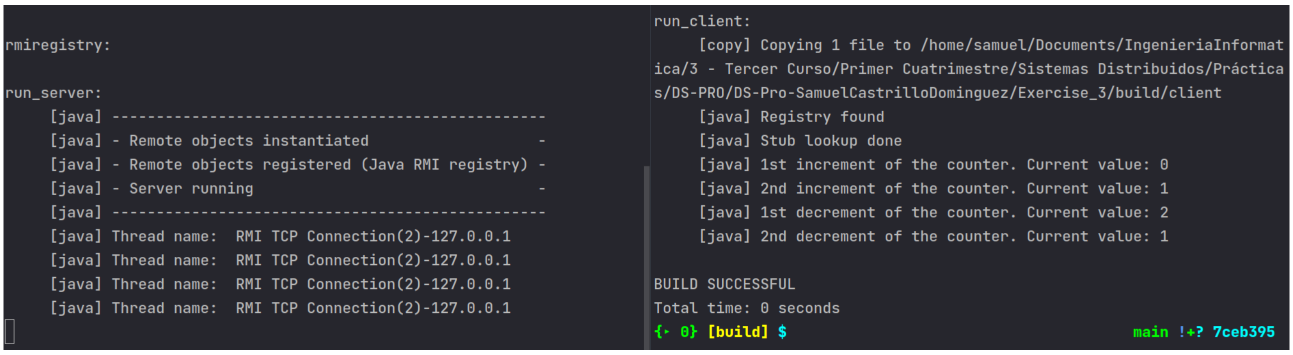
Exercise 3 – RMI C/S

Provide an extensive discussion of your software design and the tests that demonstrate that it functions correctly.

This exercise is divided in 2 folders. One for the client and other for the server. Both contain a main class that is responsible for the following functions:

- Client: First, it locates the *rmiregistry* that must be running at a specified IP address. Then, it looks in that registry for an object that has a name which is known by both the client and the server. The interface of the remote object must also be known by the client in order to execute the remote methods.
- Server: it has a class that implements the mentioned interface. After creating an instance of that class, it is assigned a name (it is stored in the rmiregistry).

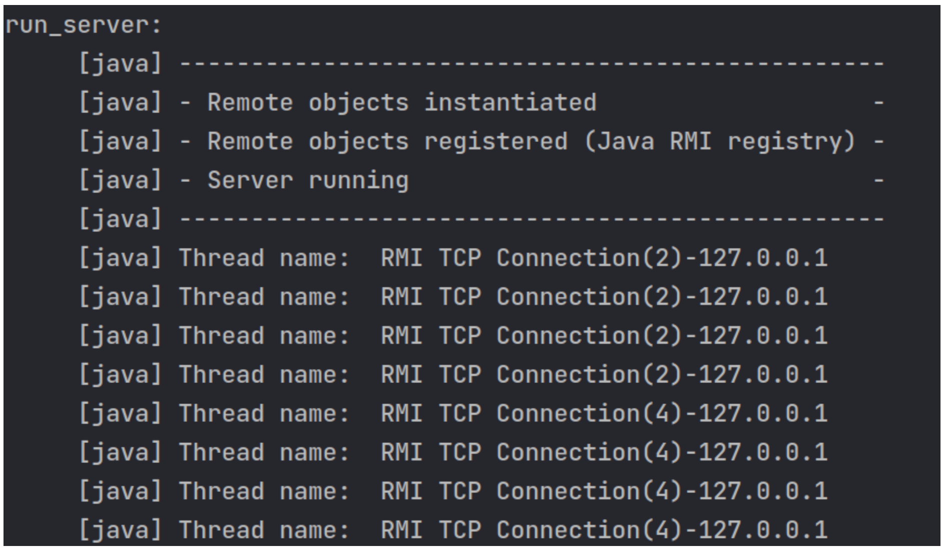


```
rmiregistry:

run_server:
     [java] -------------------------------------------------
     [java] - Remote objects instantiated                   -
     [java] - Remote objects registered (Java RMI registry) -
     [java] - Server running                                -
     [java] -------------------------------------------------
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
```

```
run_client:
     [copy] Copying 1 file to /home/samuel/Documents/IngenieriaInformat
ica/3 - Tercer Curso/Primer Cuatrimestre/Sistemas Distribuidos/Práctica
s/DS-PRO/DS-Pro-SamuelCastrilloDominguez/Exercise_3/build/client
     [java] Registry found
     [java] Stub lookup done
     [java] 1st increment of the counter. Current value: 0
     [java] 2nd increment of the counter. Current value: 1
     [java] 1st decrement of the counter. Current value: 2
     [java] 2nd decrement of the counter. Current value: 1

BUILD SUCCESSFUL
Total time: 0 seconds
{▸ 0} [build] $                                         main !+? 7ceb395
```

