentar
{ tarefa =novoPi (Integer.parseInt (args[0])); }

pegar(NumberFormatException e)
{ GenericIO.writelnString ("Exceção de instanciação Pi: " + e.getMessage ());
    e.printStackTrace();
    Sistema.exit (1);
}
```

Base de threads de computação Pi - 3

```
        /* invoca o método remoto (executa o código em um objeto remoto do ComputeEngine) */

          tentar
          { pi = (BigDecimal) (comp.executeTask (tarefa)); }

          pegar(RemoteException e)
          { GenericIO.writelnString ("exceção de invocação remota do ComputePi: " + e.getMessage ());
              e.printStackTrace();
              Sistema.exit (1);
          }

          /* imprime o resultado */ GenericIO.writelnString
          (pi.toString ());
    }
}
```

Diagramas de interação - 1

Objeto remoto para suportar o registro de objetos remotos

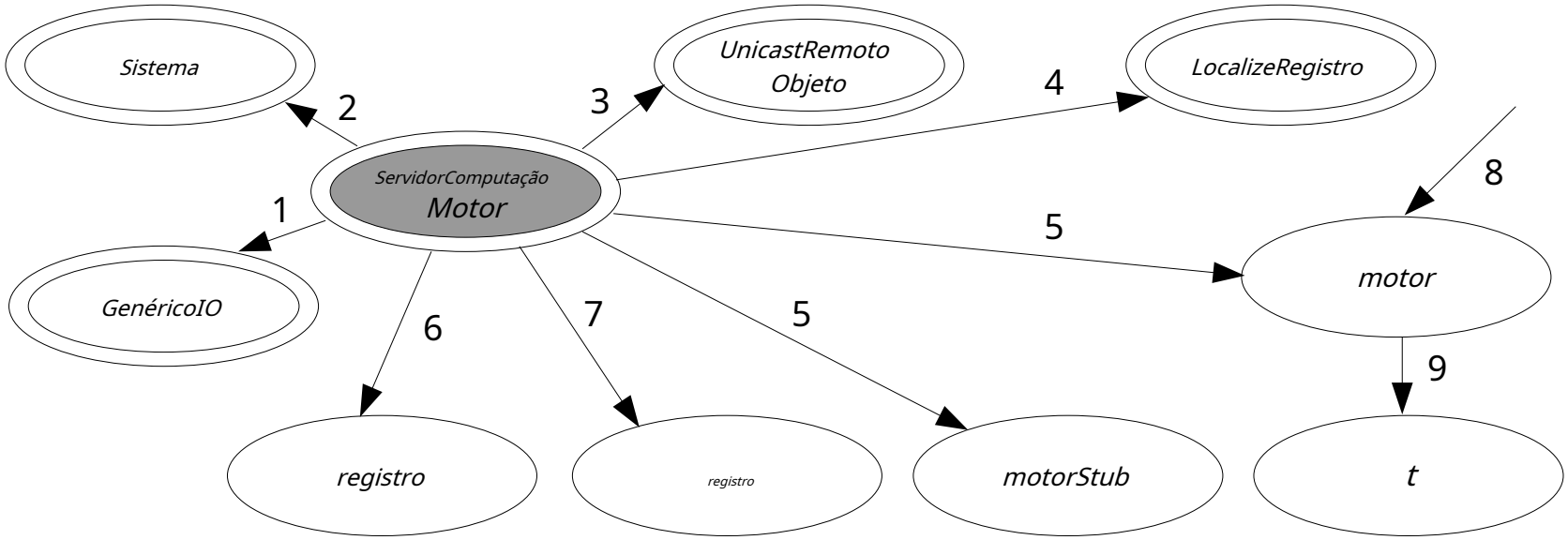


1 – readlnInt, readlnString, writeString, writelnString 2 – getSecurityManager, setSecurityManager
3 – exportarobjeto
4 – obterRegistro

5 – instanciar
6 – instanciar, religar 7 – ligar, desvincular, religar

Diagramas de interação - 2

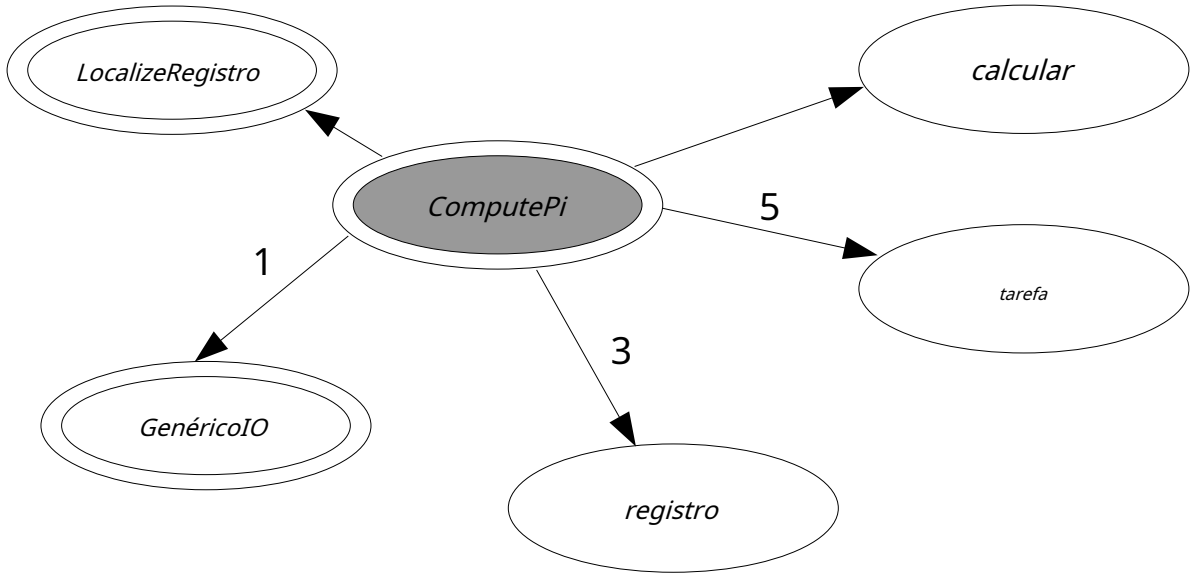
Objeto remoto para execução local de código migrado sob controle remoto



1 – readlnInt, readlnString, writeString, writelnString 2 – getSecurityManager, setSecurityManager
3 – exportarobjeto
4 – obterRegistro
5 – instanciar
6 – instanciar, localizar 7 – instanciar, vincular 8 – executarTask
9 – executar

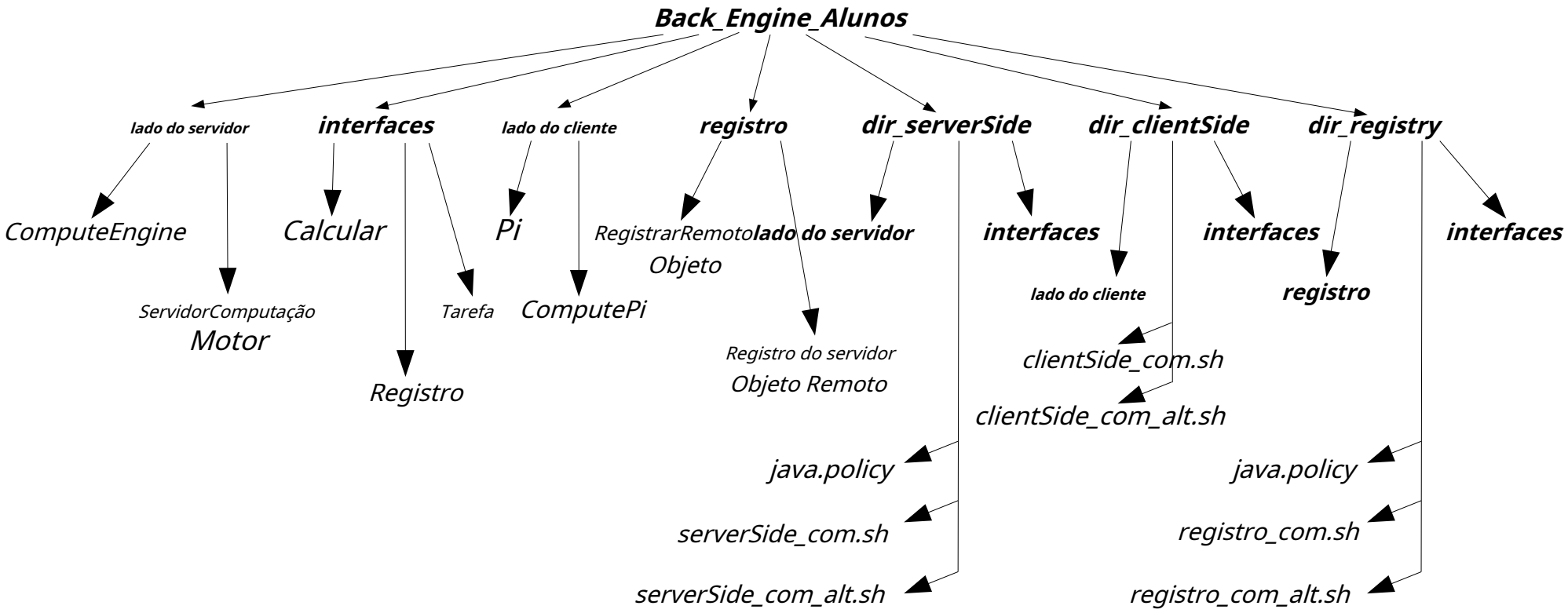
Diagramas de interação - 3

Lado do cliente



