Sistemas Distribuídos

Interprocess Communication and Synchronization

Remote Objects - 2

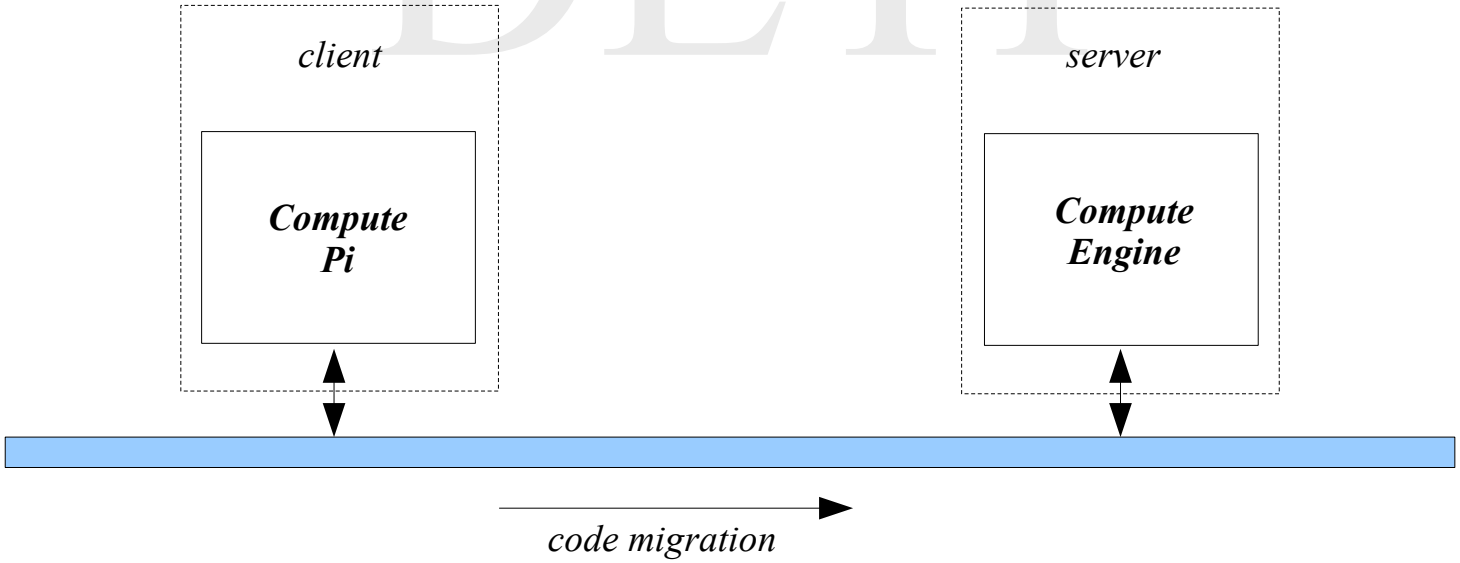
António Rui Borges

Summary

- *Characterization of the problem*
- *Code description*
- *Interaction diagrams*
- *Organization of the package BackEngine*
- *Local security policy*
- *Build and deploy*
- *Running the application*
- *Suggested reading*

Characterization of the problem - 1

- it is an example, adapted from the tutorial of Sun about *RMI*, to illustrate how code can be transferred between Java virtual machines, located on the same or on different hardware platforms, and be run in a JVM other than the one where it was formerly stored

