Compute engine generici in RMI

Esempio: Calcolo del prodotto scalare

- Un unico server offre il servizio di calcolo del prodotto scalare tra vettori di interi
- Un client richiede al server il calcolo del prodotto scalare

Esempio: Calcolo del prodotto scalare
Interfaccia del servizio

```
import java.rmi.*;

public interface ScalarProductService extends Remote
{
   public int scalarProduct(int[] v1, int[] v2) throws
   RemoteException;
}
```

Esempio: Calcolo del prodotto scalare
Implementazione del servizio

```
import java.rmi.*;
import java.rmi.server.*;

public class ScalarProductServiceImpl
  extends UnicastRemoteObject
  implements ScalarProductService
{    public ScalarProductServiceImpl()
       throws RemoteException
     {    super();
     }

     public int scalarProduct(int[] v1, int[] v2)
       throws RemoteException
     {    int ret=0;
          for(int i=0;i<v1.length;i++)
               ret+=v1[i]*v2[i];
          return ret;
     }
}
```

Esempio: Calcolo del prodotto scalare

Server

```
import java.rmi.*;

public class ScalarProductServer
{    public static void main(String args[])
     {    try
          {    ScalarProductServiceImpl spsi =
                    new ScalarProductServiceImpl();
               Naming.rebind("scalarProductService", spsi);
               System.out.println(
                    "ScalarProductServer ready.");
          }
          catch (Exception e)
          {    System.out.println(e);
          }
   }
}
```

Esempio: Calcolo del prodotto scalare

Client

```
import java.rmi.*;
import java.util.*;

public class SimpleScalarProductClient
{    public static void main(String args[])
     {    int[] v1=new int[100];
          int[] v2=new int[100];
          for(int i=0;i<=99;i++)
          {    v1[i]=i+1;
               v2[i]=1;
          }
          int result=0;

          [continua…]
```

Esempio: Calcolo del prodotto scalare

Client

```
        try
        {    ScalarProductService spe =
                  (ScalarProductService)Naming.lookup(
                  "rmi://localhost:1099/
                  scalarProductService");
             result=spe.scalarProduct(v1,v2);
        }
        catch (Exception e)
        {    System.out.println(e);
        }

        System.out.println(result);
    }
}
```

Compute engine generico

- Ogni server che offre il servizio di calcolo generico implementa l'interfaccia ComputeEngine
- ComputeEngine contiene un metodo execute per l'esecuzione, con un certo set di parametri, di metodi specificati mediante un'interfaccia Job
- Le classi che implementano l'interfaccia Job ne particolarizzano il metodo run
- I client passano ai server istanze della classe Job e set di parametri (come java.lang.Object)
- Il metodo Job.run è eseguito sui server

Compute Engine
Job

```
import java.io.*;

public interface Job extends Serializable
{     public Object run(Object parameters);
}
```

Compute Engine
Interfaccia

```
import java.rmi.*;

public interface ComputeEngine extends Remote
{     public Object execute(Job j, Object parameters)
          throws RemoteException;
}
```

Compute Engine
Implementazione

```
import java.rmi.*;
import java.rmi.server.*;

public class ComputeEngineImpl
  extends UnicastRemoteObject
  implements ComputeEngine
{    public ComputeEngineImpl() throws RemoteException
     {    super();
     }

     public Object execute(Job j, Object parameters)
       throws RemoteException
     {    return j.run(parameters);
     }
}
```

Compute Engine
Server

```
import java.rmi.*;

public class ComputeEngineServer
{    public static void main(String args[])
     {    try
          {    ComputeEngineImpl cei =
                    new ComputeEngineImpl();
               Naming.rebind("computeEngine", cei);
               System.out.println(
                    "ComputeEngineServer ready.");
          }
          catch (Exception e)
          {    System.out.println(e);
          }
   }
}
```

Compute engine generico

- I metodi remoti sono multithreaded per default
- Le loro invocazioni sono invece sincrone per i client
- Un client che suddivide il calcolo su più server deve perciò lanciare un thread per ogni invocazione di ComputeEngine.execute
- Anche la struttura di tali thread può essere resa generica