1 – readlnInt, readlnString, writeString, writelnString 2 – getRegistry
3 – instanciar, pesquisar
4 – instanciar, executeTask
5 – instanciar

Organização do pacote BackEngine - 1



Organização do pacote BackEngine - 2

Região para desenvolvimento de código da aplicação

***lado do servidor**–diretório com o código do servidor principal*

ComputeEngine[.Java]–*objeto remoto que fornece execução local de código migrado sob controle remoto*

ServidorComputeEngine[.Java]–*instanciação e registro do objeto remoto*(*serviço prestado*)

***interfaces**–diretório com as interfaces para os objetos remotos*

Calcular[.Java]–*interface para acesso ao objeto que fornece execução local do código migrado sob controle remoto*

Tarefa[.Java]–*interface para execução local de código migrado*

Registro[.Java]–*interface para acesso ao objeto que fornece suporte para registro de objetos remotos*

***lado do cliente**–direcione o código do cliente*

Pi[.Java]–*código a ser migrado e que será executado remotamente sob controle local* ***ComputePi***[.Java]–*acesso ao objeto remoto que fornece execução local do código migrado sob controle remoto*

ao controle

***registro**–diretório com o código do servidor auxiliar para suportar o registro de objetos remotos*

RegistrarRemoteObject[.Java]–*objeto remoto para suportar o registro de objetos remotos*

ServerRegisterRemoteObject[.Java]–*instanciação e registro do objeto remoto*(*serviço prestado*)

Organização do pacote BackEngine - 3

Região para implantação do aplicativo