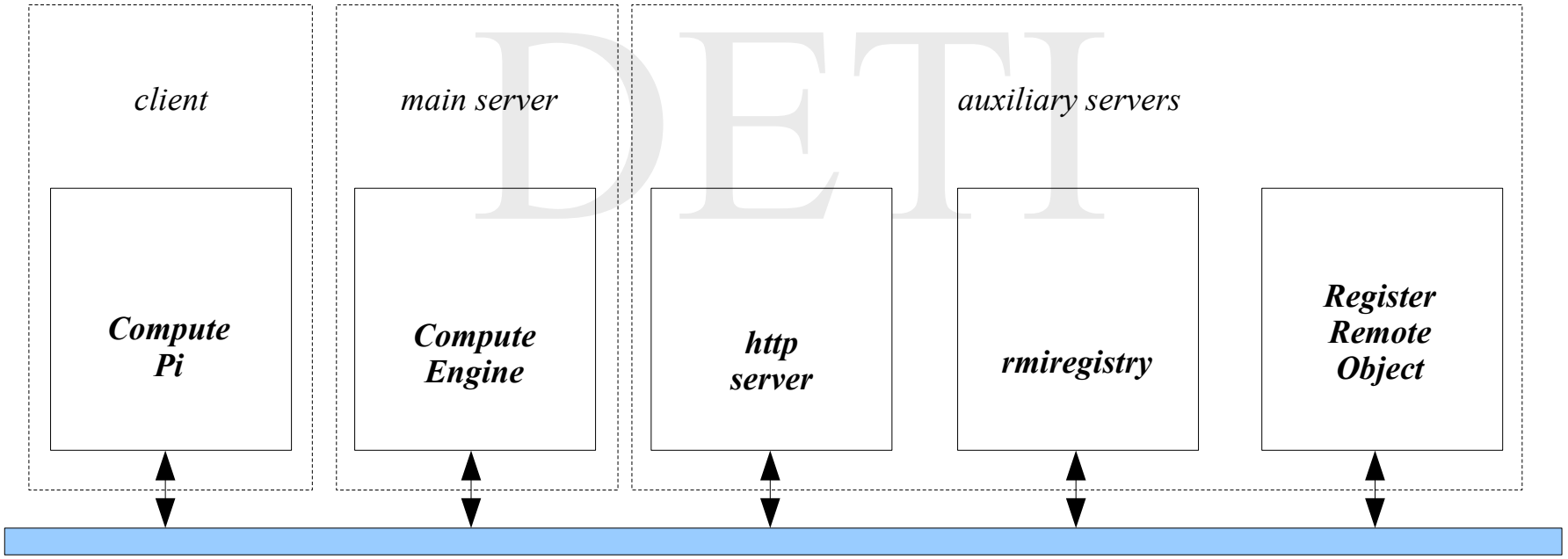


Characterization of the problem - 2

- two main classes of entities are considered
  - a *server object*, of type `ComputeEngine`, which provides a service for local execution of migrated code under remote control
  - several *client objects* to handle the local execution of migrated code under remote control (interfaces `Compute` and `Task`); in the present case, another object, of type `ComputePi`, transfers an object of type `Pi` (computation of π with a variable number of decimals) for remote execution
- and three auxiliary entities
  - a naming service for registering remote objects (`rmiregistry`)
  - a *server object*, of type `RegisterRemoteObject`, which provides support to register remote objects (those located in JVMs residing in hardware platforms other than the one the naming service is located)
  - a *http server* to assist the dynamic downloading of the data types used in remote calls.

Characterization of the problem - 3

Operational setup



Register Remote Object base thread - 1

```
public class ServerRegisterRemoteObject
{
   public static void main(String[] args)
   {
    /* get location of the registering service */

     String rmiRegHostName;
     int rmiRegPortNumb;

     GenericIO.writeString ("Name of the processing node where the registering service is located? ");
     rmiRegHostName = GenericIO.readlnString ();
     GenericIO.writeString ("Port number where the registering service is listening to? ");
     rmiRegPortNumb = GenericIO.readlnInt ();

    /* create and install the security manager */

     if (System.getSecurityManager () == null)
        System.setSecurityManager (new SecurityManager ());
     GenericIO.writelnString ("Security manager was installed!");

    /* instantiate a registration remote object and generate a stub for it */

     RegisterRemoteObject regEngine = new RegisterRemoteObject (rmiRegHostName, rmiRegPortNumb);
     Register regEngineStub = null;
     int listeningPort = 22001;                          /* it should be set accordingly in each case */

     try
     { regEngineStub = (Register) UnicastRemoteObject.exportObject (regEngine, listeningPort);
     }
     catch (RemoteException e)
     { GenericIO.writelnString ("RegisterRemoteObject stub generation exception: " + e.getMessage ());
       System.exit (1);
     }
     GenericIO.writelnString ("Stub was generated!");
```