Figure 1: Correct execution

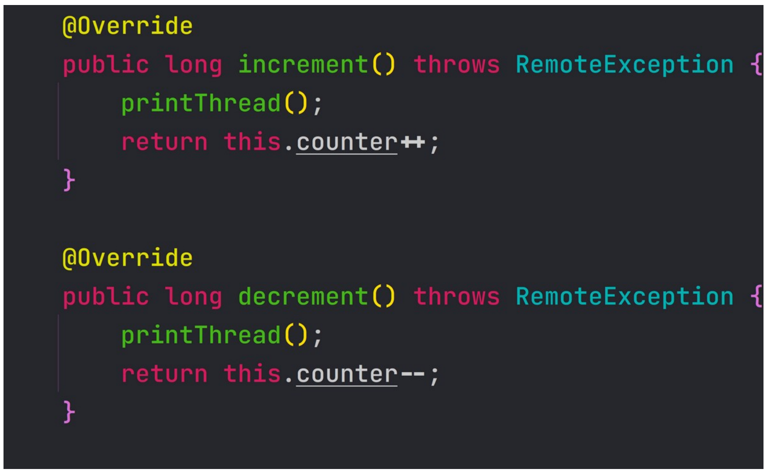
In the previous image, the server is running in the left terminal and the client in the right one. We can see that the remote object is instantiated and registered in the *rmiregistry* of the server. In the client, it is accessed in order to execute the methods. There are 4 connection lines in the server because in the increment and decrement methods of the object implementation is printed the thread name for that connection (see Figure 2). If we have annother connection, the output shows something like this:



```
run_server:
     [java] -------------------------------------------------
     [java] - Remote objects instantiated                   -
     [java] - Remote objects registered (Java RMI registry) -
     [java] - Server running                                -
     [java] -------------------------------------------------
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(2)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(4)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(4)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(4)-127.0.0.1
     [java] Thread name:  RMI TCP Connection(4)-127.0.0.1
```

Figure 2: 2 connections to the server

The new implementation of the remote object class is the following. We have a class attribute that is going to be incremented or decremented when calling the respective methods.



```
@Override
public long increment() throws RemoteException {
    printThread();
    return this.counter++;
}

@Override
public long decrement() throws RemoteException {
    printThread();
    return this.counter--;
}
```

Figure 3: New implementation

Highlight the core difficulties involved in this tiny distributed project.

A core difficulty of this project is to deploy and configure the server in Paloalto and accesing it from our home network. We have to upload the project to our account folder in paloalto.unileon.es server and then compile all the necessary files to run the server. When trying to connect from our home, we must lookup for an object that is instantiated in the server with an specific name (defined in build.xml file).

Compose a more detailed explanation about how this instance of rmiregistry is accessed from anywhere in Internet.