Compute Engine

Client thread

```
import java.rmi.*;

class ComputeThread extends Thread
{    private Job job;
     private Object parameters;
     private String host;
     private Object result;

     public ComputeThread(Job job,Object parameters,
          String host)
     {    this.job=job;
          this.parameters=parameters;
          this.host=host;
          result=null;
     }
     [continua…]
```

Compute Engine

Client thread

```
    public void run()
    {    try
         {    ComputeEngine ce = (ComputeEngine)Naming.lookup(
                   "rmi://"+host+":1099/
                   computeEngine");
              result=ce.execute(job,parameters);
              System.out.println("ComputeThread result="+
                   result);
         }
         catch (Exception e)
         {    System.out.println(e);
         }
    }

    public Object getResult()
    {    return result;
    }
}
```

Esempio: Calcolo del prodotto scalare /2

- Un unico client utilizza un unico server generico
- Il client implementa Job creando ScalarProductJob, il cui metodo run effettua il prodotto scalare

Esempio: Calcolo del prodotto scalare /2

Job di calcolo

```
import java.util.*;

public class ScalarProductJob implements Job
{    public Object run(Object parameters)
     {    Vector v=(Vector)parameters;
          Vector v1=(Vector)v.get(0);
          Vector v2=(Vector)v.get(1);
          int ret=0;
          for(int i=0;i<v1.size();i++)
               ret+=((Integer)v1.get(i)).intValue()*
                    ((Integer)v2.get(i)).intValue();
          return new Integer(ret);
     }
}
```

Esempio: Calcolo del prodotto scalare /2

Client di un unico engine

```
import java.rmi.*;
import java.util.*;

public class SimpleScalarProductCEClient
{    public static void main(String args[])
     {    Vector v1=new Vector();
          Vector v2=new Vector();
          for(int i=1;i<=100;i++)
          {    v1.add(new Integer(i));
               v2.add(new Integer(1));
          }
          int result=0;

          [continua…]
```

Esempio: Calcolo del prodotto scalare /2

Client di un unico engine

```
        try
        {    ComputeEngine ce =
                  (ComputeEngine)Naming.lookup(
                  "rmi://localhost:1099/computeEngine");
             Vector vpar=new Vector();
             vpar.add(v1);
             vpar.add(v2);
             Integer res=(Integer)ce.execute(
                  new ScalarProductJob(),vpar);
             result=res.intValue();
        }
        catch (Exception e)
        {    System.out.println(e);
        }

        System.out.println(result);
    }
}
```

Esempio: Calcolo del prodotto scalare /3

- Un unico client utilizza 3 server generici
- Il client utilizza ScalarProductJob e suddivide i vettori in 3 sottovettori, assegnandone uno ad ogni server
- Il client lancia inoltre 3 thread per seguire il calcolo sui 3 sottovettori

Esempio: Calcolo del prodotto scalare /3

Client di engine multipli

```
import java.rmi.*;
import java.util.*;

public class ScalarProductCEClient
{    public static void main(String args[])
     {    Vector v1=new Vector();
          Vector v2=new Vector();
          for(int i=1;i<=100;i++)
          {    v1.add(new Integer(i));
               v2.add(new Integer(1));
          }

          Vector vpar1=new Vector();
          vpar1.add(extract(v1,0,32));
          vpar1.add(extract(v2,0,32));
          ComputeThread ct1=new ComputeThread(
               new ScalarProductJob(),vpar1,"localhost");
          [continua…]
```

Esempio: Calcolo del prodotto scalare /3

Client di engine multipli

```
          Vector vpar2=new Vector();
          vpar2.add(extract(v1,33,66));
          vpar2.add(extract(v2,33,66));
          ComputeThread ct2=new ComputeThread(
               new ScalarProductJob(),vpar2,"localhost");

          Vector vpar3=new Vector();
          vpar3.add(extract(v1,67,99));
          vpar3.add(extract(v2,67,99));
          ComputeThread ct3=new ComputeThread(
               new ScalarProductJob(),vpar3,"localhost");

          ct1.start();
          ct2.start();
          ct3.start();
          [continua…]
```