Register Remote Object base thread - 2

```
    /* register it with the local registry service */

     String nameEntry = "RegisterHandler";
     Registry registry = null;

     try
     { registry = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
     }
     catch (RemoteException e)
     { GenericIO.writelnString ("RMI registry creation exception: " + e.getMessage ());
       System.exit (1);
     }
     GenericIO.writelnString ("RMI registry was created!");

     try
     { registry.rebind (nameEntry, regEngineStub);
     }
     catch (RemoteException e)
     { GenericIO.writelnString ("RegisterRemoteObject remote exception on registration: " +
                                e.getMessage ());
       System.exit (1);
     }
     GenericIO.writelnString ("RegisterRemoteObject object was registered!");
   }
}
```

Register Remote Object and related interfaces - 1

```
public interface Register extends Remote
{
   public void bind (String name, Remote ref) throws RemoteException, AlreadyBoundException;

   public void unbind (String name) throws RemoteException, NotBoundException;

   public void rebind (String name, Remote ref) throws RemoteException;
}
```

Register Remote Object and related interfaces - 2

```
public class RegisterRemoteObject implements Register
{
   private String rmiRegHostName;
   private int rmiRegPortNumb = 1099;

   public RegisterRemoteObject (String rmiRegHostName, int rmiRegPortNumb)
   {
     if ((rmiRegHostName == null) || ("".equals (rmiRegHostName)))
        throw new NullPointerException ("RegisterRemoteObject: null pointer parameter on instantiation!");
     this.rmiRegHostName = rmiRegHostName;
     if ((rmiRegPortNumb >= 4000) && (rmiRegPortNumb <= 65535))
        this.rmiRegPortNumb = rmiRegPortNumb;
   }

   public void bind (String name, Remote ref) throws RemoteException, AlreadyBoundException
   {
     Registry registry;

     if ((name == null) || (ref == null))
        throw new NullPointerException ("RegisterRemoteObject: null pointer parameter(s) on on bind!");
     registry = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
     registry.bind (name, ref);
   }
```

Register Remote Object and related interfaces - 3

```
    public void unbind (String name) throws RemoteException, NotBoundException
    {
      Registry registry;

      if ((name == null))
         throw new NullPointerException ("RegisterRemoteObject: null pointer parameter(s) on unbind!");
      registry = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
      registry.unbind (name);
    }
    public void rebind (String name, Remote ref) throws RemoteException
    {
      Registry registry;

      if ((name == null) || (ref == null))
         throw new NullPointerException ("RegisterRemoteObject: null pointer parameter(s) on rebind!");
      registry = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
      registry.rebind (name, ref);
    }
}
```

Compute Engine base thread - 1

```
public class ServerComputeEngine
{
   public static void main(String[] args)
   {
    /* get location of the registry service */

     String rmiRegHostName;
     int rmiRegPortNumb;

     GenericIO.writeString ("Name of the processing node where the registering service is located? ");
     rmiRegHostName = GenericIO.readlnString ();
     GenericIO.writeString ("Port number where the registering service is listening to? ");
     rmiRegPortNumb = GenericIO.readlnInt ();

    /* create and install the security manager */

     if (System.getSecurityManager () == null)
        System.setSecurityManager (new SecurityManager ());
     GenericIO.writelnString ("Security manager was installed!");

    /* instantiate a remote object that runs mobile code and generate a stub for it */

     ComputeEngine engine = new ComputeEngine ();
     Compute engineStub = null;
     int listeningPort = 22002;                          /* it should be set accordingly in each case */

     try
     { engineStub = (Compute) UnicastRemoteObject.exportObject (engine, listeningPort);
     }
     catch (RemoteException e)
     { GenericIO.writelnString ("ComputeEngine stub generation exception: " + e.getMessage ());
       e.printStackTrace ();
       System.exit (1);
     }
     GenericIO.writelnString ("Stub was generated!")
```

Compute Engine base thread - 2

```
/* register it with the general registry service */

 String nameEntryBase = "RegisterHandler";
 String nameEntryObject = "Compute";
 Registry registry = null;
 Register reg = null;

 try
 { registry = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
 }
 catch (RemoteException e)
 { GenericIO.writelnString ("RMI registry creation exception: " + e.getMessage ());
   e.printStackTrace ();
   System.exit (1);
 }
 GenericIO.writelnString ("RMI registry was created!");

 try
 { reg = (Register) registry.lookup (nameEntryBase);
 }
 catch (RemoteException e)
 { GenericIO.writelnString ("RegisterRemoteObject lookup exception: " + e.getMessage ());
   e.printStackTrace ();
   System.exit (1);
 }
 catch (NotBoundException e)
 { GenericIO.writelnString ("RegisterRemoteObject not bound exception: " + e.getMessage ());
   e.printStackTrace ();
   System.exit (1);
 }
```