***dir_serverSide**–diretório para executar o código do servidor principal*

***lado do servidor**–contém**ServidorComputeEngine**[.aula]e**ComputeEngine**[.aula]*
***interfaces**–contém**Registro**[.aula],**Calcular**[.classe] e**Tarefa**[.aula] **java.policy**–arquivo especificando a política de segurança local*
***serverSide_com.sh**–script de shell para o código em execução(os tipos de dados são localizados com a ajuda do servidor http)*
***serverSide_com_alt.sh**–script de shell para o código em execução(tipos de dados estão localizados no sistema de arquivos)*

***dir_clientSide**–diretório para executar o código do cliente*

***lado do cliente**–contém**ComputePi**[.aula]e**Pi**[.aula]*
***interfaces**–contém**Calcular**[.aula]e**Tarefa**[.aula]*
***clientSide_com.sh**–script de shell para o código em execução(os tipos de dados são localizados com a ajuda do servidor http)*
***clientSide_com_alt.sh**–script de shell para o código em execução(tipos de dados estão localizados no sistema de arquivos)*

***dir_registry**–diretório para executar o código do servidor auxiliar*

***registro**–contém**ServerRegisterRemoteObject**[.aula]e**RegistrarRemoteObject**[.aula] **interfaces**–contém**Registro**[.aula] **java.policy**–arquivo especificando a política de segurança local*

***registro_com.sh**–script de shell para o código em execução(os tipos de dados são localizados com a ajuda do servidor http)*
***registro_com_alt.sh**–script de shell para o código em execução(tipos de dados estão localizados no sistema de arquivos)*

Política de segurança local

- existe uma única regra de segurança em Java: *tudo é proibido, a menos que seja explicitamente permitido*