Esempio: Calcolo del prodotto scalare /3

Client di engine multipli

```
        try
        {    ct1.join();
             ct2.join();
             ct3.join();
        }
        catch (InterruptedException e)
        {    System.out.println(e);
        }
        System.out.println(
             ((Integer)ct1.getResult()).intValue()+
             ((Integer)ct2.getResult()).intValue()+
             ((Integer)ct3.getResult()).intValue());
    }
    [continua…]
```

Esempio: Calcolo del prodotto scalare /3

Client di engine multipli

```
    private static Vector extract(Vector v,
         int from,int to)
    {    Vector ret=new Vector();
         for(int i=from;i<=to;i++)
              ret.add(v.get(i));
         return ret;
    }
}
```

Esempio: Calcolo del prodotto tra matrici

- Si vuole calcolare il prodotto tra matrici di dimensioni $m \times n$ ed $n \times r$ (nell'esempio, 2×3 e 3×2)
- Un unico client utilizza $m * r$ server generici
- Il client utilizza sempre ScalarProductJob e assegna una coppia riga/colonna ad ogni server
- Il client lancia inoltre $m * r$ thread per seguire il calcolo di ognuno degli elementi della matrice risultato

Esempio: Calcolo del prodotto tra matrici

```
import java.rmi.*;
import java.util.*;

public class MatrixProductCEClient
{    public static void main(String args[])
     {    int[][] mat1=new int[2][3];
          int[][] mat2=new int[3][2];

          int v=0;
          for(int i=0;i<mat1.length;i++)
               for(int j=0;j<mat1[0].length;j++)
                    mat1[i][j]=++v;
          for(int i=0;i<mat2.length;i++)
               for(int j=0;j<mat2[0].length;j++)
                    mat2[i][j]=(i==j?1:0);
          [continua…]
```

Esempio: Calcolo del prodotto tra matrici

```
Vector threads=new Vector();

for(int c=0;c<mat2[0].length;c++)
     for(int r=0;r<mat1.length;r++)
     {    Vector vpar=new Vector();
          vpar.add(extractColumn(mat2,c));
          vpar.add(extractRow(mat1,r));
          ComputeThread ct=new ComputeThread(
               new ScalarProductJob(),vpar,"localhost");
          threads.add(ct);
     }

for(int i=0;i<threads.size();i++)
          ((Thread)threads.get(i)).start();
[continua…]
```

Esempio: Calcolo del prodotto tra matrici

```
try
{    for(int i=0;i<threads.size();i++)
          ((Thread)threads.get(i)).join();
}
catch (InterruptedException e)
{    System.out.println(e);
}

int[][] result=
     new int[mat1.length][mat2[0].length];
for(int i=0;i<threads.size();i++)
{    int r=((Integer)((ComputeThread)threads
          .get(i)).getResult()).intValue();
     result[i%mat1.length][i/mat1.length]=r;
}
[continua…]
```

Esempio: Calcolo del prodotto tra matrici

```
        for(int i=0;i<result.length;i++)
        {   System.out.println();
            for(int j=0;j<result[0].length;j++)
                System.out.print(result[i][j]+" ");
        }

    }
    [continua…]
```

Esempio: Calcolo del prodotto tra matrici

```
    private static Vector extractRow(int[][] mat,
            int rowIndex)
    {   Vector ret=new Vector();
        for(int c=0;c<mat[0].length;c++)
            ret.add(new Integer(mat[rowIndex][c]));
        return ret;
    }

    private static Vector extractColumn(int[][] mat,
            int columnIndex)
    {   Vector ret=new Vector();
        for(int r=0;r<mat.length;r++)
            ret.add(new Integer(mat[r][columnIndex]));
        return ret;
    }
}
```