Compute Engine base thread - 3

```
      try
      { reg.bind (nameEntryObject, engineStub);
      }
      catch (RemoteException e)
      { GenericIO.writelnString ("ComputeEngine registration exception: " + e.getMessage ());
        e.printStackTrace ();
        System.exit (1);
      }
      catch (AlreadyBoundException e)
      { GenericIO.writelnString ("ComputeEngine already bound exception: " + e.getMessage ());
        e.printStackTrace ();
        System.exit (1);
      }
      GenericIO.writelnString ("ComputeEngine object was registered!");
  }
}
```

Compute Engine remote object and related interfaces

```
public interface Compute extends Remote
{
   Object executeTask (Task t) throws RemoteException;
}

public interface Task extends Serializable
{
   public static final long serialVersionUID = 2021L;

   Object execute();
}

public class ComputeEngine implements Compute
{
   public Object executeTask (Task t)
   {
     return t.execute ();
   }
}
```

Compute Pi thread base - 1

```
public class ComputePi
{
   public static void main(String args[])
   {
    /* get location of the generic registry service */

     String rmiRegHostName;
     int rmiRegPortNumb;

     GenericIO.writeString ("Name of the processing node where the registering service is located? ");
     rmiRegHostName = GenericIO.readlnString ();
     GenericIO.writeString ("Port number where the registering service is listening to? ");
     rmiRegPortNumb = GenericIO.readlnInt ();

    /* look for the remote object by name in the remote host registry */

     String nameEntry = "Compute";
     Compute comp = null;
     Registry registry = null;

     try
     { registry = LocateRegistry.getRegistry (rmiRegHostName, rmiRegPortNumb);
     }
     catch (RemoteException e)
     { GenericIO.writelnString ("RMI registry creation exception: " + e.getMessage ());
       e.printStackTrace ();
       System.exit (1);
     }
```

Compute Pi thread base - 2

```
try
{ comp = (Compute) registry.lookup (nameEntry);
}
catch (RemoteException e)
{ GenericIO.writelnString ("ComputePi look up exception: " + e.getMessage ());
  e.printStackTrace ();
  System.exit (1);
}
catch (NotBoundException e)
{ GenericIO.writelnString ("ComputePi not bound exception: " + e.getMessage ());
  e.printStackTrace ();
  System.exit (1);
}

/* instantiate the mobile code object to be run remotely */

Pi task = null;
BigDecimal pi = null;

try
{ task = new Pi (Integer.parseInt (args[0]));
}
catch (NumberFormatException e)
{ GenericIO.writelnString ("Pi instantiation exception: " + e.getMessage ());
  e.printStackTrace ();
  System.exit (1);
}
```