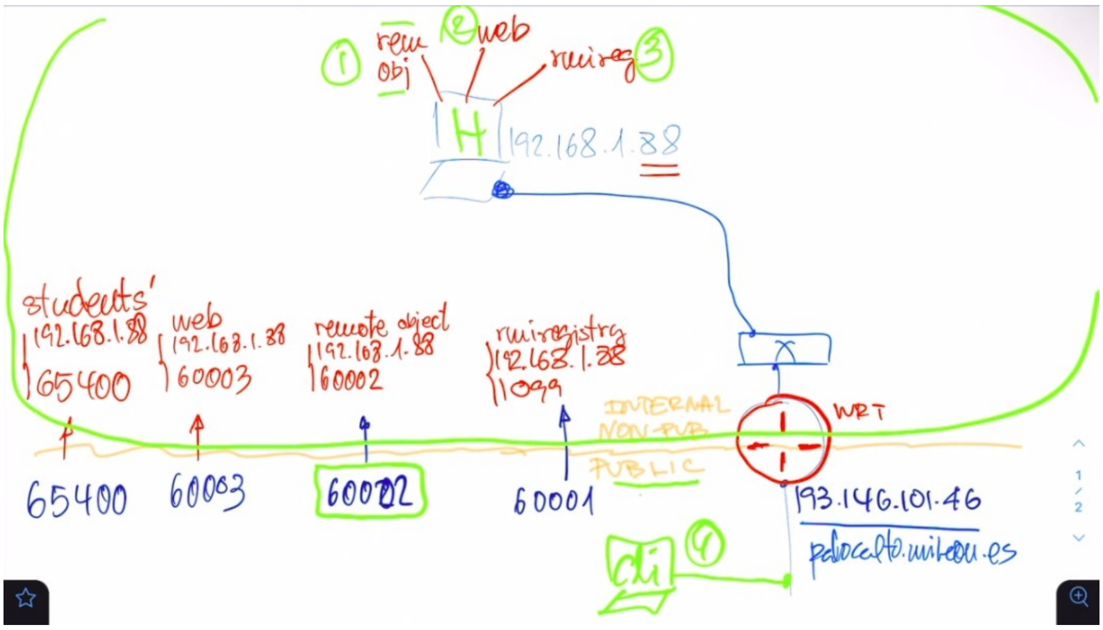


Figure 4: Configuration for the exercise (Author: José María Foces Morán)