```
conceder
{ permissão java.util.PropertyPermission "user.dir", "ler";
    permissão java.net.SocketPermission "*:1024-65535",
                                                “ouvir, resolver, conectar, aceitar”;
    permissão java.net.SocketPermission "*:80", "conectar"; permissão
    java.io.FilePermission "/-", "ler, escrever";
};
```

Política de segurança local

- existe uma única regra de segurança em Java: *tudo é proibido, a menos que seja explicitamente permitido*

```
conceder
{ permissão java.util.PropertyPermission "user.dir", "ler";
    permissão java.net.SocketPermission "*:1024-65535",
                                                “ouvir, resolver, conectar, aceitar”;
    permissão java.net.SocketPermission "*:80", "conectar"; permissão
    java.io.FilePermission "/-", "ler, escrever";
};
```

Construir e implantar - 1

- Crie um*concha*janela
- posição dentro do diretórioBack_Engine_Alunos
- substituir embuildAndDeploy.shtodas as instâncias deruibpelo seu login
- faça o mesmo nos arquivos , ntSide_com_alt.sh, serverSide_com_al , , gistry_com_alt.sh, registro_com.sh
- execute o script de shellmas

[ruib@ruib-laptop Back_Engine_alunos]$***senha*** /home/ruib/Teaching/SD/exemplos demonstrativos/ BackEngine/Back_Engine_alunos [ruib@ruib-laptop Back_Engine_Alunos]$**./buildAndDeploy.sh**
Compilando código fonte.

Distribuir código intermediário para os diferentes ambientes de execução. Compactando ambientes de execução.
Implantando e descompactando ambientes de execução.
[ruib@ruib-laptop Back_Engine_alunos]$

Construir e implantar - 2

```
[ruib@ruib-laptop Back_Engine_alunos]$gato buildAndDeploy.sh echo
"Compilando código fonte."
javac interfaces/*.java Registry/*.java serverSide/*.java clientSide/*.java echo "Distribuindo código
intermediário para os diferentes ambientes de execução." cp interfaces/Register.class dir_registry/
interfaces/
cp registro/*.class dir_registry/registry/ cp interfaces/*.class
dir_serverSide/interfaces/ cp serverSide/*.class dir_serverSide/
serverSide/
cp interfaces/Compute.class interfaces/Task.class dir_clientSide/interfaces/ cp clientSide/*.class
dir_clientSide/clientSide/
mkdir -p /home/ruib/Público/classes
mkdir -p /home/ruib/Public/classes/interfaces mkdir -p /
home/ruib/Public/classes/clientSide cp interfaces/*.class
hom                                  e/ruib/Publi          c/aula/          interfaces/
cp clientSide/Pi.class /home/ruib/Public/classes/clientSide/                          e/
echo "Compactando ambientes de execução."
rm -f dir_registry.zip dir_serverSide.zip dir_clientSide.zip zip -rq dir_registry.zip
dir_registry
zip -rq dir_serverSide.zip dir_serverSide zip -rq
dir_clientSide.zip dir_clientSide
echo "Implantando e descompactando ambientes de execução." cp
set_rmiregistry_alt.sh /home/ruib
cp set_rmiregistry.sh /home/ruib mkdir -p /
home/ruib/test/BackEngine rm -rf /home/ruib/
test/BackEngine/*
cp dir_registry.zip dir_serverSide.zip dir_clientSide.zip /home/ruib/test/BackEngine cd /home/ruib/test/
BackEngine
descompactar -q dir_registry.zip descompactar -q
dir_serverSide.zip descompactar -q
dir_clientSide.zip [ruib@ruib-laptop
Back_Engine_alunos]$
```

Executando o aplicativo

- a *http* servidor é não necessário para executar a versão onde os tipos de dados estão localizados por meio do sistema de arquivos
- são necessárias quatro janelas de shell
  - janela 1: diretório base, execute alt].sh
  - janela 2: dir_reg , executar t].sh
  - janela 3: dir_ser , executar [_alt].sh
  - janela 4: dir_clientSide, executar clientSide_com[_alt].sh

Leitura sugerida

- *On-line* documentação de suporte para ambiente de desenvolvimento de programas Java da Oracle (Java Platform Standard Edition 8)