Compute Pi thread base - 3

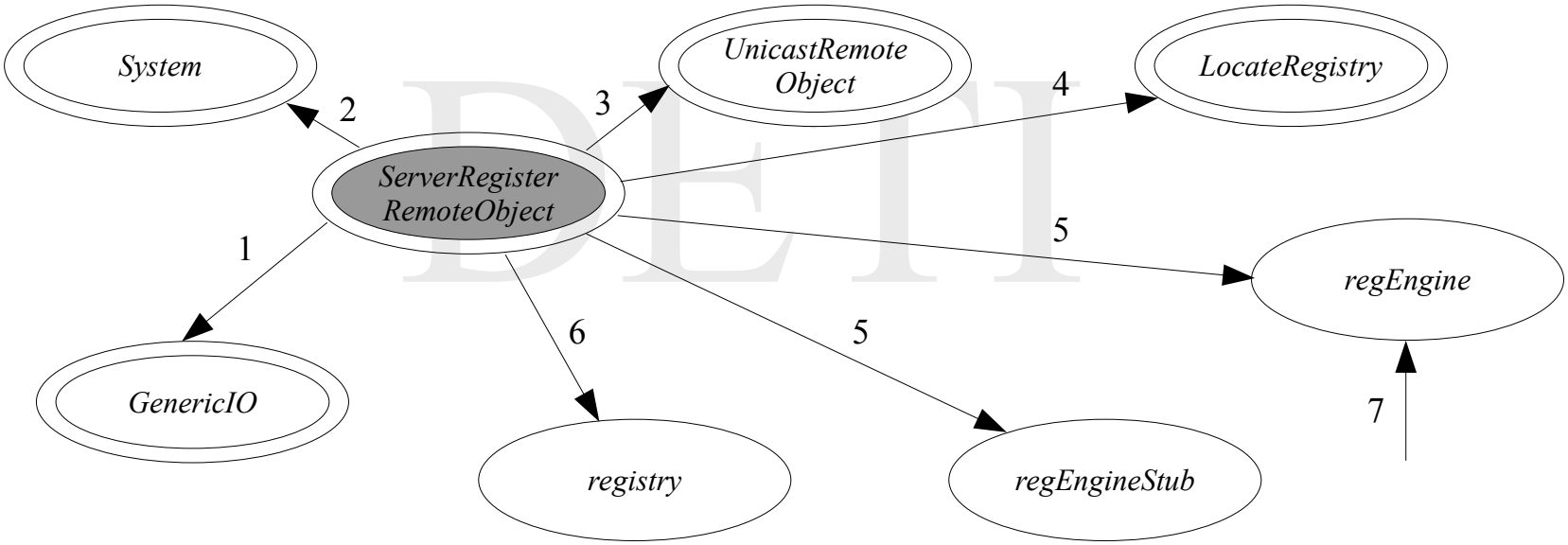
```
    /* invoke the remote method (run the code at a ComputeEngine remote object) */

     try
     { pi = (BigDecimal) (comp.executeTask (task));
     }
     catch (RemoteException e)
     { GenericIO.writelnString ("ComputePi remote invocation exception: " + e.getMessage ());
       e.printStackTrace ();
       System.exit (1);
     }

     /* print the result */
     GenericIO.writelnString (pi.toString ());
  }
}
```

Interaction diagrams - 1

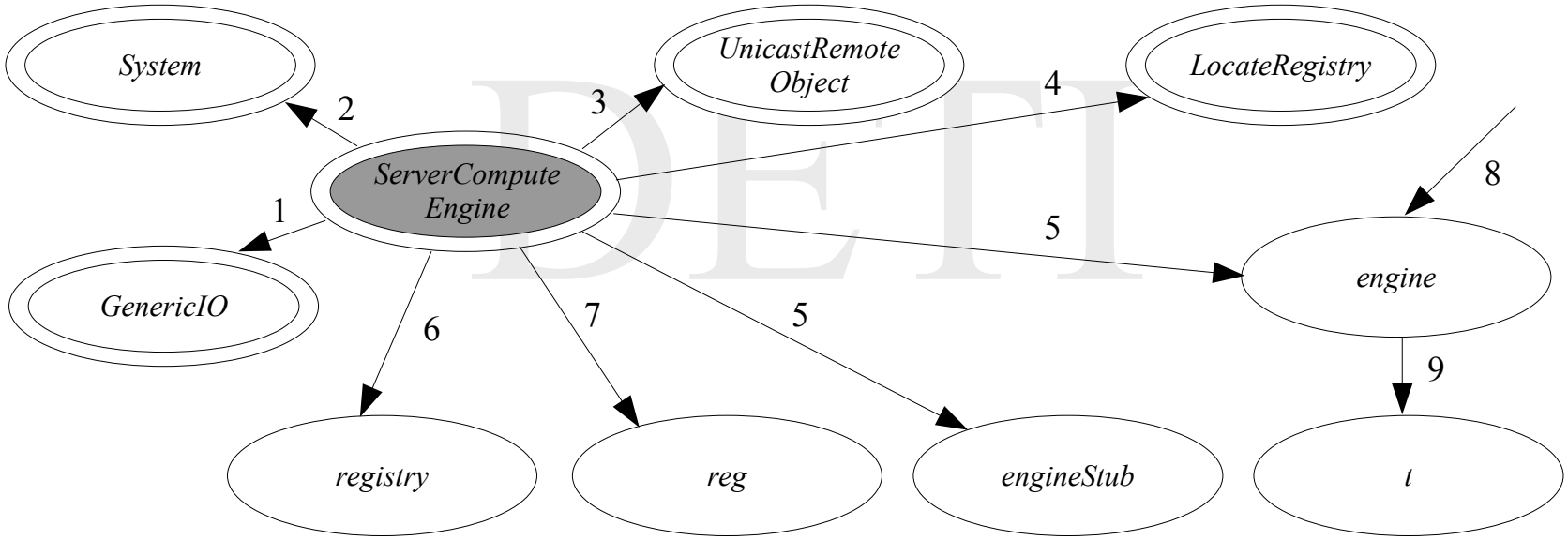
Remote object to support the registering of remote objects