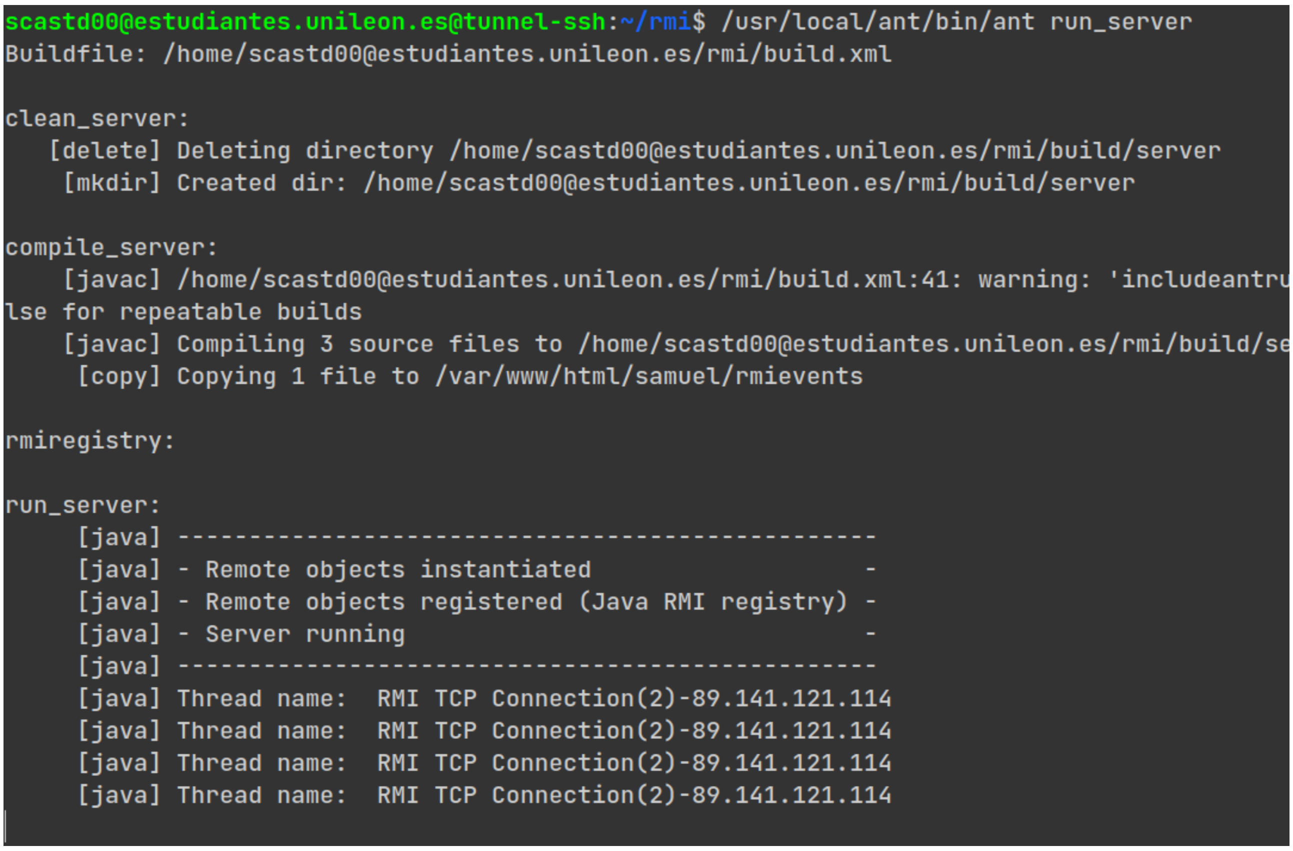
The previous figure represents the physical configuration of the laboratory for this exercise. The host at the top has several applications running on it.

Lab administrator wants to avoid that all the ports of the host H are exposed to the Internet. How can the router do this job? It applies **NAPT** (Network Address and Port Translation). **Translates** the traffic that arrives to the router by following the next mapping table:

	Public IP	Public Port	Private IP	Private Port
rmiregistry	193.146.101.46	60001	192.168.1.88	1099
Remote object		60002		60002
Web		60003		60003
Students' ports		65400-65500		65400-65500

So, we can access from anywhere in Internet to the web server that contains all the remote objects created in it.

Here is a screenshot of the project server running in paloalto. It was contacted by a client (me in my home network).



```
scastd00@estudiantes.unileon.es@tunnel-ssh:~/rmi$ /usr/local/ant/bin/ant run_server
Buildfile: /home/scastd00@estudiantes.unileon.es/rmi/build.xml

clean_server:
   [delete] Deleting directory /home/scastd00@estudiantes.unileon.es/rmi/build/server
    [mkdir] Created dir: /home/scastd00@estudiantes.unileon.es/rmi/build/server

compile_server:
    [javac] /home/scastd00@estudiantes.unileon.es/rmi/build.xml:41: warning: 'includeantru
lse for repeatable builds
    [javac] Compiling 3 source files to /home/scastd00@estudiantes.unileon.es/rmi/build/se
     [copy] Copying 1 file to /var/www/html/samuel/rmievents

rmiregistry:

run_server:
     [java] -------------------------------------------------
     [java] - Remote objects instantiated                   -
     [java] - Remote objects registered (Java RMI registry) -
     [java] - Server running                                -
     [java] -------------------------------------------------
     [java] Thread name:  RMI TCP Connection(2)-89.141.121.114
     [java] Thread name:  RMI TCP Connection(2)-89.141.121.114
     [java] Thread name:  RMI TCP Connection(2)-89.141.121.114
     [java] Thread name:  RMI TCP Connection(2)-89.141.121.114
```

Figure 5: Remote Method Invocation in Paloalto