1 – readlnInt, readlnString, writeString, writelnString
2 – getSecurityManager, setSecurityManager
3 – exportObject
4 – getRegistry
5 – instantiate
6 – instantiate, rebind
7 – bind, unbind, rebind

Interaction diagrams - 2

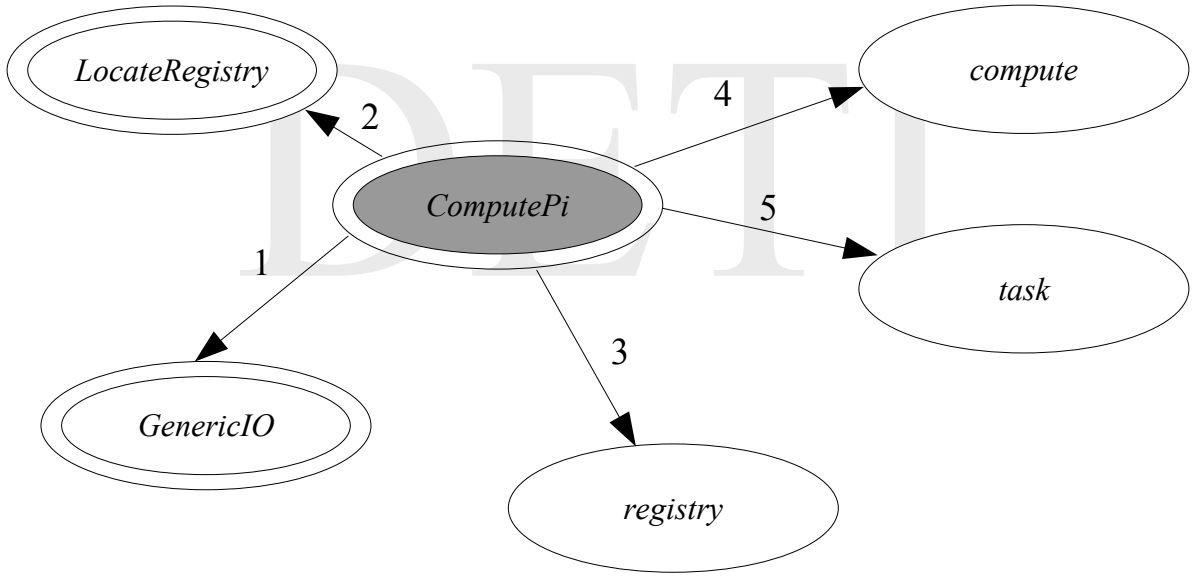
Remote object for local execution of migrated code under remote control



1 – readlnInt, readlnString, writeString, writelnString
2 – getSecurityManager, setSecurityManager
3 – exportObject
4 – getRegistry
5 – instantiate
6 – instantiate, locate
7 – instantiate, bind
8 – executeTask
9 – execute

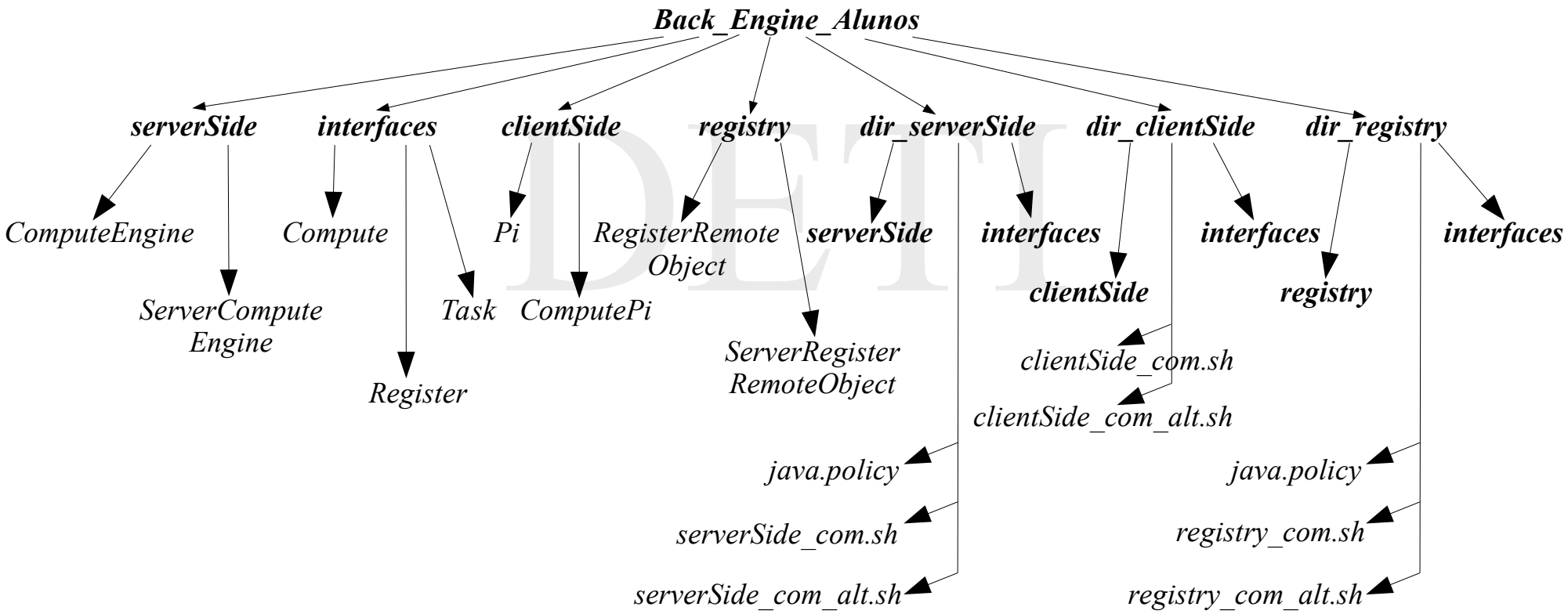
Interaction diagrams - 3

Client side



1 – readlnInt, readlnString, writeString, writelnString
2 – getRegistry
3 – instantiate, lookup
4 – instantiate, executeTask
5 – instantiate

Organization of the package BackEngine - 1



Organization of the package BackEngine - 2

Region for code development of the application

serverSide *– directory with the main server code*

- ***ComputeEngine***[.java] *– remote object which provides local execution of migrated code under remote control*
- ***ServerComputeEngine***[.java] *– instantiation and registering of the remote object* (*service provided*)

interfaces *– directory with the interfaces to the remote objects*

- ***Compute***[.java] *– interface for access to the object that provides local execution of migrated code under remote control*
- ***Task***[.java] *– interface for local execution of migrated code*
- ***Register***[.java] *– interface for access to the object that provides support for registering remote objects*

clientSide *– directory the client code*

- ***Pi***[.java] *– code to be migrated and that it is going to be executed remotely under local control*
- ***ComputePi***[.java] *– access to the remote object which provides local execution of migrated code under remote control*

registry *– directory with the auxiliary server code to support the registering of remote objects*

- ***RegisterRemoteObject***[.java] *– remote object to support the registering of remote objects*
- ***ServerRegisterRemoteObject***[.java] *– instantiation and registering of the remote object* (*service provided*)

Organization of the package BackEngine - 3

Region for deployment of the application

dir_serverSide *– directory for running the main server code*

- ***serverSide*** *– contains* ***ServerComputeEngine***[.class] *and* ***ComputeEngine***[.class]
- ***interfaces*** *– contains* ***Register***[.class], ***Compute***[.class] and ***Task***[.class]
- ***java.policy*** *– file specifying the local security policy*
- ***serverSide_com.sh*** *– shell script for the running code* (*data types are located with help of the http server*)
- ***serverSide_com_alt.sh*** *– shell script for the running code* (*data types are located through the file system*)

dir_clientSide *– directory for running the client code*

- ***clientSide*** *– contains* ***ComputePi***[.class] *and* ***Pi***[.class]
- ***interfaces*** *–contains* ***Compute***[.class] *and* ***Task***[.class]
- ***clientSide_com.sh*** *– shell script for the running code* (*data types are located with help of the http server*)
- ***clientSide_com_alt.sh*** *–shell script for the running code* (*data types are located through the file system*)

dir_registry *– directory for running the auxiliary server code*

- ***registry*** *– contains* ***ServerRegisterRemoteObject***[.class] *and* ***RegisterRemoteObject***[.class]
- ***interfaces*** *– contains* ***Register***[.class]
- ***java.policy*** *– file specifying the local security policy*
- ***registry_com.sh*** *– shell script for the running code* (*data types are located with help of the http server*)
- ***registry_com_alt.sh*** *– shell script for the running code* (*data types are located through the file system*)

Local security policy

- there is a single security rule in Java: *everything is forbidden, unless it explicitly allowed*


```
grant
{ permission java.util.PropertyPermission "user.dir", "read";
  permission java.net.SocketPermission "*:1024-65535",
                                "listen,resolve,connect,accept";
  permission java.net.SocketPermission "*:80", "connect";
  permission java.io.FilePermission "/-", "read,write";
};
```

Local security policy

- there is a single security rule in Java: *everything is forbidden, unless it explicitly allowed*


```
grant
{ permission java.util.PropertyPermission "user.dir", "read";
  permission java.net.SocketPermission "*:1024-65535",
                                "listen,resolve,connect,accept";
  permission java.net.SocketPermission "*:80", "connect";
  permission java.io.FilePermission "/-", "read,write";
};
```

Build and deploy - 1

- create a *shell* window
- position inside the directory `Back_Engine_Alunos`
- replace in `buildAndDeploy.sh` all instances of `ruib` by your login
- do the same in the files `set_rmiregistry.sh`, `clientSide_com_alt.sh`, `serverSide_com_alt.sh`, `serverSide_com.sh`, `registry_com_alt.sh`, `registry_com.sh`
- run the shell script `buildAndDeploy.sh`

```
[ruib@ruib-laptop Back_Engine_alunos]$ pwd
/home/ruib/Teaching/SD/exemplos demonstrativos/BackEngine/Back_Engine_alunos
[ruib@ruib-laptop Back_Engine_Alunos]$ ./buildAndDeploy.sh
Compiling source code.
Distributing intermediate code to the different execution environments.
Compressing execution environments.
Deploying and decompressing execution environments.
[ruib@ruib-laptop Back_Engine_alunos]$
```

Build and deploy - 2

```
[ruib@ruib-laptop Back_Engine_alunos]$ cat buildAndDeploy.sh
echo "Compiling source code."
javac interfaces/*.java registry/*.java serverSide/*.java clientSide/*.java
echo "Distributing intermediate code to the different execution environments."
cp interfaces/Register.class dir_registry/interfaces/
cp registry/*.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/Public/classes
mkdir -p /home/ruib/Public/classes/interfaces
mkdir -p /home/ruib/Public/classes/clientSide
cp interfaces/*.class /home/ruib/Public/classes/interfaces/
cp clientSide/Pi.class /home/ruib/Public/classes/clientSide/
echo "Compressing execution environments."
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 "Deploying and decompressing execution environments."
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
unzip -q dir_registry.zip
unzip -q dir_serverSide.zip
unzip -q dir_clientSide.zip
[ruib@ruib-laptop Back_Engine_alunos]$
```

Running the application

- a *http* server is <u>not</u> required to run the version where the data types are located through the file system
- four shell windows are required
  - window 1: base directory, execute `set_rmiregistry[_alt].sh`
  - window 2: `dir_registry`, execute `registry_com[_alt].sh`
  - window 3: `dir_serverSide`, execute `serverSide_com[_alt].sh`
  - window 4: `dir_clientSide`, execute `clientSide_com[_alt].sh`

Suggested reading

- *On-line* support documentation for Java program developing environment by Oracle (Java Platform Standard Edition 8)